PANTHÉON-ASSAS
UNIVERSITÉ
PARIS

Thèse de doctorat en INFORMATIQUE

Extraction automatique d'arguments par le biais de grands modèles de langage adaptés

Argument Mining with Customized and Feature-Injected Large Language Models

Présentée par UMER MUSHTAQ

LABORATOIRE D'ÉCONOMIE MATHÉMATIQUE ET DE MICROÉCONOMIE APPLIQUÉE (LEMMA)
ÉCOLE DOCTORALE D'ECONOMIE, GESTION, INFORMATION ET COMMUNICATION (ED 455)
UNIVERSITÉ PARIS-PANTHÉON-ASSAS

Octobre 2023

Contents

List of Figures

List of Tables

Acknowledgements

At the end of this challenging but fulfilling journey, I would like to take the opportunity to extend my sincere appreciations to all those who have played an integral role in the successful completion of my Ph.D.

First of all, I want to express my deep and heartfelt gratitude to my Ph.D. supervisor, Jérémie Cabessa (DAVID, Université Versailles Saint-Quentin-en-Yvelines). Over these years, your support, guidance and motivation has been absolutely crucial in helping me progress and complete this Ph.D. With your great intellectual depth and kind inter-personal demeanor, you are the best supervisor and mentor any young researcher could wish for and I am proud to have worked and collaborated with you on my research. I also want to acknowledge your patience and determination through the inevitable highs and lows of a Ph.D. journey.

I would also like to thank my co-authors, Florence Bannay and Romain Guillaume (IRIT, Université Toulouse III - Paul Sabatier) for your valued contribution to my research work. I also want to express my appreciation for the help and support you extended to me during my first weeks and months in France.

To all the respected and valued members of my Ph.D. thesis jury, Farah Benamara Zitoune (IRIT, Université Toulouse III - Paul Sabatier), Elena Cabrio (INRIA, Université Côte d'Azur), Florence Dupin de Saint-Cyr – Bannay (IRIT, Université Toulouse III - Paul Sabatier) and Maria Rifqi (LEMMA, Université Paris-Panthéon-Assas): I want to express my sincere gratitude to every one of you for your invaluable time, expertise, and thoughtful evaluation of my research work. Your constructive input and insightful feedback have been instrumental in shaping and refining this thesis and I appreciate that very much.

I also want to thank Labex France, Laboratoire d'économie mathématique et de microéconomie appliquée (LEMMA), Université Paris-Panthéon-Assas and the Czech Academy of Sciences for their generous funding of my Ph.D. research.

Finally, I would like to thank my family: my late father, my mother, my sister, my brother and his family and other members of our extended family. Without your continuous and unflinching support and encouragement, none of this would have been possible. I am eternally indebted to all of you and I love you all very much. I also want to extend my appreciation to my friends for their unwavering and consistent support.

Abstract (français)

Le traitement automatique du langage naturel (NLP) est un champ de l'intelligence artificielle d'une importance majeure. Dans ce domaine, les transformeurs et les modèles de langage de grandes tailles (LLMs) subséquents ont représenté un changement de paradigme significatif. Dans le cadre du NLP, l'extraction d'arguments (AM) se concentre sur la détection automatique d'arguments structurés et de leurs relations au sein d'un corpus textuel. L'extraction d'arguments possède de multiples applications, telles que l'analyse de textes juridiques ou d'articles scientifiques.

Ce travail de thèse concerne l'extraction d'arguments (AM) et les modèles de langage de grande taille (LLMs). Notre première contribution concerne la classification d'arguments. Nous proposons un modèle unifié de type BERT enrichi de caractéristiques contextuelles, structurelles et syntaxiques additionnelles, données sous forme de texte. Notre deuxième contribution concerne également la classification d'arguments ainsi que l'identification des liens entre ces derniers. Cette fois, nous proposons BERT–MINUS, un modèle composé de plusieurs sous-modules de type BERT, capable d'intégrer des caractéristiques additionnelles sous forme de texte ainsi que d'accomplir du transfert d'apprentissage. Notre troisième contribution se situe à la jonction des systèmes argumentatifs et de la théorie de la décision. Dans ce contexte, nous définissons un formalisme appelé "Bipolar Layered Framework with Support and Weights" (BLFSW), qui offre une modélisation des structures argumentatives sous forme de graphes. L'expressivité étendue de ce formalisme permet à la fois d'affiner l'évaluation des processus décisionnelles.

Mots-clés: Intelligence artificielle (IA), Apprentissage automatique, Traitement automatique du langage (TALN), Extraction automatique d'arguments, Classification de texte, Grands modèles de langage, BERT, Apprentissage par transfert, Features as Text (FeaTxt), Prompt ingénierie.

Abstract (English)

Natural Language Processing (NLP) is a field of artificial intelligence of major importance. In this domain, Transformers and Large Language Models (LLMs) represent a significant paradigm shift. By enabling parallel processing and capturing long-range dependencies, LLMs have achieved state-of-the-art results in NLP tasks. Argument Mining (AM) focuses on the automatic detection of structured arguments and their relationships within a textual corpus. Argument Mining has multiple applications, such as the analysis of legal texts or scientific articles.

This PhD thesis focuses on Argument Mining (AM) and Large Language Models (LLMs). Our first contribution concerns argument classification. We propose a unified BERT model enriched with additional contextual, structural, and syntactical features provided – in textual form – instead of the usual numerical form. Our second contribution focuses on argument classification and argument relation identification tasks. In this context, we propose BERT–MINUS, a model composed of multiple BERT sub-modules capable of integrating additional text-based features as well as enabling transfer learning between the two tasks. Our third contribution lies at the intersection of argumentative systems and decision theory. Here, we define a framework called "Bipolar Layered Framework with Support and Weights" (BLFSW), which provides a graph-based modeling for argumentation-based decisions. The extended expressiveness of this formalism allows for a refined evaluation of decision-making processes.

Keywords: Artificial Intelligence (AI), Machine Learning, Natural Language Processing (NLP), Argument Mining, Text Classification, Large Language Models (LLMs), BERT, Transfer Learning, Features as Text (FeaTxt), Prompt Engineering.

Résumé

Le traitement automatique du langage naturel (Natural Language Processing - NLP) est un domaine de recherche actuel d'une importance majeure en intelligence artificielle. Le NLP vise à doter des machines de la capacité à synthétiser, traiter et comprendre le langage naturel. Le NLP utilise un large éventail de méthodes, telles que l'apprentissage automatique, l'apprentissage profond et la modélisation statistique, pour traiter et analyser les données textuelles. Depuis son application pionnière dans le domaine de la traduction automatique des langues, les modèles de NLP sont devenus essentiels dans des domaines aussi divers que la reconnaissance vocale, la génération de langage naturel et les systèmes de recommandation. D'autre part, les modèles de langage de grandes tailles (Large Language Models - LLMs) sont des modèles d'apprentissage profond basés sur l'architecture dite des transformeurs. Ces modèles sont fortement parallélisables et capables de capturer des dépendances de long terme au sein des données. Les LLMs sont entraînés de manière auto-supervisée sur de vastes corpus de données non-annotées, ce qui leur permet de capturer des motifs statistiques, syntaxiques et sémantiques du langage naturel, et ainsi d'apprendre des représentations extrêmement riches de données textuelles. Les LLMs ont révolutionné le domaine du NLP en atteignant des performances de pointe dans une grande variété de tâches de compréhension et de génération du langage naturel.

Le développement de modèles informatiques efficaces et robustes pour l'analyse du raisonnement humain est une préoccupation centrale de l'intelligence artificielle. À la jonction de la représentation et du raisonnement des connaissances (KRR), de la linguistique computationnelle (LC) et du NLP, le domaine de l'extraction d'arguments (Argument Mining - AM) se focalise sur la détection automatique d'arguments structurés et de leurs relations au sein d'un corpus textuel. L'extraction des arguments traite divers types de données, qui vont de textes juridiques relativement structurés jusqu'à des données plus désorganisées provenant de plateformes telles X/Twitter. Un pipeline complet d'AM ingurgite un texte brut en entrée et produit une représentation graphique des arguments de ce texte, ces derniers étant connectés par des relations de type preuve, justification ou réfutation. Cette représentation graphique peut, par la suite, être fournie à des systèmes d'analyse de raisonnement qui permettent d'en extraire des inférences rationnelles. L'extraction d'arguments concerne diverses tâches de NLP telles que la classification de mots et de textes, ainsi que la prédiction de relations entre entités textuelles.

Les transformeurs, ces modèles géant d'apprentissage profond introduits par Google Brain en 2017, représentent une révolution dans le domaine du NLP. Un transformeur est constitué d'une architecture encodeur-décodeur feed-forward augmentée d'un mécanisme d'attention extrêmement performant. L'encodeur construit progressivement une représentation contextuelle enrichie de la séquence d'inputs. Le décodeur, quant à lui, utilise ce vecteur de contexte pour produire une séquence d'outputs décodée, pas à pas. Le mécanisme d'attention pondère les représentations cachées des inputs afin de que le modèle puisse se concentrer efficacement sur les parties les plus pertinentes de ces dernières. Les transformeurs peuvent traiter les data en parallèle, ce qui accélère considérablement leur temps d'entraînement et d'inférence, et sont capables de capturer des relations de dépendances de longues portées au sein des leurs inputs textuelles. Plus récemment, les transformeurs ont été adaptés au traitement de données multimodales, telles que le texte et les images. Le paradigme "pré-entraînement et affinage" (pre-training, fine-tuning) utilisé par les LLMs de types transformeurs, tels que BERT et GPT, permet un transfert d'apprentissage extrême-

ment performant entre les tâches de pré-entraînement (pre-training) et les tâches d'affinage (fine-tuning) ultérieures. Les LLMs de types transformeurs ont atteint des capacités de pointe dans une large gamme de tâches de NLP.

Au-delà des paradigmes d'apprentissage supervisé et du "pré-entraînement et affinage", l'apprentissage basé sur des requêtes (prompt-based learning) est considéré comme le troisième paradigme d'importance majeure dans le domaine du NLP. Par le biais de la nouvelle stratégie qualifiée de "pré-entraînement, requête, prédiction" (pre-trained, prompt, predict), une tâche de NLP peut être reformulée en une seconde tâche sur laquelle le LLM est déjà pré-entraîné. Par exemple, avec l'aide d'une requête textuelle (prompt) appropriée, une tâche de classification de texte peut être conçue comme une tâche de modélisation de langage masqué (MLM) ou de prédiction de phrase suivante (NSP). Cette stratégie présente deux avantages importants. Premièrement, les données massives de pré-entraînement peuvent être ré-utilisées telles quelles pour résoudre de nouvelles tâches, dans un contexte d'apprentissage parsimonieux (few-shot learning). Deuxièmement, l'utilisation de requêtes (prompts) élimine la nécessité d'un raffinage ultérieur (fine-tuning) du LLM sur les nouvelles tâches.

Ce travail de thèse se concentre sur l'extraction d'arguments (AM) et les modèles de langage de grande taille (LLMs). Notre première contribution concerne la tâche essentielle que représente la classification d'arguments en AM. La classification des composantes argumentatives fait référence à la classification d'arguments en deux classes principales: affirmations ou prémisses. Dans ce contexte, la construction de représentations enrichies des composantes argumentatives revêt une importance cruciale pour l'utilisation de modèles basés sur les transformeurs. Cependant, il est important de noter que la considération de la composante argumentative seule ne suffit pas pour parvenir à construire une telle représentation, et donc, pour prédire avec précision la nature argumentative de ladite composante. De fait, la considération de caractéristiques lexicales, contextuelles et structurelles supplémentaires sont nécessaires. Face à ce constat, nous proposons un modèle unifié pour la classification d'argument basé sur le modèle BERT (Bidirectional Encoder Representations from transformeurs) et inspiré du nouveau paradigme en terme de requêtes (prompt engineering) en NLP. Notre modèle associe la composante argumentative à des caractéristiques contextuelles, structurelles et syntaxiques additionnelles, toutes données *sous forme de texte*, au lieu de la forme numérique habituelle. Cette nouvelle technique permet à BERT de construire une représentation enrichie pertinente des composantes argumentatives. Nous évaluons notre modèle sur trois datasets qui reflètent une diversité textuelle, entre discours écrits et parlés. Nous obtenons des résultats de pointe sur deux datasets et atteignons une performance de 95% des meilleurs résultats sur le troisième. Notre approche montre que BERT est capable d'exploiter des informations non textuelles, alors que ces dernières sont données de manière textuelle.

Notre deuxième contribution concerne la classification des composantes argumentatives ainsi que l'identification des liens entre ces dernières. En nous inspirant de méthodes de représentation précédentes basées sur l'encodage d'intervalles textuels, nous proposons BERT–MINUS, un modèle modulaire capable de considérer des caractéristique additionnelles sous forme de texte, ainsi que de transfert d'apprentissage entre diverses tâches d'AM. Le modèle BERT–MINUS se compose d'un module conjoint dédié au traitement du des paragraphes entiers, ainsi que d'un module dédié, composé de trois sous-modèles personnalisés de type BERT, qui contextualisent les marqueurs argumentatifs, les arguments eux-mêmes, et les caractéristiques supplémentaires données sous forme de texte. Le modèle BERT–MINUS implémente deux types de transfert d'apprentissage: l'auto-transfert (transfert d'une tâche vers elle-même) et le cross-transfert (transfert classique, d'une tâche à une autre), et ce par le biais d'un nouveau mécanisme d'affinage sélectif (selective fine-tuning). Le modèle BERT–MINUS atteint des résultats de pointe sur la tâche d'identification des liens argumentatifs et des résultats compétitifs sur la tâche de classification des composants argumentatives. La synergie entre les caractéristiques non-textuelles données sous forme de texte et les mécanismes d'affinages sélectifs améliore significativement les performances du modèle. Notre travail met en évidence l'importance et le potentiel du transfert d'apprentissage par affinage sélectif pour les LLMs modulaires. De plus, cette étude s'intègre naturellement dans le paradigme à base de requêtes en NLP (prompt engineering).

Notre troisième contribution se situe à la jonction des systèmes argumentatifs et de la théorie de la décision, deux domaines qui s'entrecroisent volontiers dans des contextes de raisonnements explicables. En effet, il est courant que des agents décisionnels fassent appel à des systèmes argumentatifs pour modéliser des informations incertaines complexes, avant d'appliquer des outils de théorie de la décision servant à sélectionner une séquence d'actions optimale, en fonction des résultats du processus argumentatif. Le processus de décision est dit explicables lorsque les données, la logique et les règles qui aboutissent au résultat décisionnels sont compréhensibles et interprétables par les humains. Notre travail se situe dans le domaine de la prise de décisions explicables, dans le context spécifique d'une connaissance incomplète. Plus précisément, nous définissons un formalisme appelé 'Bipolar Layered Framework with Support and Weights' (BLFSW), qui correspond à une modélisation des structures argumentatives sous forme de graphes. Ce modèle contient également des informations sur les utilités/non-utilités des structures argumentatives. Ce formalisme étend celui des 'Bipolar Layered Frameworks' en permettant l'expression de relations de *soutiens* des principes qui sous-tendent les prises de décisions, et en offrant à l'agent la possibilité de moduler la force des inhibiteurs et des soutiens par l'entremise d'un processus de *pondération.* L'expressivité étendue de ce formalisme permet à la fois d'affiner l'évaluation des alternatives décisionnelles et d'être plus synthétique. Le principal résultat de ce travail est de permettre une automatisation de l'explication d'un contexte décisionnel en terme de BLFSW, un formalisme qui permet de rendre explicite les principes qui sous-tendent les prises de décisions.

Summary

Natural Language Processing (NLP) is a popular and highly-influential current area of research in Artificial Intelligence. NLP seeks to endow a machine with the ability to synthesize, process and understand human produced language and text. NLP utilizes a wide range of methods, such as machine learning, deep learning, and statistical modeling, to process and analyze language data. From its pioneering application in automated machine language translation, NLP systems have become essential for such diverse areas as speech recognition, language generation, and recommender systems. Large Language Models (LLMs) are deep learning models based on the highly parallelizable and long dependency-capturing Transformer architecture. LLMs are trained in a self-supervised manner on massive corpus of unlabeled data, which enables them to model the statistical patterns, syntax and semantics of human language as well as to learn rich, context-aware word representations and features of text. LLMs have revolutionized the field of NLP by achieving state-of-the-art performance on a wide range of language understanding and generation tasks.

Developing efficient and robust computational models for human reasoning is a core concern of Artificial Intelligence. At the confluence of knowledge representation and reasoning (KRR), computational linguistics (CL) and NLP, Argument Mining (AM) involves the automated detection of structured arguments and their relations in text. Argument mining deals with diverse data sources, from relatively structured legal domain to web sources like X/Twitter. A complete end-to-end AM pipeline takes raw text as input and produces a structured map with connected arguments, evidences/warrants and rebuttals, which can then be fed into task-specific reasoning engines to draw rational inferences. Argument mining involves several NLP tasks such as token classification, text classification, and relation prediction.

The Transformer architecture, introduced in 2017 by Google Brain, has been a game changer for NLP. The Transformer model utilizes a feedforward encoder-decoder structure enhanced with an attention mechanism. The encoder processes input words one by one, progressively building a context vector that represents the entire input sequence's embedding. Subsequently, the decoder utilizes this context vector to produce a sequence of decoded words, following a step-by-step approach. The attention mechanism allocates weights to all the input hidden representations in order to effectively discern the most contextually salient part of the input data. Transformers can therefore process text in parallel, thereby significantly speeding up training and inference times, and are able to capture long-range dependencies and relationships in text. More recently, Transformers have been adapted to handle multimodal data, such as text and images. The 'pre-train, fine-tune' paradigm employed by Transformer-based LLMs, such as BERT and GPT, enables transfer learning between pre-training tasks and downstream fine-tuning tasks. Transformer-based LLMs have achieved state-of-the-art results across a wide range of NLP tasks.

After the pioneering supervised learning and the 'pre-train, fine-tune' paradigms, prompt-based learning is considered as the third paradigm in NLP. In what has been dubbed as the 'pre-train, prompt, predict' strategy, a downstream task is reformulated as one on which the LLM is pre-trained. For example, with the help of a suitable textual prompt, a text classification task can be reformulated as a Masked Language Modeling (MLM) or Next Sentence Prediction (NSP) task. This strategy has two vital benefits: firstly, the massive data available for pre-training tasks can be utilized to solve downstream tasks in a few-shot learning setting. Secondly, prompting eliminates the necessity of fine-tuning the LLM on downstream tasks.

Our work in this doctoral thesis focuses on Argument Mining (AM) and Large Language Models (LLMs). Our first contribution focuses on the integral AM sub-task of argument component classification. Argument component classification refers to the classification of arguments as claims or premises. In this context, the construction of rich and meaningful textual representations of the argument components are of vital importance in using Transformer-based models. However, the content of the component alone does actually not suffice to build such an enriched representation, and thus, to accurately predict its corresponding class. In fact, additional lexical, contextual, and structural features are needed. We propose a unified model for argument component classification based on the Bidirectional Encoder Representations from Transformers (BERT) LLM and inspired by the new prompting NLP paradigm. Our model incorporates the component itself together with contextual, structural and syntactic features – given *as text* – instead of the usual numerical form. This new technique enables BERT to build a customized and enriched representation of the components. We evaluate our model on three datasets that reflect a diversity of written and spoken discourses. We achieve state-of-art results on two datasets and 95% of the best results on the third. Our approach shows that BERT is capable of exploiting non-textual information given in a textual form.

Our second contributions concerns the argument component classification and the link identification sub-tasks of AM. Taking inspiration from previously introduced span representation methods, we propose BERT–MINUS, a modular, feature-enriched and transfer learning enabled model for AM. BERT–MINUS consists of: 1) a joint module which embeds the paragraph text, and 2) a dedicated module, composed of three customized BERT models, which contextualize the argument markers, argument components and additional features given as text, respectively. BERT–MINUS implements two kinds of transfer learning – auto-transfer (transfer from a task to itself) and cross-transfer (classical transfer from one task to another) – by means of a novel selective fine-tuning mechanism. BERT–MINUS achieves state-of-the-art results on the link identification task and competitive results on the argument component classification task. The synergy between the features as text and the selective fine-tuning mechanisms significantly improves the performance of the model. Our work reveals the importance and potential of transfer learning via selective fine-tuning for modular large language models. Moreover, this study dovetails naturally into the prompt engineering paradigm in NLP.

Our third contribution lies at the confluence of Argumentation Systems and Decision Theory, which often intersect in real life explainable reasoning approaches. In fact, decision-makers often use argumentation systems to model complex, uncertain information and then apply decision theory to select the most optimal course of action based on the outcomes of the argumentation process. Explainable decisions refer to a decision-making process in which the rationale behind the outcome, such as data, rules, and logic, is understandable and interpretable by humans. Our work concerns the problem of explainable decisions in a context of incomplete knowledge. For this purpose, we define a framework called Bipolar Layered Framework with Support and Weights (BLFSW) that represents the set of argument graphs that can be used in the domain, enabling us to compute the different results that can be obtained in various decision situations. This framework also contains information about the utilities/disutilities of these tangible results. This work extends Bipolar Layered Frameworks by enabling the expression of *supports* for decision principles and by

giving the user the possibility to fix the strength of inhibitors and supports with *weights*. This increased expressiveness of the framework is important both for refining the evaluation of alternatives and to improve the compactness of the representation. The main result of this paper is to provide an automatic way to explain a possibilistic decision setting in terms of a BLFSW which makes explicit the principles that govern the decision.

Part I

Theoretical Background

Chapter 1

Machine Learning

1.1 Introduction

Machine Learning is a field of study that focuses on developing algorithms and statistical models that enable computer systems to automatically learn and improve from experience, without being explicitly programmed. Machine Learning algorithms learn from data, by identifying patterns, regularities, and relationships within large datasets, and using them to make predictions or decisions. Machine Learning has numerous applications across various domains, including but not limited to computer vision, natural language processing, data analytics, and robotics. The field has rapidly evolved in recent years, driven by advancements in computing power, big data, and deep learning techniques, enabling machines to learn and perform increasingly complex tasks.

In algorithmic terms, a machine learning algorithm is a process of optimizing a mathematical model based on observed data. This process involves selecting a set of features, or variables, that are relevant to the problem at hand, defining a cost function that measures the error or difference between the predicted output of the model and the actual output in the data, and then iteratively updating the model parameters to minimize the cost function. This optimization process is typically achieved using a technique called gradient descent, which involves computing the gradient of the cost function with respect to the model parameters and adjusting the parameters in the direction of the negative gradient. This process is repeated until the model achieves a satisfactory level of accuracy or the cost function converges to a minimum. Once the model has been trained on a set of data, it can be used to make predictions on new, unseen data. The performance of the model is evaluated using metrics such as accuracy, precision, recall, and F1 score, depending on the nature of the problem.

1.2 Machine Learning Paradigms

Machine learning has emerged as a significant field in computer science, with a wide range of applications in various crucial domains. The core concept of machine learning is to develop algorithms that can learn from data and make predictions without being explicitly programmed. There are various machine learning paradigms, each with their unique learning strategies and underlying assumptions. These paradigms include supervised learning, unsupervised learning, reinforcement learning, and semi-supervised learning. Understanding the strengths and limitations of each paradigm is crucial for designing efficient machine learning models for a given task. In this section, we provide an overview of the different paradigms in machine learning, their key characteristics, and their applications.

Supervised Learning

In the *supervised learning* paradigm, the machine learning model is trained on labeled data, where each input sample is associated with a corresponding output label or target variable. The goal of supervised learning is to learn a mapping between the input features and the

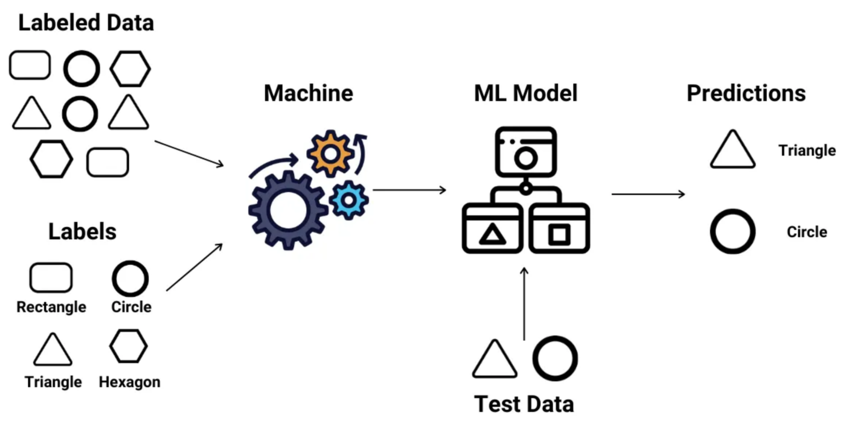


Figure 1.1: A supervised learning algorithm [148]. The input data for the supervised learning algorithm consists of data points $(\mathbf{x_i}, y_i)$ where $\mathbf{x_i}$ is the input features vector and y_i is the corresponding target. The supervised learning model learns the mapping between the input features vectors and the targets. For example, the input vector here consists of the coordinates of the points in a 2D plane for a given geometric shape and the target consists of the shape these points correspond to. The machine learning model learns the mapping between the input data and the targets. This model can then be used to compute inferences on unseen (test) data.

output labels, so that the model can accurately predict the output for new, unseen input data. The target variables can be either discrete (quantitative or categorical) or continuous (qualitative). When target variables are of the former kind, we call this a classification problem whereas in the latter case we call it a regression problem. Examples of supervised learning include image classification, speech recognition, and spam detection. The supervised learning process is depicted in Figure 1.1.

Suppose we are given a training dataset $\mathbf{X} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \ldots, n\}$ where $\mathbf{x}_i$ denotes the i-th training sample and y_i denotes its corresponding label. The purpose of a supervised learning problem is to learn a function

$$f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}$$

which depends on parameters $\boldsymbol{\theta}$ and maps any data $\mathbf{x}$ to a corresponding label $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$. The learning of the parameters $\boldsymbol{\theta}$ is achieved by minimizing an objective function of the form:

$$\mathcal{L}\left(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i\right) + \lambda\Omega(\boldsymbol{\theta})$$

where $\mathcal{L}$ represents a predefined loss function measuring the error between the predicted label $\hat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$ and the real label y_i, respectively, and Ω is a regularization term with weight λ. The optimal parameters $\boldsymbol{\theta}^*$ are therefore given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) + \lambda\Omega(\boldsymbol{\theta})\,.$$

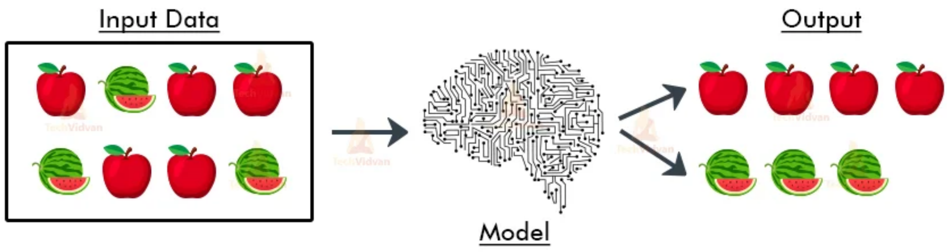


Figure 1.2: An unsupervised learning algorithm (TechVivdan). The input data for the unsupervised learning algorithm consists of unlabeled data points. The unsupervised learning model learns to identify structure and patterns in the data. For example, the input data here consist of apples and watermelons, and the machine learning model learns to discriminate the former from the latter without using labels.

Unsupervised Learning

Unsupervised learning is a machine learning paradigm where the model is trained on data that does not have any labeled outputs. Unlike supervised learning, where the model is trained on a labeled dataset to predict an output, unsupervised learning algorithms work on raw, unlabeled data to identify patterns and relationships within the dataset. Unsupervised learning is an important area of machine learning, as it allows machines to identify structures and patterns in data, and make predictions without any human intervention or guidance. The most common unsupervised learning techniques include clustering, dimensionality reduction, and anomaly detection. The unsupervised learning process is depicted in Figure 1.2.

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a dataset. Each $\mathbf{x}_i \in \mathbb{R}^d$ is a data point in some feature space of dimension d that is not associated with any label. The goal of unsupervised learning is to find a parametric transformation

$$f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}^{d'}$$

which maps any data $\mathbf{x}$ into into a new representation $\mathbf{z} \in \mathbb{R}^{d'}$, usually of smaller dimensionality $d' < d$. The parameters $\boldsymbol{\theta}$ are also generally obtained by minimizing of some criterion which depends on the unsupervised algorithm that is considered. The transformed dataset $\mathbf{X'} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$, where $\mathbf{z}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$, should reveal structures and patterns in the data that were not identifiable in the original dataset $\mathbf{X}$ [216].

Reinforcement Learning

Reinforcement learning is a type of machine learning where an agent learns to make decisions based on the feedback he receives from its environment. Unlike supervised learning, where the model is trained on a labeled dataset to predict an output, and unsupervised learning, where the model is trained on raw, unlabeled data to identify patterns and relationships within the dataset, reinforcement learning algorithms learn through trial and error, and are commonly used in areas such as robotics, gaming, and decision making. In a reinforcement learning setting, a agent interacts with his environment, taking actions and receiving feedback in the form of rewards or penalties. The goal of the agent is to learn a policy, which is a mapping from states to actions that maximizes his expected reward over time.

Specifically, at each interaction time step t, the agent is in a state $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}$, according to a policy $\pi_\theta(s_t) = \pi_\theta(\cdot \mid s_t) : \mathcal{S} \to \mathcal{A}$ parameterized by θ. Then the agent receives an immediate reward $r_t = r(s_t, a_t) \in \mathbb{R}$, where $r(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and moves to the next state s_{t+1} according to the environment dynamics. For each episode, this process continues until the agent reaches a terminal state. The return for each state is the discounted accumulated reward given by $R_t = R(s_t, a_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where $\gamma \in (0, 1]$ is the discount factor. The policy of the agent is learned by maximizing the expectation of the long-term return from each state [216]:

$$\max_\theta \mathbb{E}_{s_t} \left[R_t \mid s_t, a_t = \pi_\theta(s_t) \right] .$$

Reinforcement learning algorithms can be classified into two main categories: value-based and policy-based methods. Value-based methods learn a value function that maps states to expected rewards. The value function is then used to determine the optimal action to take in a given state. Examples of value-based methods include Q-learning and SARSA. Policy-based methods learn a policy function that maps states to actions directly (as presented above). The policy function is then used to determine the optimal action to take in a given state. Examples of policy-based methods include policy gradient methods and actor-critic methods. The reinforcement learning paradigm is illustrated in Figure 1.3.

Semi-Supervised Learning

Semi-supervised learning is a machine learning paradigm that combines both supervised and unsupervised learning techniques. In semi-supervised learning, the model is trained on a dataset that contains both labeled and unlabeled data. The primary goal of semi-supervised learning is to improve the accuracy of the model by utilizing both labeled and unlabeled data. This is particularly useful in situations where labeled data is expensive or difficult to obtain, but unlabeled data is abundant. Some popular techniques for semi-supervised learning include self-training, co-training, and label propagation models, semi-supervised Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

We now suppose that the training dataset is composed of two sub-datasets, a labeled set $\mathbf{X} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i = 1, \ldots, n\}$ and an unlabeled set $\mathbf{X'} = \{\mathbf{z}_i \in \mathbb{R}^p : i = 1, \ldots, m\}$. Typically, the labeled set $\mathbf{X}$ is smaller than the unlabeled one $\mathbf{Z}$. The semi-supervised learning problem can thus be formulated as

$$\boldsymbol{\theta}^*, \boldsymbol{\theta'}^* = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\theta'}} \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i\right) + \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}'\left(f'(\mathbf{z}_i; \boldsymbol{\theta'}), R(\mathbf{z}_i; \mathbf{X})\right) + \lambda \Omega(\boldsymbol{\theta}) \right),$$

where $f(\cdot; \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}$ and $f'(\cdot; \boldsymbol{\theta'}) : \mathbb{R}^p \to \mathbb{R}^{p'}$ are the supervised and unsupervised mappings that depend on parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta'}$, respectively, $\mathcal{L}$ and $\mathcal{L}'$ are the supervised

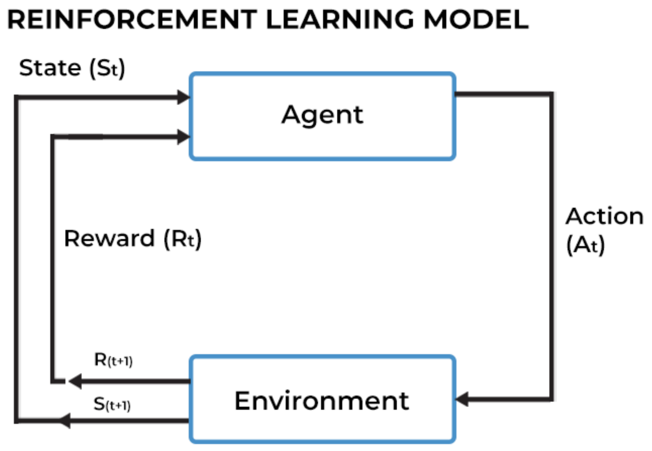


Figure 1.3: Reinforcement Learning [80]. A simple maze game is an example of a Reinforcement Learning algorithm. In this game, the reward function r is the opposite of the solving time, i.e., the longer it takes to solve the maze, the lesser the reward. The set of actions is given by the four directions the agent can move on a maze block, i.e. $\mathcal{A} = \{N, S, E, W\}$, and the set of states comprises all possible positions of the agent on the maze, i.e. $\mathcal{S} = \{s_1, s_2, \ldots s_n\}$. At every time step t, the agent chooses one action from $\mathcal{A}$ based on his expected reward and the environment (the position of the agent and the state) is updated based on this action. The 'reinforcement' the agent receives form the environment enables it to make choices that maximize his reward.

and unsupervised loss functions, respectively, and R is a function that defines pseudo-labels for the unlabeled data which are thus integrated into the end-to-end training process. Note that, while the learning related to labeled data $\mathbf{x}_i$ is based on the associated labels y_i, that related to unlabeled data $\mathbf{z}_i$ that is merely based on their internal properties $R(\mathbf{z}_i; \mathbf{X})$, like the distance or designed pretext tasks for instance [216].

There are several techniques that can be used for semi-supervised learning, including:

- Self-training: This technique involves training the model on a small set of labeled data, and then using the model to make predictions on the unlabeled data. The model's predictions on the unlabeled data are then added to the labeled data, and the model is retrained on the expanded dataset.

- Co-training: This technique involves training multiple models on different subsets of the data, and then using the models to label each other's unlabeled data. This allows each model to learn from the other's labeled data and improve its accuracy.

- Transductive learning: This technique involves using the unlabeled data to improve the model's predictions on the labeled data. The model is trained on the labeled data, and then the unlabeled data is used to refine the model's predictions on the labeled data.

Self-Supervised Learning

Self-supervised learning is a type of machine learning technique in which a model is trained to predict a missing or corrupted part of its input data, without the need for explicit supervision from human annotators. In self-supervised learning, the model uses the structure and patterns in the input data to create its own labels and train itself. In contrast to supervised learning, where labeled examples are used to train a model, self-supervised learning relies on unlabeled data to learn useful representations of the data. By leveraging the inherent structures and relationships in the data, a self-supervised model can learn to perform a wide range of tasks, such as image recognition or natural language processing, without the need for human-labeled training data. Examples of self-supervised learning include predicting missing words in a sentence, predicting the next frame in a video, predicting missing parts of an image (image inpainting). Self-supervised learning is a promising approach for training deep learning models on large amounts of unlabeled data, which is often readily available, and has been shown to outperform supervised learning approaches in some cases.

In self-supervised learning, the task-specific labels or targets are generated from the data itself. More specifically, a so-called pretext task creates a surrogate label for each data point. The model is trained to predict this surrogate label while the true target (usually hidden) serves as the learning signal. A common pretext task is context prediction. Given a data point $\mathbf{x}_i$, a portion of the data is masked, and the model is trained to predict the masked portion from the remaining context.

Formally, let $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^p : i = 1, \dots, n\}$ be a dataset containing n data points. In case of context prediction for instance, any data point $\mathbf{x}_i$ is split into two parts, i.e. $\mathbf{x}_i = [\mathbf{x}_{i,\text{context}}, \mathbf{x}_{i,\text{mask}}]$, where $\mathbf{x}_{i,\text{context}}$ contains the context information and $\mathbf{x}_{i,\text{mask}}$ is the masked portion. The goal of self-supervised learning is then to train a model f that builds a representation $\mathbf{h}_i = f(\mathbf{x}_{i,\text{context}}; \boldsymbol{\theta})$ of the context information, in order to predict the masked portion $\mathbf{x}_{i,\text{mask}}$. Towards this purpose, a loss function $\mathcal{L}'$ of the following form is considered [216]:

$$\mathcal{L}'(\mathbf{x}_i) = \mathcal{L}\left(f(\mathbf{x}_{i,\text{context}}; \boldsymbol{\theta}), \mathbf{x}_{i,\text{mask}}\right) + \lambda\Omega(\boldsymbol{\theta})$$

where $\mathcal{L}$ is a loss function measuring the error between $f(\mathbf{x}_{i,\text{context}}; \boldsymbol{\theta})$ and $\mathbf{x}_{i,\text{mask}}$.

Self-supervised learning has had important applications in NLP. Self-supervised learning is broadly divided into three categories: 1) Contrastive, where an encoder is trained to encode the input into an explicit vector to measure similarity; 2) Generative, where an encoder is trained to encode the input into an explicit vector and a decoder is trained to reconstruct the input from the explicit vector; and 3) Generative-Contrastive (Adversarial): where an encoder is trained to generate fake samples and a discriminator distinguishes them from real samples [109]. Two most popular self-supervised learning models are the auto-encoding model Bidirectional Encoder Representations from Transformers (BERT) and the auto-regressive model Generative Pre-trained Transformer (GPT). The BERT model uses a large amount of unlabeled text data to train a language model to understand the relationships between words in a sentence. The pre-training process involves masking a portion of the input text and training the model to predict the masked words based on the context of the surrounding words. This process enables BERT to learn contextualized word representations that can be fine-tuned for a variety of downstream tasks in natural language processing, such as text classification, and language translation. GPT models, on the other hand, are used for text generations tasks. We will introduce the BERT model in detail in Chapter 7.

Multi-Task Learning

Multi-task learning is a machine learning technique that involves training a single model to perform multiple related tasks simultaneously. In multi-task learning, the model is trained on a shared representation of the input data that is relevant to all the tasks, and also learns task-specific parameters for each task. The shared representation is learned jointly across all tasks, and it captures the common patterns and relationships between the input data and the output variables. By sharing the representation, the model can leverage the shared knowledge between tasks and improve the overall performance of the model on all tasks.

For example, in natural language processing, a multi-task learning model can be trained to perform multiple related tasks such as sentiment analysis, named entity recognition, and text classification, all from the same input text. The model would learn to represent the input text in a way that captures the relevant information for all tasks, and also learns task-specific parameters for each task, allowing it to perform well on each task individually. Multi-task learning has become popular in machine learning, since it can improve the performance of a model on multiple related tasks while reducing the amount of data and computational resources required for training separate models for each task.

Formally, consider $T_1, T_2, \dots, T_T$ to be a set of tasks denoted by T. The training dataset for each task T_t is denoted by $\mathbf{D}_t = \{(\mathbf{x}_{t,i}, y_{t,i}) : i = 1, \dots, n_t\}$, where $\mathbf{x}_{t,i}$ is the input data and $y_{t,i}$ is the corresponding target or label for the i-th example in task T_t. Multi-task learning involves a shared representation $h(\mathbf{x}; \boldsymbol{\theta})$ of the data $\mathbf{x}$ which captures an information that is common to all tasks. The parameters $\boldsymbol{\theta}$ is formed by the concatenation of task-specific parameters $\boldsymbol{\theta}_t$ for all $t = 1, \dots, T$, i.e. $\boldsymbol{\theta} = [\boldsymbol{\theta}_1; \boldsymbol{\theta}_2; \dots; \boldsymbol{\theta}_T]$. In addition to the shared representation, multi-task learning also involves task-specific representations $f_t(h(\mathbf{x}; \boldsymbol{\theta}); \boldsymbol{\theta}_t)$ of the data, where the task-specific component f_t depends on the task-specific parameters $\boldsymbol{\theta}_t$ for $t = 1, \dots, T$. The objective of multi-task learning is to learn the model parameters $\boldsymbol{\theta} = [\boldsymbol{\theta}_1; \boldsymbol{\theta}_2; \dots; \boldsymbol{\theta}_T]$ that optimize the performance across all tasks simultaneously. This can be formulated using a joint objective function of the form:

$$J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \alpha_t \bar{\mathcal{L}}_t(\mathbf{D}_t; \boldsymbol{\theta}_t) = \sum_{t=1}^{T} \alpha_t \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}_t\left(y_{t,i}, f_t(h(\mathbf{x}_{t,i}; \boldsymbol{\theta}); \boldsymbol{\theta}_t)\right)$$

where $\mathcal{L}_t$ is a task-specific loss function that measures the discrepancy between the predicted output and the true target and α_t is a task-specific weight. The optimal parameters $\boldsymbol{\theta}^*$ are then given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) .$$

Transfer Learning

Transfer learning is a machine learning technique where the knowledge gained from solving one problem is applied to solve a different but related problem. The underlying intuition motivating transfer learning is that knowledge gained or patterns recognized while training on one task may be useful for training on the related task. The idea is that the pre-trained model has learned to extract high-level features that are useful for a wide range of related tasks, and by reusing those features, we can improve the performance of the new model.

For example, a pre-trained image recognition model that has been trained on a large dataset of images can be used as a feature extractor for a new task such as detecting tumors in medical images. By fine-tuning the pre-trained model on the new task, the model can learn to recognize specific patterns and features that are relevant to the new task more quickly and accurately than as if it had been trained from scratch. Transfer learning has become a popular technique in machine learning due to its ability to reduce the amount of data and computational resources required for training a new model and improve the performance of the model on a new task.

Formally, let S denote the source task and $\mathbf{D}_S = \{(\mathbf{x}_{S,i}, y_{S,i}) : i = 1, \dots, n_S\}$ be its dataset associated, where $\mathbf{x}_{S,i}$ is the input data and $y_{S,i}$ is the corresponding target or label for the i-th example. Let T denote the target task and $\mathbf{D}_T = \{(\mathbf{x}_{T,j}, y_{T,j}) : j = 1, \dots, n_T\}$ be its associated dataset. The feature space for both source and target tasks is denoted by $\mathbf{X}$. Let $f(\mathbf{x}; \mathbf{T})$ be a model with parameters $\boldsymbol{\theta}$. After the model f has been trained on the source task S, the parameters $\boldsymbol{\theta}_S$ are obtained by minimization of some loss function $\mathcal{L}$, as usual. The goal of transfer learning consists in learning the target task T using the optimal parameters $\boldsymbol{\theta}_S$ of the source task as initial configuration of the model f. The problem can be formalized as finding the optimal target model parameters $\boldsymbol{\theta}_T^*$ that minimize the loss of the target task T:

$$\boldsymbol{\theta}_T^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{n_T} \sum_{j=1}^{n_T} \mathcal{L}\left(y_{T,j}, f(\mathbf{x}_{T,j}; \boldsymbol{\theta})\right),$$

where $\mathcal{L}$ is the loss function that measures the discrepancy between the prediction and the target, and $f(\cdot; \boldsymbol{\theta}_S)$ is the initial training configuration of the model.

1.3 Machine Learning Algorithms

Machine learning has emerged as a transformative technology, enabling computers to learn from data and make intelligent decisions without explicit programming. Therefore, various machine learning algorithms have been proposed and refined over the decades ranging from classic techniques like linear regression and decision trees to more advanced methods such as neural networks and support vector machines. In this section, we will present the main machine learning algorithms in detail.

1.3.1 Linear Regression

Linear regression is a supervised Machine Learning algorithm. Since it is a paradigmatic model, we believe that it is insightful to describe it in detail. Intuitively, linear regression models the relationship between one or more input variables and an output variable in a linear way. In order to learn this relationship, the algorithm makes use of observed data. The linear regression works by fitting a linear function (line, plane, hyperplane) that best fits the data points. This is achieved by minimizing the summed distance between the linear function and the data. A linear regression is illustrated in Figure 1.4.

Formally, let $X_1, \dots, X_p$ by some input random variables, generally called independent variables, predictors or features, and let Y be some output variable referred to as the dependent variable, response or target. We make the strong hypothesis that the true but unknown relationship between the input and the output variables is of the following form:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon \tag{1.1}$$

where ϵ is a random noise with zero mean, i.e., $\mathbb{E}(\epsilon) = 0$. Usually, each coefficient β_i is interpreted as the average effect on Y of a one unit increase in X_i, if all other input variables are kept fixed. Since they are unknown, the regression coefficients $\beta_0, \dots, \beta_p$ need to be estimated by some $\hat{\beta}_0, \dots, \hat{\beta}_p$. These estimations yield the following linear model

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p \tag{1.2}$$

where $\hat{Y}$ is the prediction associated to inputs $X_1, \dots, X_p$, Note that the noise ϵ has been discarded, as it is not intended to be estimated (cf. Equation (1.1)).

For this purpose, we dispose of a set of data points

$$S = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_N}, y_N)\}.$$

where each point $(\mathbf{x_i}, y_i) \in \mathbb{R}^p \times \mathbb{R}$ is a realization of the variables $(\mathbf{X}, Y)$, with $\mathbf{X} = (X_1, \cdots, X_p)$. A key principle of Machine Learning is that the sets of data used to train and to evaluate the performance of the model must be different. In fact, the model might provide good results on the data it has been trained on, but poorer results on data that it has never encountered during training. According to these considerations, the data set S is generally split into two sets in a random way, namely,

$$\begin{aligned} S_{\text{train}} &= \{(\mathbf{x_{i_1}}, y_{i_1}), \dots, (\mathbf{x_{i_n}}, y_{i_n})\} \\ S_{\text{test}} &= \{(\mathbf{x_{j_1}}, y_{j_1}), \dots, (\mathbf{x_{j_m}}, y_{j_m})\} \\ S &= S_{\text{train}} \cup S_{\text{test}} \end{aligned}$$

where S_{train} and S_{test} are called the train set and the test set, and will be used to train and to evaluate the model, respectively.

Note that, for any data point $(\mathbf{x_i}, y_i)$, the prediction $\hat{y}_i$ of the model (1.2) for the input data $\mathbf{x_i}$ is given by

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip} = (1, \mathbf{x_i})\hat{\boldsymbol{\beta}}^T$$

where x_{ij} is the j-th component of $\mathbf{x}_i$ (for $j = 1, \dots, p$), $(1, \mathbf{x_i})$ is the vector $\mathbf{x_i}$ with an additional component 1 appended at the beginning, and $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \dots, \hat{\beta}_p)$. Accordingly, the train set S_{train} can be written in the following matricial form:

$$\mathbf{X}_{\text{train}} = \begin{pmatrix} 1 & \mathbf{x_{i_1}}^T \\ \vdots & \\ 1 & \mathbf{x_{i_n}}^T \end{pmatrix} = \begin{pmatrix} 1 & x_{i_1 1} & \cdots & x_{i_1 p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i_n 1} & \cdots & x_{i_n p} \end{pmatrix} \quad \text{and} \quad \mathbf{y}_{\text{train}} = \begin{pmatrix} y_{i_1} \\ \vdots \\ y_{i_n} \end{pmatrix}.$$

For some model with non-optimal parameters $\boldsymbol{\beta} = (\beta_0, \dots, \beta_p)$, its performance on the train set S_{train} is given by the residual sum of squares (RSS), which represents the sum of squared distances between the predictions $\hat{y}_i$ and the responses y_i:

$$\text{RSS}_{\text{train}}(\boldsymbol{\beta}) = \sum_{k=1}^{n} (\hat{y}_{i_k} - y_{i_k})^2 = \sum_{k=1}^{n} \left(\mathbf{x_{i_k}}^T \boldsymbol{\beta} - y_{i_k}\right)^2 = \|\mathbf{X}_{\text{train}}\boldsymbol{\beta} - \mathbf{y}_{\text{train}}\|^2$$

The residual sum of squares for the test set $\text{RSS}_{\text{test}}(\boldsymbol{\beta})$ is defined analogously.

For simplicity, let $\mathbf{X}_{\text{train}}$ be denoted as $\mathbf{X}$ and $\mathbf{y}_{\text{train}}$ as $\mathbf{y}$. The linear regression finds the parameters $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \dots, \hat{\beta}_p)$ that minimize $\text{RSS}_{\text{train}}(\boldsymbol{\beta})$, namely:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \text{RSS}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|^2 .$$

In order to solve this minimization problem, the gradient of $\text{RSS}_{\text{train}}(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is computed and set to 0:

$$\begin{aligned} \frac{\partial \text{RSS}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \frac{\partial \left(\|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|^2\right)}{\partial \boldsymbol{\beta}} = \frac{\partial \left((\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})\right)}{\partial \boldsymbol{\beta}} \\ &= \frac{\partial \left(\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} + \mathbf{y}^T\mathbf{y}\right)}{\partial \boldsymbol{\beta}} \\ &= 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y}, \\ \frac{\partial \text{RSS}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= 0 \text{ iff } \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y} \text{ iff } \hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

Consequently, the estimations of the real parameters $\boldsymbol{\beta}$ in Equation (1.1) are given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

The model with optimal parameters $\hat{\boldsymbol{\beta}}$ is given by

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p ,$$

and the performance of this model on the train and test sets are given by the values of $\text{RSS}_{\text{train}}(\boldsymbol{\beta})$ and $\text{RSS}_{\text{test}}(\boldsymbol{\beta})$, respectively.

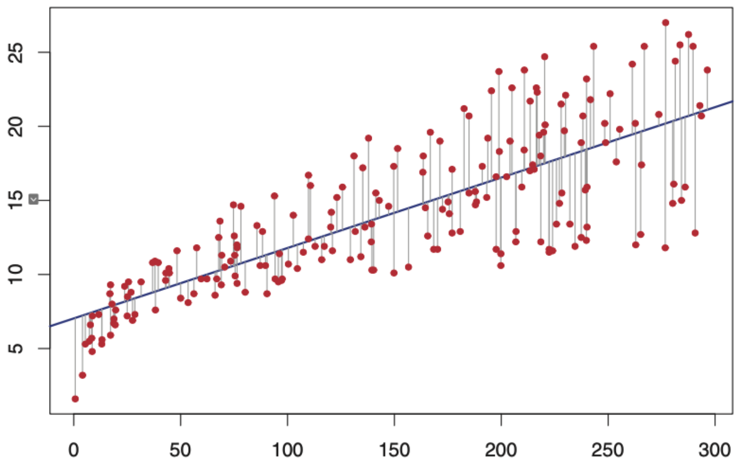


Figure 1.4: Linear regression. Here, a linear relationship between one input variable X (horizontal axis) and an output variable Y (vertical axis) is computed. The red dots represent the data points used to fit the regression. The sum of the vertical lines between the points and the line is the residual sum of squares. The straight line represents the equation of the regression after being trained on the data.

1.3.2 Logistic Regression

Logistic regression is also a supervised Machine Learning classification algorithm. Logistic regression models the relationship between a binary or categorical output variable and several input variables. As opposed to linear regression where the input variable is modeled directly, in logistic regression, the variable is modelled according to the probability of it belonging to a certain class. In formal terms, in the case of a binary output variable Y (taking value 0 or 1), the logistic regression models the relationship between the input variable X and the probability $Pr(Y = 1 \mid X)$. To model this relationship, logistic regression uses the logistic function:

$$p(x) = Pr(Y = 1 \mid X = x) = \frac{\mathrm{e}^{(\beta_0 + \beta_1 x)}}{1 + e^{(\beta_0 + \beta_1 x)}}$$

This logistic function produces an S-shaped curve (see Figure 1.5) where all probabilities are between 0 and 1, thereby ensuring that theoretically sound predictions are obtained for all values of X. Note that the probability $Pr(Y = 0 \mid X = x)$ can easily be obtained from $Pr(Y = 1 \mid X = x)$ by the formula $Pr(Y = 0 \mid X = x) = 1 - Pr(Y = 0 \mid X = x)$. This model ensures that the logit function, the logarithm of the quotient of the two probabilities, is linear:

$$\log\left(\frac{Pr(Y = 1 \mid X = x)}{Pr(Y = 0 \mid X = x)}\right) = \log\left(\frac{\mathrm{e}^{(\beta_0 + \beta_1 x)}}{1 + e^{(\beta_0 + \beta_1 x)}} \cdot \frac{1 + e^{(\beta_0 + \beta_1 x)}}{1}\right) = \beta_0 + \beta_1 x$$

To estimate the coefficients β_0 and β_1, the maximum likelihood function is utilized.

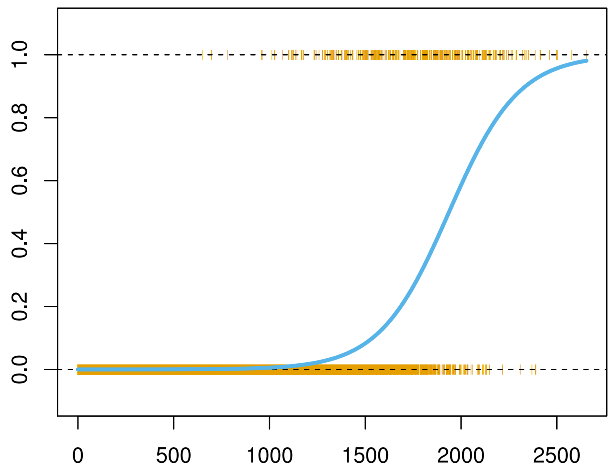


Figure 1.5: Logistic regression. This figure shows the logistic regression function. The x-axis represents the predictor and the y-axis represents the predicted probabilities of the dependent variable using logistic regression. For example, in the *Default* dataset, the predictor variable is the balance of a bank customer and the dependent variable is whether they default on the loan or not. Using logistic regression, the probability of a custom defaulting on their loan, given their balance, is computed and a subjective threshold is applied to classify a custom as likely to default or not [74].

Intuitively, the coefficients are selected such that the predicted probability $p(x_i)$ is as close as possible to it's actual observed class y_i. Formally, the likelihood function is given by:

$$l(\beta_0, \beta_1) = \prod_{\{i:y_i=1\}} p(x_i) \prod_{\{i':y_{i'}=0\}} (1 - p(x'_i))$$

The closer the predicted probabilities $p(x_i)$ are to the actual classes y_i, the larger the likelihood function. Hence, the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize this likelihood function:

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0,\beta_1} l(\beta_0, \beta_1)$$

Unlike the minimization of the residual sum of squares (RSS), the likelihood maximization problem has no closed-form solution. It is generally solved by means of iterative methods, like the expectation–maximization (EM) algorithm, which converge to local maxima.

In order to make predictions on unseen samples of data, the logistic function is used, along with the estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$, to compute the predicted probability for a sample. A threshold value (usually 0.5) can then be used to determine the corresponding predicted class for the sample.

In *multiple logistic regression*, the predictors or independent variables are more than one. Therefore, for a set of p predictors, $X = \{X_1, X_2, \ldots, X_p\}$, the logistic function is given by:

$$p(x_1, \dots, x_p) = Pr(Y = 1 \mid X_1 = x_1, \dots, X_p = x_p) = \frac{\mathrm{e}^{(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}{1 + e^{(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}$$

Once again, this model ensures that the logit function is linear:

$$\log\left(\frac{Pr(Y = 1 \mid X_1 = x_1, \dots, X_p = x_p)}{Pr(Y = 0 \mid X_1 = x_1, \dots, X_p = x_p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

The coefficients $\hat{\beta}_0, \hat{\beta}_1, \dots \hat{\beta}_p$ are computed as above using the maximum likelihood function:

$$\hat{\beta}_0, \dots, \hat{\beta}_p = \arg \max_{\beta_0, \dots, \beta_p} l(\beta_0, \dots, \beta_p)$$

Finally, in *multinomial logistic regression*, the dependent variable can have more than two classes. For a set of $K > 2$ classes, the K-th class is selected as the baseline class and the logistic functions are given by:

$$Pr(Y = k \mid X_1 = x_1, \dots, X_p = x_p) = \frac{\mathrm{e}^{(\beta_{k0} + \beta_{k1} x_1 + \cdots + \beta_{kp} x_p)}}{1 + \sum_{l=1}^{K-1} e^{(\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p)}}$$

for $k = 1, \dots, K - 1$, whereas for $k = K$, it is given by:

$$Pr(Y = K \mid X_1 = x_1, \dots, X_p = x_p) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{(\beta_{l0} + \beta_{l1} x_1 + \cdots + \beta_{lp} x_p)}} .$$

The likelihood function can be generalized to the multinomial context, and the parameters $\hat{\beta}_{li}$ for $l = 1, \dots, K - 1$ and $i = 0, \dots, p$ are obtained by maximizing this function.

1.3.3 Decision Trees

Decision trees are supervised Machine Learning algorithms used for regression and classification. For classification problems (where the target variable has discrete values), the leaves in the classification tree represent the values of the target variable, and the branches represent conjunctions of features in the observed data leading to these target variable values. For regression problems (where the target variable has continuous values), the leaves in the regression tree are the predicted values of the target variable. A decision tree is illustrated in Figure 1.6. Both cases are explained in more detail below.

Regression Trees

Regression trees are decision trees used to solve regression problems. Regression trees work by building a tree structure where every internal node is a predictor variable. At each

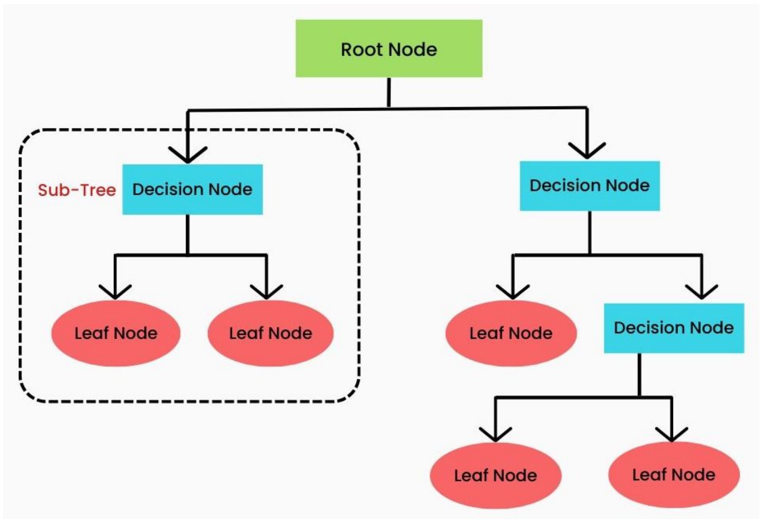


Figure 1.6: An example of a decision tree (turing.com).

internal node, the data is split into a left sub-tree and a right sub-tree based on the value of the predictor variable. In this way, the regression tree divides the data into distinct regions. The leaf nodes are assigned the mean value of the data points that fall in their region. Formally, for a set of predictor variables $X = \{X_1, X_2, \ldots, X_p\}$, a regression tree is built as follows:

- The predictor space, $X_1 \times X_2 \times \cdots \times X_p$ is divided into J distinct and over-lapping regions $R = \{R_1, R_2, \ldots, R_j\}$. By construction, the regions have the shape of high-dimensional rectangles (boxes). More precisely, the regions $\{R_1, R_2, \ldots, R_j\}$ are built so that they minimize the Residual Sum of Squares (RSS), given by:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where $\hat{y}_{R_j}$ is the mean value of all the observations in the region R_j.

- For every observation that falls into the region R_j, the value assigned to it is the mean value of the training observations in the region R_j.

Considering every possible partition of the feature space into J boxes is computationally infeasible. Hence, to avoid computational blowup, a top-down greedy technique called recursive binary splitting is used. Recursive binary splitting works by selecting a predictor X_j and a cutoff value s such that splitting the predictor space into regions $\{X : X_j < s\}$ and $\{X : X_j \geq s\}$ leads to the largest possible reduction in Residual Sum of Squares (RSS). In this way, all predictors $\{X_1, X_2, \ldots, X_p\}$ and all possible cutoff values s for each are considered and then the predictor and cutoff are chosen such that resulting regression tree has the lowest RSS [74]. An example of a regression tree is shown in Figure 1.7.

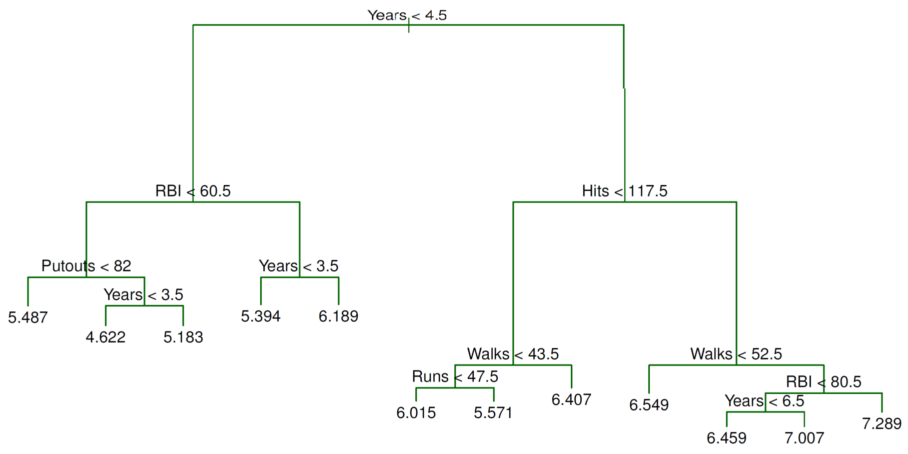


Figure 1.7: Regression Trees [74]. This figure shows the regression tree obtained from the *Hitters* dataset. This dataset consists of salaries of sports players, based on different predictors like years they have played, their hits records and so on. In the tree, the internal nodes represent the predictors and their cutoff values and the leaf nodes represent their salaries (qualitative, continuous).

Classification Trees

Classification trees are decision trees used for predicting categorical variables. Since categorical variables do not have continuous values, the mean of the training observations in a particular region cannot be used as the predicted response. Therefore, in classification trees, the predicted response for an observation is the class which is the most commonly occurring in the region to which the observation belongs.

As with regression trees, recursive binary splitting is also used to build classification trees. Since categorical variables do not have continuous values, RSS cannot be used as a loss function. Instead, classification trees used classification error rate as the criterion. Classification error rate is the fraction of training samples which do not belong to the most commonly occurring class of the region:

$$E = \sum_{m=1}^{M} \left(1 - \max_{k}(\hat{p}_{mk})\right)$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.

In addition to classification rate error, two other metrics are also used: Gini index and entropy which are given, respectively, by:

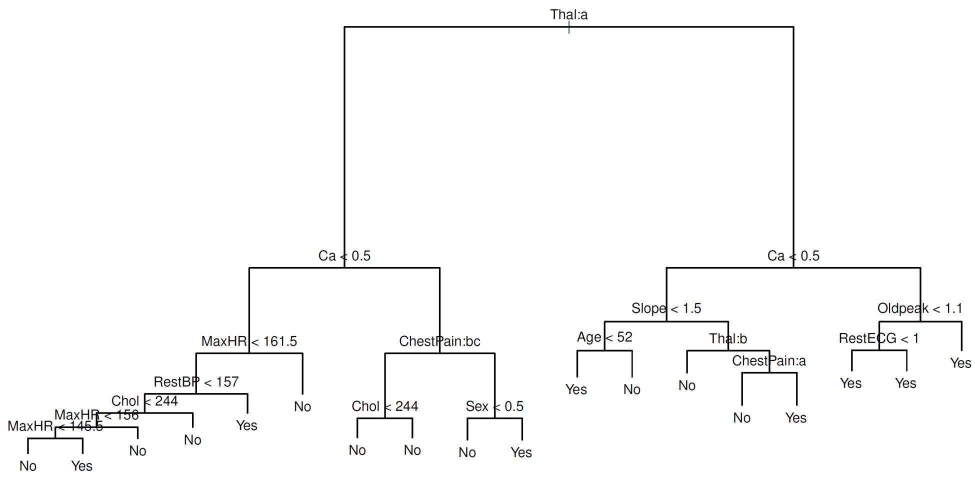


Figure 1.8: Classification Trees [74]. This figure shows the classification tree obtained from the *Heart* dataset. This dataset consists of observations of whether individuals have heart disease or not, based on different predictors like Sex, ChestPain and so on. The internal nodes of the classification tree represent different predictors (Calcium, RestECG etc). The leaf nodes represent the class value (qualitative, binary, Yes or No) of the observation.

$$\begin{aligned} G &= \sum_{m=1}^{M}\sum_{k-1}^{K}\hat{p}_{mk}(1-\hat{p}_{mk}) \\ D &= -\sum_{m=1}^{M}\sum_{k=1}^{K}\hat{p}_{mk}\log(\hat{p}_{mk}) \end{aligned}$$

Both Gini index and entropy can be seen as a measure of the 'purity' of a node: smaller values of the Gini index/entropy of a node means that observations from a single class predominates the specific region [74]. An example of a classification tree is depicted in Figure 1.8.

1.3.4 Random Forest

Random forests are a powerful and popular machine learning algorithm for both classification and regression tasks. They are based on the concept of an ensemble of decision trees, where each tree is trained on a random subset of the data and features. Decision trees, while powerful, have the downside of having a high variance. Consequently, different random splits of training data can produce significantly different results. To remedy this problem, a technique called *bagging* (bootstrap aggregation) is used to reduce variance.

Given a set of n independent random variables $Z = \{Z_1, \ldots, Z_n\}$ each with variance σ^2, the variance of the mean $\bar{Z}$ is given by $\frac{\sigma^2}{n}$. This means that averaging a set of observations reduces its variance. This observation, when applied to learning methods, leads to the conclusion that taking several subsets from the training set, using separate learning models for each subset, and averaging their predictions will reduce the variance. In bagging, B different bootstrapped training data sets, consisting of samples from the (single) training data set, are created. Then, for each $b = 1, \ldots, B$, the model is trained on the b-th training set to obtain the prediction function $f_b(X)$. The final predictions are then obtained by averaging all predictions as follows:

$$f_{bag}(X) = \frac{1}{B}\sum_{b=1}^{B} f_b(X)$$

Random forests represent an improvement over the simple bagging technique. Let $\mathbf{X}$ be the input data of size $n \times p$, where n is the number of observations, p is the number of features, and $\mathbf{x_i}$ is the i-th observation. Let $\mathbf{y}$ be the corresponding output vector of size $n \times 1$, where y_i is the target of the i-th observation. The goal of a random forest is to create an ensemble of decision trees that can accurately predict the target value y for new, unseen input data $\mathbf{x}$. To achieve this, the random forest algorithm creates a set of B decision trees T, referred to as a forest, where each tree t in T is trained on a random subset of the data and features. For each tree t in T, a bootstrap sample of the input data is created by randomly selecting n observations from the original data with replacement. This means that some observations may be selected multiple times, while others may not be selected at all. Once the bootstrap sample is created, a decision tree is grown as follows. At each split node in the tree, a random subset of m amongst the p features is selected, and the split is made on the feature that maximizes the information gain or reduction in impurity. The process is repeated recursively until a stopping criterion is met, such as a maximum tree depth or a minimum number of observations at each leaf node.

After training all B trees in the random forest, the algorithm aggregates their predictions by taking the majority vote for classification tasks or the average for regression tasks. This produces a single prediction for each input data point. The performance of a random forest depends on several hyperparameters, including the number of trees B, the maximum tree depth, the minimum number of observations at each leaf node, and the number of features considered at each split node. These hyperparameters can be tuned using cross-validation techniques to optimize the model performance. An example of a Random Forest is shown in Figure 1.9.

1.3.5 Support Vector Machines (SVM)

Support vector machines (SVMs) are powerful machine learning algorithms used for both classification and regression tasks. SVMs are based on the concept of finding an optimal separating hyperplane that maximizes the margin between two classes of data.

Let $X = \{X_1, \ldots, X_p\}$ be a set of p input features, and Y be the corresponding target variable. For a dataset of size n, the observations and their associated targets are denoted by $\mathbf{x_i}$ and y_i, respectively, for $i = 1, \ldots, n$. For simplicity, we assume that the output is binary, with $y_i = -1$ or $y_i = 1$ representing the two classes. The goal of a linear SVM is to find the hyperplane that separates the two classes of data with the maximum margin.

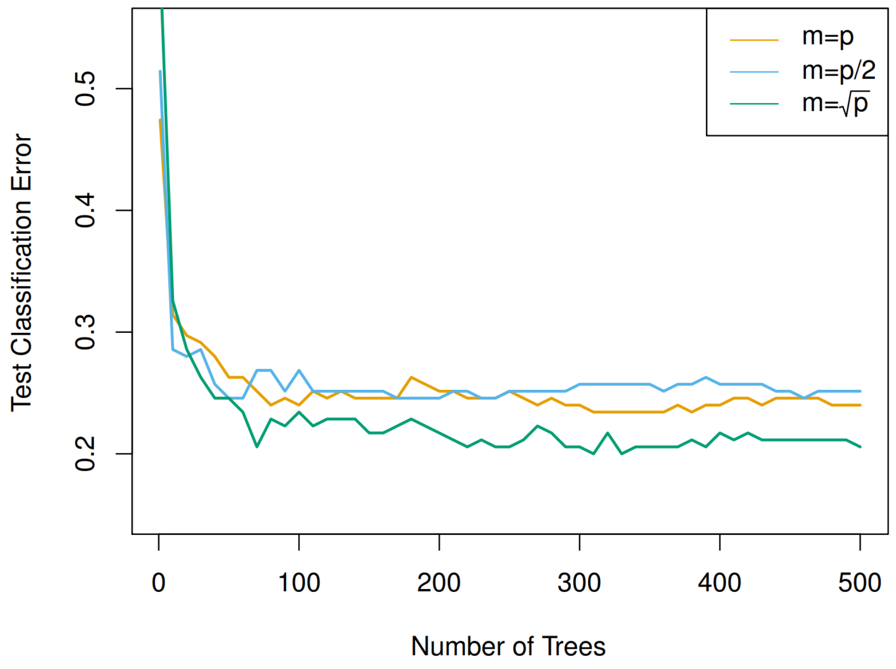


Figure 1.9: Random Forest [74]. This figure shows the results of the Random Forest algorithm on a dataset. The x-axis represents the number of trees generated by the algorithm and the y-axis represents the test classification error. The three curves represent the different values of m chosen (top right) for the algorithm. Notice that random forests $m < p$ show an improvement over bagging $m = p$.

The margin is defined as the distance between the hyperplane and the closest point in each class. The hyperplane is defined as a linear combination of the input features:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where $\mathbf{w}$ is the weight vector and b is the bias.

The SVM finds the optimal values of w and b that maximize the margin while ensuring that all data points are correctly classified. This can be formulated as the following optimization problem:

$$\begin{aligned} \underset{\mathbf{w},\, b}{\text{minimize}} \quad & \|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 \quad \forall i = 1, \ldots, n \end{aligned}$$

where $\|\mathbf{w}\|_2$ is the L_2-norm of the weight vector.

The first term in the objective function represents the regularization term, which encourages the weight vector to be small and prevents over-fitting. The second term represents the constraint that each data point $\mathbf{x_i}$ are correctly classified in its class y_i, expressed by the fact that $y_i(\mathbf{w}^\top\mathbf{x}_i - b) \geq 1$. Note that only the data points that are closest to the separating hyperplane, known as the support vectors, contribute to the optimization problem. This allows the SVM to be very efficient even for high-dimensional data. The optimization problem can be solved using a variety of methods, including quadratic programming and gradient descent. Once the optimal values of $\mathbf{w}$ and b are found, the hyperplane can be used to predict the class of new, unseen data points.

SVMs can be extended to handle non-linearly separable data by using a kernel function. The kernel function maps the input data into a higher-dimensional feature space, where the data may become separable by a hyperplane. The optimization problem is then solved in the feature space, rather than the input space.

1.3.6 K-Nearest Neighbors (KNN)

K-nearest neighbors (KNN) is a supervised machine learning algorithm used for classification and regression analysis. KNN works by finding, in the feature space, the K closest training data points to a given test data point, and then using these neighbors to classify or predict the output variable of the test data point. In the case of classification, KNN determines the class of the test point by computing the mode of the classes of its K nearest neighbors. In the case of regression, KNN determines the value of the test point by computing the mean or median of the outputs of its K nearest neighbors.

Formally, for a positive integer K, a class $j \in Y$ and a test observation $\mathbf{x} = (x_1, \ldots, x_p) \in X_1 \times \cdots \times X_p$, the KNN classifier identifies the K points in the training data that are closest to $\mathbf{x}$. Then, given this set of points N_0, it estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

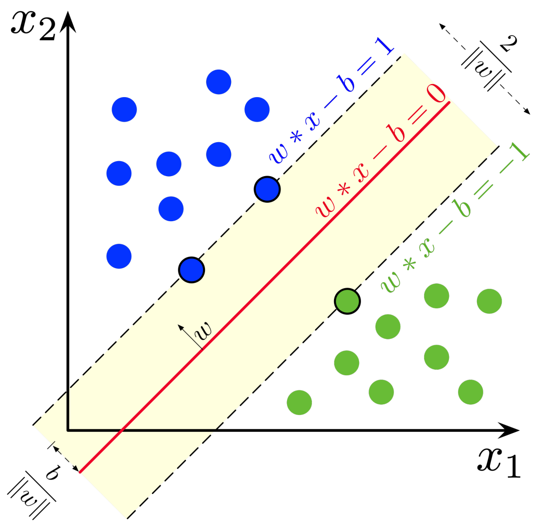


Figure 1.10: Illustration of the support vector machine (SVM) algorithm (Wikipedia).

$$Pr(Y = j \mid X_1 = x_1, \ldots, X_p = x_p) = \frac{1}{K} \sum_{\mathbf{x_i} \in N_0} I(y_i = j)$$

where $I(y_i = j)$ is the indicator function that equals 1 if $y_i = j$ and 0 if $y_i \neq j$. KNN classifies the test observation $\mathbf{x}$ to the class y with the largest probability from the above equation [74]:

$$y = \arg\max_{j \in Y} Pr(Y = j \mid X_1 = x_1, \ldots, X_p = x_p)\,.$$

In the KNN algorithm, the choice of the parameter K is important. When K is kept low, the classifier's decision boundary is overly flexible and the classifier that has low bias but very high variance. As K increases, the decision boundary become increasingly closer to the linear boundary and the classifier has low variance and high bias. A KNN classifier with $K = 3$ is depicted in Figure 1.11.

1.3.7 Naive Bayes Classifier

The *naive Bayes classifier* is a simple and popular machine learning algorithm used for classification tasks, especially in natural language processing and text analysis. It is based on applying Bayes' theorem with a 'naive' assumption of independence among the features or attributes.

Formally, let $\mathbf{X} = (X_1, \ldots, X_P)$ be feature variables and Y be a discrete response variable taking values in $C = \{c_1, \ldots, c_K\}$. Consider also the train set

$$S = \left\{(\mathbf{x_i}, y_i) \in \mathbb{R}^P \times C : i = 1, \ldots, N\right\}.$$

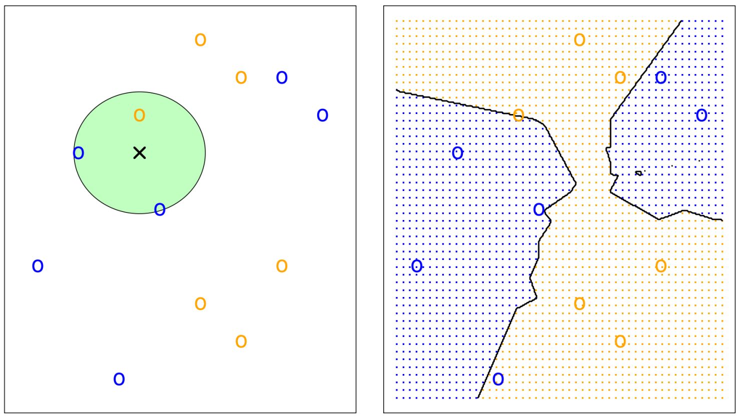


Figure 1.11: K-Nearest Neighbours (KNN) [74]. This figure shows a KNN classifier with $K = 3$. There are a total of 12 training observations, six belonging to the class *blue* and six belonging to the class *orange*. The test observation is depicted by a black cross. On the left, the test observation and the 3 nearest neighbours are shown in a colored circle. Since, the probabilities of the blue class and the orange class are $2/3$ and $1/3$ respectively, the test observation is assigned the class *blue*. On the right, the KNN decision boundary, achieved by applying the above method to all observations, is shown. Test observations which fall in the blue region will be assigned class *blue* and those which fall in the orange region will be assigned *orange*.

The goal of a probabilistic classifier is to compute the conditional probability

$$p(Y = c_k \mid \mathbf{X} = \mathbf{x})$$

for each data point $\mathbf{x}$ and for all $k = 1, \dots, K$. Then, the predicted class $\hat{c}$ associated to the data point $\mathbf{x}$ is the one with maximal conditional probability, i.e.

$$\hat{c} = \arg\max_{c_k \in C} p(Y = c_k \mid \mathbf{X} = \mathbf{x})\,.$$

Unfortunately, while the conditional probabilities $p(Y = c_k \mid \mathbf{X} = \mathbf{x})$ are difficult to compute, their counterparts $p(\mathbf{X} = \mathbf{x} \mid Y = c_k)$, and in particular the individual $p(X_j = x_j \mid Y = c_k)$ for $j = 1, \dots, P$, can generally be estimated from the data. The naive Bayes classifier makes uses these considerations to compute the required probabilities $p(Y = c_k \mid \mathbf{X} = \mathbf{x})$ from the individual $p(X_j = x_j \mid Y = c_k)$. The derivation of this computation is provided below.

For any $\mathbf{x}$, let us denote $p(Y = c_k \mid \mathbf{X} = \mathbf{x})$ by $p(c_k \mid \mathbf{x})$. A repeated application of the rule of conditional probabilities ensures that

$$\begin{aligned}
p(c_k, x_1, \ldots, x_P) &= p(x_1, \ldots, x_P, c_k) \\
&= p(x_1 \mid x_2, \ldots, x_P, c_k)\, p(x_2, \ldots, x_P, c_k) \\
&= p(x_1 \mid x_2, \ldots, x_P, c_k) \\
&\quad\ p(x_2 \mid x_3, \ldots, x_P, c_k)\, p(x_3, \ldots, x_P, c_k) \\
&= \cdots \\
&= p(x_1 \mid x_2, \ldots, x_P, c_k) \\
&\quad\ p(x_2 \mid x_3, \ldots, x_P, c_k) \\
&\quad\ \cdots \\
&\quad\ p(x_{P-1} \mid x_P, c_k)\, p(x_P \mid c_k)\, p(c_k)
\end{aligned}$$

In addition, assuming the following naive hypothesis of conditional independence

$$p(x_j \mid x_{j+1}, \ldots, x_P, c_k) = p(x_j \mid c_k)$$

the above equalities rewrite as

$$\begin{aligned}
p(c_k, x_1, \ldots, x_P) &= p(x_1 \mid c_k)\, p(x_2 \mid c_k)\, p(x_3 \mid c_k) \cdots p(c_k) \\
&= p(c_k) \prod_{j=1}^{P} p(x_j \mid c_k)
\end{aligned}$$

By using Bayes' theorem and the previous equality, the required conditional probabilities are given by

$$p(c_k \mid x_1, \ldots, x_P) = \frac{p(c_k, x_1, \ldots, x_P)}{p(x_1, \ldots, x_P)} = \frac{p(c_k) \prod_{j=1}^{P} p(x_j \mid c_k)}{p(x_1, \ldots, x_P)}$$

The prediction $\hat{c}$ associated to the data point $\mathbf{x}$ is then given by

$$\hat{c} = \arg\max_{c_k \in C} p(c_k \mid \mathbf{x}) = \arg\max_{c_k \in C} p(c_k) \prod_{j=1}^{P} p(x_j \mid c_k)\,.$$

In this formula, the class priors $p(c_k)$ and the feature distributions $p(x_j \mid c_k)$ can be estimated from the data, and require only the computation of only $\mathcal{O}(PK)$ parameters.

1.3.8 K-Means

Clustering is a set of techniques that finds clusters in data. Observations from data that are clustered in the same cluster (or subgroup) are assumed to be similar to each other and distinct to those observations which are grouped in other clusters. The K*-means* algorithm is one such clustering technique. K-means clustering algorithm works by partitioning the data into K distinct, non-overlapping clusters. The algorithm assigns each observation to exactly one of the defined clusters.

Formally, let $C = \{C_1, C_2, ..., C_K\}$ be the set of K clusters where each C_i is a set containing the indices of the observations belonging to a cluster. These sets must satisfy two properties: first, that each observation must belong to at least one of the K clusters, and secondly, that no observation belongs to more than one cluster. K-means clustering algorithm works by assigning observations to clusters such that the within-cluster variation within each cluster is minimized. In other words, the algorithm assigns the observations into clusters such that the total within-cluster variation, measured over all K clusters, is minimized. Formally, the optimal clusters $\hat{C}_1, \hat{C}_2, ..., \hat{C}_K$ are given by:

$$\hat{C}_1, \ldots, \hat{C}_K = \arg \min_{C_1, \ldots, C_K} \sum_{k=1}^{K} W(C_k)$$

where $W(C_k)$ is the within-cluster variation for cluster C_k. There are several definition in use for measuring within-cluster distance. The popular choice is the squared Euclidean distance:

$$W(C_k) = \frac{1}{C_k} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

This reduces the K-means algorithm to the following optimization problem:

$$\hat{C}_1, \ldots, \hat{C}_K = \arg \min_{C_1, \ldots, C_K} \sum_{k=1}^{K} \frac{1}{C_k} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

A brute force approach to this problem is computationally infeasible, as there are approximately K^N possible cluster assignations for a dataset of size N. Hence, this optimization problem is solved by the following procedure with converges to a local minimum of the optimization problem: first, each observation is randomly assigned a cluster index from 1 to K. Then, iteratively, the following two sub-steps are carried out: for each cluster, its centroid is calculated as the mean value of all the observations assigned to that cluster. Then, each observation is assigned to the cluster with the closest centroid, as measured by squared Euclidean distance. The latter two steps continue until centroids stabilize, which ensures that a local minimum is reached.

The initial cluster assignments for the observations play an important role in the calculation of centroids. Therefore, it is typical to run the K-means algorithm using several initial configurations. The value of K – the number of expected clusters in the data – is also an important consideration. An example of a K-means clustering algorithm is depicted in Figure 1.12. Figure 1.13 shows the results of a K-mean algorithm with different initial configurations.

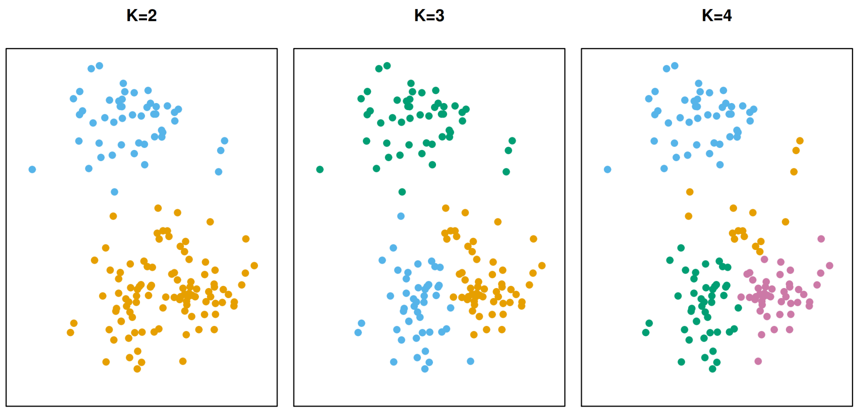


Figure 1.12: K-Means Clustering [74]. This figure shows the application of K-means clustering algorithm to a dataset with $K = 2$, $K = 3$ and $K = 4$ (L to R). The colors assigned to each observation depict the cluster to which it is assigned. The assignment of observations to different clusters is done so as to minimize the within-cluster variation between the observations.

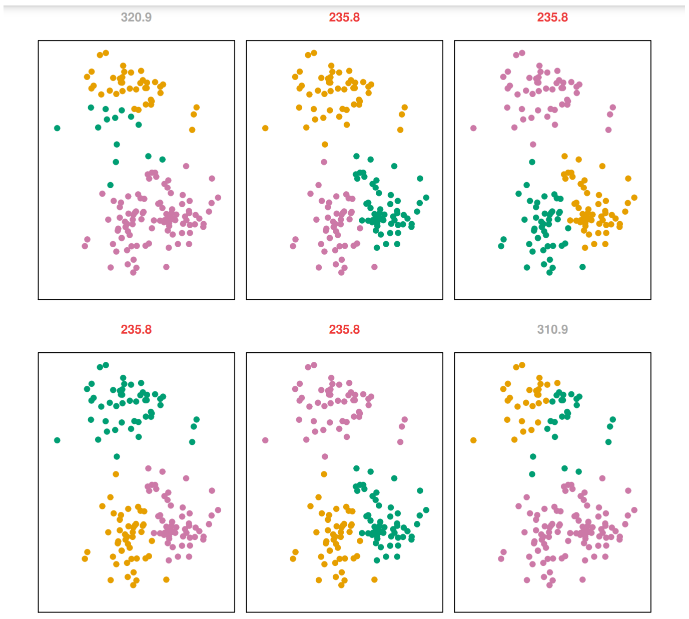


Figure 1.13: K-Means Clustering initial configurations [74]. This figure shows the results of the K-means algorithm with $K = 3$ with six different initial configurations – different random initial assignments of observations to clusters. The within-cluster value for every run of the algorithm is also given. Those with the minimum values are chosen as the best run.

Chapter 2

Natural Language Processing

2.1 Introduction

Natural Language Processing (NLP) is a field of computer science and artificial intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and computational models that enable computers to understand, interpret, and generate human language. NLP is used in a wide range of applications, such as speech recognition, machine translation, sentiment analysis, and text classification. NLP has become increasingly important in the development of intelligent systems, chatbots, and virtual assistants, as it enables machines to communicate with humans in a more natural and intuitive way. NLP is a complex and challenging field, as human language is diverse, ambiguous, and context-dependent. Researchers in NLP continue to develop new techniques and approaches to improve the accuracy and effectiveness of language-based systems. The history of NLP dates back to the 1950s and has since then evolved significantly with the development of new technologies and techniques. The NLP timeline is illustrated in Figure 2.1.

The earliest work in NLP dates back to the late 1950s and early 1960s. In 1950, Alan Turing proposed the "Turing Test," which is a way to determine if a machine is capable of exhibiting intelligent behavior that is indistinguishable from that of a human. This idea served as a catalyst for research into natural language understanding. In the 1960s, the first significant NLP program was developed. The program, called ELIZA, was created by Joseph Weizenbaum in 1966 at MIT. ELIZA simulated a conversation between a user and a therapist. It was based on the idea of using pattern matching and substitution to generate responses to user input. ELIZA was a groundbreaking development in the field of NLP and is considered one of the earliest examples of a chatbot.

In the 1970s and 1980s, NLP research shifted toward the development of knowledge-based systems. These systems used explicit rules and knowledge representations to understand and process natural language. One of the earliest examples of a knowledge-based system was SHRDLU, developed by Terry Winograd at MIT in 1972. SHRDLU was a program that could understand and manipulate a virtual world of blocks. In the 1980s, the introduction of statistical approaches to NLP led to significant progress in the field. One of the most influential works in this area was the development of Hidden Markov Models (HMMs) for speech recognition by L.R. Rabiner in 1986 [79][141].

In the 1990s and 2000s, the introduction of machine learning and corpus linguistics led to significant advances in NLP. Machine learning algorithms, such as neural networks and support vector machines, were used to train models that could automatically learn patterns in natural language data. Corpus linguistics, the study of language as it occurs in large collections of text, also became a popular area of research. The availability of large corpora of text, such as the Brown Corpus and the Penn Treebank, enabled researchers to develop models that could learn from vast amounts of data.

In the 2010s, the introduction of deep learning and neural networks revolutionized the field of NLP. These techniques allowed researchers to build models that could learn to represent language at a higher level of abstraction. The development of Word2vec by Mikolov et al. in 2013 [117] and the subsequent explosion of research in deep learning for NLP led to

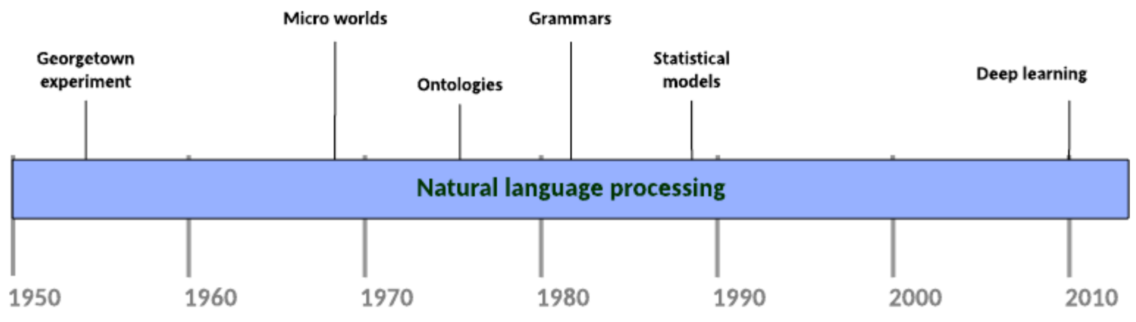


Figure 2.1: The Natural Language Processing (NLP) timeline [124].

significant advances in areas such as sentiment analysis, machine translation, and question answering. In recent years, attention has shifted toward the development of models that can understand and generate more complex forms of language, such as natural language reasoning and dialogue. The development of large-scale pre-trained models, such as Transformers [184], BERT [37], GPT [142], GPT-2 and GPT-3, has opened up new possibilities in these areas.

2.2 NLP Pipeline

The NLP pipeline is a series of steps that a natural language processing system goes through to process and understand human language (see Figure 2.2). It involves several stages of processing, including text normalization, tokenization, part-of-speech tagging, syntactic parsing, semantic analysis, and discourse processing. A brief overview of each stage is provided below:

- Text normalization: The first stage of the NLP pipeline involves text normalization, where the raw text is converted into a standard form. This includes tasks such as removing punctuation, converting all text to lowercase, and expanding contractions.

- Tokenization: Once the text is normalized, the next stage is tokenization, where the text is divided into individual words or tokens. This is an important step for many NLP tasks, as it enables the system to process the text at a more granular level.

- Stop words removal: Stop word removal involves filtering out common words that are considered to have little semantic value and are unlikely to contribute to the meaning of a text. Examples of stop words include articles (e.g., “the”, “a”, “an”), conjunctions (e.g., “and”, “or”), and prepositions (e.g., “in”, “on”, “at”).

- Lemmatization and Stemming: Lemmatization and stemming involve reducing words to their base or root form. Stemming involves removing the suffixes of words to reduce them to their base or stem form. For example, the word “running” can be stemmed to “run”, and the word “jumps” can be stemmed to “jump”. Stemming is

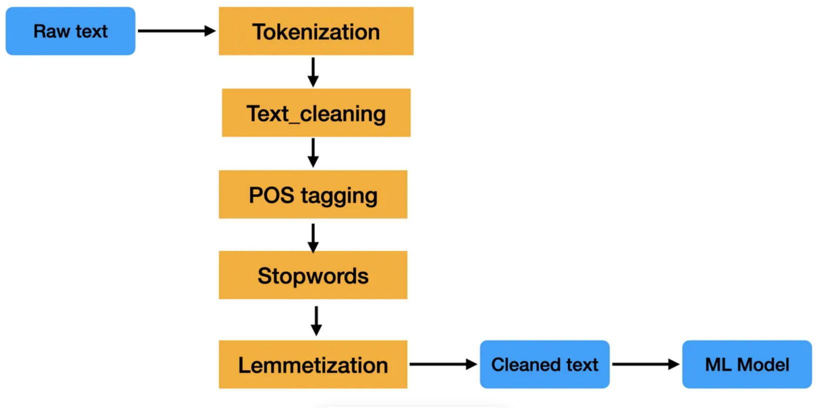


Figure 2.2: The Natural Language Processing (NLP) pipeline [124].

a simple and computationally efficient technique that can help reduce the dimensionality of the text data, which can be useful for certain NLP tasks such as text classification.

- Part-of-speech tagging: In this stage, each token is assigned a part-of-speech tag based on its grammatical role in the sentence. This is important for many NLP tasks, such as text classification and information extraction.

- Syntactic parsing: The next stage is syntactic parsing, where the system analyzes the grammatical structure of the sentence. This involves identifying the relationships between words and grouping them into phrases and clauses.

- Semantic analysis: After parsing, the system moves on to semantic analysis, where it attempts to understand the meaning of the sentence. This involves tasks such as named entity recognition, semantic role labeling, and sentiment analysis.

- Discourse processing: The final stage of the NLP pipeline is discourse processing, where the system attempts to understand the broader context of the text. This involves tasks such as coreference resolution, which identifies when two words or phrases refer to the same entity, and discourse analysis, which attempts to understand the overall meaning of a text.

Syntactic Analysis

Syntactic analysis, also known as syntactic parsing or parsing, is a key component of natural language processing that involves analyzing the grammatical structure of a sentence or phrase. Syntactic analysis attempts to identify the relationships between words in a sentence and group them into meaningful phrases and clauses based on their syntactic roles. Syntactic analysis typically involves the use of grammatical rules or models to parse sentences. There are two main types of syntactic models used in NLP: rule-based models and statistical models.

Rule-based models rely on predefined grammatical rules to analyze the structure of a sentence. These rules are typically based on formal grammar, such as context-free grammar, which defines the syntax of a language in terms of a set of production rules. Rule-based models can be very accurate but may require significant linguistic expertise to develop and maintain. Statistical models, on the other hand, use machine learning algorithms to learn the syntactic structure of a language from large corpora of text. These models can be trained to identify patterns and relationships between words based on statistical patterns in the training data. While statistical models may not be as accurate as rule-based models, they can be more flexible and easier to scale to new languages or domains.

Once a sentence has been parsed, the resulting syntactic structure can be used for a variety of NLP tasks, such as text classification, information extraction, and machine translation. For example, syntactic analysis can be used to identify the subject and object of a sentence, which can be useful for information extraction tasks. Syntactic analysis includes the following:

- Sentence parsing: Sentence parsing is the process of analyzing the grammatical structure of a sentence to identify its constituent parts and their syntactic relationships. Sentence parsing involves breaking down a sentence into its constituent parts, such as phrases and clauses, and identifying the relationships between those parts based on their syntactic roles. The resulting parse tree represents the structure of the sentence in terms of its grammatical units and their relationships. Once a sentence has been parsed, the resulting parse tree can be used for a variety of NLP tasks. For example, the parse tree can be used to extract information from the sentence, such as the subject and object of a sentence, or to generate a machine translation of the sentence.

- Word segmentation: Word segmentation, also known as tokenization, is the process of dividing a continuous text into individual words, or tokens. To perform word segmentation, various techniques can be used, depending on the language and the specific NLP task. For example, rule-based techniques can be used to identify spaces, punctuation marks, or other delimiters between words. Statistical techniques, such as machine learning algorithms, can also be used to learn patterns of word boundaries from large amounts of text data.

- Sentence breaking: Sentence breaking, also known as sentence boundary disambiguation, is the process of identifying the boundaries between sentences in a text. As in word segmentation, various techniques can be used, depending on the language and the specific NLP task. Rule-based techniques can be used to identify common sentence-ending punctuation marks, such as periods, question marks, and exclamation points.

- Morphological segmentation: Morphological segmentation is the process of dividing words into their smallest meaningful units, or morphemes. Morphemes are the smallest units of language that carry meaning, and they can be combined to form words. For example, the word “unhappiness” can be divided into three morphemes: “un-”, “happy”, and “-ness”. Once words have been segmented into their constituent morphemes, various other NLP techniques can be applied to analyze the text. Syntactic analysis and named entity recognition can also be used to identify the grammatical structure of the text and extract important information such as names, places, and dates.

Semantic Analysis

Semantic analysis, also known as semantic processing, is a subfield of natural language processing that focuses on understanding the meaning of language. Semantic analysis goes beyond syntactic analysis, which focuses on the grammatical structure of language, to extract meaning from text and speech. Syntactic analysis includes the following:

- Word sense disambiguation: Word sense disambiguation (WSD) is the task of identifying the correct meaning of a word in a given context. Many words have multiple meanings or senses, and WSD is important in NLP applications such as machine translation, information retrieval, and text-to-speech conversion. WSD can be a challenging task, as the meaning of a word can depend on the context in which it is used. For example, the word "bank" can refer to a financial institution or the edge of a river, and the correct meaning depends on the context of the sentence.
- Named entity recognition: Named entity recognition (NER) is a subtask of NLP that involves identifying and categorizing important entities, such as people, places, organizations, and dates, in a piece of text. NER is used in various applications, such as information extraction, question answering, and text classification. Various techniques can be used for NER, including rule-based systems and machine learning algorithms. In a rule-based system, pre-defined rules and patterns are used to identify entities in the text. In a machine learning approach, a model is trained on annotated data to predict the likelihood of each word being an entity.
- Textual entailment: Textual entailment is a subfield of NLP that involves determining whether one piece of text (the hypothesis) logically follows from another piece of text (the premise). In other words, textual entailment involves determining whether the meaning of the hypothesis can be inferred from the meaning of the premise. Textual entailment can be modeled as a binary classification task, where the system predicts whether the hypothesis is entailed by the premise or not. More advanced approaches can also be used to model the degree of entailment between the two pieces of text.

2.3 Text Classification

Text classification is a task in natural language processing that involves categorizing a piece of text into one or more pre-defined categories. The goal of text classification is to automatically assign a label or category to a document, based on its content. Text classification has various applications, such as spam filtering, sentiment analysis, topic classification, and document categorization. Text classification can be modeled as a supervised machine learning problem, where a model is trained on a labeled dataset and used to make predictions on new, unseen data. The labeled dataset consists of documents, each associated with a label or category. The model learns to associate the features of the text with the corresponding labels or categories [86].

The first step in text classification is to pre-process the text. This typically involves tokenization, which is the process of breaking the text into individual words or tokens. The tokens are then typically converted to lowercase, and stop words (such as "the" and "and") are removed. Stemming or lemmatization may also be performed to reduce words to their base form. Once the text has been pre-processed, features are extracted from the text. These features are used to represent the text in a numerical format that can be used by machine learning algorithms. Common features used in text classification include: Bag of Words (BoW), TF-IDF (term frequency-inverse document frequency) and word embeddings. Once the features have been extracted, a machine learning model is trained on the labeled dataset. Various machine learning algorithms can be used for text classification, such as naive Bayes, decision trees, random forests, support vector machines, and neural networks [86].

In general, the text classification system contains four different levels of scope that can be applied:

- Document level: In the document level, the algorithm obtains the relevant categories of a full document.
- Paragraph level: In the paragraph level, the algorithm obtains the relevant categories of a single paragraph (a portion of a document)
- Sentence level: In the sentence level, obtains the relevant categories of a single sentence (a portion of a paragraph)
- Sub-sentence level: In the sub-sentence level, the algorithm obtains the relevant categories of sub-expressions within a sentence (a portion of a sentence) [86].

Text Classification Pipeline

The standard text classification pipeline consists of four steps (see Figure 2.3 [86]):

- Feature extraction: Feature extraction is a crucial step in text classification, where the goal is to represent the text in a numerical format that can be used by machine learning algorithms. The choice of features can have a significant impact on the performance of the text classification model, and various feature extraction techniques can be used depending on the nature of the data and the task at hand. Some common feature extraction techniques used in text classification include: Bag of Words (BoW), TF-IDF (term frequency-inverse document frequency), word embeddings, n-grams and Part-of-speech (POS) tagging.
- Dimensionality reduction: Dimensionality reduction is a common technique used in text classification to reduce the high-dimensional feature space to a lower-dimensional

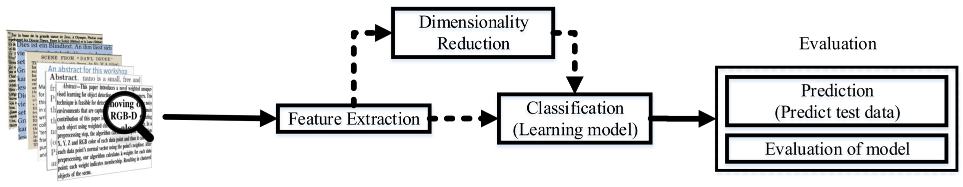


Figure 2.3: Text classification pipeline [86].

space, while still retaining the most relevant information. This can help to improve the performance of the text classification model by reducing the noise and redundancy in the data, and speeding up the training process. Some methods used for dimensionality reduction include: Principal component analysis (PCA), Latent semantic analysis (LSA) and Non-negative matrix factorization (NMF).

- Classification: The most important step of the text classification pipeline is choosing the best classifier. The choice of classifier depends on the nature of the data, the complexity of the task, and the trade-off between performance and interpretability. In practice, multiple classifiers may be trained and compared to choose the best performing model. Several classes of classifiers have been used for text classification: ensemble-based learning techniques, non-parametric techniques, tree-based classifiers and deep learning.
- Evaluation: Text classification evaluation is the process of assessing the performance of a text classification model. The aim of evaluation is to measure how well the model is able to classify new and unseen text data. The common metrics used to evaluate classifiers include: accuracy, F1-score and area under the receiver operating characteristic curve (AUC-ROC).

2.4 Embeddings

Word embedding is a feature learning technique in which each word or phrase from the vocabulary is mapped to an N-dimensional vector of real numbers. This approach has revolutionized natural language processing (NLP) by providing a means to represent words in a continuous, semantic space, where words with similar meanings or contextual relationships are positioned closer to each other. These word vectors, often referred to as word embeddings or word representations, capture the underlying linguistic structures and semantic information hidden in textual data. Word embeddings capture semantic relationships between words based on their co-occurrence patterns in the training data. Words with similar meanings or usage tend to have similar vector representations, which allows machine learning models to understand the context and semantics of words in a text. At the heart of embeddings is the concept of representing words as vectors in a continuous vector space. This vector space is designed to capture semantic relationships between words, such that similar words are closer together in the vector space. To formalize this, let's denote:

Vocabulary: V, the set of all unique words in the corpus.

Vector Dimension: d, the number of dimensions in the vector space.

Each word w in the vocabulary V is associated with a d-dimensional vector $\mathbf{v}_w \in \mathbb{R}^d$. These vectors are typically initialized randomly and then adjusted during training to capture the desired semantic properties.

2.4.1 Static Embeddings

Static word embeddings, also known as fixed word embeddings, are a type of word representation where each word in a vocabulary is associated with a fixed, pre-trained vector of real-valued numbers. These embeddings are determined during a pre-training phase on a large text corpus. Static word embeddings are based on word co-occurrence statistics and do not change or adapt during the course of a specific task.

Word2Vec

T. Mikolov et al. [115] introduced word embedding technique known as the 'word to vector' representation. The Word2Vec method employs shallow neural networks containing two hidden layers, incorporating both the continuous bag-of-words (CBOW) and Skip-gram models to generate high-dimensional vectors for individual words [86]. The Continuous Bag of Words (CBOW) model aims to predict a target word based on its context words (words that appear within a fixed window around the target word). The objective is to maximize the likelihood of observing the target word given its context words. The probability of observing the target word $\mathbf{w_i}$ given its context c can be expressed using the softmax function:

$$P(\mathbf{w_i} \mid c) = \frac{\exp(\mathbf{w_i}' \cdot \text{avg}(\mathbf{w_j}, \text{for } \mathbf{w_j} \in c))}{\sum_{j=1}^{|V|} \exp(\mathbf{w_j}' \cdot \text{avg}(\mathbf{w_k}, \text{for } \mathbf{w_k} \in c))}$$

where $\mathbf{w_i}$ is the target word we want to predict, c is the context, which consists of context words $\mathbf{w_j}$ for $\mathbf{w_j}$ in C and $\mathbf{w_i}'$ is the output vector for the target word [117].

In the Skip-gram model, the objective is reversed. Given a target word, we aim to predict its context words. This approach is particularly useful when we want to generate word embeddings that capture rich, context-based information. The probability of observing a context word $\mathbf{w_j}$ given the target word $\mathbf{w_i}$ can be expressed as:

$$P(\mathbf{w_j} \mid \mathbf{w_i}) = \frac{\exp(\mathbf{w_j} \cdot \mathbf{w_i}')}{\sum_{k=1}^{|V|} \exp(\mathbf{w_k} \cdot \mathbf{w_i}')}$$

where $\mathbf{w_j}$ is a context word, $\mathbf{w_i}$ is the target word and $\mathbf{w_i}'$ is the output vector for the target word $\mathbf{w_i}$ [117].

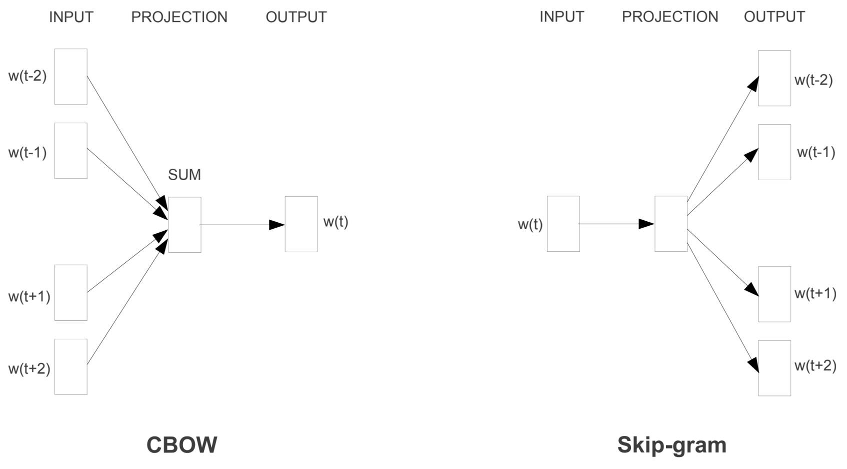


Figure 2.4: Word2Vec word emedding: CBOW (left) and skip-gram (right) [117].

GloVe

Global Vectors (GloVe) is another robust word embedding technique extensively applied in text classification. It closely resembles the Word2Vec method, where each word is represented by a high-dimensional vector and undergoes training based on its contextual words within an extensive corpus. In many studies, the pre-trained word embeddings are derived from a vocabulary of 400,000 words, trained using Wikipedia 2014 and Gigaword 5 as the corpus, with each word represented in 50 dimensions. Additionally, GloVe offers alternative pre-trained word vectorizations in 100, 200, and 300 dimensions, trained on even larger corpora, including Twitter content. The technique's objective function is as follows [134, 86]:

$$f(\mathbf{w_i} - \mathbf{w_j}, \mathbf{w_k}) = \frac{P_{ik}}{P_{jk}}$$

where $\mathbf{w_i}$ refers to the word vector of word i, and P_{ik} denotes the probability of word k to occur in the context of word i.

FastText

FastText is a embedding technique introduced by Facebook AI Research. Each word, w, is represented as a bag of character n-grams. Given a dictionary of n-grams of size G, and a word w which is associated as a vector representation z_g to each n-gram g and context c, the scoring function is given by [17, 86]:

$$s(w, c) = \sum_{g \in g_w} z_g^T v_c$$

where $g_w \in \{1, 2, \ldots, G\}$.

2.4.2 Dynamic Embeddings

Dynamic embeddings, on the other hand, also referred to as contextual word embeddings, are word representations that adapt to the context in which a word appears within a sentence or document. Unlike static embeddings, which assign a single fixed vector to each word, dynamic embeddings vary depending on the surrounding words and sentence structure. These embeddings are typically generated by deep learning models, such as recurrent neural networks (RNNs), BiLSTMS or Transformer-based models, which process text in a sequential manner and consider the context of each word when computing its embedding. One popular approach to dynamic embeddings is contextualized word representations, where each word's embedding depends on its context within a sentence or document.

ELMO

ELMo (Embeddings from Language Models) is a contextual word embedding technique that captures the contextual meaning of words in a sentence [135]. ELMo models, which are typically based on bidirectional LSTMs (Long Short-Term Memory networks), generate word embeddings that depend on the entire sentence context. ELMo typically uses a deep, bidirectional LSTM architecture. This means that for each word in a sentence, there are two LSTM networks: one processing the words from left to right (forward LSTM) and another processing the words from right to left (backward LSTM). The final word embedding produced by ELMo is a combination of the outputs from both LSTMs.

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ represent the words in the input sentence, where n is the sentence length. Each word $\mathbf{x}_i$ is represented as a vector (e.g., word embeddings) denoted as $\mathbf{x}_i$. At each time step t, it computes a hidden state $\mathbf{h}_t^{\text{forward}}$ based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}^{\text{forward}}$. The forward LSTM equations are:

$$\mathbf{h}_t^{\text{forward}} = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}^{\text{forward}})$$

Similarly, at each time step t, it computes a hidden state $\mathbf{h}_t^{\text{backward}}$ based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}^{\text{backward}}$. The backward LSTM equations are similar to the forward LSTM:

$$\mathbf{h}_t^{\text{backward}} = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}^{\text{backward}})$$

ELMo combines the hidden states from both the forward and backward LSTMs at each time step t:

$$\gamma \cdot (\mathbf{h}_t^{\text{forward}} + \mathbf{h}_t^{\text{backward}})$$

where γ is a learnable scalar weight. ELMo computes contextual word embeddings by summing the word representations across all time steps t in the sentence:

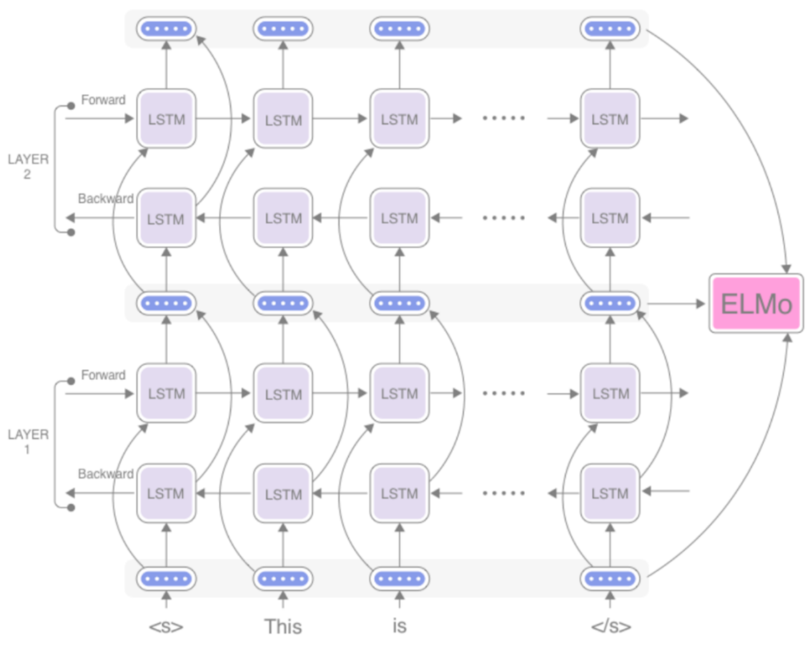


Figure 2.5: ELMo embedding [98].

$$\sum_{t=1}^{n} \gamma \cdot (\mathbf{h}_t^{\text{forward}} + \mathbf{h}_t^{\text{backward}})$$

BERT

BERT (Bidirectional Encoder Representations from Transformers) embeddings are contextually rich word representations or word embeddings generated by pre-training a deep bidirectional transformer-based neural network on a large corpus of text. These embeddings capture not only the meaning of individual words but also their contextual relationships within sentences and documents. For a detailed treatment of how BERT embeddings are constructed, see Chapter 7.

Chapter 3

Neural Networks

3.1 Introduction

The history of neural networks dates back to the early days of computer science, when researchers were trying to create machines that could perform tasks that required human-like intelligence. The idea of creating artificial neural networks (ANNs) was inspired by the workings of the human brain, which is a complex network of interconnected neurons that communicate with each other to process information.

The first artificial neuron was created in 1943 by Warren McCulloch and Walter Pitts [113]. Their model of a neuron was inspired by the biological neuron, which consists of a cell body, dendrites, and an axon. More specifically, they proposed a computational model of a neuron using binary logic to simulate neural firing. In the following years, several researchers worked on developing artificial neural networks based on the McCulloch-Pitts model. However, the limitations of this model, such as its inability to learn, led to the development of new models.

In the late 1950s, Frank Rosenblatt developed the Perceptron [152], which was a type of neural network that could learn to classify inputs into different categories. The Perceptron was able to perform simple image recognition tasks, but it was limited in its capabilities. During the 1960s and 1970s, interest in neural networks declined, as other machine learning techniques such as decision trees and rule-based systems gained popularity. However, in the 1980s, several breakthroughs were made in the field of neural networks.

In 1982, John Hopfield introduced the Hopfield network [72], which was a type of recurrent neural network that implement a content-addressable or associative memory. The Hopfield network was able to store and recall patterns, making it useful for tasks such as image and speech recognition.

In 1986, Geoffrey Hinton, David Rumelhart, and Ronald Williams developed the backpropagation (BP) algorithm [155], which allowed neural networks to learn and improve their performance. Backpropagation is an efficient gradient-based supervised learning algorithm that adjusts the weights of the connections between neurons to minimize the error between the network's output and the desired output.

In the 1990s, neural networks were used for a variety of applications, including speech recognition, handwriting recognition, and image recognition. However, the limitations of neural networks, such as the difficulty of training large networks, led to a decline in interest in the field. In the early 2000s, new techniques such as deep learning, convolutional neural networks, and recurrent neural networks were developed, which allowed for the training of larger and more complex neural networks. Deep learning refers to the class of machine learning algorithms achieved by deep neural networks, namely, feedforward neural networks composed of many layers of neurons. Convolutional neural networks are deep neural networks with adapted architectures and information propagation rules that are designed for image processing tasks. Recurrent neural networks use recurrent instead of a feedforward architectures, which makes them capable of memorizing their successive inputs. They are trained by means of an adaptation of backpropagation called backpropagation through time (BPTT) [151, 197]. They are well adapted for tasks that involve sequential data, such as speech and text.

These techniques led to a resurgence of interest in neural networks and their applications. Today, neural networks are used for a variety of applications, including image and speech recognition, natural language processing, and autonomous vehicles. For an extensive and detailed survey about deep learning methods, see the work by Schmidhuber [162].

The field of neural networks continues to evolve, and new techniques and architectures are being developed to improve their capabilities. A major breakthrough in the field has been achieved with the introduction of large-scale pre-trained language models, such a the Transformer [184] and its offsprings GPT [142], GPT-2, GPT-3 and BERT [37]. These models are discussed in detail in the next chapters.

Besides, in recent years, there has been growing interest in neuromorphic computing, which aims to create computer systems that function like the human brain [163]. Neuromorphic computing involves building hardware that mimics the behavior of neurons and synapses, and it has the potential to create highly efficient and powerful computing systems.

3.2 Perceptrons

A perceptron is a type of artificial neural network used for classification and prediction tasks. It is a single-layer neural network that consists of a set of input nodes, a set of output nodes, and a set of weights that are used to determine the output of the network.

3.2.1 Single Perceptron

A single perceptron is a basic building block of an artificial neural network. It is a mathematical model that simulates the behavior of a single neuron in the brain and acts as a binary classifier. Formally, a single perceptron is an object $(\mathbf{x}, \mathbf{w}, b, y, \sigma)$ where:

- $\mathbf{x} = (x_1, x_2, \ldots, x_M) \in \mathbb{R}^M$ is the set of inputs to the cell
- $\mathbf{w} = (w_1, w_2, \ldots, w_M) \in \mathbb{R}^M$ is the set of weights associated with the connections from the inputs to the cell
- b is the bias of the cell
- $y \in \{-1, +1\}$ is the binary output of the cell
- σ is the hard-threshold activation function defined by $\sigma(z) = 1$ if $z \geq 0$ and $\sigma(z) = -1$ otherwise.

The output or activation value y of the perceptron is given by:

$$y = \sigma\left(\sum_{i=1}^{M} w_i x_i + b\right) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} +1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{3.1}$$

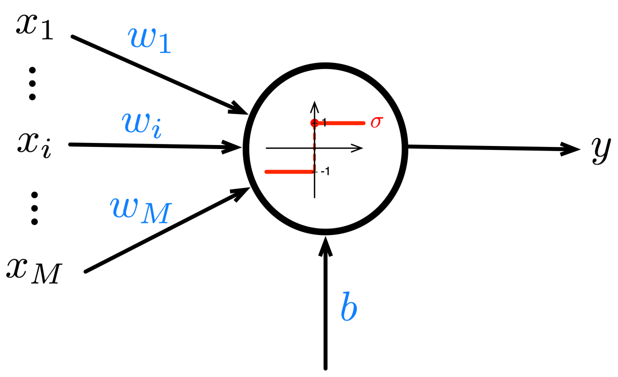


Figure 3.1: A single perceptron. The perceptron is composed of a set of inputs $\mathbf{x} = (x_0, \ldots, x_M)$, a cell, and a set of weights $\mathbf{w} = (w_0, \ldots, w_M)$, where each weight w_i corresponds to the strength of the connection from input x_i ot the cell. The perceptron also has its bias value b. The value of the perceptron is given by $(\mathbf{w}^T\mathbf{x} + b)$. The activation function σ then computes the discrete output of the perceptron based on this value: the output is -1 if this value is less than 0 and $+1$ if it is larger than or equal 0.

A perceptron is depicted in Figure 3.1. The perceptron takes one or more inputs, multiplies each input by a weight, and then sums up the weighted inputs and the bias. The sum is then passed through the hard-threshold activation function to produce a discrete output. This output can be interpreted as a binary decision, such as whether an input belongs to a certain category or not. As shown in the above equation, note that the dynamics of the perceptron can be written in the compact vectorial form $y = \sigma(\mathbf{w}^T\mathbf{x} + b)$.

3.2.2 Multi-Layer Perceptron

A multi-layer perceptron (MLP) is an artificial neural network composed of multiple layers of perceptrons or neurons, where each layer is connected to the next in a feedforward manner. In an MLP, the first layer takes the input data, and subsequent layers transform the data through a series of non-linear transformations, before producing the final output.

Before the multi-layer perceptron is introduced, the single-layer perceptron is presented. A single-layer perceptron consists a number of perceptrons or neurons stacked together in the form of a layer. Each neuron in this layer receives weighted connections from the inputs and has its own bias. According to Equation (3.1), the outputs of a single-layer perceptron composed of M inputs and N cells is given by:

$$a_i = \sigma(\mathbf{w_i}^T\mathbf{x} + b_i), \text{ for } i = 1, \ldots, N$$

where a_i is the output of the i-th cell, and $\mathbf{w_i} = (w_{i1}, \ldots, w_{iM})$ and b_i are the weights and biases associated with the i-th cell, for $i = 1, \ldots, N$, as described in the previous section. Note that this dynamics can be written in the following compact matricial form:

$$\mathbf{a} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{3.2}$$

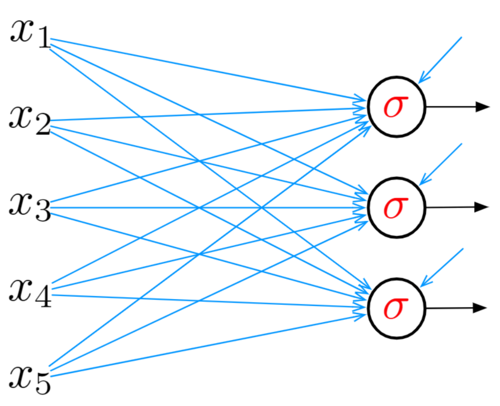


Figure 3.2: A single-layer perceptron.

where $\mathbf{x} = (x_0, \ldots, x_M)$ is the vector of inputs, $\mathbf{W} = (w_{ij})$ is the weight matrix, where w_{ij} is the weight from input j to cell i, for $j = 1, \ldots, M$ and $i = 1, \ldots, N$, $\mathbf{b} = (b_1, \ldots, b_N)$ is vector of biases of the cells, σ is the hard-threshold activation value applied component by component, and $\mathbf{a} = (a_0, \ldots, a_N)$ is output vector of the cells. For instance, the dynamics of a single-layer perceptron with 3 neurons and 5 inputs is given by:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \right]$$

A single-layer perceptron is shown in Figure 3.2.

The concept of a single-layer perceptron can be extended to the multi-layer context. Here, the outputs of the first layer (represented by $\mathbf{a} = (a_1, a_2, a_3)$ above) are input to the second layer of perceptrons, and the outputs of the second layer are input to the third layer, and so forth. Based on Equation (3.2), the dynamics of an MLP with L layers is formally given by:

$$\begin{cases} \mathbf{a^{(0)}} = \mathbf{x} \\ \mathbf{z^{(l)}} = \mathbf{W^{(l)}} \mathbf{a^{(l-1)}} + \mathbf{b^{(l)}}, \text{ for } l = 1, \ldots, L \\ \mathbf{a^{(l)}} = \sigma(\mathbf{z^{(l)}}) \end{cases} \tag{3.3}$$

where:

- $\mathbf{x}$ is the input of the MLP
- $\mathbf{a^{(L)}}$ is the output of the MLP
- $\mathbf{W^{(l)}} \in \mathbb{R}^{\lambda_l} \times \mathbb{R}^{\lambda_{l-1}}$ is the weight matrix, with $w_{ij}^{(l)}$ being the weight of neuron j in layer $l-1$ (of size λ_{l-1}) to neuron i in layer l (of size λ_l)

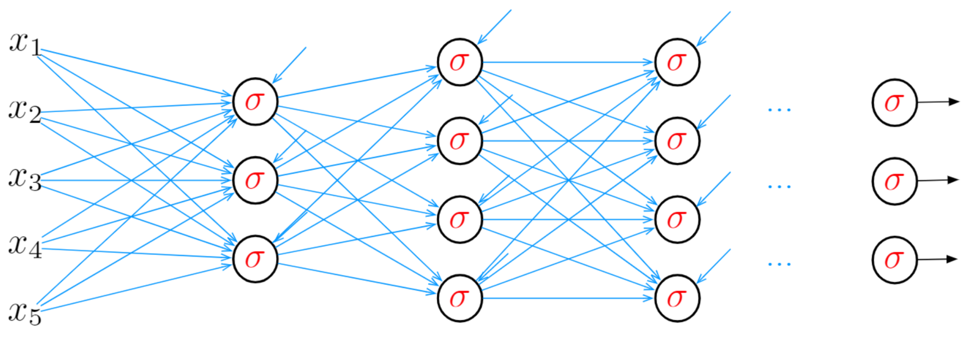


Figure 3.3: A multi-layer perceptron.

- $\mathbf{b}^{(l)} \in \mathbb{R}^{\lambda_l}$ is the bias vector
- σ is the hard-threshold activation function applied component by component.

A multi-layer perceptron is shown in Figure 3.3.

3.3 Neural Networks

Artificial Neural Networks (ANNs) is a class of Machine Learning algorithms loosely inspired by biological neurons. The network are made of input, hidden and output artificial neurons related together by weighted synaptic connections. The network is said to be feedforward or recurrent depending on whether its topology is acyclic or cyclic, respectively. Information is processed from the input to the output neurons by traveling through the synaptic connections. Artificial networks, unlike perceptrons, use continuous instead of discrete non-linear activation functions, which implies that the activation values of the neurons are continuous. The training process of artificial neural networks consists of an adjustment of their synaptic weights according to some algorithmic task to be achieved. In the feedforward and recurrent contexts, the weights are usually updated by means of the backpropagation (BP) or backpropagation through time (BPTT) algorithm, respectively, which are efficient gradient descent-based minimization processes of the error function of the network. Nowadays, artificial neural networks are among the most successful Machine Learning techniques, thanks to their highly efficient training and learning capabilities. An artificial neural network is illustrated in Figure 3.4.

In general, a neural network consists of the following:

- an input layer: this layer of the network receives the input data. Each neuron in the input layer corresponds to a feature or attribute of the input data. The dimension of the input layer is the same a that of the data.
- the hidden layer(s): the hidden layers of the network perform computations on the input data. Each neuron in the hidden layers applies a non-linear activation function, such as the sigmoid or ReLU function, to the weighted sum of its inputs.

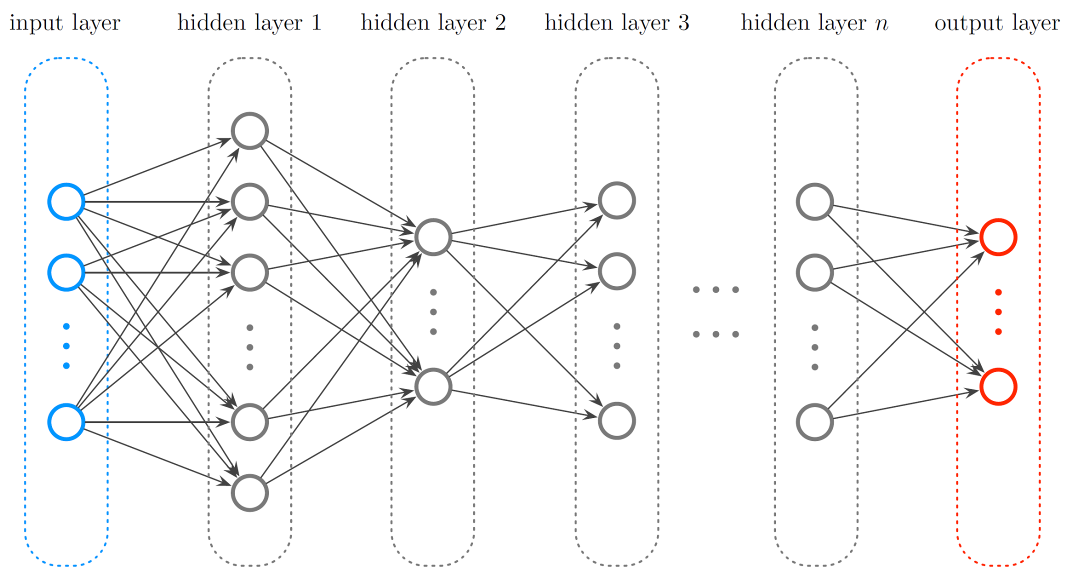


Figure 3.4: A feedforward neural network. The nodes and edges represent the neurons and synaptic connections between them, respectively. Information is processed in a feedforward manner, i.e., from one layer to the next, without any recurrent connection. The training of the neural network consist of an adjustment of its synaptic weights according to some algorithmic task to be achieved.

- an output layer: The output layer of the network produces the final prediction or classification of the input data. The number of neurons in the output layer depends on the problem being solved.

The training of an ANN is achieved by means of the Stochastic Gradient Descent (SGD) method. This method is implemented efficiently using the backpropagation (BP) algorithm in the context of feedforward neural networks, and the backpropagation through time (BPTT) algorithm in the case of recurrent neural networks. In the main, the training process consists of a succession of two alternating phases: the forward pass and the backward pass. In the forward pass, the network processes a batch of inputs throughout its layers to compute associated output predictions. In the backward pass, the error between the predictions and the actual outputs is computed according to some pre-determined loss/objective function. Then, the gradients of the loss function with respect to the network parameters – the weights and biases – are computed in the backward direction, i.e., from the last layer to the first. Finally, the network parameters are updated in the inverse direction of their gradients. These phases are repeated until some local minimum of the loss/objective function is hopefully reached. In the context of recurrent networks, this process is applied to the graph of the network "unfolded in time". The training process of a neural network is presented in Section 3.7.

3.4 Feedforward Neural Networks

A feedforward neural network (FFNN) is a neural network whose topology is organized into successive layers of neurons, such that the output of one layer serves as the input to the next layer (cf. multi-layer perceptron). When the network contains more than one hidden layer, it is usually referred to as a deep neural network (DNN). Feedforward neural networks are used to build complex mapping between the input data and the predicted output using hidden layers of various sizes. *Through a chain of transformations, the network is able to build up fairly complex transformations of [the input] X that ultimately feed into the output layer as features* [74].

Formally, a feedforward neural network is a generalization of a multi-layer perceptron where the activation functions of the neurons are continuous rather than discrete. Popular activation functions include the sigmoid function, the hyperbolic tangent, and the rectified linear unit (ReLU) function, which are respectively defined by:

$$g(z) = \frac{1}{1+e^{-z}}\,, \qquad g(z) = \tanh(z) \quad \text{and} \quad g(z) = \begin{cases} 0 & \text{if z < 0} \\ z & \text{otherwise.} \end{cases}$$

Following the governing equations of an MLP (3.3), the dynamics of a feedforward neural network composed of L layer is given by the following equations:

$$\begin{cases} \mathbf{a^{(0)}} = \mathbf{x} \\ \mathbf{z^{(l)}} = \mathbf{W^{(l)}}\mathbf{a^{(l-1)}} + \mathbf{b^{(l)}}, \ \text{for } l = 1, \ldots, L \\ \mathbf{a^{(l)}} = g^{(l)}(\mathbf{z^{(l)}}) \end{cases} \tag{3.4}$$

where $\mathbf{x}$ is the input of the network, $\mathbf{a^{(L)}}$ is the output vector of the network, $\mathbf{W^{(l)}} \in \mathbb{R}^{\lambda_l} \times \mathbb{R}^{\lambda_{l-1}}$ is the weight matrix, with $w_{ij}^{(l)} \in \mathbf{W^{(l)}}$ being the weight of neuron j in layer $l-1$ (of size λ_{l-1}) to neuron i in layer l (of size λ_l), $\mathbf{b^{(l)}} \in \mathbb{R}^{\lambda_l}$ is the bias vector, and $g^{(l)}$ is the activation function of the neurons of layer l applied component by component. According to these equations, the prediction of the network over some input $\mathbf{x}$ is given by

$$\mathbf{\hat{y}} = f(\mathbf{x}) = \mathbf{a^{(L)}}\,. \tag{3.5}$$

A feedforward neural network is illustrated in Figure 3.4.

Given some dataset $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$, many loss functions can be considered to measure the error between the predictions $\mathbf{\hat{y}_i}$ of the network and the actual responses $\mathbf{y_i}$ of the dataset (for $i = 1, \ldots, n$). Note that the loss function depends on the parameters of the network. Hence, training the network is the process of finding the parameters that minimize this loss function. For regression problems, the most common loss function is the mean squared error (MSE) given by:

$$MSE\left(\Theta;\mathcal{D}\right)=\frac{1}{n}\sum_{i=1}^{n}\left\|\hat{\mathbf{y}}_{\mathbf{i}}-\mathbf{y}_{\mathbf{i}}\right\|^{2}.$$

where Θ is the set of all parameters of the network, i.e., the weights and the biases. For classification problems, the the categorical cross-entropy (CCE) loss, also known as the negative multinomial log-likelihood, is generally considered:

$$CCE\left(\Theta;\mathcal{D}\right)=-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{y}_{\mathbf{i}_j}\log(\hat{\mathbf{y}}_{\mathbf{i}_j})$$

where m is the number of classes for this task, $\mathbf{y}_{\mathbf{i}}$ is a one-hot encoding vector encoding the class of $\mathbf{x}_{\mathbf{i}}$, and $\mathbf{y}_{\mathbf{i}_j}$ and $\hat{\mathbf{y}}_{\mathbf{i}_j}$ are the j-th component of $\mathbf{y}_{\mathbf{i}}$ and $\hat{\mathbf{y}}_{\mathbf{i}}$, respectively. In both cases, the closer the predictions of the networks are to the actual target values, the lower the loss values.

To demonstrate the working of a feedforward neural network, consider the MNIST handwritten digit dataset. The task is to build a neural network model that classifies images of handwritten digits into their correct class from 0 to 9. Each image consists of $28 \times 28 = 784$ pixels, where each pixel is associated with a number between 0 and 255 representing its grey-scale value. A feedforward neural network for this task satisfies the following requirements: its input layer is of dimension 784; its hidden layers compute successive transformations of the 784-dimensional inputs; and its output layer is of dimension 10, where each of the 10 neurons represents a digit from 0 to 9 whose value is 1 if the input vector represents the respective digit and 0 otherwise. One such feedforward neural network model is shown in Figure 3.5.

In this example, the feedforward neural network for classifying the MNIST handwritten image dataset has two hidden layers L_1 and L_1 containing 256 and 128 units, respectively. The model works as follows: if $\mathbf{x}$ denotes a 784-dimensional input, the output $\mathbf{a^{(1)}}$ of the first hidden layer $L1$ of dimension 256 is given by:

$$\begin{aligned}\mathbf{a^{(0)}} &= \mathbf{x}\\ \mathbf{z^{(1)}} &= \mathbf{W^{(1)}}\mathbf{a^{(0)}}+\mathbf{b^{(1)}}\\ \mathbf{a^{(1)}} &= g^{(1)}(\mathbf{z^{(1)}})\end{aligned}$$

where $\mathbf{W^{(1)}} \in \mathbb{R}^{256\times 784}$ and $\mathbf{b^{(1)}} \in \mathbb{R}^{256}$ are the weights and biases of the first hidden layer, and $g^{(1)}$ is the activation function of the hidden neurons. Recall that each element $w_{ij}^{(1)} \in \mathbf{W^{(1)}}$ represents the weight of the connection from input j to cell i. The output $\mathbf{a^{(1)}}$ of the first hidden layer L_1 is then input to the second hidden layer L_2 of dimension 128 whose output $\mathbf{a^{(2)}}$, in turn, is given by:

$$\begin{aligned}\mathbf{z^{(2)}} &= \mathbf{W^{(2)}}\mathbf{a^{(1)}}+\mathbf{b^{(2)}}\\ \mathbf{a^{(2)}} &= g^{(2)}(\mathbf{z^{(2)}})\end{aligned}$$

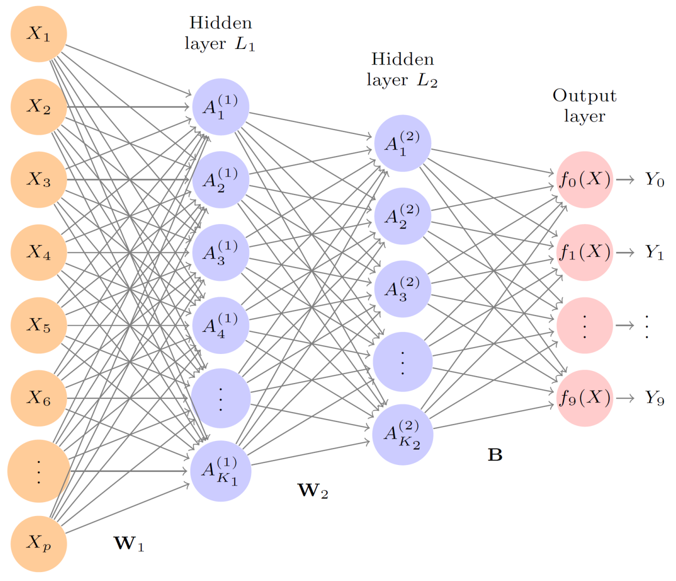


Figure 3.5: A feedforward neural network for classifying the MNIST handwritten dataset images [74]. An input is a 28×28 pixel matrix flattened into a 784-dimensional vector $\mathbf{x} = (x_1, \dots, x_p)$. An output is a 10-dimensional vector $f(\mathbf{x}) = (f_0(\mathbf{x}), \dots, f_9(\mathbf{x}))$. The successive hidden layers compute successive non-linear transformations of the input.

where again, $\mathbf{W^{(2)}} \in \mathbb{R}^{128\times256}$, $\mathbf{b^{(2)}} \in \mathbb{R}^{128}$ and $g^{(2)}$ are the weights, biases and activation. function of the second hidden layer, respectively. Finally, the output $\mathbf{a^{(2)}}$ of the second hidden layer L_2 is fed to the output layer of dimension 10 whose output $\mathbf{a^{(3)}}$ is given by:

$$\begin{aligned}\mathbf{z^{(3)}} &= \mathbf{W^{(3)}a^{(2)}} + \mathbf{b^{(3)}}\\ \mathbf{a^{(3)}} &= g^{(3)}(\mathbf{z^{(3)}})\end{aligned}$$

where $\mathbf{W^{(3)}} \in \mathbb{R}^{10\times128}$, $\mathbf{b^{(3)}} \in \mathbb{R}^{10}$ and $g^{(3)}$ are the weights, biases and activation function of the output layer, respectively. For the output $\mathbf{a^{(3)}}$ of the network to be a probability vector, we use the softmax activation function for $g^{(3)}$, which imposes that the components of $\mathbf{a^{(3)}}$ are non-negative and sum to 1:

$$\mathbf{a^{(3)}} = g^{(3)}(\mathbf{z^{(3)}}) = \text{softmax}(\mathbf{z^{(3)}}) = \left(\frac{e^{z_0^{(3)}}}{\sum_{i=0}^{9} e^{z_i^{(3)}}}, \dots, \frac{e^{z_9^{(3)}}}{\sum_{i=0}^{9} e^{z_i^{(3)}}}\right)^T$$

where $z_i^{(3)}$ it the i-th component of $\mathbf{z^{(3)}}$ for $i = 0, \dots, 9$. In this way, the i-th component $a_i^{(3)}$ of $\mathbf{a^{(3)}}$ represents the probability that the output variable $Y = i$ given the fact that the input variable $\mathbf{X} = \mathbf{x}$:

$$Pr(Y = i \mid \mathbf{X} = \mathbf{x}) = a_i^{(3)} = \frac{e^{z_0^{(3)}}}{\sum_{i=0}^{9} e^{z_i^{(3)}}}$$

for $i = 0, \ldots, 9$. Putting all together, the dynamics of the network is given by the following equations:

$$\begin{aligned}
\mathbf{a^{(0)}} &= \mathbf{x} \\
\mathbf{z^{(1)}} &= \mathbf{W^{(1)}a^{(0)}} + \mathbf{b^{(1)}} \\
\mathbf{a^{(1)}} &= g^{(1)}(\mathbf{z^{(1)}}) \\
\mathbf{z^{(2)}} &= \mathbf{W^{(2)}a^{(1)}} + \mathbf{b^{(2)}} \\
\mathbf{a^{(2)}} &= g^{(2)}(\mathbf{z^{(2)}}) \\
\mathbf{z^{(3)}} &= \mathbf{W^{(3)}a^{(2)}} + \mathbf{b^{(3)}} \\
\hat{\mathbf{y}} = f(\mathbf{x}) = \mathbf{a^{(3)}} &= \mathrm{softmax}(\mathbf{z^{(3)}})
\end{aligned}$$

Since we are in the context of a classification problem, the categorical cross-entropy loss is considered

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=0}^{9} \mathbf{y_i}_j \log(\hat{\mathbf{y}}_{\mathbf{i}j})$$

where n is the number of data, $(\mathbf{x_i}, y_i)$ are the data, $\mathbf{y_i}$ is the 10-dimensional one-hot encoding of y_i and $\mathbf{y_i}_j$ its j-th component, $\hat{\mathbf{y}}_\mathbf{i}$ is the prediction of the network associated with $\mathbf{x_i}$ and $\hat{\mathbf{y}}_{\mathbf{i}j}$ its j-th component. The model is then trained on the train dataset until a weight and bias configuration that minimizes the cross-entropy of the model is reached. The training process is described in more detail in Section 3.7.

Once trained, the model can be used in a pure prediction mode. The output $y = f(\mathbf{x})$ of the network is then given by the class associated with the highest output probability. This corresponds to the index of the neuron with maximal probability:

$$\hat{y} = \arg\max_{i=0,\ldots,9} \hat{\mathbf{y}}\,.$$

3.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process sequential data. Unlike feedforward neural networks, RNNs have loops in their architecture that allow information to persist within the hidden cells across time steps. In this way, RNNs implement some form of memory. These memory capabilities enable RNNs to handle variable-length inputs and outputs and to capture dependencies between elements in the input sequence. Several types of data are sequential in nature. This includes text documents, where the meaning of the words is dependent on it's relative position and context, as well as time series data, such as weather forecasting and financial and stock forecasting, and recorded speech.

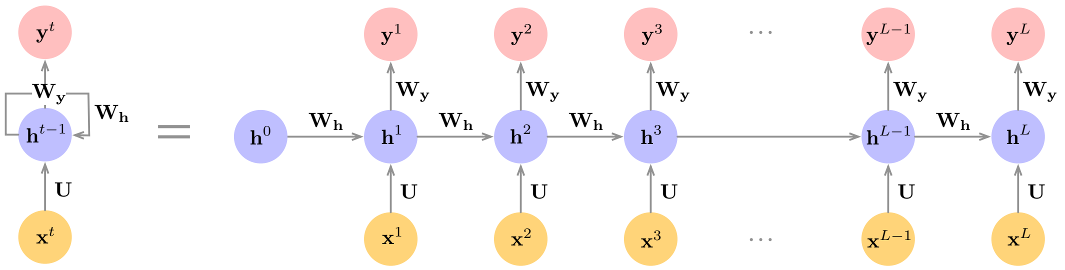


Figure 3.6: A recurrent neural network (RNN). To the left of the equal sign is the regular representation of the RNN. To the right is the dynamics of the RNN unfolded in time. The input sequence to be processed $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^L)$ produces an associated output sequence $\mathbf{y} = (\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^L)$. The activation of the hidden layer at time t, $\mathbf{h}^t$, is computed based on the input at time t, $\mathbf{x}^t$, and the activation of the hidden layer at previous time step $t-1$, $\mathbf{h}^{t-1}$. This activation is then used to produce the output at time t, $\mathbf{y}^t$. Typically, the last output of the hidden layer $\mathbf{y}^L$ is considered as the output of the network. The set of input-to-hidden weights $\mathbf{U}$, the hidden-to-hidden weights $\mathbf{W_h}$ and hidden-to-output weights $\mathbf{W_y}$ are learned during the training of the RNN.

We will now formally define a Recurrent Neural Network (RNN). An RNN is composed of an input layer of N neurons $x_0, \dots, x_{N-1}$, a hidden layer of K neurons $h_0, \dots, h_{K-1}$, and an output layer of M neurons $y_0, \dots, y_{M-1}$. The input layer is connected to the hidden layer means of (feedforward) synaptic connections associated with the weight matrix $\mathbf{U} \in \mathbb{R}^{K \times N}$, where $u_{ji} \in \mathbf{U}$ represents the weight of the connection from the input cell x_i to the hidden cell h_j. The hidden layer is recurrently interconnected by means of synaptic connections with weight matrix $\mathbf{W_h} \in \mathbb{R}^{K \times K}$, where $w_{h_{ji}} \in \mathbf{W_h}$ represents the weight of the connection from the hidden cell h_i to the hidden cell h_j and $\mathbf{b_h} \in \mathbb{R}^K$ is the bias vector associated with the hidden cells. The hidden layer is connected to the output cells by means of (feedforward) synaptic connections with weight matrix $\mathbf{W_y} \in \mathbb{R}^{K \times N}$, where $w_{y_{ji}} \in \mathbf{W_y}$ represents the weight of the connection from the hidden cell h_i to the output cell y_j and $\mathbf{b_y} \in \mathbb{R}^M$ is the bias vector of the output cells. The activation values of the neurons in these layers will evolve across time according to the dynamics of the network described below. The activation values of the input, hidden and output layer at time t are denoted by the vectors $\mathbf{x}^t$, $\mathbf{h}^t$ and $\mathbf{y}^t$, respectively. An RNN in illustrated in Figure 3.6.

The input object to an RNN is a sequence, $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^L)$. For example, $\mathbf{x}$ could be a textual sequence consisting of L words, where each $\mathbf{x}^t$ represent the word received at time t. As the successive inputs are received, network update the activation value of its hidden layer, and successive outputs are produced. The output of the RNN $\mathbf{y}$ is an associated sequence $\mathbf{y} = (\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^L)$. Depending on the task at hand, either the full sequence $\mathbf{y}$ or its last element $\mathbf{y}^L$ can be considered as the network's prediction.

The dynamics of an RNN is given as follows: at each time step t, the network update the activation value of its hidden layer $\mathbf{h}^t$ based on the input $\mathbf{x}^t$ received at time t and the activation value of the hidden layer $\mathbf{h}^{t-1}$ at time $t-1$. Then, the network uses the activation of its hidden layer $\mathbf{h}^t$ at time t to produce an output $\mathbf{y}^t$ at time t. Assuming that the initial hidden activation $\mathbf{h}^0$ is given, the dynamic of an RNN is therefore given by the following equations:

$$\begin{cases} \mathbf{h^t} = \sigma_h\left(\mathbf{Ux^t} + \mathbf{W_h h^{t-1}} + \mathbf{b_h}\right) \\ \mathbf{y^t} = \sigma_y\left(\mathbf{W_y h^t} + \mathbf{b_y}\right) \end{cases}$$

for $t = 1, \dots, L$ where σ_h and σ_y are the non-linear activation functions of the hidden and output cells applied component-wise, respectively.

An important dynamic in RNNs is weight sharing: same set of weights is used for multiple time steps in the network's computation. In other words, instead of having separate weights for each time step in the sequence, the same weights are used for every step. Specifically, same weight matrices and biases, $\mathbf{U}$, $\mathbf{W_h}$, $\mathbf{W_y}$, $\mathbf{b_h}$ and $\mathbf{b_y}$, are used while processing each element in the sequence. In this way, the last output at the ouptut layer $\mathbf{y}^L$ is a reflection of the 'accumulated' activations, ($\mathbf{h}^t$, for $t = 1, 2, \dots, L-1$), which allows the RNN to capture the context of the elements in the sequence.

Finally, in the context of RNNs, different loss functions can be considered depending on the task to be solved. For classification tasks over a dataset $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), \dots, (\mathbf{x_n}, \mathbf{y_n})\}$ of size n, the loss function is the usual MSE given by

$$MSE\left(\Theta; \mathcal{D}\right) = \frac{1}{n}\sum_{i=1}^{n} \left\|\mathbf{y_i}^L - \mathbf{y_i}\right\|^2$$

where $\mathbf{y_i}^L$ is the last output prediction of the RNN associated with input sequence $\mathbf{x_i} = (\mathbf{x_i}^1, \dots, \mathbf{x_i}^L)$, and Θ regroups the parameters $\mathbf{U}$, $\mathbf{W_h}$, $\mathbf{W_y}$, $\mathbf{b_h}$ and $\mathbf{b_y}$ of the RNN.

3.6 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is specifically designed to handle the challenge of preserving information over long sequences of data. Specifically, LSTMs were devised to solve the vanishing gradient problem that RNNs suffer from. The vanishing gradient problem arises when the gradients calculated during backpropagation become very small as they are propagated through multiple layers of the network, making it difficult for the network to learn and optimize the weights of the layers closest to the input. Therefore, classical RNN find it difficult to learn the relationship between input sequences and target outputs over large time lags. Indeed, time lags of 5 to 10 steps already diminishes the learning capability of the RNN [69].

These long term dependencies in sequential tasks can be explained with an example. Consider a next word prediction task. This task involves sequentially processing a text and predicting the next word based on the previous words. Consider two text sequences: 'The clouds are in the sky.' and 'I grew up in France, which is the reason why I speak fluent French.'. In the former sentence, predicting the word 'sky' after reading the previous words is straightforward. In the latter sentence, however, in order to predict the word 'French', the model needs context which is a lot further back in the sequence. In some tasks, this gap between relevant input elements and the current output becomes too large for standard

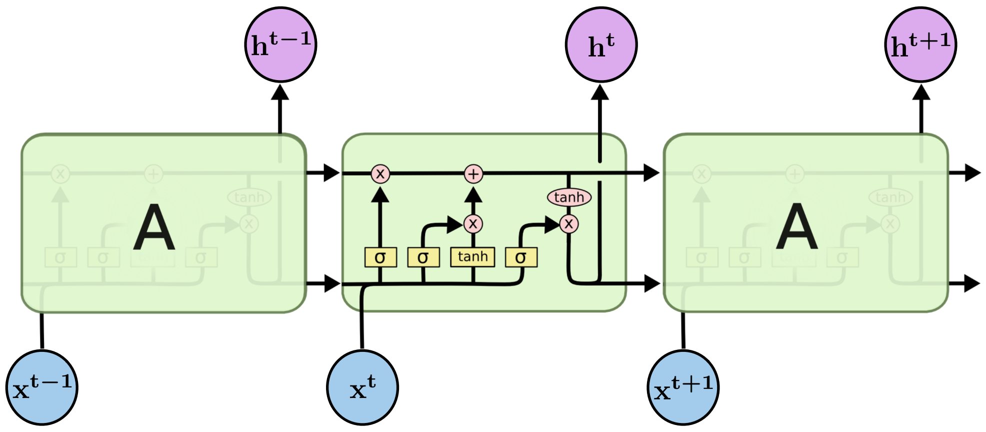


Figure 3.7: An LSTM block with one cell shown in detail. The successive activations of the LSTM block are represented. The vectors $\mathbf{x^t}$ and $\mathbf{h^{t-1}}$ represent the network's input and the state of the LSTM block at time t and $t-1$, respectively. The inward connection at the bottom left denotes the processing of $\mathbf{h^{t-1}}$ while the inward connection at the top left is that of thes CEC/cell state. The units inside the cell, from left to right respectively, represent the forget, input and output gates. The outward connection at the top right represents the new cell state and the outward connection at the bottom right represents the output of the cell [128].

RNNs to handle. LSTMs are able to effectively capture long-term dependencies in data by using a sophisticated gating mechanism.

The basic block of an LSTM network is a memory block containing one or more memory cells and three gate units (forget, input and output). The core unit of an LSTM block is the so-called 'Constant Error Carousel' (CEC) which is a recurrently self-connected linear unit. The CEC provides the LSTM block with a short-term extended time memory by 'retaining' it's activation/error signals until modified by interaction with one of the three gate units, each of which performing a specific operation. The input gate controls what information is to be stored in the memory, the forget gate decided how much information is to be stored in the memory, and the output gate decides how much information is output by the block and input to the next block. In practice, multiple LSTM blocks are considered in parallel to form an LSTM layer. Hence, an LSTM layer processes vectors of activations. For the sake of simplicity, an LSTM block will refer from now on to a layer of LSTM blocks. An LSTM block is shown in Figure 3.7.

We now describe the dynamics of the recurrent LSTM block by formally defining each of the gates as well as the CEC/cell state.

Forget gate The forget gate of the LSTM block is used to determine and control how the cell state is modified in this specific cell. The forget gate unit considers the output of the LSTM block at previous time step and the current input, and it outputs a value between 0 and 1. This value can be thought of as the proportion of the existing cell state that is to be retained. Formally:

$$\mathbf{f^t} = \sigma\left(\mathbf{W_f}[\mathbf{h^{t-1}}; \mathbf{x^t}] + \mathbf{b_f}\right)$$

where $\mathbf{f^t}$ is the state (vector) of the forget gate at time t, $\mathbf{h^{t-1}}$ is the output (vector) of the LSTM block at time $t-1$, $\mathbf{x^t}$ is the input (vector) of the network's input at time step t, $[.;.]$ denotes the vector concatenation, $\mathbf{W_f}$ and $\mathbf{b_f}$ are the weights and biases of the forget gate, respectively, and σ is the logistic sigmoid function applied component-wise.

Input gate Having decided the proportion of the cell state to retain/omit using the forget gate, the input gate of the LSTM block defines new values for the cell state. The input cell takes the output of the LSTM block at previous time step and the current network's input, and it produces two outputs: a vector $\mathbf{i^t}$ defining the values to be updated and a vector $\tilde{\mathbf{c}}^{\mathbf{t}}$ of the new candidate values for the cell state. Formally, the two vectors are given, respectively, by:

$$\begin{aligned}\mathbf{i^t} &= \sigma\left(\mathbf{W_i}[\mathbf{h^{t-1}};\mathbf{x^t}] + \mathbf{b_i}\right)\\ \tilde{\mathbf{c}}^{\mathbf{t}} &= \tanh\left(\mathbf{W_c}[\mathbf{h^{t-1}};\mathbf{x^t}] + \mathbf{b_c}\right)\end{aligned}$$

where $\mathbf{W_i}$ and $\mathbf{b_i}$ are the weights and biases of the input gate, respectively, and $\mathbf{W_c}$ and $\mathbf{b_c}$ are the weights and biases CECs, respectively.

Cell state Based on the activations obtained from the forget gate and the input gate, the cell state is updated. The cell state is then obtained by, first, multiplying the previous cell state by the current output of the forget gate and then adding the activation of the current input gate. Formally:

$$\mathbf{c^t} = \mathbf{f^t} \odot \mathbf{c^{t-1}} + \mathbf{i^t} \odot \tilde{\mathbf{c}}^{\mathbf{t}}$$

where $\odot$ denotes the component-wise multiplication (Hadamard product).

The cell state retains its current value in proportion to the activation of the forget gate, i.e., if the activation of the forget gate is closer to 0, most of the current cell state is replaced while if its activation is closer to 1, most of the current cell state is retained. Thus, using the CEC/cell state together with the input and forget gates, the LSTM block is able to store information for long dependencies as well as to update and discard it if and when necessary [57].

Output gate This gate produces the output of the LSTM block. This output is based on the updated cell state and is produced in two steps: first, a sigmoid layer decides which part of the cell state the cell wants to output and second, the activation of this sigmoid layer is multiplied by cell state activation, scaled using the hyperbolic tangent function. Formally:

$$\begin{aligned}\mathbf{o^t} &= \sigma\left(\mathbf{W_o}[\mathbf{h^{t-1}};\mathbf{x^t}] + \mathbf{b_o}\right)\\ \mathbf{h^t} &= \mathbf{o^t} \odot \tanh(\mathbf{c^t})\end{aligned}$$

where $\mathbf{W_o}$ and $\mathbf{b_o}$ are the weights and biases of the output gate, respectively.

Since the seminal work from Hochreiter and Schmidhuber, 1997 [69], several improvements and variations on the basic LSTM structure have been proposed. One such improvement, proposed by Gers & Schmidhuber, 2000 [56], adds the so-called 'peephole connections' where each of the input, forget and output gate gets a copy of the cell state while computing it's activation.

Bidirectional LSTMs

In natural language texts, the meaning of a word depends both on it's past and future context. LSTMs, while powerful, are able to capture only the past context of a word. Therefore, it makes sense to devise a model that is able to capture the context of a text in both directions. Bidirectional LSTMSs (BiLSTMs) are an extension of the traditional LSTM architecture that allows the model to take into account both past and future contexts when making predictions. This is achieved by processing the input sequence in both forward and backward directions, using two separate LSTM layers.

To capture the left and right context of an element in a sequence, the forward LSTM network of a BiLSTM processes the input sequence in left-to-right direction and the backward LSTM network processes the sequence in the right-to-left direction. In this way, the forward network gathers the context to the left of each sequence element and the backward network gathers the context to the right of each sequence element. After processing the sequence in both directions, the outputs of these two LSTM layers are concatenated, allowing the BiLSTM model to have access to both past and future contexts when making predictions [182].

For example, consider an input sequence $\mathbf{x} = (\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^n})$. Then, the forward LSTM layer computes a sequence of hidden states, $\mathbf{h_f} = (\mathbf{h_f^1}, \mathbf{h_f^2}, \ldots, \mathbf{h_f^n})$, where:

$$\mathbf{h_f^t} = \mathrm{LSTM_f}(\mathbf{x^t}, \mathbf{h_f^{t-1}})$$

where $LSTM_f$ is the forward LSTM. In similar fashion, the backward LSTM layer computes a sequence of hidden states, $\mathbf{h_b} = (\mathbf{h_b^1}, \mathbf{h_b^2}, \ldots, \mathbf{h_b^n})$, where:

$$\mathbf{h_b^t} = \mathrm{LSTM_b}(\mathbf{x^t}, \mathbf{h_b^{t-1}})$$

The final output of the BiLSTM is obtained by concatenating the forward and backward hidden states at each time step:

$$\mathbf{h^t} = \mathrm{BiLSTM}[\mathbf{h_f^t}; \mathbf{h_b^t}]$$

where $[.;.]$ is the concatenation operation. A BiLSTM is depicted in Figure 3.8.

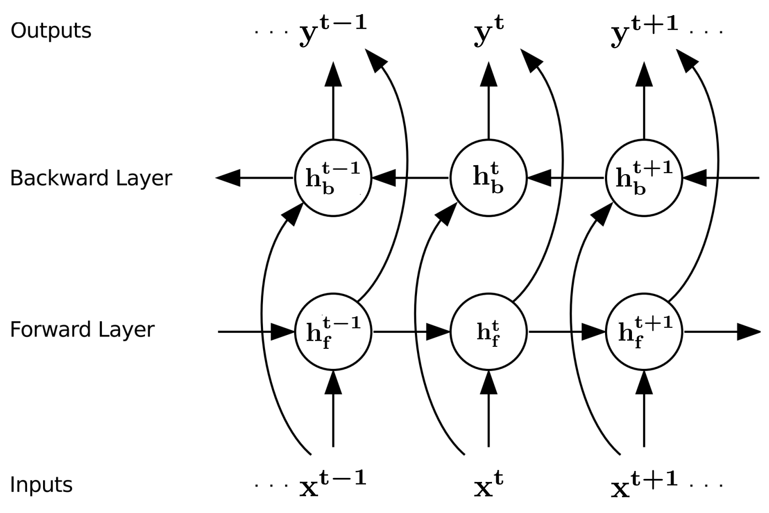


Figure 3.8: A BiLSTM. The forward LSTM layer processes the input sequence in from left to right while the backward LSTM layer processes the sequences from right to left. The outputs of both forward and backward LSTM layers are concatenated to obtain the output at each layer. Typically, the output at the last layer is considered the output associated with the input sequence [58].

3.7 Network Fitting

In neural networks, network fitting refers to the process of adjusting the parameters of the model to minimize the loss function, which measures the difference between the actual and the predicted outputs for a given dataset. This process is also known as model training or model optimization. More specifically, during network fitting, the model is presented with a set of training data, and its parameters – the weights and biases – are adjusted iteratively using an optimization algorithm to minimize the loss function.

The goal of network fitting is to find the optimal set of parameters for the model which can accurately predict the output for new input data that the model has not seen before – the test set. This is accomplished by balancing the tradeoff between underfitting (where the model is too simple and fails to capture the complexity of the data) and overfitting (where the model is too complex and fits the noise in the data instead of the underlying patterns).

3.7.1 Gradient Descent

In machine learning, gradient descent (GD) is used to optimize the parameters of a model so that it can make accurate predictions on new data. The optimized parameters are those that minimize the loss function of the model for the task at hand.

More specifically, let

$$\mathcal{D} = \left\{(\mathbf{x_i}, \mathbf{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, n\right\}$$

be some dataset of size n, and let

$$\begin{aligned} f(\cdot;\boldsymbol{\Theta}) : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{d_2} \\ \mathbf{x} &\longmapsto \hat{\mathbf{y}} := f(\mathbf{x};\boldsymbol{\Theta}) \end{aligned}$$

be some model (a neural network for instance) with parameters $\boldsymbol{\Theta}$ which, for any input $\mathbf{x}$, computes the prediction $\hat{\mathbf{y}} = f(\mathbf{x};\boldsymbol{\Theta})$. Let also the loss function associated with the model f and the dataset $\mathcal{D}$ be given by

$$\begin{aligned} \mathcal{L} : \mathbb{R}^{|\boldsymbol{\Theta}|} &\longrightarrow \mathbb{R} \\ \boldsymbol{\Theta} &\longmapsto \mathcal{L}(\boldsymbol{\Theta}) \end{aligned}$$

where $|\boldsymbol{\Theta}|$ is the cardinality of $\boldsymbol{\Theta}$, i.e., the number of parameters of the model. Note that for a fixed dataset $\mathcal{D}$, the loss function $\mathcal{L}$ depends on the model's parameters $\boldsymbol{\Theta}$, since this function computes some error between the actual outputs (which do not depend on the model) and the predicted outputs which do depend on the model. Typically, the loss function can the mean squared error (MSE) or the categorical cross-entropy (CE). According to these considerations, the optimal parameters of the model f are those that minimize the loss function, namely:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}) \ .$$

A well known result from real analysis states that the gradient $\nabla\mathcal{L}(\boldsymbol{\Theta})$ is a vector pointing in the direction of the steepest ascent of $\mathcal{L}(\boldsymbol{\Theta})$. Consequently, for any point $(\boldsymbol{\Theta}, \mathcal{L}(\boldsymbol{\Theta}))$ on the loss surface, the minimum $\mathcal{L}(\boldsymbol{\Theta})$ lies in opposite direction of $\nabla\mathcal{L}(\boldsymbol{\Theta})$, namely, in the direction $-\nabla\mathcal{L}(\boldsymbol{\Theta})$. The gradient descent algorithm is based on this result. The basic idea is to start from a random vector of parameters of the model $\boldsymbol{\Theta}$, and then, iteratively adjust the latter by making a little step of size λ in the direction of the steepest descent of the loss function: $\boldsymbol{\Theta} := \boldsymbol{\Theta} - \nabla\mathcal{L}(\boldsymbol{\Theta})$. Hopefully, the algorithm will converge to a local minimum $\hat{\boldsymbol{\Theta}}$ of $\mathcal{L}(\boldsymbol{\Theta})$. The gradient descent algorithm is given in Algorithm 1 and illustrated in Figure 3.9.

Algorithm 1: Gradient descent (GD)

```
Inputs: model $f(\cdot;\boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}$;
        dataset $\mathcal{D} = \{(\mathbf{x_i}, \mathbf{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\}$;
        differentiable loss function $\mathcal{L} : |\boldsymbol{\Theta}| \longrightarrow \mathbb{R}$;
        random initial parameters $\boldsymbol{\Theta}$; learning rate $\lambda > 0$; nb of epochs $nb\_epochs$.

for $e = 1, \dots, nb\_epochs$ do
    for $i = 1, \dots, N$ do                                  // compute predictions (dataset)
        $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i};\boldsymbol{\Theta})$
    end
    $\mathcal{L}(\boldsymbol{\Theta}) := \mathcal{L}(\hat{\mathbf{y}}_\mathbf{1}, \dots, \hat{\mathbf{y}}_\mathbf{N}, \mathbf{y_1}, \dots, \mathbf{y_N}; \boldsymbol{\Theta})$      // compute loss (dataset)
    $\boldsymbol{\Theta} := \boldsymbol{\Theta} - \lambda\nabla\mathcal{L}(\boldsymbol{\Theta})$                      // update gradient (dataset)
end
return $f(\cdot;\boldsymbol{\Theta})$
```

Minimizing the loss function of the neural model is not a trivial task. The loss function generally contains multiple local minima. In addition, with the increasing complexity of

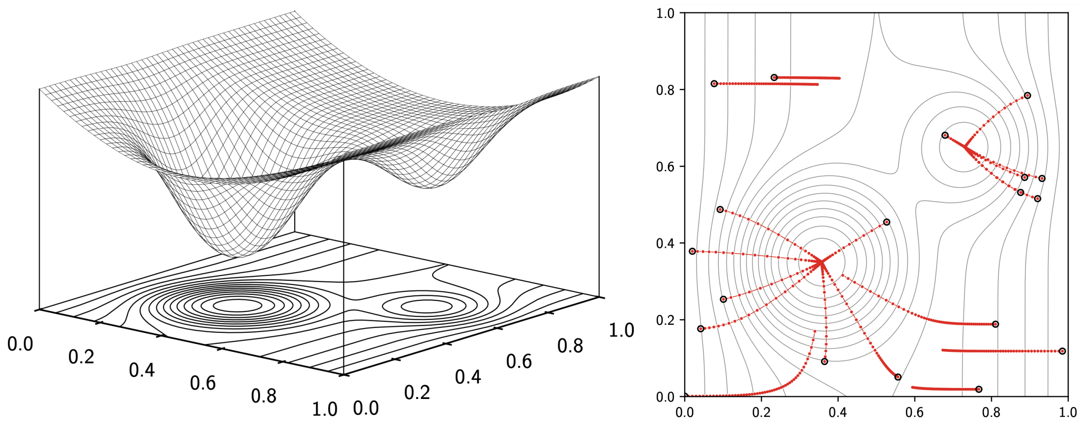


Figure 3.9: Gradient descent (GD) algorithm. (Left) Illustration of a loss function $\mathcal{L}(\mathbf{\Theta})$ with 2 minima. The 'ground' is the parameter space and the surface is the loss function. (Right) Illustration of several runs of the GD algorithm in the contour plot of the loss function. The GD algorithm starts from random values of $\mathbf{\Theta}$ and makes successive steps in the direction of the steepest descent of $\mathcal{L}(\mathbf{\Theta})$, until reaching a local minimum.

neural networks, the number of parameters increases drastically, and so does the dimensionality of the loss function, and with it the complexity and difficulty of finding a minimum of it.

Stochastic Gradient Descent Stochastic Gradient Descent is a variation of the traditional gradient descent algorithm. In traditional Gradient Descent, the entire dataset is used to compute the gradient of the loss function with respect to the parameters, which can be computationally expensive for large datasets. In contrast, in stochastic gradient descent, the gradient is computed based on a randomly selected subset (or minibatch) of the data. In each iteration of SGD, a random batch of samples is selected from the training dataset, and the gradients of the loss function with respect to the parameters are computed using only that batch. The parameters are then updated based on the computed gradients, and the process is repeated for multiple epochs until convergence.

3.7.2 Backpropagation

Backpropagation is the algorithm for training artificial neural networks. It is used to efficiently compute the gradients of the loss function with respect to the weights and biases of the network. The backpropagation algorithm relies on the following crucial property: the gradients of the loss with respect to the weights and biases of layer $l-1$ can be computed easily from those related to the next layer l. Hence, backpropagations computes the loss gradients associated with the last layer L, and then, step by step, use the gradients related to layer l to compute those related to layer $l-1$. More specifically, the gradients associated with the successive layers, from the last to the first, are:

$$\nabla_{\mathbf{W}^{(l)}}\mathcal{L}(\mathbf{\Theta}) \quad \text{and} \quad \nabla_{\mathbf{b}^{(l)}}\mathcal{L}(\mathbf{\Theta}) \quad \text{for all layer } l = L, \ldots, 1.$$

These gradients are then used to update the network's parameters according to the GD Algorithm 1 as follows:

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \lambda \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\Theta) \quad \text{and} \quad \mathbf{b}^{(l)} := \mathbf{b}^{(l)} - \lambda \nabla_{\mathbf{b}^{(l)}} \mathcal{L}(\Theta) \quad \text{for all layer } l = L, \ldots, 1.$$

In this way, all parameters of the network are updated after each co-called 'backward pass'.

Formally, consider some feedforward neural network with L layers governed by the following equations (cf. Equations 3.4):

$$\begin{cases} \mathbf{a}^{(0)} = \mathbf{x} \\ \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \text{ for } l = 1, \ldots, L \\ \mathbf{a}^{(l)} = g^{(l)}(\mathbf{z}^{(l)}) \end{cases}$$

We define the error at layer l by the following expression

$$\boldsymbol{\delta}^{(l)} := \nabla_{\mathbf{z}^{(l)}} \mathcal{L}(\Theta) \; .$$

Using the multidimensional generalization of the chain rule for derivatives, it can be shown that the gradients of the loss $\mathcal{L}$ with respect to the weights $\mathbf{W}^{(l)}$ and biases $\mathbf{b}^{(l)}$ of the successive layers can be computed in the backward order as follows:

$$\begin{aligned} &\text{For } l = L, \ldots, 1 : \\ &\boldsymbol{\delta}^{(l)} = \begin{cases} \nabla_{\mathbf{a}^{(l)}} \mathcal{L}(\Theta) \odot \boldsymbol{\sigma}'(\mathbf{z}^{(l)}), & \text{if } l = L \\ \left[\mathbf{W}^{(l+1)}\right]^T \boldsymbol{\delta}^{(l+1)} \odot \boldsymbol{\sigma}'(\mathbf{z}^{(l)}), & \text{if } L > l \geq 1 \end{cases} \\ &\nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\Theta) = \boldsymbol{\delta}^{(l)} \left[\mathbf{a}^{(l-1)}\right]^T \\ &\nabla_{\mathbf{b}^{(l)}} \mathcal{L}(\Theta) = \boldsymbol{\delta}^{(l)} \end{aligned}$$

where $\odot$ is the Hadamard product. Note that $\boldsymbol{\delta}^{(l)}$ can be computed from $\boldsymbol{\delta}^{(l+1)}$ which allows for an efficient computation of the gradients by successively saving the vectors $\boldsymbol{\delta}^{(\mathbf{L})}, \boldsymbol{\delta}^{(\mathbf{L-1})}, \boldsymbol{\delta}^{(\mathbf{L-2})}, \ldots$

Backpropagation played a crucial role in the success of neural networks, by making them easier to train than other machine learning algorithms with comparable number of parameters.

3.7.3 Hyperparameter Tuning

In contrast to parameters that are learned during the training phase of a network, hyperparameters are the settings that are determined before training a neural network are not learned. Hyperparameters control the architecture of the neural network, its optimization algorithm, and its training process. Hyperparameters include:

- The number of hidden layers, and the number of units per layer.

- Regularization tuning parameters.
- Details of stochastic gradient descent. These includes the batch size, the number of epochs, the learning rate etc.

Chapter 4

Argument Mining

4.1 Introduction

Argumentation is the process of presenting arguments, claims, or reasons to persuade someone of a particular point of view, opinion, or conclusion. It is a fundamental aspect of human communication and critical thinking. In argumentation, individuals use logical reasoning, evidence, and rhetoric to support their position on a topic or issue and to counter opposing viewpoints. Abstract argumentation is a formal framework used in artificial intelligence, computer science, and philosophy to model and analyze argumentation and reasoning processes. It provides a structured way to represent, manipulate, and evaluate arguments and their relationships. Abstract argumentation is often used to handle uncertainty, conflicts, and incomplete information in decision-making and reasoning systems.

Argument mining is a natural language processing (NLP) and computational linguistics technique that involves automatically extracting and analyzing arguments from text, such as articles, essays, debates, and online discussions. The goal of argument mining is to identify and structure the arguments within a given text, helping to understand the reasoning and persuasive elements presented in the content. Argument mining can be used in various applications, including information retrieval, sentiment analysis, debate analysis, and content summarization.

NLP techniques empower argument mining by providing the tools to process and parse text, extract linguistic features, and identify the structural elements of arguments within a given corpus. NLP plays a pivotal role in pre-processing textual data, which is a crucial step in argument mining. It involves tasks like sentence segmentation, part-of-speech tagging, named entity recognition, and syntactic parsing, all of which aid in discerning the boundaries and relationships among statements in a text. Moreover, NLP models and algorithms are instrumental in recognizing argumentative markers, such as claim indicators, premise keywords, and transitional phrases, which are vital in distinguishing argumentative content from non-argumentative text. By leveraging these linguistic cues, argument mining systems can accurately identify and categorize arguments, assigning roles like claims and premises to individual statements. Furthermore, sentiment analysis, a common NLP application, plays a complementary role in argument mining. By determining the sentiment expressed within arguments, NLP can help assess the persuasiveness and emotional tone of the arguments, enriching the understanding of the rhetorical strategies employed.

4.2 Argument Mining Sub-Tasks

A complete end-to-end Argument Mining pipeline consists of the following related sub-tasks [24, 132]: 1) Argument Component Detection (ACD): given a token, classify whether it is part of an argument component or not; 2) Argument Type Classification (ATC): given an argument component, classify it as a *Major Claim*, *Claim* or *Premise*; 3) Link Identification (LI): given an argument component, classify it as either *Linked* or *Not Linked* to another argument component and 4) Link Type Classification (LTC): given a linked argument component, classify whether the link is of a *Support* or of an *Attack* type (See Figure 4.1). The

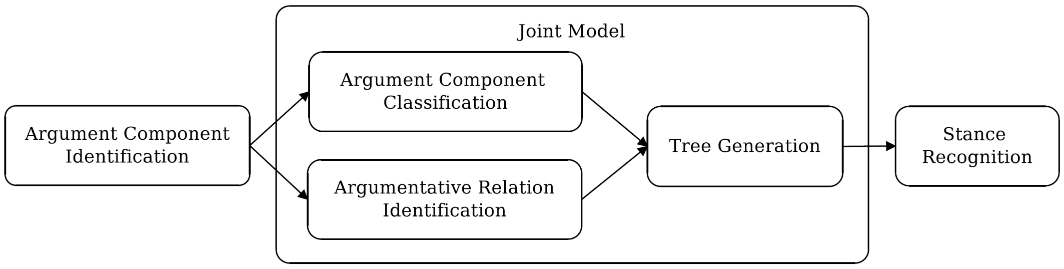

Joint Model
Argument Component Identification
Argument Component Classification
Argumentative Relation Identification
Tree Generation
Stance Recognition

Figure 4.1: Argument Mining pipeline. Taking a body of text as input, the AM pipeline proceeds through various sub-tasks to produce a structured argumentative map of the text which can then be used in task-specific reasoning engines [171].

end output of the Argument Mining pipeline is a tree-like structure of the argumentative text where the classified argument components are the nodes and links between argument components are the edges [171] (See Figure 4.2). In this subsection, we explain each of the Argument Mining sub-task.

Argument Component Detection

The component identification sub-task focuses on the separation of argumentative from non-argumentative text units and the identification of argument component boundaries [171]. This involves identifying and classifying individual tokens or spans of text to determine whether they belong to an argument component or not. This identified argument component represents a single unit of argumentation which interacts with other argument components in the text.

Example: Consider the following paragraph [171]:

'It is no secret that advertising become inseparable part of our modern life. We come across with ads almost everywhere and everyday. Some people argue that **companies, by advertising their goods or services, spend their money for nothing**. Other critics say that ads have no purpose, they only cause damages to society and all forms of advertising should be banned. On my part I would like to solve this issue on a different way. **Society does need an advertising but it is our responsibility to control the content and what kind of goods and services we would like to offer to our customers.**'

In this example, the paragraph contains two argument components (bold). The objective of the argument component identification sub-task is to separate the argument components in bold from the rest of the paragraph and determine the boundaries of the argument components. When considered as a sequence tagging/token classification task, this amounts to assigning to each successive token one of the tags: 'B', 'I', 'O', meaning, respectively, that the token belongs to the beginning of an argument component, is inside an argument component or is not a part of an argument component.

Argument Type Classification

The argument type classification sub-task addresses the function of argument components. It aims at classifying argument components into different types such as major claims, claims and premises [171]. Classifying an argument component as a "claim" or "premise" involves identifying its role within an argumentative structure. These roles are essential for understanding the structure and content of arguments in a text. A major claim addresses the the main topic of the text directly and asserts a stance on it. A claim is an assertion or statement made in support of the stance (major claim). They are often the focal points of a reasoned stance on a topic and are likely supported by premises or evidence. A premise is a statement or piece of evidence that is presented to support a claim. Premises are used to provide reasons or justification for accepting the claim as true or reasonable. They help establish the logical or persuasive basis for the claim. In the previous example, for instance, the argument component 'Society does need an advertising but it is our responsibility to control the content and what kind of goods and services we would like to offer to our customers' is a major claim that directly presents the author's stance on the topic of the essay ('Society should ban all forms of advertising'). This is then followed by a series of paragraphs, each presenting claim(s) and supporting premises for the claim(s).

Link Identification

Having identified argumentative components in the text and classified them according to their argumentative role, the next step is to identify which argument components are linked. The link identification sub-tasks seeks to determine whether two argument components are linked or unlinked. A link represents an argumentative relationship between two arguments that has semantic and structural implications. Intuitively, major claims are unlinked because they are standalone assertions of the author's position on the topic. Claims, on the other hand, are linked to major claims in the sense that they provide support for the major claim. Furthermore, premises are linked to claims because they are a piece of evidence, statistic or warrant presented for a claim.

Example: Consider the following paragraph [171]:

'[**These days, not only many businesses, but also governments have to rely on advertising.**]$_{claim}$. For example, tourism makes up one-third of the Czech Republic's economy. In order to promote the country's attractions, the Czech government has to advertise and sell its services to foreign consumers. [**Only well planned and well targeted advertising will bring more foreign tourists to the country.**]$_{premise}$'

This example paragraph also contains two argument components: a claim and a premise. In this case, the second argument component (premise) is linked to the first argument component (claim) because it is provides a justification or warrant for the author to present the said claim.

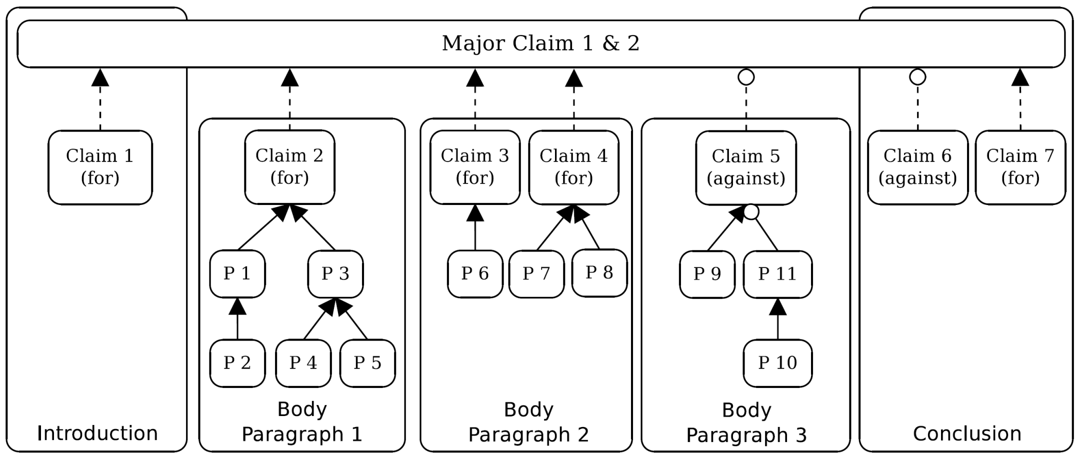


Figure 4.2: An illustration of the output of the AM pipeline on an example text from the Persuasive Essays dataset containing several paragraphs. Argument components have been separated from non-argumentative text and then classified as major claims, claims or premises. Then, argument component pairs within paragraphs are classified as linked or not-linked. Finally, the links between components are classified as *attacking* or *supporting* [171].

Link Type Classification

The link type classification sub-tasks classifies link(s) between two linked argument components as either a supporting or an attacking link. A supporting link means that the source argument component supports the argumentative stance of the target component and, similarly, an attacking link means that the source components disagrees with the target component. In the previous example, for instance, the link between the premise and the claim is a supporting link.

4.3 Argument Mining Approaches

In this section, we will present the datasets and the various approaches to Argument Mining. In his seminal paper, Dung [45], introduced the of argumentation frameworks. Argumentation frameworks provide a formal framework for reasoning about conflicting pieces of information, opinions, or arguments. An argumentation framework is a graph where the nodes are the arguments and the edges represent attacks from one argument to another. In this way, argumentation frameworks model the inter-relationship between the arguments. Argumentation semantics are rules to determine which arguments are accepted in an argumentation framework. Several argumentation semantics have been proposed, establishing different criteria for acceptance of (sets of) arguments, called extensions [13]. Intuitively, these extensions can be seen as the outcomes of the argumentation process modeled by the argumentation framework according to the criteria set out in the particular semantics.

Argument Mining is described as the broad process of examining discourse at the pragmatics level and employing a specific argumentation theory for the purpose of modeling and automatically analyzing the available data [24, 59]. From simple classifiers to more complicated neural models, various model architectures have been developed to tackle the various AM sub-tasks. Argument Mining has been approached from both as a particular task specific perspective and as in end-to-end AM pipeline. Teufel et al., 2009 [181] introduced the notion of argument zoning which they defined as: 'analysis of the argumentative and rhetorical structure of a scientific paper'. Subsequently, Mochales and Moens, 2011 [120] employ a combination of state-of-the-art machine learning methods and context-free grammars to address the complexities inherent in argumentation mining in the legal domain.

With progress in word representation and deep learning techniques, different architectures have been proposed for Argument Mining on datasets constructed from diverse data sources. From an algorithmic perspective, classical techniques like Support Vector Machines [120, 60, 12] and Logistic Regression [96, 125] have been used. Eger et al., 2017 [50] and Niculae et al., 2017 [126] used the more advanced Recurrent Neural Networks (RNN) architecture. More recently, Potash et al. [138] present a Joint Neural Model, based on Bidirectional Long Short-Term Memory (BiLSTM) architecture as well as pre-trained word embeddings, to simultaneously identify of argument component classification and link extraction between argument components. Building on the work of Wang and Chang [192] on span representations of argument components, Kuribayashi et al. [89] introduce a novel modelling approach where they create separate contextualized representations of the argumentative markers and argumentative component present in the argumentative unit using distinct BiLSTMs for both and a suitable initial embedding (GloVe/ELMo). These contextualized representations are then concatenated to obtain an enriched representation of the argumentative unit for both separate and joint learning of AM sub-tasks. Mayer et al. [112] use neural network-based architectures for argument mining in healthcare applications. They merge boundary detection and component classification into a single sequence tagging task. They experiment with several embeddings such as GloVe, ELMo, fastText, FlairPM, etc. with various combinations of LSTMs, GRUs and CRFs as well as fine-tuning the Transformer-based BERT model.

From a data perspective, Argument Mining has been used to model and extract argumentative structure of data from diverse sources and with diverse formats. Scientific articles [181], political debates and parliamentary proceedings [114, 49, 123, 61], legal arguments and judgments [120, 180] and structured essays [171] all present a natural avenue for AM implementation. Furthermore, online debates, forums social media platforms, although more unstructured and polemic in nature, have also been used as a data source for Argument Mining [133]. For a detailed study on Argument Mining from a data perspective, we refer the reader to 'Five Years of Argument Mining: a Data-driven Analysis', Cabrio and Villata [24]. Here, we reproduce two tables (with minor modifications) from the study: Table 4.1 which lists the datasets and data types and Table 4.2 which lists the various algorithms and architectures used by various researches for AM tasks.

Dataset	**Document Sources**	**Size**	**SC**	**BD**	**RP**
Stab and Gurevych, 2017 [171]	persuasive essays	402 essays	Yes	Yes	Yes
Peldszus and Stede, 2015 [133]	microtexts	112 short texts	Yes	No	Yes
Bar-Haim et al., 2017 [12]	debate motions DB	55 topics	Yes	No	No
Rinott et al., 2015 [150]	Wikipedia, debate motions DB	58 topics, 547 articles	Yes	No	No
Bar-Haim et al., 2017 [12]	Wikipedia, debate motions DB	33 topics, 586 articles	Yes	No	No
IAC [186]	4forums.com	11,800 discussions	No	No	No
Habernal and Gurevych, 2017 [60]	comments, forum, blog posts	524 documents	Yes	No	No
Khatib et al., 2016 [3]	i-debate	445 documents	No	Yes	No
NoDE	online debates	260 pairs	No	No	Yes
DART	Twitter	4,713 tweets	No	Yes	Yes
Araucaria	newspapers, legal, debates	660 arguments	Yes	No	No
Teruel et al., 2018 [180]	ECHR judgments	7 judgments	Yes	Yes	Yes
Mochales and Moens, 2011 [120]	ECHR judgments	47 judgments	Yes	Yes	Yes
Niculae et al., 2017 [126]	eRule-making discussion forum	731 comments	No	No	Yes
Menini et al., 2018 [114]	Nixon-Kennedy Presid. campaign	5 topics (1,907 pairs)	No	No	Yes
Lippi and Torroni, 2016a [103]	Sky News debate for UK elections	9,666 words	Yes	No	No
Duthie et al., 2016 [49]	UK parliamentary record	60 sessions	Yes	No	No
Naderi and Hirst, 2015 [123]	Canadian Parliament speeches	34 sent., 123 paragr	No	No	Yes

Table 4.1: Available datasets for AM (sub-)tasks. SC represents Sentence Classification, BD represents Boundary Detection and RP represents Relation Prediction [24].

Approach	*Sentence Classification*	*Boundaries Detection*	*Relations prediction*
SVM	Mochales and Moens, 2011 [120], Duthie et al., 2016 [49] Lippi and Torroni, 2016a; 2016c [102, 103] Habernal and Gurevych, 2017 [60] Bar-Haim et al., 2017 [12]	Mochales and Moens, 2011 [120] Lippi and Torroni, 2016c [103]	Naderi and Hirst, 2015 [123] Niculae et al., 2017 [126] Stab and Gurevych, 2017 [171] Menini et al., 2018 [114]
P	Villalba and Saint-Dizier, 2012 [185] Peldszus and Stede, 2015 [133] Eger et al., 2017 [50]	Eger et al., 2017 [50]	Villalba, Saint-Dizier, 2012 [185] Peldszus and Stede, 2015 [133] Eger et al., 2017 [50]
LR	Levy et al., 2014 [96], Rinott et al., 2015 [150] Nguyen and Litman, 2018 [125]	Dusmanu et al., 2017 [48] Ibeke et al., 2017 [73] Nguyen and Litman, 2018 [125]	Nguyen and Litman, 2018 [125]
RNN	Eger et al., 2017 [50]	Eger et al., 2017 [50]	Niculae et al., 2017 [126] Eger et al., 2017 [50]
ME	Mochales and Moens, 2011 [120], Duthie et al., 2016 [49]	Mochales and Moens, 2011 [120]	
CRF	Stab and Gurevych, 2017 [171]		
NB	Duthie et al., 2016 [49]		
RF		Dusmanu et al., 2017 [48]	
TES			Cabrio and Villata, 2013 [23]
ML		Levy et al., 2014 [96]	

Table 4.2: Available approaches for AM sub-tasks [24]

Part II

Large Language Models

Chapter 5

LLM Basics

5.1 Introduction

Pretrained Foundation Models (PFMs) serve as the cornerstone for a wide array of downstream tasks that involve diverse data modalities. PFMs, like BERT, ChatGPT, and GPT-4, undergo training on extensive text corpora, endowing them with a valuable parameter initialization that are useful for an extensive range of downstream applications. Unlike earlier methodologies relying on convolutional and recurrent modules for feature extraction, BERT learns bidirectional encoder representations from Transformers. Transformers, in turn, are training on large datasets as contextual language models. Likewise, the generative pretrained transformer (GPT) methodology leverages Transformers as feature extractors and adopts an autoregressive approach during training, employing massive datasets. More recently, ChatGPT has demonstrated remarkable success by implementing an autoregressive language model framework, demonstrating exceptional performance even in situations where it encounters zero-shot or few-shot prompts. The development and progress in PFMs have ushered in significant breakthroughs across a multitude of AI domains in recent years.

Large Language Models (LLMs) have emerged as ‘the’ go-to asset in NLP and AI. LLMs are deep learning frameworks that are trained on vast corpora of textual data which in turn provides them with the capability to produce coherent and contextually apt text as responses to user inputs. These models have sparked transformative advancements in a multitude of domains, including text generation, language translation, sentiment assessment, and question resolution. LLMs harness their initial pre-training on diverse textual sources to grasp intricate linguistic patterns and semantic interconnections, thereby capturing the subtleties inherent in human discourse. By using advanced methodologies such as attention mechanisms and transformer architectures, LLMs have achieved noteworthy performance across a broad spectrum of language-related tasks. However, certain challenges, including bias present in training data, ethical considerations, and the need for interpretability, persist as important research topics in the design and implementation of LLMs. As LLMs evolve, their prospective applications in natural language comprehension and generation are set to exert substantial influence in academic, industrial, technology and everyday contexts [216].

Broadly, Language Modeling (LM) seeks to create models that can estimate the likelihood of sequences of words, thereby enabling the predictions of forthcoming or missing tokens. The research landscape in Language Models (LMs) can be divided into four distinct phases, each marked by its unique developments and insights [216].

- *Statistical Language Models (SLM)*: SLMs have their roots in statistical learning techniques that gained prominence during the 1990s [75, 53, 153, 172]. The fundamental concept involves constructing a model for word prediction by leveraging the Markov assumption. In simpler terms, this means making predictions about the next word in a sequence based on the most recent context. SLMs, when designed with a predetermined context length n, are commonly referred to as n-gram language models. Examples include bigram and trigram language models. SLMs have found extensive use in improving task performance in information retrieval (IR) [212, 211] and natural language processing (NLP) [131, 11, 20]. Nonetheless, they have to deal with the ‘curse of dimensionality’: accurately estimating high-order language models becomes a formidable task due to the requirement to estimate an exponential number of transition probabilities. Consequently, specialized smoothing techniques, including backoff estimation [55], and Good–Turing estimation [52], have been introduced to mitigate the issue of data sparsity.

- *Neural Language Models (NLM)*: NLMs employ neural networks, including recurrent neural networks (RNNs), to characterize the probabilities associated with sequences of words [129, 85, 118]. An influential contribution in this field introduced the concept of distributed word representations [129]. This work involved constructing a word prediction function, which relied on contextual features derived from aggregated distributed word vectors. Expanding the idea of learning meaningful representations for words and sentences, a comprehensive neural network approach was formulated to provide a unified solution for various natural language processing (NLP) tasks [34]. Additionally, the advent of word2vec introduced a simplified shallow neural network designed for acquiring distributed word representations [115, 117]. These representations demonstrated exceptional efficacy across a diverse spectrum of NLP tasks. These studies have laid the foundation for the adoption of language models in representation learning and significantly reshaping the landscape of NLP.

- *Pre-trained language models (PLM)*: In an early pioneering effort, ELMo aimed at capturing context-aware word representations [135]. It accomplished this by initially pre-training a bidirectional LSTM (biLSTM) network, rather than relying on fixed word representations. Subsequently, the biLSTM network is fine-tuned, tailoring it to specific downstream tasks. Furthermore, drawing from the highly parallelizable Transformer architecture with self-attention mechanisms, BERT was introduced [184, 37]. BERT involved the pretraining of bidirectional language models using specially crafted pre-training tasks applied to extensive unlabeled corpora. These pre-trained word representations, infused with contextual awareness, have proven highly effective as versatile semantic features. They have significantly elevated the performance benchmarks for a broad spectrum of natural language processing (NLP) tasks. This landmark study has served as a catalyst for a multitude of subsequent endeavors, all operating under the "pre-training and fine-tuning" learning paradigm. Following this paradigm, numerous studies on Pre-trained Language Models (PLMs) have come to fruition. These studies have introduced either novel architectures [97, 51], such as GPT-2 [143] and BART [97], or refined pre-training strategies [110, 157, 191]. Within this paradigm, the common practice often necessitates fine-tuning the PLM to adapt it to various downstream tasks.

- *Large language models (LLM)*: Researchers have observed that the scaling of pre-trained language models (PLMs), whether by increasing the model size or the amount of data, often results in enhanced model capacity for handling downstream tasks [82]. Several investigations have ventured to push the boundaries of performance by training increasingly larger PLMs. Notable examples include the 175-billion-parameter GPT-3 and the colossal 540-billion-parameter PaLM. Even if scaling predominantly pertains to augmenting model size while maintaining similar architectures and pre-training tasks, these massive PLMs actually exhibit distinctive behaviors when compared to their smaller counterparts, such as the 330-million-parameter BERT and the

1.5-billion-parameter GPT-2. This divergence in behavior showcases their surprising capabilities, referred to as “emergent abilities” [196], which enable them to excel in tackling a gamut of intricate tasks. For instance, GPT-3 demonstrates its prowess by effectively addressing few-shot tasks through in-context learning, a feat beyond the reach of GPT-2. Consequently, the research community has coined the term “large language models (LLM)” to categorize these generously proportioned PLMs [165, 195, 70, 176]. One notable achievement stemming from LLMs is ChatGPT, which harnesses the potential of LLMs from the GPT series to engage in dialogues, showcasing an astonishing capacity for conversing with humans. A comprehensive list of popular LLMs is given in Table 5.1.

5.2 LLMs for NLP

Large Language Models (LLMs), exemplified by Transformer-based architectures like GPT and BERT, have demonstrated impressive skills in comprehending, generating, and manipulating natural language. These LLMs are trained on extensive corpora of textual data, endowing them with the capacity to learn linguistic patterns, syntactical structures, and the subtle semantic relationships that underpin language. They harness the power of self-supervised learning techniques to capture contextual information and create meaningful word representations. LLMs excel in a wide array of Natural Language Processing (NLP) tasks, spanning language translation, sentiment analysis, named entity recognition, text summarization, and question answering. Their adeptness at processing and generating coherent and contextually meaningful text has ushered in significant progress within the NLP domain.

The architecture of LLMs is generally composed of a stacking of encoder and/or decoder blocks of the Transformer. In addition, LLMs and are pre-trained on different textual tasks, before being fine-tuned on specific downstream tasks. These features are described in the following sections.

5.2.1 LLM Architectures

Encoder-Decoder or Encoder-only

The widespread availability of natural language data and the introduction of unsupervised training paradigms designed to harness exceedingly vast datasets have ignited a surge of interest in unsupervised natural language learning. One prevalent approach entails predicting masked words within a sentence while taking into account the contextual surroundings, constituting the Masked Language Model training paradigm. This method empowers models to cultivate a better understanding of the relationships between words and the context in which they are used. The models are trained on large text corpora using this method. The results have been nothing short of remarkable, with Masked Language Models consistently achieving state-of-the-art performance across numerous Natural Language Pro-

Model	Release Time	Size	Base Model	Pre-train Data Scale	Training Time
T5	Oct-2019	11	-	1T tokens	-
mT5	Oct-2020	13	-	1T tokens	-
PanGu-α	Apr-2021	13*	-	1.1TB	-
CPM-2	Jun-2021	198	-	2.6TB	-
T0	Oct-2021	11	T5	-	27h
CodeGen	Mar-2022	16	-	577B tokens	-
GPT-NeoX-20B	Apr-2022	20	-	825GB	-
Tk-Instruct	Apr-2022	11	T5	-	4h
UL2	May-2022	20	-	1T tokens	-
OPT	May-2022	175	-	180B tokens	-
NLLB	Jul-2022	54.5	-	-	-
GLM	Oct-2022	130	-	400B tokens	60d
Flan-T5	Oct-2022	11	T5	-	-
BLOOM	Nov-2022	176	-	366B tokens	105d
mT0	Nov-2022	13	mT5	-	-
Galactica	Nov-2022	120	-	106B tokens	-
BLOOMZ	Nov-2022	176	BLOOM	-	-
OPT-IML	Dec-2022	175	OPT	-	-
LLaMA	Feb-2023	65	-	1.4T tokens	21d
CodeGeeX	Sep-2022	13	-	850B tokens	60d
Pythia	Apr-2023	12	-	300B tokens	-
GPT-3	May-2020	175	-	300B tokens	-
GShard	Jun-2020	600	-	1T tokens	4d
Codex	Jul-2021	12	GPT-3	100B tokens	-
ERNIE 3.0	Jul-2021	10	-	175B tokens	-
Jurassic-1	Aug-2021	178	-	300B tokens	-
HyperCLOVA	Sep-2021	82	-	300B tokens	13.4d
FLAN	Sep-2021	137	LaMDA-PT	-	60h
Yuan 1.0	Oct-2021	245	-	180B tokens	-
Anthropic	Dec-2021	52	-	400B tokens	-
WebGPT	Dec-2021	175	GPT-3	-	-
Gopher	Dec-2021	280	-	300B tokens	920h
ERNIE 3.0 Titan	Dec-2021	260	-	300B tokens	28d
GLaM	Dec-2021	1200	-	280B tokens	574h
LaMDA	Jan-2022	137	-	2.81T tokens	57.7d
MT-NLG	Jan-2022	530	-	270B tokens	-
AlphaCode	Feb-2022	41	-	967B tokens	-
InstructGPT	Mar-2022	175	GPT-3	-	-
Chinchilla	Mar-2022	70	-	1.4T tokens	-
PaLM	Apr-2022	540	-	780B tokens	-
AlexaTM	Aug-2022	20	-	1.3T tokens	120d
Sparrow	Sep-2022	70	-	-	-
WeLM	Sep-2022	10	-	300B tokens	24d
U-PaLM	Oct-2022	540	PaLM	-	5d
Flan-PaLM	Oct-2022	540	PaLM	-	37h
Flan-U-PaLM	Oct-2022	540	U-PaLM	-	-
GPT-4	Mar-2023	-	-	-	-
PanGu-σ	Mar-2023	1085	PanGu-α	329B token	100d

Table 5.1: Popular Large Language Models (LLMs) [216].

cessing (NLP) tasks, including sentiment analysis and named entity recognition. Prominent instances of Masked Language Models encompass BERT [37], RoBERTa [110], and T5 [145].

Decoder-only

While language models typically maintain a task-agnostic architecture, their effective deployment necessitates fine-tuning on task-specific datasets. Researchers have observed that the substantial scaling up of language models leads to significant enhancements in few-shot and even zero-shot performance [21]. Among the most successful models for achieving improved few-shot and zero-shot performance are Autoregressive Language Models. These models are trained by generating the subsequent word in a sequence based on the preceding words, a methodology that has found widespread utility in downstream tasks like text generation and question answering. Prominent examples of Autoregressive Language Models include GPT-3 [21], OPT [213], PaLM [32], and BLOOM [202]. GPT-3, often regarded as a game-changer, marked a significant milestone by demonstrating commendable few-/zero-shot performance through the utilization of prompting and in-context learning. This achievement underscored the ascendancy of autoregressive language models. Additionally, there exist models like CodeX [1], optimized for specific tasks such as code generation, and BloombergGPT [203], tailored for the financial domain.

5.2.2 Pre-training Tasks for NLP

The pre-training tasks for language models can be categorized into five distinct groups, each based on specific learning methods. These categories include Mask Language Modeling (MLM), Denoising AutoEncoder (DAE), Replaced Token Detection (RTD), Next Sentence Prediction (NSP), and Sentence Order Prediction (SOP). Notably, RTD, NSP, and SOP are characterized as contrastive learning methods, predicated on the assumption that the observed samples exhibit greater semantic similarity than randomly selected samples. These diverse pre-training tasks contribute to the multifaceted training strategies employed in developing advanced language models [216].

- **Mask Language Modeling (MLM):** MLM involves randomly masking some words in the input sequence and then requires the model to predict these masked words during the pre-training phase. Well-known examples of MLM include BERT [37] and SpanBERT [78].
- **Denoising AutoEncoder (DAE):** DAE introduces noise into the original corpus and tasks the model with reconstructing the original input from the noisy version. BART [97] serves as a notable illustration of this approach.
- **Replaced Token Detection (RTD):** RTD constitutes a discriminative task designed to ascertain whether the language model has replaced the current token. This task was introduced in ELECTRA [33]. By training the model to differentiate between replaced and unchanged tokens, it acquires a deeper understanding of language.

- **Next Sentence Prediction (NSP):** NSP is introduced to facilitate the model's comprehension of the relationship between two sentences and to capture sentence-level representations. In this task, a Pre-trained Foundation Model (PFM) takes two sentences from different documents and determines whether their order is correct. BERT is a well-known example of a model that employs NSP.
- **Sentence Order Prediction (SOP):** Differing from NSP, SOP employs two adjacent fragments from a document as positive samples and considers the exchange of the order of these fragments as negative samples. PFMs that utilize SOP can effectively model sentence correlations, as demonstrated by ALBERT [90].

5.2.3 Data for NLP

Pre-training data

The significance of pre-training data in the evolution of large language models (LLMs) is axiomatic. Serving as the foundation upon which LLMs exhibit their impressive capabilities [4, 82], the quality, quantity, and diversity of this foundational data wield a profound influence over the performance of LLMs [210]. This pre-training data comprises a vast array of text sources, encompassing everything from books to articles and websites. Meticulously curated, this data repository is designed to offer a comprehensive representation of human knowledge, linguistic intricacies, and cultural perspectives. Its importance lies in its ability to imbue language models with a deep comprehension of word knowledge, grammar, syntax, and semantics. Furthermore, it equips them with the aptitude to discern context and generate coherent responses [216].

The diversity inherent in the pre-training data also assumes a pivotal role in shaping the model's efficiency. The selection of an appropriate LLM often hinges on the specific components included in the pre-training data. For instance, models like PaLM [32] and BLOOM [202] excel in multilingual tasks and machine translation due to the abundant presence of multilingual pre-training data. PaLM's exceptional performance in Question Answering tasks benefits from the incorporation of substantial volumes of social media conversations and the Books corpus [32]. Similarly, GPT-3.5 (code-davinci-002) harnesses its code execution and code completion capabilities, which are augmented by the inclusion of code-related data in its pre-training dataset. In summary, when selecting LLMs for specific downstream tasks, it is prudent to opt for models pre-trained on datasets closely aligned with the target domain, as the richness and relevance of pre-training data significantly impact model performance [216].

Fine-tuning data

When deploying a model for downstream tasks, it is crucial to take into account three primary scenarios based on the availability of annotated data: zero, limited, and abundant. Here, we present a concise overview of the appropriate models to consider for each scenario [216].

Zero annotated data: In situations where annotated data is absent, employing Large Language Models (LLMs) in a zero-shot setting proves to be the most suitable approach. Research has demonstrated that LLMs outperform prior zero-shot methods [207]. Furthermore, the absence of a parameter update process ensures that issues like catastrophic forgetting [84] are avoided since the language model parameters remain unaltered.

Limited annotated data: In this scenario, a few-shot approach is adopted, where a small number of examples are directly integrated into the input prompt of LLMs, referred to as in-context learning. These examples effectively guide LLMs to generalize to the task at hand. As reported in [22], this few-shot approach results in substantial performance improvements, sometimes even matching the performance of state-of-the-art fine-tuned open-domain models. Additionally, LLMs' zero/few-shot capabilities can be further enhanced through scaling [22]. Alternatively, certain few-shot learning techniques, such as meta-learning [92] or transfer learning [154], have been developed to bolster fine-tuned models. However, it's worth noting that the performance of these approaches might lag behind that of LLMs due to the smaller scale and susceptibility to overfitting of fine-tuned models.

Abundant annotated data: When there is a substantial amount of annotated data available for a specific task, both fine-tuned models and LLMs become viable options. In most instances, fine-tuning the model can yield a good fit for the data. However, LLMs can be employed to address certain constraints, such as privacy considerations [175]. In this scenario, the choice between using a fine-tuned model or an LLM is contingent upon the specific task and various factors, including desired performance, computational resources, and deployment constraints.

Test data

When deploying Large Language Models (LLMs) for downstream tasks, we frequently encounter challenges arising from distributional disparities between the test/user data and the data used for training. These disparities may encompass domain shifts [217], out-of-distribution variations [39], or even adversarial examples [140]. Such challenges can significantly impede the effectiveness of fine-tuned models in real-world applications, as they are often tailored to specific data distributions and struggle to generalize to out-of-distribution (OOD) data [216].

However, LLMs tend to excel when faced with these scenarios due to their lack of an explicit fitting process. Furthermore, recent advancements have further augmented the capacity of language models in this regard. Notably, the Reinforcement Learning from Human Feedback (RLHF) method has significantly enhanced LLMs' generalization capabilities [130]. For instance, InstructGPT exhibits proficiency in comprehending and following a wide range of instructions across various tasks, occasionally even complying with instructions in different languages, despite the scarcity of such instructions. Similarly, ChatGPT demonstrates consistent advantages in most adversarial and out-of-distribution (OOD) clas-

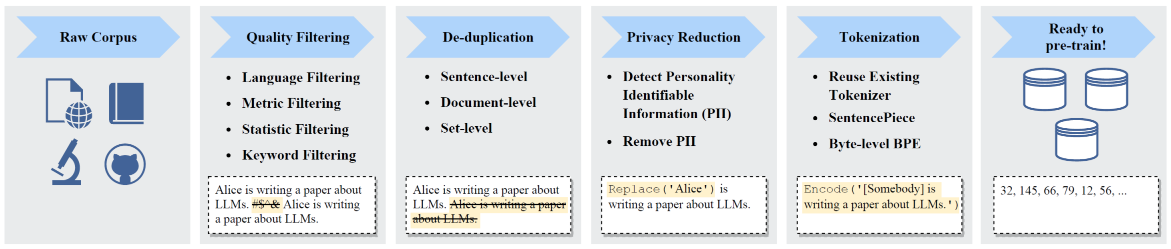

Raw Corpus
Quality Filtering
Language Filtering
Metric Filtering
Statistic Filtering
Keyword Filtering
Alice is writing a paper about LLMs. ~~#$^&~~ Alice is writing a paper about LLMs.
De-duplication
Sentence-level
Document-level
Set-level
Alice is writing a paper about LLMs. ~~Alice is writing a paper about LLMs.~~
Privacy Reduction
Detect Personality Identifiable Information (PII)
Remove PII
Replace('Alice') is writing a paper about LLMs.
Tokenization
Reuse Existing Tokenizer
SentencePiece
Byte-level BPE
Encode('[Somebody] is writing a paper about LLMs.')
Ready to pre-train!
32, 145, 66, 79, 12, 56, ...

Figure 5.1: Illustration of a typical data pre-processing pipeline for pre-training large language models [214].

sification and translation tasks [190]. Its exceptional ability to comprehend dialogue-related texts has resulted in impressive performance on the DDXPlus dataset [179], a medical diagnosis dataset specifically designed for OOD evaluation.

5.2.4 Data Pre-processing

Upon amassing a substantial volume of textual data, it becomes imperative to engage in data pre-processing before constructing the pre-training corpus. This pre-processing phase is particularly crucial for the removal of noisy, redundant, irrelevant, and potentially harmful data [32, 144]. Such data elements have the potential to significantly impact the capacity and performance of Large Language Models (LLMs). In this section, we present a range of data pre-processing strategies aimed at enhancing the quality of the collected data [32, 202, 40]. Figure 5.1 depicts a typical pre-training data pre-processing pipeline tailored for LLMs.

Quality Filtering

To eliminate low-quality content from the collected corpus, existing approaches commonly employ two methods: (1) classifier-based and (2) heuristic-based. In the former approach, a selection classifier is trained using high-quality texts and then used to identify and filter out low-quality data [22, 32, 40]. Typically, these methods train a binary classifier with well-curated data (e.g., Wikipedia pages) as positive instances and candidate data as negative instances, predicting a score that assesses the quality of each data example. However, some studies also caution that a classifier-based approach may inadvertently remove high-quality texts in dialectal, colloquial, and sociolectal languages, potentially introducing bias in the pre-training corpus and reducing corpus diversity [144, 40].

De-duplication

Literature highlights that duplicate data within a corpus can reduce the diversity of language models, potentially destabilizing the training process and impacting model performance [67]. Consequently, de-duplication of the pre-training corpus becomes necessary. Specifically, de-duplication can occur at various granularities, including sentence-level, document-level, and dataset-level de-duplication. Initially, low-quality sentences with repeated words and phrases should be eliminated, as they may introduce repetitive language patterns [71]. At the document level, existing studies often rely on the overlap ratio of surface features (e.g., words and n-grams overlap) between documents to identify and remove duplicate documents containing similar content [183, 144, 202, 93].

Privacy Reduction

A significant portion of pre-training text data is sourced from the web, including user-generated content containing sensitive or personal information, increasing the risk of privacy breaches [25]. Therefore, it's essential to sanitize the pre-training corpus by removing personally identifiable information (PII). An effective approach involves rule-based methods, like keyword spotting, to detect and remove PII such as names, addresses, and phone numbers [91]. Additionally, researchers have observed that the vulnerability of LLMs to privacy attacks can be attributed to the presence of duplicate PII data in the pre-training corpus [81]. Thus, de-duplication also contributes to reducing privacy risks.

Tokenization

Tokenization represents a crucial step in data preprocessing, involving the segmentation of raw text into sequences of individual tokens for subsequent use as input for LLMs. While leveraging an existing tokenizer can be convenient (e.g., OPT [213] and GPT-3 [22] use GPT-2's tokenizer [143]), employing a tokenizer specifically designed for the pre-training corpus offers significant benefits, particularly for corpora comprising diverse domains, languages, and formats [202]. Consequently, several recent LLMs train custom tokenizers explicitly for their pre-training corpora using SentencePiece [87]. The byte-level Byte Pair Encoding (BPE) algorithm [164] is employed to ensure lossless information representation post-tokenization [32, 144].

5.3 Word Representations Methods

In natural language processing and computational linguistics, word representation methods assume a pivotal role. Their primary objective is to efficiently capture the meaning and contextual associations of words. The current landscape of pre-training Language Models (LMs) can be broadly categorized into three branches based on their approach to word representations: (1) autoregressive LM, (2) contextual LM, and (3) permuted LM. Among these three branches, the direction of word prediction and the incorporation of contextual information emerge as the most significant determining factors [216].

Autoregressive Language Model An Autoregressive Language Model falls under the category of natural language processing (NLP) models. It operates by generating text through the prediction of the subsequent word or token in a sequence, primarily guided by the words that precede it. In autoregressive models, the likelihood of the next word is determined by considering the entire prior sequence of words. As a result, the model produces text incrementally, with each word generated while considering the context formed by the words it has previously generated. As a result, it exhibits superior performance in Natural Language Generation (NLG) tasks, including tasks like text summarization and machine translation.

The GPT models are example of an autoregressive model. GPT follows a two-stage approach involving self-supervised pre-training and supervised fine-tuning [142]. Its architecture consists of stacked decoder blocks of the Transformer [184]. Building on this foundation, the OpenAI team expanded upon GPT, giving rise to GPT-2 [143], which boasts an impressive amount of 48 stacked decoder blocks with a total of 1.5 billion parameter. Notably, GPT-2 introduced the concept of multi-task learning, allowing it to adapt to various task models without requiring fine-tuning [26]. GPT-2's primary performance boost arises from a combination of multi-task pre-training, the utilization of extensive datasets, and the employment of large models. However, it is important to note that task-specific datasets are still necessary for fine-tuning on particular downstream tasks. Recognizing the potential for further improvements, GPT-3 was developed, featuring a massive model size of 175 billion parameters and trained on a staggering 45 terabytes of data [22]. This extensive training enables GPT-3 to exhibit strong performance across a wide range of tasks without the need for fine-tuning on specific downstream applications [216].

Contextual Language Model Autoregressive Language Models, while powerful and versatile, do have some limitations. The most important limitation is the lack of bidirectionality: autoregressive models consider only the preceding words (unidirectional context) when generating the next token. Thus they are unable to capture the bidirectional context of words, thereby limiting their understanding of certain language nuances. Contextual Language Model are trained to capture the meaning and nuances of words, phrases, and sentences based on the words that come before and after them in a given context. Contextual language models build their words' representations based on their bi-directional contexts. CMLs are trained on a masked language modeling task where it learns to predict masked-out words using both preceding and following context. This comprehensive view of context helps them understand the meaning of a word in a sentence more accurately. Furthermore, CMLs are better able to capture very long-range dependencies due to their bidirectional attention mechanisms [216].

BERT is an example of a contextual language model [37]. It is composed of stacked decoder blocks of the Transformer operating bi-directionally, which allows it to address the limitations of ELMO and GPT by considering both preceding and succeeding context. However, BERT is not without its drawbacks. It bi-directionality does not entirely eliminate the constraints of self-encoding models. Additionally, the language modeling acquired during pre-training may lead to discrepancies with the model's data used during fine-tuning. Many pre-trained foundation models (PFMs) require additional training tasks and access to more extensive corpora. To address the issue of inadequate training, Liu et al. introduced RoBERTa [110]. RoBERTa employs a larger batch size and leverages unlabeled data. It also extends the model's training duration, eliminates the Next Sentence Prediction (NSP) task, and introduces long sequence training. For text input processing, RoBERTa differs from

BERT by using Byte Pair Encoding (BPE) [164] for word segmentation.

Permuted Language Model Permuted Language Models are a variant of autoregressive language models. The core idea behind a Permuted Language Model is to encourage the model to predict words that have been randomly shuffled within a sentence. This training objective aims to enhance the model's understanding of word order and sentence structure. During training, instead of presenting the model with the original sentences as-is, sentences are randomly permuted by shuffling the order of words or subword tokens within each sentence. The model is then trained to predict the correct order of the shuffled words within each sentence. This prediction task is in addition to the standard language modeling objective, where the model predicts the next word in a sentence based on the preceding context. PLMs, by training on permuted sentences, encourage the model to understand contextual relationships between words in a more comprehensive way including the syntactic and grammatical structures of language such as word order and sentence structure.

Masked Language Modeling (MLM), as demonstrated by BERT, excels in bi-directional encoding. However, MLM employs masking during pretraining but not during fine-tuning, leading to inconsistencies in the data between these phases. To address these issues and achieve robust bi-directional encoding, permuted LM departs from sequential modeling. Instead, it considers all possible permutations of sequences to maximize the expected logarithmic likelihood of sequences. This approach enables any position within the sequence to benefit from contextual information across all positions. Among the prominent permuted LM models, XLNET [206] and MPNet [169] stand out. XLNET incorporates essential techniques from Transformer-XL, including relative positional encoding and the segment recurrence mechanism. On the other hand, MPNet combines Masked Language Modeling (MLM) and permuted language modeling to predict token dependencies. It leverages auxiliary positional information as input, allowing the model to comprehensively analyze entire sentences and reduce positional disparities [216].

Chapter 6

Transformers

Textual data is inherently sequential. Accordingly, *recurrent neural networks* represent a natural fit for handling most NLP tasks. Until recently, the state-of-the-art NLP models were mainly based on recurrent architectures, like LSTM or GRU neural networks [69, 31]. But recurrent architectures present major limitations. First, the processing of input sequences cannot be fully parallelized. Secondly, long-term dependencies between inputs cannot be optimally captured.

In 2017, in a seminal paper entitled "Attention Is All You Need", the *Transformer* model was proposed as a relevant solution to these issues [184]. The model consists of an encoder-decoder feedforward architecture augmented with a self-attention mechanism. According to the authors, their work "propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely" [184]. The feedforward architecture alleviates the parallelization issue and the self-attention mechanism is capable of focusing on any part of the input, to the limit of the model's dimensionality. The main message of the paper is that the attention mechanism can on its own outperform complex architectures. The Transformer model achieved impressive improvements on machine translation tasks. Most of all, it opened the way for a new generation of language models that broke the barriers of NLP.

In particular, the *Generative Pre-Training (GPT)* models (GPT, GPT-2, GPT-3, GPT-4) are children of the Transformer [142, 143, 21]. Their architecture consists of several decoder blocks of the Transformer stacked upon one another. These models can leverage an impressive amount of linguistic information from unlabeled text data, which in turn, can be efficiently transferred to the learning of most common NLP tasks. The GPT model employs a 2-stage training process: an unsupervised pre-training followed by a supervised fine-tuning. During pre-training, a language model is learned using a large corpus of unlabeled text. During fine-tuning, the architecture of the pre-trained model is slightly modified to learn a specific downstream task, using labeled data [142]. In contrast to GPT, the GPT-2 and GPT-3 models are not fine-tuned, but pre-trained on larger and larger text corpora. By means of a simple task conditioning procedure, they can achieve multitask learning via unsupervised pre-training only. In fact, these models show remarkable performance in zero-shot, one-shot and few-shot learning on most common downstream NLP tasks [143, 21].

In 2019, another offspring of the Transformer, the *Bidirectional Encoder Representations from Transformers (BERT)* model has been released [37]. It consists of several encoder (rather than decoder) blocks stacked together. It also uses the 2-stage pre-training and fine-tuning procedure as well. But in contrast to GPT, the left-to-right pre-training process is replaced by a *bidirectional* pre-training consisting of two steps: (i) a masked language model (MLM) task, and (ii) a next sentence prediction task (NSP). This pre-training provides the language model with a representation of both left and right contexts. Fine-tuning is also achieved via minimal modifications of the language model's architecture. With these improvements, BERT achieves impressive performance on most NLP tasks.

The pre-trained BERT model can not only be used as a language model, but also as a word or sentence dynamic *embedding*. In this case, the input text is encoded into a vectorial representation at the character, word, or sentence level, before being fed to a subsequent model for the task at hand. The BERT embedding is *dynamic* is the sense that a same word will not yield the same embedding depending on its left and right contexts. This feature contrasts with the previous embeddings which are mostly *static*. In fact, the BERT embedding takes over most previous pre-trained embeddings like word2vec [116], GloVe [134], FastText [18], ELMo [135] and Flair [2], and appears nowadays as a best choice.

But like all Transformer-based models, BERT is highly resource consuming, containing from 110M to 340M parameters. And while the pre-trained model is available off-the-shelf, the fine-tuning process remains computationally expensive. In an effort to address these issues, lighter and faster versions of the model have been proposed [156, 173]. An enlightening presentation of several Transformer-based models can be found in Jay Alammar's blog [5].

6.1 Encoder-Decoder

Before the Transformer architecture is introduced, some considerations about the encoder-decoder architecture are recalled. An *encoder-decoder* architecture is composed of two models assembled together, an *encoder* and a *decoder* (see Figure 6.2). In summary, the encoder encodes the successive input words, step by step, and generates a *context vector* that represents the embedding of the whole input sequence. The decoder takes the context vector as a hidden state, and outputs a sequence of decoded words, step by step. A typical application for the encoder-decoder architecture is machine translation, where the input sequences are English sentences and the output ones are their corresponding French translations, for instance.

More specifically, consider the case where the encoder-decoder architecture is composed of two recurrent neural networks (RNNs). Recall that the dynamics of an RNN is given by the following equations

$$\begin{aligned} \mathbf{h^t} &= f(\mathbf{x^t}, \mathbf{h^{t-1}}; \mathbf{\Theta_f}) \\ \mathbf{y^t} &= g(\mathbf{h^t}; \mathbf{\Theta_g}) \end{aligned} \tag{6.1}$$

where $\mathbf{h^{t-1}}$ and $\mathbf{h^t}$ are the hidden states of the RNN at time steps $t-1$ and t, respectively, $\mathbf{x^t}$ and $\mathbf{y^t}$ are the input and output of the RNN at time step t, respectively, f is the equation of some recurrent layers, (e.g., LSTMs or GRUs) involving parameters $\mathbf{\Theta_f}$ (biases and weights), and g is the equation of some feedforward layers (e.g., fully-connected or softmax) involving parameters $\mathbf{\Theta_g}$ (biases and weights). According to this equation, the network computes its current state $\mathbf{h^t}$ as a function f of its previous state $\mathbf{h^{t-1}}$ and its current input $\mathbf{x^t}$, and it produces the current output $\mathbf{y^t}$ as a function g of its current state $\mathbf{h^t}$, as illustrated in Figure 6.1.

Suppose that the encoder and decoder are two RNNs given by the functions f_e, g_e and f_d, g_d, respectively. The encoding-decoding process, illustrated in Figure 6.2, can be described as follows. Consider some input sentence, $w_e^1, \dots, w_e^N$, where each w_e^i is a token (word) of the input sentence. The input tokens $w_e^1, \dots, w_e^N$ are first embedded into input vectors $\mathbf{x_e^1}, \dots, \mathbf{x_e^N}$. These input vectors are then passed to the encoder RNN with some initial state $\mathbf{h_e^0}$, producing the sequence of states and outputs $\mathbf{h_e^1}, \dots, \mathbf{h_e^N}$ and $\mathbf{y_e^1}, \dots, \mathbf{y_e^N}$, respectively. The outputs of the encoder are generally discarded. It is worth noting that

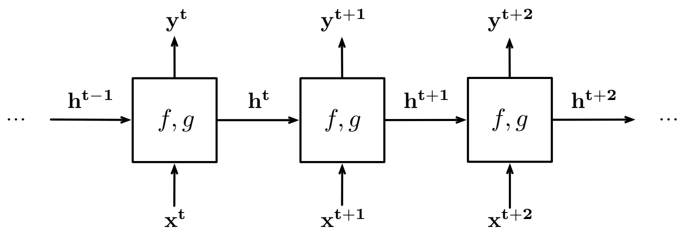


Figure 6.1: Dynamics of an RNN unfolded in time. The current state $\mathbf{h^t}$ of the network is computed as a function f of the previous state $\mathbf{h^{t-1}}$ and the current input $\mathbf{x^t}$, for each $t = 1, \ldots, N$. The current output $\mathbf{y^t}$ is computed as a function g of the current state $\mathbf{h^t}$, for each $t = 1, \ldots, N$.

the last hidden state $\mathbf{h_e^N}$ contains some memory (i.e., some dependence) of all the input vectors $\mathbf{x_e^1}, \ldots, \mathbf{x_e^N}$, by construction. It is usually called the *context vector*, and represents an *embedding* of the whole input sequence. This context vector is then taken as the initial state of the decoder $\mathbf{h_d^0} = \mathbf{h_e^N}$, and the decoding process can begin. A first input token $w_d^1 = \text{[START]}$ is embedded into the input vector $\mathbf{x_d^1}$. When $\mathbf{h_d^0}$ and $\mathbf{x_d^1}$ are passed to the decoder, it produces a hidden state $\mathbf{h_d^1}$ and an output $\mathbf{y_d^1}$. The output $\mathbf{y_d^1}$ corresponds to some decoded word w_d^1. This word is embedded and passed to the decoder as a second input vector $\mathbf{x_d^2}$. With $\mathbf{h_d^1}$ and $\mathbf{x_d^2}$, the decoder produces a hidden state $\mathbf{h_d^2}$ and an output $\mathbf{y_d^2}$. The output $\mathbf{y_d^2}$ corresponds to some decoded word w_d^2, which is embedded and passed to the decoder as a third input vector $\mathbf{x_d^3}$. The process continues until the output token $w_d^M = \text{[END]}$ is produced. The sequence of tokens $w_d^1, \ldots, w_d^M$ obtained via this process is the decoding of the input sequence $w_e^1, \ldots, w_e^N$.

Note that, in the case where the encoder and the decoder are RNNs, the encoding-decoding process cannot be parallelized. More specifically, an RNN cannot process its successive inputs $\mathbf{x^0}, \mathbf{x^1}, \mathbf{x^2}, \ldots$ in parallel, since each current input $\mathbf{x^t}$ requires the preceding hidden state $\mathbf{h^{t-1}}$ to be computed before being processed, according to Eq. (6.1). And the state $\mathbf{h^{t-1}}$ precisely comes from the processing of the previous input $\mathbf{x^{t-1}}$.

6.2 Model

Transformer architecture The *Transformer* model consists of an encoder-decoder architecture, where both the encoder and the decoder are feedforward instead of recurrent neural networks [184]. This architecture makes intensive use of the attention mechanism. The feedforward architecture enables parallelization, and the attention implements memory capabilities, to the limit of the model's dimensionality. This architecture is illustrated in Figure 6.3 and 6.4.

The *encoder* consists of a stacking of 6 identical blocks, each of which being composed of two sub-blocks: a multi-head self-attention mechanism, and a simple fully connected layer. Residual connections followed by layer normalization around each of the two sub-blocks are also added. An encoder block is illustrated in Figures 6.5 and 6.6.

The *decoder* also consists of a stacking of 6 identical blocks, each of which being com-

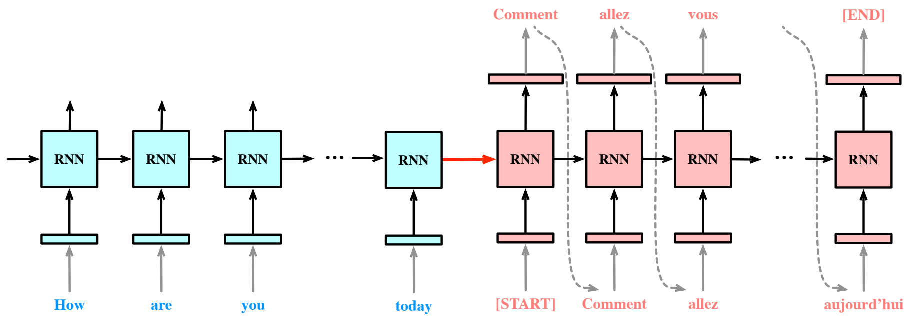


Figure 6.2: Encoder-decoder architecture. The input words (in blue) are embedded (blue vectors) and passed to the RNN encoder (blue boxes). At each step, the RNN encoder computes its current state from its previous state and its current embedded input. After all inputs have been processed, the final state reached by the RNN encoder (red arrow), the context vector, represents an embedding of the whole input sequence. The context vector (red arrow) is then plugged into the RNN decoder (red boxes). At each step, the RNN decoder computes its current state and current output (upper red vector) from its previous state and from the the embedding of the word that has been previously decoded (lower red vector).

posed of three sub-blocks: a multi-head self-attention mechanism similar to that of the encoder, a new multi-head masked attention mechanism operating over the output of the encoder stack, and a simple fully connected layer. Here again, residual connections followed by layer normalization around each of the three sub-blocks are added. In the decoder, the self-attention mechanism includes an additional masking functionality, which ensures that the attention is computed only on the basis of the backward context, since the subsequent tokens have, in theory, not been decoded yet. An decoder block is illustrated in Figures 6.5 and 6.6.

Tokenization, embedding and positional encoding The first stage of the encoder and decoder involves the tokenization, embedding and positional encoding of a given text, illustrated in Example 1 and Figure 6.7. *Tokenization* is the process of splitting a text into a list of tokens. These tokens are then replaced by integers, their token ids, which correspond to indices in a pre-defined vocabulary of about 37 K sub-words. The *embedding* process converts the input ids into dense vectors of dimension 512. Towards this purpose, a pre-defined lookup dense matrix of dimension 30K $\times$ 512 is employed. The k-th row of this matrix corresponds to the embedding of the sub-word with token id k (cf. Figure 6.7). Using this static embedding, the input text is converted into a sequence of embedded inputs. But this embedding process does actually not capture the positions of the tokens in the initial text, which is a crucial information. Hence, the relative positions of the token ids (i.e., $0, 1, 2, \dots$) are encoded into dense vectors of dimension 512 called *positional encodings*, using the following formula

$$
\begin{aligned}
PE(pos, 2i) &= \sin\left(pos/10000^{\frac{2i}{d}}\right) \\
PE(pos, 2i+1) &= \cos\left(pos/10000^{\frac{2i}{d}}\right)
\end{aligned}
$$

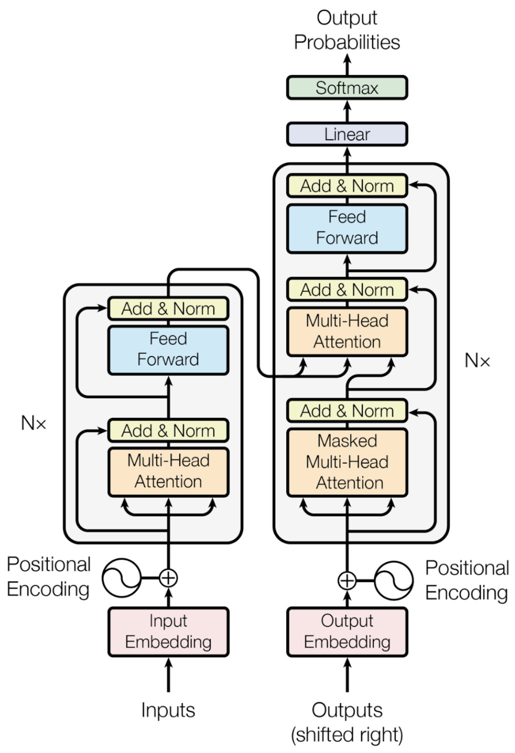


Figure 6.3: The Transformer model. The Transformer is a feedforward neural network arranged in an encoder-decoder architecture. It is composed of $N = 6$ encoding blocks stacked upon one another, followed by $N = 6$ decoding blocks also stacked together. The model makes intensive use of attention. Figure taken from the original paper [184].

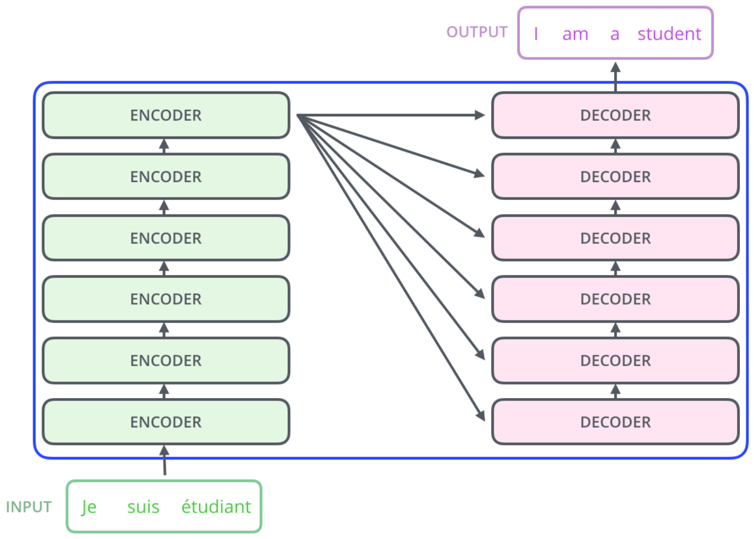


Figure 6.4: Encoder-decoder block architecture of the Transformer. Figure taken from Jay Alammar's blog [5].

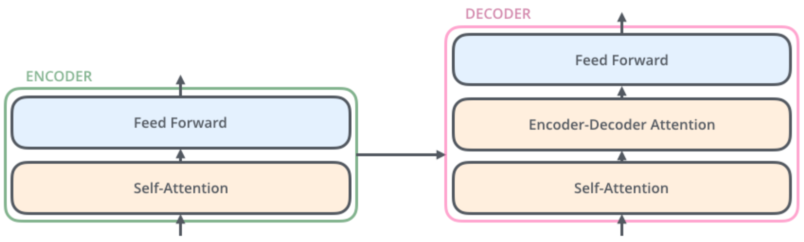


Figure 6.5: An encoder and a decoder block of the Transformer. Figure taken from Jay Alammar's blog [5].

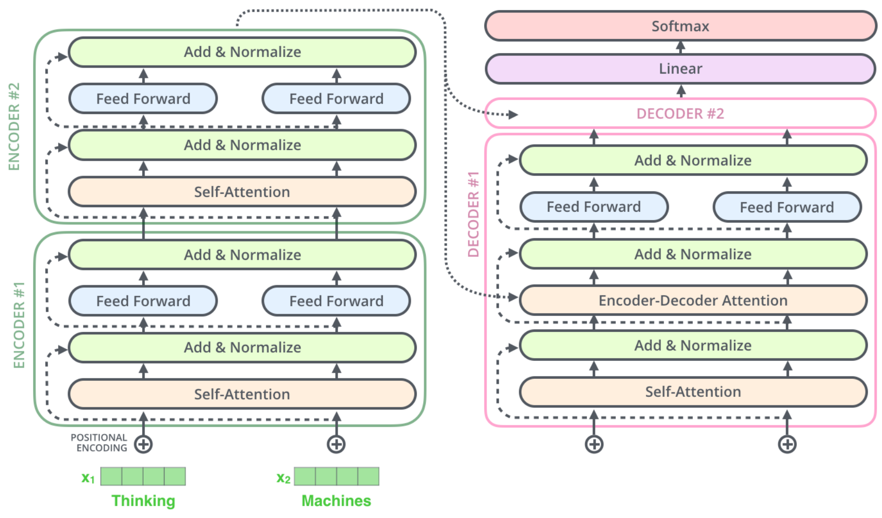


Figure 6.6: Details of two encoder and one decoder blocks of the Transformer. Figure taken from Jay Alammar's blog [5].

where and pos is the position of the input token $(0, 1, 2, \dots)$, i is the encoding's dimension $(0 \leq i \leq 255)$, and $d = 512$ is the embedding's dimension. The positional encodings are then added to the embeddings to form the *embedded inputs*, that can further be passed to the encoder or the decoder (cf. Figure 6.7).

Example 1. *This example shows how an input text is tokenized, converted into token ids, embedded into input vectors, and added to positional encoding vectors. After this process, the inputs can be passed to the Transformer.*

```
# Original Sentence
"Let's learn deep learning!"
# Tokenized Sentence
["Let", "'", "s", "learn", "deep", "learning", "!"]
# Adding [CLS] and [SEP] Tokens
["[CLS]", "Let", "'", "s", "learn", "deep", "learning", "!",
"[SEP]"]
# Padding
```

```
["[CLS]", "Let", "'", "s", "learn", "deep", "learning", "!",
"[SEP]", "[PAD]"]
# Converting to IDs
[101, 2421, 112, 188, 3858, 1996, 3776, 106, 102, 0]
```

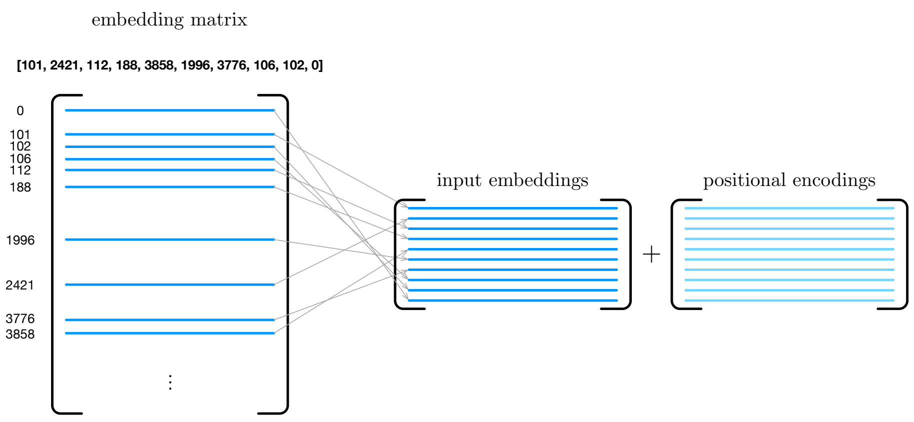


Figure 6.7: The sequence of token ids are embedded into dense vectors of dimension 512, using a static embedding lookup matrix of dimension 37 K × 512 (large matrix). The k-th row of this matrix corresponds to the dense embedding of the sub-word with token id k. For each embedded token id (dark blue vector), its positional encoding of dimension 512 is computed (light blue vector) and added to it.

Attention, self-attention, and multi-head attention We now introduce the *multi-head attention* involved in the Transformer. We describe the self-attention present in each encoder block, as well as the masked self-attention and the attention that happen at the first and second stage of each decoder block, respectively.

Intuitively, an attention mechanism is a process which, for any input sequence $(\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^N})$ and any element $\mathbf{y}$, computes a sequence of weights $(w_1, w_2, \ldots, w_N)$, called *attention weights*. Each attention weight w_i represents the importance that element $\mathbf{y}$ attaches to – or equivalently, the *attention* that $\mathbf{y}$ puts on – the element $\mathbf{x^i}$ of the sequence. The computation ensures that the attention weights sum to 1, i.e., $\sum_{i=1}^{N} w_i = 1$, like probabilities. The larger the attention weight w_i, the more attention is put on $\mathbf{x^i}$ by element $\mathbf{y}$. Whenever $\mathbf{y}$ is an element of the sequence $(\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^N})$ itself (i.e., $\mathbf{y} = \mathbf{x^k}$ for some $1 \leq k \leq N$), we encounter a situation where the attention of $\mathbf{y} = \mathbf{x^k}$ is put on every element of the sequence $(\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^N})$, including itself. This process is referred to as *self-attention*.

The attention mechanism used in the original paper is referred to as *scaled dot-product attention* [184]. This mechanism has been conceived by analogy with a *retrieval system* based on *queries (Q)*, *keys (K)* and *values (V)*. Suppose that a set of objects is stored in a key-value database. The *keys* are the objects identifiers and the *values* are the objects themselves. For a given database *query*, the retrieval system will first return a list of importance scores, the *attention weights*, which describe how much do the successive *keys* match with the *query*. Then, using these attention weights, the answer to the query will be computed as the weighted combination of the *values* associated to the *keys*. In practice, the *queries*, *keys* and *values* are all vectors, and multiple *queries* are performed in parallel. Most importantly, the generation of the *queries*, *keys* and *values* is learned by the model in a precise sense described below.

Example 2. *Consider the following movie review:*

> *"Not a bad* **movie**, *typical action movie with a reasonable storyline."*

Suppose that this review is represented as sequence of embedded words (or embedded tokens), and stored in a key-value database of the following form:

keys	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
values	*Not*	*a*	*bad*	*movie*	*,*	*typical*	*action*	*movie*	*with*	*a*	*reasonable*	*storyline*	*.*

Now, suppose also that the embedded word "movie" (in bold) makes the following query to the database: "Which are the adjectives associated to myself in this review?" As an answer, the retrieval system will first provide a list of attention weights such that the keys 2, 5 *and* 10, *corresponding to the values "*$x_2 =$ *bad", "*$x_5 =$ *typical" and "*$x_{10} =$ *reasonable", obtain high attention weights* w_2, w_5 *and* w_{10}, *respectively, while the other keys will obtain low or zero weights. Then, the answer given to this query by the retrieval system will be the weighted combination of values given by these attention weights, namely,* $\sum_{i=0}^{12} w_i x_i$. *This corresponds to some weighted average of the adjectives associated with the word "movie" in this review. This process describes the attention mechanism associated to a single query, but in practice, every element of the sequence makes its own query, and all the queries are performed in parallel.*

Formally, the *scaled dot-product attention* is composed of three fully-connected layers, a softmax layer, a scaling and a matrix multiplication operations, and potentially, an additional masking operation in the case of *masked attention.* This attention mechanism is illustrated in Figures 6.8, 6.9, 6.10 and 6.11 and given by the following formula explained below:

$$\text{Attention}(Q, K, T) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V. \tag{6.2}$$

Before entering into the detailed description of the mechanism, note that a sequence of vectors is represented and implemented as a matrix, where the successive rows of the matrix are the successive vectors of the sequence. To begin with, three sequences of vectors, the matrices Q_0, K_0 and V_0, associated with queries (Figure 6.9, bottom orange matrix), keys and values (Figure 6.9, bottom blue matrix), are given as input. In this context, the input matrices associated with keys and values are the same, i.e., $K_0 = V_0$. The three matrices are passed through three fully connected layers with weights W^Q, W^K and W^K, respectively, which transforms them into three corresponding matrices Q, K and V called *queries* (Figure 6.9, middle orange matrix), *keys* and *values* (Figure 6.9, middle blue matrix), respectively, i.e.

$$Q = W^Q Q_0 \quad K = W^K K_0 \quad V = W^V V_0. \tag{6.3}$$

Note that the weights W^Q, W^K and W^K are learned during training, meaning that the model learns how the queries, keys and values should be generated from the inputs in order to achieve good performance, in the sense of reducing the loss function. At this stage, each row $(Q)_i$ of Q, $(K)_j$ of K and $(V)_k$ of V represents a single *query*, *key* and *value*, respectively, and each query $(Q)_i$ operates on all the keys and values, i.e., on the full matrices K and V. The matrix representation ensure that all queries are performed in parallel.

As the next step, the *attention scores* are computed as the matrix product QK^T (Figure 6.9, first red matrix). Accordingly, the attention score that the i-th query $(Q)_i$ puts on the j-th key $(K)_j$ is computed as the *dot-product* $(Q)_i \cdot (K)_j$. The attention scores that the i-th query $(Q)_i$ puts on all the keys K is given by the i-th row of QK^T. The attention that all the queries Q puts on all the keys K is given by the matrix product QK^T.

Afterwards, the attention scores are *rescaled* for stability purposes by dividing QK^T by $\sqrt{d_k}$, where $d_k = 512$ is the dimension of the model. The reason for this rescaling is empirical and justified in original paper as follows [184]: "We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$." Then, the *attention weights* are obtained from the attention scores by applying a *softmax* operation row-wise to the matrix $\frac{1}{\sqrt{d_k}}QK^T$ (Figure 6.9, second red matrix). We recall that the softmax operation is defined by

$$\operatorname{softmax}(x_0, \dots, x_n) = \left(\frac{e^{x_0}}{\sum_{i=0}^n e^{x_i}}, \dots, \frac{e^{x_n}}{\sum_{i=0}^n e^{x_i}} \right).$$

In this way, the attention weights associated to each query $(Q)_i$, which correspond to the i-th row of the matrix $\operatorname{softmax}\left(\frac{1}{\sqrt{d_k}}QK^T\right)$, sum to 1.

Finally, the answer $(A)_i$ to each query $(Q)_i$ is given by a weighted sum of all the values V, where the weights are precisely the attention weights that have just been computed (Figure 6.9, third red matrix). Formally,

$$(A)_i = \operatorname{softmax}\left(\frac{1}{\sqrt{d_k}}QK^T\right)_i (V)_i.$$

When all the queries are performed in parallel, the matrix of answers A is given by

$$A = \operatorname{softmax}\left(\frac{1}{\sqrt{d_k}}QK^T\right) V. \tag{6.4}$$

To summarize, the attention mechanism applied to the input sequences Q_0, K_0 and V_0 are given by

$$\text{Attention}\,(Q_0, K_0, V_0) = \operatorname{softmax}\left(\frac{1}{\sqrt{d_k}}(W^Q Q_0)(W^K K_0)^T\right)(W^V V_0), \tag{6.5}$$

according to Equations 6.4 and 6.3. In this way, the input sequence Q_0 associated with the queries is transformed into another sequence A, whose every element $(A)_i$ puts attention on all elements of the sequence associated with the values K_0.

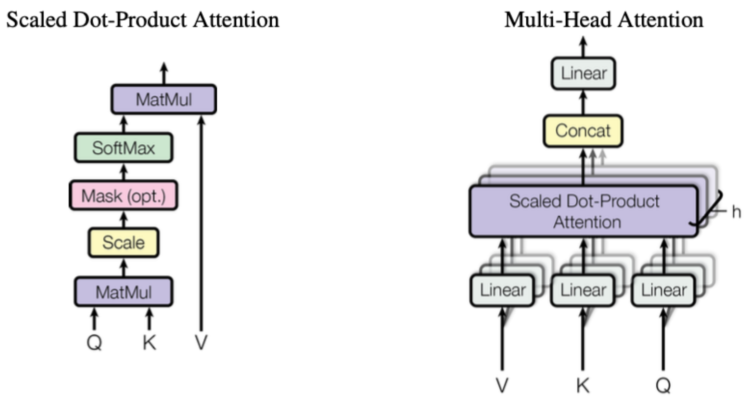


Figure 6.8: Scaled dot-product attention and its multi-headed version. Figure taken from the original paper [184].

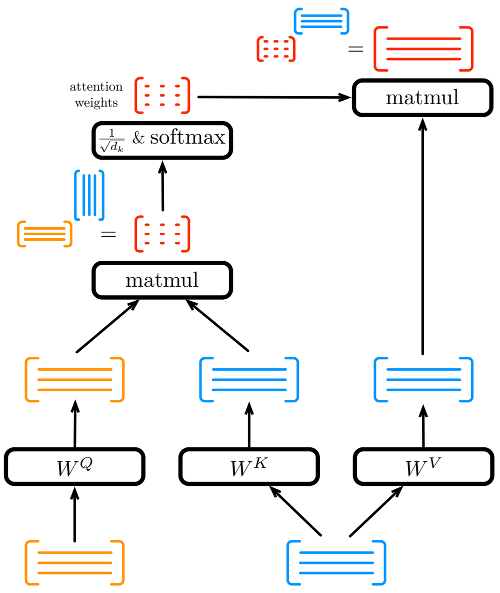


Figure 6.9: Scaled dot-product attention. Sequences of input vectors associated with queries (bottom orange matrix), keys and values (bottom blue matrices) are provided to the system in the form of matrices. These input matrices are passed through fully connected layers to obtain queries (middle orange matrix), keys and values (middle blue matrices). The queries and keys are multiplied to obtain attention scores (first red matrix). The attention scores are rescaled and passed through a softmax to obtain attention weights (second red matrix). The attention weight are multiplied by the values to obtain the answers to the queries (third red matrix). In this way, the input sequence (bottom orange matrix) is transformed into another attention-based sequence (top red matrix).

The mechanism of attention is called *self-attention* when the sequence Q_0 associated with queries is the same as those associated with keys and values K_0 and V_0, respectively. Self-attention is illustrated in Figure 6.10. Intuitively, this means that every element of the input sequence puts attention on itself as well as all other elements of the sequence to which it belongs. Self-attention is employed in the encoder, to compute attention on the input sequence, and in the decoder to compute attention on the sequence of words that have been decoder so far. By contrast, the attention computed in the second part of the decoder is not self-attention. It builds its queries on the basis of the words that have been decoded so far, but builds its keys and values based on the sequence computed by the encoder.

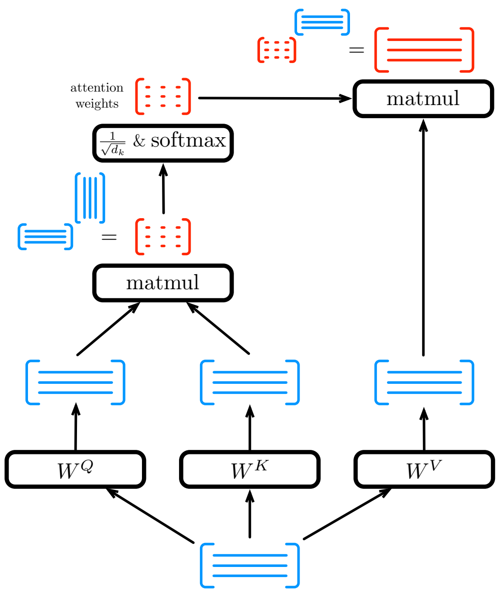


Figure 6.10: Scaled dot-product self-attention. In this case, the queries, keys and values all come from the same input sequence. The input sequence (bottom blue matrix) is transformed into another attention-based sequence (top red matrix).

Regarding the self-attention used in the decoder, a *masking* mechanism needs to be introduced in order to prevent the words that have been decoded so far to put attention on the words that will be decoded in the next steps. Masking is illustrated in Figure 6.11. Towards this purpose, the upper right diagonal of the attention scores is masked with "$-\infty$" values, so that when the softmax is applied, the associated attention weights become equal to 0. Hence, each row i of the attention weight matrix puts zero attention on the next positions $i+1, i+2, \ldots, N$. This means that each query $(Q)_i$ puts attention only on the keys and values $(K)_j$ and (V_j), for $j \leq i$.

Finally, multiple attention processes are applied in parallel, a mechanism referred to as *multi-head attention* (see Figure 6.8). In this case, the results of all attention mechanisms – or *attention heads* – are concatenated row-wise before being passed into a fully-connected layer with weights W^O. Therefore, the full attention mechanism is given by the following equations:

$$\text{MultiHead}\left(Q_0, K_0, V_0\right) = \left[\text{Concat}_{i=1}^{h}\left(\text{Attention}_\text{i}\left(Q_0, K_0, V_0\right)\right)\right] W^O \tag{6.6}$$

$$\text{Attention}_\text{i}\left(Q_0, K_0, V_0\right) = \text{softmax}\left(\frac{1}{\sqrt{d_k}}(W_i^Q Q_0)(W_i^K K_0)^T\right)(W_i^V V_0) \tag{6.7}$$

where W_i^Q, W_i^K and W_i^V are the weights associated with attention head i.

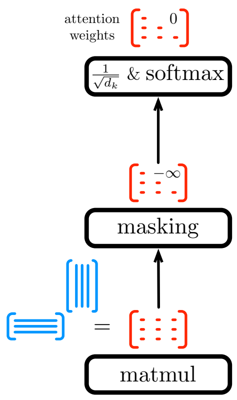


Figure 6.11: Masking mechanism involved in self-attention. A masking operation is inserted between the matmul and the scaling & softmax operations. The upper right diagonal of the attention scores is masked with $-\infty$, so that when the softmax is computed, the associated attention weights become 0.

Fully connected layers, residual connections and layer normalization In each encoder and decoder block, a fully connected layer is added on top of the the attention mechanism. These layers transform the vectors obtain via multi-head attention into vectors of the same dimension, namely $d_k = 512$ in this case. In addition, *residual connections* (or *skip connections*) are employed around each attention and fully connected sub-block [66]. We recall a residual connection around a block consists in summing up the input of the block with its output in order to bypass it. Residual connections have been empirically shown to help convergence in training. This process is followed by *layer normalization* [10]. This technique aims at maintaining the mean and standard deviation of the previous layer activations, within each example, close to 0 and 1, respectively. It has been empirically shown to stabilize the hidden state dynamics, and in turn to reduce the training time of recurrent networks. It also has benefits for feedforward networks, like transformers.

Finally, on top of the last decoder block is a fully connected layer followed by a softmax operation, which transform the output vectors of dimension $d_k = 512$ to probability vectors of dimension equal to the vocabulary size, namely around 37 K. Accordingly, the index of the largest value in each output probability vector corresponds to the token id of the word output by the decoder.

6.3 Training and Inference

Training The training and inference modes of the transformer are different. The training process is performed via *teacher forcing*, as described below and illustrated in Figure 6.12. Suppose that the Transformer is being trained on an English-to-French translation task. Consider some sentence pair (e.g., “Hello, how are you doing?" and “Bonjour, comment allez-vous?”). The English sentence (“Hello, how are you doing?") is first passed to the encoder. Next, the output of the encoder and the target sentence (“Bonjour, comment allez-vous?”) are passed to the decoder. The decoder thus produces a sequence of output probabilities, which corresponds to the translated sentence. Afterwards, the *cross entropy loss* between the sequence of probabilities output by the decoder and the sequence of probabilities associated with the target sequence is computed, the gradients are computed, and the model is trained via backpropagation. This process is performed by batch.

The approach of feeding the target sequence to the decoder during training is referred to as *teacher forcing*, due to the analogy of a teacher that would provide us with the answer to some problem in order to learn from it. This approach has two advantages over a more straightforward step-by-step inference-like mode. First and foremost, the model can be trained in parallel, which significantly speeds up the training process. Indeed, with the help of the target sequence, the model is able to output all the decoded words in parallel, before applying backpropagation. Secondly, the model learns to predict each word based on the correct preceding words, instead of on potentially erroneous previous predictions. This feature prevents the errors made by the model from getting accumulated along the decoding process.

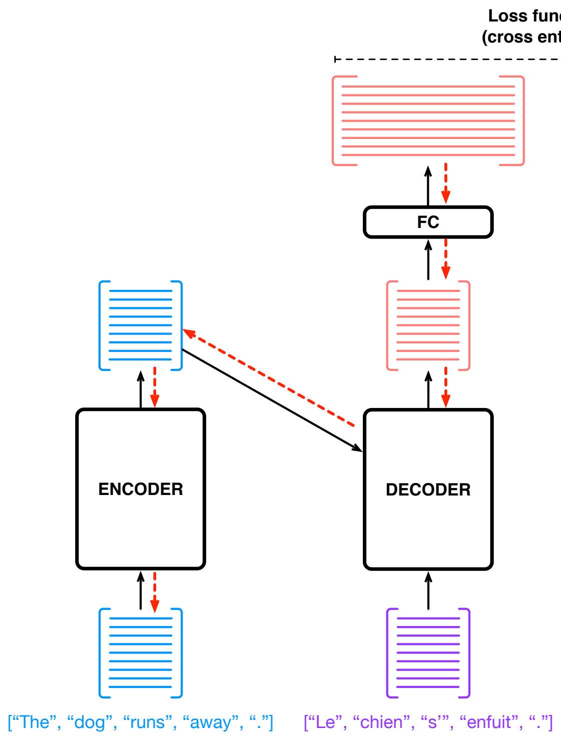

Loss function
(cross entropy)
FC
ENCODER
DECODER
["The", "dog", "runs", "away", "."]
["Le", "chien", "s'", "enfuit", "."]

Figure 6.12: Training process of a Transformer. The input sequence ("The dog runs away.") is passed to the encoder, and the target sequence ("Le chien s'enfuit.") is provided to the decoder in a teacher forcing way. With this information, the network outputs a decoded sequence. The probability sequence output by the model (upper red matrix) is then compared to that associated with the target sequence (upper purple matrix). The cross entropy loss between the two sequences is then computed and the network is trained via backpropagation (red dashed backward arrows).

In the original paper, the Transformer model is trained on two datasets. First, the WMT 2014 English-German dataset which consist of about 4.5 M sentence pairs. In this case, the sentences are encoded using *byte-pair encoding* with a shared source-target vocabulary of about $37\,\mathrm{K}$ tokens. Secondly, the significantly larger WMT 2014 English-French dataset consisting of $36\,\mathrm{M}$ sentences and a $32\,\mathrm{K}$ word-piece vocabulary. Sentence pairs are batched together by approximate sequence length. The chosen optimizer is Adam with parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$ and a variable learning rate. The loss function is the cross entropy (or a variant of it). The larger models were trained for $300K$ steps, which took approximately 3.5 days. For more details, the reader is invited to refer to the original paper [184].

Inference The inference mode of the Transformer is achieved in a step-by-step mode, as in a Seq2Seq model. This process is illustrated in Figure 6.13. At the beginning, the whole input sequence is passed to the encoder, producing a corresponding encoded sequence. This encoded sequence as well as a [START] token is then given to the decoder. The decoder generates an output probability vector corresponding to the first decoded word. At the next time step, this first decoded word is appended to the [START] token and fed back to the decoder, together with the encoded sequence. The decoder then generates a sequence output probability vectors whose last element corresponds to the second decoded word. At the following time step, this second decoded word is appended to the sequence of tokens that have been decoded so far and fed back to the decoder, together with the encoded sequence. The decoder then produces the third decoded word. And so on and so forth. The decoding process continues until the end-of-sentence token [END] is reached.

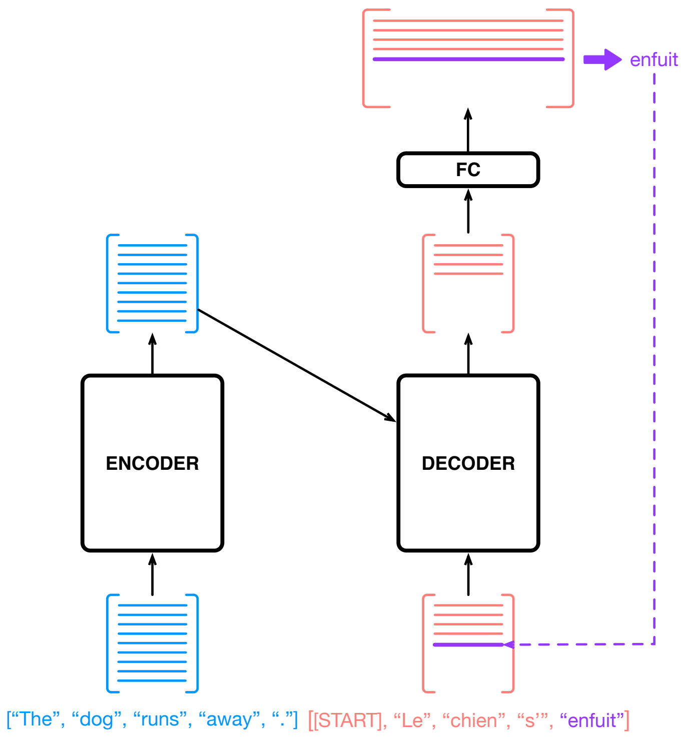

enfuit
FC
ENCODER
DECODER
[“The”, “dog”, “runs”, “away”, “.”]
[[START], “Le”, “chien”, “s’”, “enfuit”]

Figure 6.13: Inference process of a Transformer. The input sequence (“The dog runs away.”) is passed to the encoder, producing an encoded sequence (upper blue matrix). At each time step, this encoded sequence together with the sequence of words that have been decoded so far (“[START] Le chien s’") are passed to the decoder. The decoder generates a corresponding sequence of probability outputs, whose last element (purple vector) corresponds to the last decoded word (“enfuit”). Afterwards, this last decoder word is appended to the previous decoded words, and this new sequence of decoded words is fed back to the decoder (purple arrow). The decoding process continues until the end-of-sentence token is reached.

Chapter 7

Bidirectional Encoder Representations from Transformers (BERT)

7.1 Introduction

BERT, or Bidirectional Encoder Representations from Transformers, is a neural network architecture designed for natural language processing (NLP) tasks. It was introduced by researchers at Google in 2018 and quickly became one of the most widely used NLP models. BERT had a revolutionizing effect on NLP because it proved the importance of bidirectional pre-training of language representations, made it possible to use one unified architecture for distinct NLP tasks, and advanced the state of the art on relevant NLP tasks [37].

Before the introduction of BERT, the two dominant pre-trained language representation strategies were the features-based and fine-tuning approaches. In the features-based approach, task-specific neural architectures are used along with pre-trained word embeddings like ELMO as additional features [135]. In the fine-tuning approach, such st the Generative Pre-trained Transformer (GPT), the pre-trained architecture is fine-tuned on a downstream task by fine-tuning all its parameters [142].

In natural language, the meaning of a word/text sequence depends not only on the context of the previous tokens but also on the context of the upcoming tokens. Consequently, a good language models needs the capability to take in to account the bidirectional context of a token/sequence. Traditional models, such as word embeddings and recurrent neural networks, and existing techniques (above) suffer from the following limitation: they could only process language in a single direction (either left-to-right or right-to-left), which limited their ability to capture the full context of a sentence. BERT addressed this limitation by using a transformer-based architecture that could analyze the entire sentence in both directions, allowing it to better understand the relationships between words and the context in which they were used.

7.2 BERT Architecture

In this subsection, we will explain the BERT model architecture. The BERT model architecture is a multi-layer bidirectional Transformer encoder based on the the self-attention Transformer mechanism of Vaswani et al., 2017 [184]. More specifically, BERT is composed of several encoder blocks of the Transformer piled one upon another. The architecture of particular BERT models is defined by their number of layers (transformer blocks), L, the hidden size of the layers, H, and the number of self-attention heads, A. The original authors experimented with two BERT model configurations:

- $\text{BERT}_{\text{BASE}}$: 12 layers, 768 hidden size dimension and 12 self-attention heads ($L = 12$, $H = 768$, $A = 12$). This configuration has a total of 110 million parameters.
- $\text{BERT}_{\text{LARGE}}$: 24 layers, 1024 hidden size dimension and 16 self-attention heads ($L = 24$, $H = 1024$, $A = 16$). This configuration has a total of 340 million parameters.

BERT models combine a feed-forward architecture with a self-attention mechanism, which allow to parallelize the computation and focus on any part of the input, respectively.

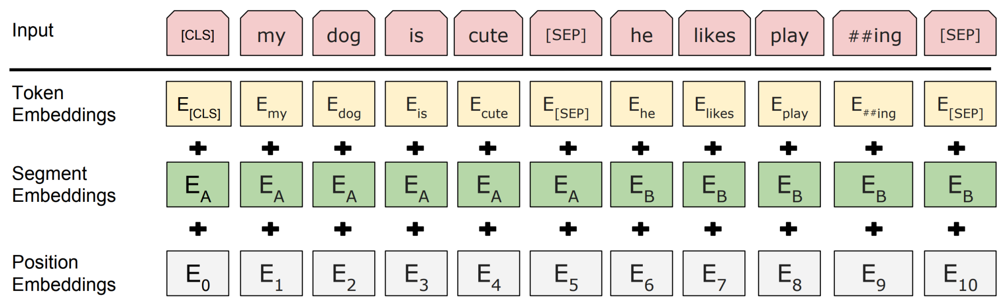


Figure 7.1: BERT input representation for a text sequence consisting of two sentence ('my dog is cute' and 'he likes playing'). Notice that the token 'playing' in the second sentence has been subword tokenized. The start of the text sequence is denoted by the [CLS] token and the two sentences are separated by the [SEP] token. For every token, its input representation is the sum of its token embedding, its segment embedding and it's positional embeddings [37].

BERT architecture consists of twelve encoder blocks of the Transformer model stacked together and 12 self-attention heads [37]. The self-attention heads enable BERT to incorporate bidirectional context and focus on any part of the input sequence. BERT builds a 768 dimensional representation – or embedding – of the input text sequence. BERT architecture also involves a novel input/output representation scheme. For BERT to be able to handle a variety of downstream tasks, it has to be able to provide accurate representations for both a single sentence (for the text classification task, for example) and a pair of sentence (for a text entailment task, for example). BERT uses the 30,000 token vocabulary WordPiece embedding [204], which uses an algorithm to break down words into smaller subword units, called word pieces. In this way, the model can handle words that it has not seen before by recognizing the similarity of the word pieces to those it has already learned [37].

Furthermore, BERT uses two special tokens in it's input representations: the '[CLS]' token and the '[SEP]' token. The [CLS] token denotes the first token of every sequence. For text classification, the final hidden state, or the embedding, of this token is also used as the combined sequence representation. The [SEP] token, on the other hand, is used to separate two sentences in the sequence. Additionally, for every token, BERT adds learned embeddings signifying which of the two sentences the token belongs to. Then, for a given token, its input representation is defined as the sum of the token embedding, the segment embedding (which encodes the sentence in the sequence it it belongs to) and the position embedding (which encode the position of the token in the input sequence) of the token. The construction of this input representation is shown in Figure 7.1.

BERT model is trained in a two step process: In the first step, called pre-training, the BERT model is trained in unsupervised fashion on a huge corpus of data on different pre-training tasks. In the second step, called fine-tuning, the pre-trained BERT model is fine-tuned on using labeled data from a dataset on a particular downstream task. The pre-training and fine-tuning steps of the BERT model are shown in Figure 7.2. In the following subsections, we explain these two steps in detail.

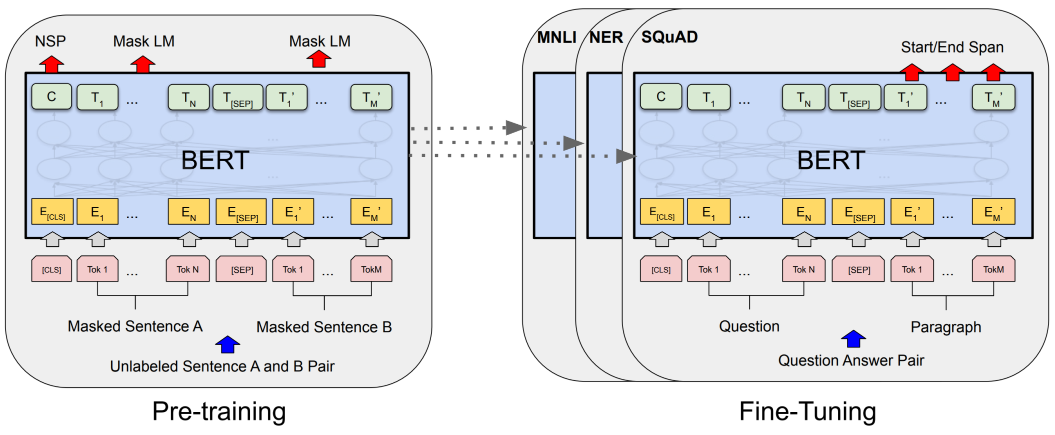


Figure 7.2: The pre-training (left) and fine-tuning (right) processes of BERT. In pre-training, the value of the [CLS] token is used for next sentence prediction (NSP) task and token predictions are obtained for the masked tokens (15%) in the masked language model (MLM) task. In fine-tuning, the pre-trained BERT model is initialized and optimized on a specific dataset for a downstream NLP task. For example, on the right, the starting and ending token positions of the answer span for a SQuAD dataset sample are shown [37].

7.3 BERT Pre-training

BERT is pre-trained using a huge corpus of data on two unsupervised tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP).

Masked Language Model (MLM) task In MLM, some of the input tokens in a sequence are randomly masked, and the objective is to predict the masked tokens based on the context provided by the surrounding tokens. The pre-training setup of the BERT model on the MLM tasks words as follows: First, some percentage of the input tokens are masked (replaced by the '[MASK]' token). Then, the model is asked to predict the masked tokens by feeding the final hidden states of the latter to a softmax output layer. In the original BERT experiments, 15% of all the tokens in each input sequence are masked at random and the masked words are predicted.

This masked prediction procedure has one limitation. Since the fine-tuning process does not have a [MASK] token, this leads to an inconsistency between pre-training and fine-tuning. To rectify this inconsistency, BERT uses a mixed strategy for the masked tokens. This mixed strategy involves selecting 15% of the tokens for masking and then replacing the i-th chosen token by one the of the following three tokens: either the [MASK] token, for 80% of the times; or a random token, for 10% of the times; or the unchanged token itself, for the remaining 10% of the times. In this way, the last hidden vector for the i-th token $\mathbf{T_i}$ is used to predict the original token with cross-entropy as the loss function [37].

Next Sentence Prediction (NSP) task Several NLP tasks, like Question Answering, require the model to be able to capture the relationship between two sentences. In NLP, Next Sentence Prediction (NSP) is the task whose goal is to predict whether two sentences in a given text segment are consecutive or not. In addition to the MLM task, which endows BERT with the ability to understand context and dynamics based on token relationship, BERT is also trained on an unsupervised NSP task to enable it to understand sentence relationships.

This NSP pre-training task works by generating sentence pairs from a huge text corpus. For each pre-training sample (A, B), in 50% of the cases, the chosen sentences B corresponds to actual sentence following sentence A in the corpus, while for the 50% of the cases, the sentence B is randomly chosen from the corpus. These labels are called *IsNext* and *NotNext* respectively. The model then learns to predict, for each sample (A, B), whether sentence B follows sentence A in the original corpus text. The MLM and NSP tasks are illustrated in Figure 7.2.

Pre-training data As mentioned above, to achieve meaningful, deep bidirectional representations, the BERT model is trained on MLM and NSP tasks, utilizing a huge corpus of textual data. Specifically, BERT is pre-trained using the BooksCorpus and English Wikipedia. BookCorpus is a large collection of free novel books written by unpublished authors, which contains 11,038 books (around 74M sentences and 1G words) of 16 different sub-genres (e.g., Romance, Historical, Adventure, etc.) [218]. The English Wikipedia dataset consists of 2,500M words.

7.4 BERT Fine-tuning

The pre-training process enables BERT to learn deep, contextualized representations of text. However, the pre-trained BERT model is a general-purpose language model. To optimized a pre-trained BERT model for a downstream task, such as sequence tagging and natural language inference, BERT is fine-tuned in a supervised fashion on a particular dataset with a task specific loss function. This process is so-called because the parameters of the pre-trained BERT model are 'fine-tuned' end-to-end during the training phase on the downstream task dataset.

More specifically, for every downstream task, a dataset consisting of input/output samples is fed into the pre-trained BERT model. For example, in the Question Answering task, the input to BERT consists of a 'question' text sequence and a 'passage' text sequence. In the text classification task, on the other hand, the input consists of a single text sequence which is to be classified into a pre-defined class. The text sequences are first tokenized using an appropriate tokenizer and a pre-trained BERT model is initialized. Then, the tokenized dataset is fed into the BERT model in a batched fashion, and the model is trained. In this step, the weights of the pre-trained BERT model are updated using backpropagation with respect to the loss function. The token/sentence representations from BERT are fed into an output layer for answering/classification. Once the model has been trained on the downstream test dataset (i.e. the loss function has been minimized), it can be used for inference: to predict the output for the input/output samples in the test dataset.

In their original contribution, the authors of the BERT model experimented and achieved state-of-the-art results on 11 NLP tasks. We briefly describe these in the following. The BERT fine-tuning process on these tasks is depicted in Figure 7.3. For details and results, we refer the interested reader to the original paper [37].

GLUE The GLUE benchmark consists of several natural language understanding tasks used for evaluating different language models in the NLP community. The GLUE benchmark consists of the following datasets: Multi-Genre Natural Language Inference (MNLI) [200], Quora Question Pairs (QQP) [30], Question Natural Language Inference (QNLI) [188], Stanford Sentiment Treebank (SST-2) [167], Corpus of Linguistic Acceptability (COLA) [194], Semantic Textual Similarity Benchmark (STS-B) [27], Microsoft Research Paraphrase Corpus (MRPC) [38], Recognizing Textual Entailment (RTE) [16] and Winograd NLI (WNLI) [95] [189].

For fine-tuning on a GLUE task, the input representations of the text sequences (single sentences/sentence pairs, depending on the task) are computed. Then, the final hidden vectors corresponding to the [CLS] tokens of these texts are selected as the aggregate representations, or the embeddings, of the input texts. Finally, these aggregate representations are fed into a classification layer [37].

SQuAD v1.1 SQuAD v1.1 (Stanford Question Answering Dataset) is a dataset consisting of 100k crowd-sourced question-answer pairs. For a given question and a Wikipedia passage containing the answer, the task is to predict the answer span in the passage. The input is represented as a joint packed sequence. Then, the probability of a word i being the start or the end token of the answer span (i, j) are given by:

$$P_i^s = \frac{e^{\mathbf{S}\cdot\mathbf{T_i}}}{\sum_j e^{\mathbf{S}\cdot\mathbf{T_j}}} \text{ and } P_i^e = \frac{e^{\mathbf{E}\cdot\mathbf{T_i}}}{\sum_j e^{\mathbf{E}\cdot\mathbf{T_j}}}$$

where $\mathbf{S}, \mathbf{E} \in \mathbb{R}^H$ are the start and end vectors respectively, and the sum is taken over all tokens in the paragraph. The score of a span (i, j) (with $j \geq i$) is then defined by $\mathbf{S} \cdot \mathbf{T_i} + \mathbf{E} \cdot \mathbf{T_j}$, and the maximum scoring span is chosen as the prediction. To compute the loss, the true start and end positions of the answer are converted into one-hot vectors, and the cross-entropy loss is computed between the predicted and true distributions. The negative log-likelihood of the true start and end positions is used as the loss function [37].

SQuAD v2.0 SQuAD v2.0 is an updated version of the SQuAD dataset. In addition to the data in SQuAD v1.1 dataset, each question in SQuAD v2.0 is labelled either ‘answerable’ or ‘unanswerable’. If a question is labelled unanswerable, it means that the answer to it does not exist in the given passage. In this way, SQuAD v2.0 provides a more realistic and challenging scenario for the Question Answering task than the previous version.

When fine-tuning BERT on SQuAD v2.0, questions which do not have an answer in the provided passage are assigned the span starting and ending with the [CLS] token. To find the best predicted span, the score of the no-answer span $s_{null} = \mathbf{S} \cdot \mathbf{C} + \mathbf{E} \cdot \mathbf{C}$ is compared with the best score of the non-null span $\hat{s}_{i,j} = \max_{j \geq i} \mathbf{S} \cdot \mathbf{T_i} + \mathbf{E} \cdot \mathbf{T_j}$. A non-null answer is predicted when $\hat{s}_{i,j} > s_{null} + \tau$, for some threshold τ selected so as to maximize the F1 score [37].

SWAG The Situation With Adversarial Generations (SWAG) consists of 113k sentence-pair completion examples. For a sentence, the task is to choose the most likely completion/continuation among the four given choices. For the SWAG dataset, four input sequences are constructed during fine-tuning. Each sequence is a concatenation fo the given sentence (sentence A) with one of the four continuation options (sentence B). For each choice, a normalized score is obtained by computing the dot product of a vector assigned to each continuation with the final hidden representation of the [CLS] token followed by a softmax layer.

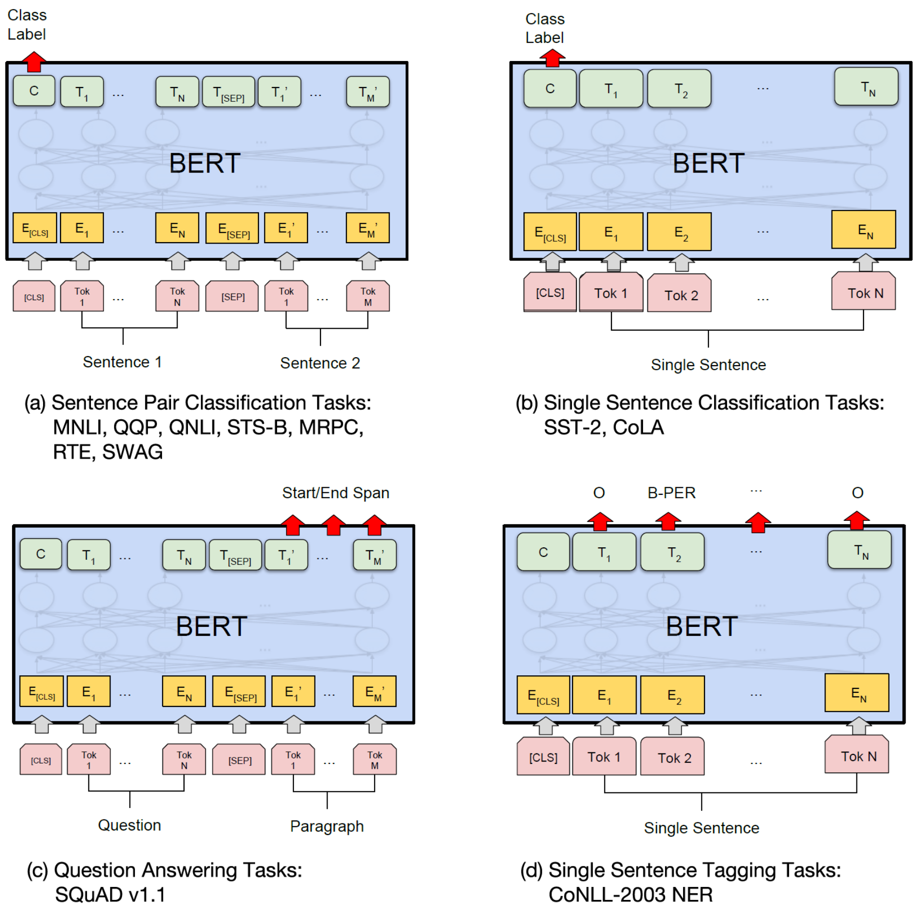


Figure 7.3: BERT fine-tuning on different NLP tasks. Panels (a) and (b): Either sentence pair or single sentence classification tasks on various datasets. The [CLS] token is considered as the aggregate representation of the input sequence and fed to a classification layer to obtain the predicted class label. Panel (c): Question answering tasks. The scores of various answer spans are computed and the best one selected as the predicted span. Panel (d): Token-level sequence tagging task (e.g., Named Entity Recognition). Every token i is assigned with a class label by feeding its token representation $\mathbf{T_i}$ into a classification layer [37].

Chapter 8

Prompt Engineering

8.1 Introduction

Natural Language Processing (NLP) has progressed within two dominant frameworks or paradigms: the classical fully supervised learning paradigm and the more recent ‘pre-train, fine-tune’ paradigm. In *fully supervised learning*, a task-specific model is trained on a labelled dataset consisting of input-output samples. The mapping learned during the training phase is then used to run inference on unseen data samples. In *pre-train, fine-tune* paradigm, a large language model (LLM) is first pre-trained in an unsupervised way son a large data corpus so as to learn general purpose language representations. Then, this pre-trained LLM is fine-tuned on a labeled dataset and a custom objective function so as to capture and learn dataset and task specific representations, respectively. This fine-tuned model is then used for inferences on the unseen data samples.

The defining characteristic of both paradigms narrowed the focus of research in particular ways. In fully-supervised learning, two sub-paradigms emerged: feature engineering and architecture engineering. Feature engineering was motivated by the limited availability of high-quality, labelled and annotated data. In feature engineering, researchers used their domain knowledge and expertise to define and design hand-crafted features from raw data that help the full-supervised model learn representations of data. For example, for text classification task, researchers designed several feature groups like structural, syntactic and lexical features, which were used to build meaningful representations of available data. Architecture engineering, on the other hand, involved developing better and more efficient neural network architectures, where features are learned in the hidden layers of the network during the training of the model. For example, to process sequential data where the representation one element depends on that of its previous element(s), recurrent neural networks (RNNs) were introduced. Thereafter, Long Short-Term Memory (LSTM) neural networks were introduced to process long-term dependencies in data. LSTMs were enhanced with deal with bidirectional context, resulting in the BiLSTM neural architecture.

In the ‘pre-train, fine-tune’ paradigm, focus has been put on objective engineering, which involves designing objective functions for both the pre-training and fine-tuning phases. The idea behind this approach is that optimizing different objective functions in both phases will lead to improvement in model performance on downstream tasks. The transition from the fully-supervised to the ‘pre-train, fine-tune’ learning paradigm has represented a transformative change in NLP. Since raw data which can be used for pre-training large language models (LLMs) is ubiquitous and plentiful, fully-supervised learning is playing an increasingly smaller role in NLP, and attention and research focus has shifted to LLMs and their capabilities.

More recently, NLP is undergoing another transformative change. A third paradigm, called ‘pre-train, prompt, predict’ has received increasing attention. Instead of fine-tuning a pre-trained LLM on a downstream task with a labeled dataset and a custom objective function, a textual prompt is introduced which reformulates the downstream task into one on which the LLM was trained during its training phase. For example, a text classification task can be reformulated as a masked token prediction task: the text sequence ‘I like this movie!’, which is to be classified as either positive or negative, can be reformulated as ‘I like this movie! It was a [MASK] movie.’, and the LLM task consists in predicting the masked token using the prompt 'It was a [MASK] movie.'. In this way, the 'pre-train, prompt, predict' paradigm obviates the need for fine-tuning the LLM on a downstream task: the LLM is trained entirely in an unsupervised fashion and then used to predict the output for several downstream tasks. The diversity of downstream NLP tasks and dynamics necessitates research into Prompt Engineering, which is the subject of this chapter.

8.2 Basics

In this section, we will introduce the basics of the 'pre-train, prompt, predict' paradigm. In traditional supervised learning, the task is to predict the output y associated to some input x based on a model $P(y \mid x; \theta)$, where θ represents the parameters of the model. The parameters θ are learned by fine-tuning a model on a dataset consisting of (x, y) pairs. Prompt-based learning methods, on the other hand, work by learning an LM that models the probability $P(x; \theta)$ directly, and uses it to predict the output y associated with input x. Prompt-based learning follows a three-step process which is presented below.

8.2.1 Prompt Addition

Prompt addition involves modifying the the input text sequence x into a prompted text sequence x'. Formally, it involves two steps:

- Edit the input text x using a textual template which has two slots: slot [X] for the input text x, and an answer slot [Z] for a generated answer text z, which we will use to obtain the output prediction y.

- Fill in the input text x into slot [X].

For example, suppose x is the input sequence 'I like this movie!'. Then, one possible format for the template could be: '[X] Overall, it was a [Z] movie.'. Then, the prompted text sequence is: 'I like this movie! Overall, it was a [Z] movie.'. In this way, a text classification task has been reformulated into a token prediction task by transforming the input sequence x into the prompt addition-based sequence x'.

The definition and form of the template admit several possibilities. Firstly, the position of the slot for the generated answer text z has two options: 1) cloze prompts: where the slot for z is in the middle of the prompted text sequence and 2) prefix prompt: where the slot for z is placed exactly at the end of the input sequence. Secondly, it is possible for the prompting template to consist of tokens which represent other information, such as numeric ids. Lastly, the number of [X] and [Z] slots can be customized for the particular task at hand.

8.2.2 Answer Search

In this step, we use the LM to generate candidate predicted answers. We will denote the set of permissible values of z by $\mathbb{Z}$, from which we will select the highest-scoring one as $\hat{z}$. The range of $\mathbb{Z}$ depends on the particular task. For example, for the text classification example above where movie reviews are classified into classes, we can have $\mathbb{Z} = \{\text{`}\texttt{good}', \text{`}\texttt{ok}', \text{`}\texttt{bad}'\}$.

Answer search proceeds as follows. We define a function $f_{fill}(x', z)$ that takes the prompt x' and fills the answer slot [Z] with a potential answer z (filled prompt). Formally:

$$\hat{z} = \text{search}_{z \in \mathbb{Z}} P\left(f_{fill}(x', z); \theta\right)$$

where $P(\cdot; \theta)$ is a pre-trained LM model and $P(f_{fill}(x', z); \theta)$ represents the probability that the filled prompt with candidate x' generates answer z. For selecting the highest-scoring candidate from $\mathbb{Z}$, an argmax function is used for search. A prompt filled with the true answer is called the answered prompt.

8.2.3 Answer Mapping

In this step, the highest scoring selected candidate answer $\hat{z}$ is used to obtain the highest scoring output $\hat{y}$. For this purpose, a mapping between the candidate answers (z) and the desired outputs (y) is employed. This mapping depends on the task that is being solved as well as the label definitions of the task. For example, for the text classification example above, we could have a mapping $\mathbb{Z} = \{\text{`}\texttt{good}', \text{`}\texttt{ok}', \text{`}\texttt{bad}'\} \rightarrow \mathbb{Y} = \{+, \ , -\}$, where $\mathbb{Y}$ is the set of labels for the text classification task. For a generation task, such as language translations, the predicted token themselves are the required labels, and hence no mapping is necessary. Finally, it is also possible to map several answers to one output class label. The terminologies and notations used in Prompting are summarized in Table 8.1.

8.3 Prompt Engineering

Prompt Engineering is the process of creating a prompting function $f_{prompt}(x)$ which transforms a text sequence into a prompted text resulting in the most effective performance on downstream tasks. First, a prompt shape is defined, followed by selection between manual or automated prompt template engineering.

8.3.1 Prompt Shape

The prompt shape is determined by the task and the model under consideration. As mentioned above, cloze prompts introduce answer slots within the prompted text sequence, while prefix prompts introduce them at the end of the prompted text sequence. Prefix

Name	Notation	Example	Description
Input	x	I love this movie.	One or multiple texts
Output	y	++ (very positive)	Output label or text
Prompt Function	$f_{prompt}(x)$	[X] Overall, it was a [Z] movie.	A function that converts the input into a specific form the input x and adding by inserting a slot [Z] where the answer z may be filled later.
Prompt	x'	I love this movie. Overall, it was a [Z] movie.	A text where [X] is instantiated by input x but answer slot [Z] is not.
Filled Prompt	$f_{fill}(x', z)$	I love this movie. Overall, it was a bad movie.	A prompt where slot [Z] is filled with any answer.
Answered Prompt	$f_{fill}(x', z^*)$	I love this movie. Overall, it was a good movie.	A prompt where slot [Z] is filled with a true answer.
Answer	z	'good', 'fantastic', 'boring'	A token, phrase or sentence that fills [Z]

Table 8.1: An explanation of terminologies and notations used in prompting methods [105].

prompts are the natural choice for text generation tasks, as the left-to-right direction of the generated text fits in well with the end of sequence position of the prefix prompt. Cloze prompts, on the other hand, are more suitable for tasks that are solved by masked language models. Tasks that involve more than one sentence inputs, like next sentence prediction and text pair classification, requires multiple input slots [Z] in the prompted text sequence.

8.3.2 Manual Template Engineering

In manual template engineering, the prompted template is manually created by considering the task and the model. Several manual prompting templates have been introduced in the literature for different categories of NLP tasks. In Petroni et al. (2019), the authors introduce the LAnguage Model Analysis (LAMA) dataset, which consists of manually crafted cloze templates for knowledge querying in a langauge model (LM) [136]. For NLP tasks such as question answering and translation, Brown et al. (2020) introduce manually defined prefix prompts [22]. Finally, Schick and Schütze (2020, 2021) use pre-defined templates for text classification and conditional text generation tasks in few-short learning scenario [159, 160, 161].

8.3.3 Automated Template Engineering

Manual template engineering suffers from two drawbacks. Firstly, creating and experimenting with the prompts through trial and error is a time consuming and costly process. Secondly, finding optimal prompts manually is a difficult task. These considerations necessitate the automation of the template design process.

Automatically created prompts fall into two categories: 1) discrete prompts, where the prompt is an actual text sequence, and 2) continuous prompts, where the prompt is defined in the embedding space of the language model (LM). Another important consideration in automated template engineering is whether the prompting function $f_{prompt}(x)$ is static, where the same prompt template is used for each input sample, or dynamic, where a custom template is generated for each input sample.

Discrete Prompts

In discrete prompts engineering, the prompts are generated in a defined discrete space, normally consisting of a natural language vocabulary. We present several discrete prompt engineering methods below.

Prompt Mining The MINE approach was introduced by Jiang et al. (2020) [76]. In this approach, given a set of training inputs $\mathbf{x}$ and associated outputs $\mathbf{y}$, MINE scrapes a large text corpus to find text sequences containing $\mathbf{x}$ and $\mathbf{y}$. It then finds the frequent middle words or dependency paths between these inputs and outputs, and constructs a suitable prompt from them of the form '[X] middle words [Y]'.

Prompt Paraphrasing Paraphrasing based prompt engineering works by selecting a seed prompt, and then constructing a set of prompts through paraphrasing this seed prompt. There are several paraphrasing techniques that can be used. Jiang et al. (2020) use full-circle translation, whereby the prompt is translated to another language and then translated back into the original language [76]. Similarly, Yuan et al. (2021) use phrase replacement using a thesaurus [208], whereas Haviv et al. (2021) use a neural prompt re-writer that is optimized to produce high accuracy models [65]. The prompt paraphrasing technique of Haviv et al. (2021) is dynamic, because the paraphrasing takes place after the input x is inserted into the template. Accordingly, every input sequences has a different paraphrased prompted text sequence.

Prompt Generation Prompt generation method treats creation of prompts as a text generation task, hence using natural language text generation language models. Gao et al. (2021) use T5, a seq2seq model which is pre-trained on filling in missing spans task [54]. In this case, the prompt generation process consists in generating template tokens as follows: 1) specify their position within a template, and 2) provide training samples to the T5 model to decode them. Ben-David et al. (2021), on the other hand, train the T5 model to generate unique domain relevant features (DRFs) for each input [14]. These DRFs are then concatenated with the input to obtain the desired prompt. Note that this method is also a dynamic prompt engineering method.

Prompt Scoring Prompt scoring is a dynamic method which works by generating several candidate prompts and then choosing the prompt that gets the highest probability in a language model (LM). Davison et al. (2019), for example, create a set of potential prompts and fill the input and answer slots to form a filled prompt [36]. The highest scoring prompt is selecting by processing the filled prompts using an LM.

Continuous Prompts

Continuous prompts work by defining prompts directly in the embedding space of the model. Continuous prompt engineering is motivated by the fact that prompts do not necessary have to be in natural language, although they are processed by language models (LM). This observations brings two main benefits: 1) it expands the prompting space by removing the constraint that the embeddings of template words should be the embeddings of natural language (e.g., English) words, and 2) it also removes the constraint the template is parameterized by the pre-trained LM's parameters. Some continuous prompts methods are described in the next paragraphs.

Prefix Tuning Prefix tuning was introduced by Li and Liang (2021) [101]. The basic idea of this method is to prepend a sequence of continuous task-specific vectors to the input. Formally, this teechnique amounts to optimizing the following log-likelihood objective:

$$\max_{\phi} \log P\left(y \mid x; \theta; M_{\phi}\right) = \max_{\phi} \sum_{y_i} \log P\left(y_i \mid h_{<i}; \theta; M_{\phi}\right)$$

where: M_{ϕ} is a trainable prefix matrix, θ are the parameters of the model and $h_{<i} = [h^1_{<i}, h^2_{<i}, \ldots, h^n_{<i}]$ is the concatenation of all neural network layers at time step i.

Tuning Initialized with Discrete Prompts In this method, a prompt that has already been created using discrete prompt search is utilized to create a continuous prompt. One such method is followed by Zhong et al. (2021) [215]. In their approach, a discrete prompt is first defined using AUTOMPROMPT (Shin et al. (2020) [166]) and virtual tokens are initialized using this discrete prompt. Then, the embeddings are fine-tuned to improve task accuracy. Another method is that of Qin and Eisner (2021) [139]. In this approach, a mixture of soft templates for each input is learned and the weights and parameters of each template are jointly learned using training samples. In this setting, the initial set of templates can either be manually created or obtained using the MINE method described above. Finally, Hambardzumyan et al. (2021) proposed an approach involving a continuous template whose shape follows that of a manual prompt template [62].

Hard-Soft Prompt Hybrid Tuning In this case, some tunable embeddings are inserted into a hard prompt template. This method comprises two dominant techniques: First, the P-tuning, introduced by Liu et al. (2021) [108, 58]. Here, continuous prompts are learned by inserting trainable variables into the embedded inputs, and prompt embeddings are represented as the output of a BiLSTM. Secondly, the prompt tuning with rules (PTR), introduced by Han et al. (2021) [63]. In this context, manually crafted sub-templates are used to compose a complete template based on logic rules. The resulting prompt from is hybrid, in the sense of containing both actual and virtual tokens whose embeddings can be tuned along with the model parameters.

8.4 Answer Engineering

The Answer Engineering step has two objectives: 1) search for an appropriate answer in the answer space $\mathbb{Z}$, and 2) build a mapping from the answer space $\mathbb{Z}$ to the set of original output labels $\mathbb{Y}$. Answer engineering involves two main deciding factors: deciding the answer shape and choosing an answer design method. We will now explain both factors in more details.

8.4.1 Answer Shape

The first factor is to decide the shape and form that the candidate answers will take. The exact shape depends on the task but common choices include:

- Token: each answer candidate is a token drawn from the vocabulary of the language model (LM).
- Span: each candidate answer is a short multi-token span.
- Sentence: each candidate answer is a sentence or a document.

Span answer shapes are usually used with cloze prompts, and sentence answer shapes with prefix prompts. The choice of the answer shape is dependent on the particular task to be performed. For classification tasks, such as sentiment classification, relation classification, and named entity recognition (NER), token and span answer shapes are usually used [207, 137, 35]. For text generation and multiple-choice question answering tasks, long phrasal or sentential answer shapes are usually used [143, 83].

8.4.2 Answer Space Design

Answer space design involves the construction of an appropriate answer space $\mathbb{Z}$ as well as a mapping from this answer space $\mathbb{Z}$ to the set of true labels $\mathbb{Y}$.

Manual Design

In manual design, the answer space and the mapping are hand-crafted manually.

Unconstrained Spaces The answer space $\mathbb{Z}$ is often defined as the collection of all tokens, fixed-length spans, or token sequences. This approach, as described by Petroni et al. (2019), Jiang et al. (2020), and Radford et al. (2019) typically involves a direct mapping from the answer z to the final output y using the identity function [137, 76, 143].

Constrained Spaces In some task settings, text classification or entity recognition, and multiple choice question answering, the answer space in constrained because the allowed labels of the true output are limited. Yin et al. (2019) created lists of words related to specific topics, such as “health”, “finance”, “politics”, and “sports”, or emotions, like “anger”, “joy”, “sadness”, and “fear”, as well as other characteristics of the input text to be classified [207]. Similarly, for NER tasks, Cui et al. (2021) designed lists of entities such as “person” and “location” [35]. In such cases, it is essential to establish a mapping between the answer space $\mathbb{Z}$ and the output labels set $\mathbb{Y}$ (for instance, the word “Paris” in $\mathbb{Z}$ would be mapped to the token “LOC” in $\mathbb{Y}$).

Discrete Answer Search

It is possible that manually created answers may not be the most effective way for ensuring optimal prediction performance of the language model. Hence, there is ongoing research on automatic answer search. Some of these methods are described below.

Answer Paraphrasing In answer paraphrasing, the initial answer space $\mathbb{Z}$ is expanded using paraphrasing. For a pair of answer and associated output (z, y), a set of paraphrased answers $para(z)$ is generated using some function. For example, Jiang et al. (2020) construct the set of multiple paraphrased answers using back-translation [76]. The probability of the final output is then defined as the marginal probability of all answers in this paraphrase set:

$$P(y \mid x) = \sum_{z' \in para(z)} P(z \mid x) \,.$$

Prune-then-Search These approaches involve creating an initial pruned answer space $\mathbb{Z}'$ comprising multiple plausible answers, followed by a search algorithm that further sifts through this space to choose the final set of answers. In some cases a function, called verbalizer, which maps a label y to a single answer token z, is used. Schick and Schütze (2021) and Schick et al. (2020) find frequently occurring tokens containing at least two alphabetic characters in an unlabeled dataset [159, 158]. Then, during the search step, they iteratively evaluate a word’s suitability as a representative answer z for a label y, by maximizing the likelihood of the label over training data. Shin et al. (2020) train a logistic classifier using the contextualized representation of the [Z] token as input [166]. During the search step, they choose the top-k tokens with the highest probability score by using the learned logistic classifier. These tokens form the answer. Gao et al. (2021) create the pruned search space $\mathbb{Z}'$ by selecting top-k vocabulary words, based on their generation probability at the [Z] position in training samples [54]. Then, they further prune the search space by selecting only a subset of words within $\mathbb{Z}'$, based on their zero-shot accuracy on training samples. Finally, they fine-tune the language model with fixed templates and every answer mapping using training data during the search step, by selecting the best label word as the answer based on its accuracy on the development set.

Label Decomposition Label decomposition involves each label into its component tokens, and use them as an answer [29]. For example, for the output label 'per:city of death', the set of decomposed label tokens can be defined as { person, city, death }. Then, the probability of the answer span is computed as the sum of each token's probability.

Continuous Answer Search

Continuous Answer Search involves the use of soft answer tokens (drawn from the model's embedding space) which can then be optimized using gradient descent. In Hambardzumyan et al. (2021), for example, a virtual token is assigned for each class label and then the token embedding for each class and prompt token embeddings are optimized [62].

8.5 Multi Prompt Learning

The prompt engineering techniques discussed so far have concentrated on creating a single prompt for any input. Nonetheless, ample research has shown that utilizing several prompts can enhance the effectiveness of prompting methods. We refer to these methods as multi-prompt learning and give an overview in the next subsections.

8.5.1 Prompt Ensembling

Prompt Ensembling involves using multiple unanswered prompts, as opposed to a single prompt, for an input text sequence. These multiple prompts can be a combination of both discrete and continuous prompts. Prompt ensembling brings two main benefits: 1) it leverages the benefits of different prompting techniques, and 2) the cost of choosing the best performing prompt through expensive and costly trial-and-error process can be reduced. We now explain the main prompt ensembling techniques.

Uniform averaging When utilizing multiple prompts, the most straightforward method to merge the predictions is by computing the average of the probabilities from the various prompts. Formally, this amounts to:

$$P(z \mid x) = \frac{1}{K} \sum_{i}^{K} P\left(z \mid f_{prompt,i}(x)\right)$$

where $f_{prompt,i}(\cdot)$ is the i-th prompting function of the prompt ensemble. In their approach, Jiang et al. (2020) begin by filtering their prompts and selecting the K prompts that achieve the highest accuracy on the training set [77]. Then, they compute the average log probabilities from the top K prompts, and use this to determine the probability for a single token at the [Z] position during factual probing tasks. Schick and Schütze (2021) perform a simple average when using an ensemble model to annotate an unlabeled dataset [159]. In the text generation evaluation scenario, Yuan et al. (2021) frame this task as a text generation problem, and compute the average of the final generation scores acquired by utilizing different prompts [208].

Weighted average Weighted average method is an intuitive extension of simple averaging method. The weighted average prompt ensembling method works by associating a weight to every prompt in the ensemble. The weights are either optimized using a training set or pre-specified based on prompt performance. There are several approaches for assigning weights to specific prompts. For example, in Jiang et al. (2020), the weight for each prompt are learned by maximizing the probability of the target output over training data [77]. Qin and Eisner (2021) make two contributions: first, they jointly learn the prompt weights with continuous prompt parameters and second, they introduce a new data-dependent weighting strategy which takes into account the probability of the input appearing in a prompt while weighting different prompts [139]. Finally, in Schick and Schütze (2021), the weight of each prompt is set in proportion to the accuracy on the training set before training [159, 160].

Majority voting In classification tasks, majority voting is another method that can be employed to merge the outcomes from various prompts, as shown by Lester et al. (2021) and Hambardzumyan et al. (2021) [94, 62].

Knowledge distillation Using an ensemble of deep learning models can often enhance performance, and this enhanced performance can be condensed into a single model using knowledge distillation, as demonstrated by Allen-Zhu and Li (2020) [6]. Schick and Schütze (2021, 2020) adopt this concept by training a distinct model for each template-answer pair that is manually constructed [159, 161, 158]. These models are then combined to label an unlabeled dataset, and the knowledge is distilled from the annotated dataset to train the final model. Similarly, Gao et al. (2021) utilize an ensemble method on their automatically generated templates [54].

Prompt ensembling for text generation In text generation tasks, the answers are a string of tokens instead of a single token. Ensembling in this scenario can be performed by utilizing conventional techniques that produce the output by relying on the ensembled probability of the subsequent word in the answer sequence. Formally:

$$P(z_t \mid x, z_{<t}) = \frac{1}{K} \sum_{i}^{K} P(z_t \mid f_{prompt,i}(x), z_{<t})\,.$$

Schick and Schütze (2020), on the other hand, opt for a different approach by training a distinct model for each prompt, $f_{prompt,i}(x)$ [158]. As storing these fine-tuned LMs in memory is impractical, they instead decode generations utilizing each model and score each generation by computing the average of their generation probabilities across all models.

8.5.2 Prompt Augmentation

Prompt Augmentation is also known in literature as demonstration learning [54]. In Prompt Augmentation, the language model (LM) is provided with a few answered prompts as an example of how it is expected to fill in the unanswered input prompt sequence. In this way, answered prompts are prepended to the unanswered prompted input sequence. To illustrate this, instead of simply giving the prompt "The capital of China is [Z].", it would be helpful to include a few examples beforehand, such as "London is the capital of Great Britain. Tokyo is the capital of Japan. The capital of China is [Z].". These few-shot demonstrations capitalize on the capacity of robust language models to recognize repetitive patterns (Brown et al. (2020) [22]. Prompt Augmentation involves two main considerations: 1) Sample selection: what indicative answered prompts should be provided to the LM?, and 2) Sample order: in what order should these indicative answered prompts be prepended to the input prompt.

Sample Selection Empirical evidence suggests that the choice of example answered prompts in few-shot scenario have enormous effect on the model performance [111]. To obtain the most effective example samples, Gao et al. (2021) and Liu et al. (2021) use sentence embeddings to generate sample examples that are similar to the input in the embedding space [54, 104]. Alternatively, Mishra et al. (2021), provide both positive and negative samples in order to indicate to the LM which patterns to look for and which to avoid [119].

Sample Ordering The sequence in which answered prompts are presented to the model also has a significant impact on its performance. Lu et al. (2021) propose entropy-based techniques to evaluate various potential permutations [111]. Meanwhile, Kumar and Talukdar (2021) explore various permutations of training examples as augmented prompts, and introduce a separator token between prompts [88].

8.5.3 Prompt Composition

For tasks that can be broken down into more basic sub-tasks, it is possible to utilize prompt composition. This involves using several sub-prompts, each one corresponding to a specific sub-task. A composite prompt is then constructed by combining these sub-prompts. To illustrate this, consider relation extraction task, which involves identifying the relationship between two entities. This task can be deconstructed into several sub-tasks, such as identifying the attributes of the entities and categorizing the types of relationships that exist between them. By creating sub-prompts for each of these sub-tasks and then combining them, we can develop a composite prompt that addresses the overall relation extraction task. For example, Han et al. (2021), utilize manually created sub-prompts for entity recognition and relation classification, and the use logic rules to compose them into a complete prompt [63].

8.5.4 Prompt Decomposition

Prompt decomposition works in opposite way to prompt composition. For tasks such as sequence labeling, where multiple predictions are required for a single sample, one full prompt may be unable to fully consider the entire input text. To tackle this challenge, one approach is to deconstruct the full prompt into several sub-prompts, and then answer each sub-prompt individually. Cui et al. (2021) show a prompt decomposition technique for named entity recognition [35]. In their work, the input text is first be divided into a series of smaller text segments or spans. The model can then be prompted to predict the entity type for each span individually. This approach allows the model to focus on each segment separately, making it easier to produce accurate predictions for the entire input sequence.

Part III

Articles

Chapter 9

Argument Classification with BERT plus Contextual, Structural and Syntactic Features as Text

9.1 Introduction

Human argumentation proceeds by the participants presenting arguments in favor of or opposing a certain stance on a topic. Argumentative discourse can be both written and spoken. Computational argumentation involves arriving at a coherent understanding of an argumentative text by identifying the underlying argumentational structure of the text and applying an argumentation model to it. In general, the argumentation process proceeds by one interlocutor presenting his or her stance on the topic and then presenting one or more pieces of evidence in support of their stance.

Natural Language Processing (NLP) is a broad branch of Artificial Intelligence (AI) concerned with the automated processing, understanding and analysis of natural language texts. In NLP, texts from different sources, like written and spoken etc, from different domains, like legal and medical, and different structures, like structured essays and social media micro-texts, are parsed and analyzed in order to be processed and understood by computational language models. Natural Language Processing (NLP) researched is informed by diverse and broad areas like linguistics, computational linguistics and computer science. NLP has several increasingly vital real-world applications like Speech Recognition, Sentiment Analysis, Language Translation and Predictive Text.

In NLP, Argument Mining (used interchangeably with Argumentation Structure Parsing (ASP)) is the process of analyzing argumentative discourse and identifying the underlying argumentational structure of it. Argument Mining involves automated detection of argument units and their relations from natural language text which can then be utilized by computational argumentation models for specific applications. Essential sub-tasks in AM/ASP include: 1) separating argumentative discourse units (argument components) from non-argumentative text, 2) classifying argument components to determine their role in the argumentative process, 3) given two argument components, deciding whether they are linked or not and, 4) given two linked component, decide whether the link is a supporting or attacking link. An end-to-end AM pipeline process leads to a tree-structure representation of the argumentative text which can be utilized for several popular downstream applications like Stance Recognition, Sentiment Analysis, Discourse Analysis and E-Commerce Feedback.

Text Classification involves classifying input text into one of several pre-defined classes. In the context of argument mining (AM), text classification takes on a vital role because it is essential, once an argument unit is separated from non-argumentative text, to understand the role of the argument unit in the argumentation process. In argumentative texts, argument units are generally classified as either claims or premises. Claims represent assertions made and positions taken by the person on a given topic while Premises are pieces of evidences and warrants a person gives for taking a certain stance. In our work, we focus on text classification task using a novel contextual-structural text representation model. For example, claims can represents a person's position on a moral question (immigration, taxes, war) and premises represent the justifications (facts, statistics, historic examples, moral or normative principles) he/she presents for support his claim.

Argumentative discourse happens in many interesting settings. In *written discourse* such as essays and articles, an agent seeks to convince the audience of his/her position on a certain topic by presenting several claims and justifications in a structured manner, leading to a logical conclusion. Similarly, *organized political debates* are an argumentative process where each candidate seeks to convince the audience of the validity of his/her position on several issues in a dialogical manner. Lastly, in *internet fora and social platforms* users argue, discuss and debate controversial topics. Each of these discursive settings have their own structure and dynamics which influence how they can be processed by an AM/ASP. We work with one dataset of each of these kinds to see how our model performs on them.

Transformer language models have been game changers in NLP. Transformer models are sequential language models consisting of an encoder/decoder architecture which accept natural language text as input and generate class probabilities as output. The attention layer mechanism in the encoder component allows the model to take into account the context of a word when processing an input text sequence. Bidirectional Encoder Representations from Transformers (BERT) is one such model. BERT models are pre-trained on huge amounts of data in a self-supervised manner. Using a transfer learning process called fine-tuning, this pre-trained BERT model is then utilized for a specific NLP task on a specific dataset. BERT models have been successfully used for several NLP taks like Named Entity Recognition (NER), Question Answering, Sentiment Analysis, Text Generation and Machine Translation and have produced state-of-the-art performance. Our novel text representation model is based on BERT.

For argument component classification, however, the use of different embeddings (GloVe, ELMo, FastText, etc.) alone as sentence representation do not suffice. The role of an argument component depends, among others, on its context and position in the text and thus cannot be captured by its content alone. Therefore, additional features like lexical, indicator, discourse, syntactic, contextual and structural features have been used to enrich the sentence representation of the components [171, 61, 89]. Accordingly, an efficient BERT-based model for AM requires both architecture customization and enhancement to capture the contextual, structural and syntactic features necessary for the task at hand.

Representation Design is an essential part in any AM system. In Machine Learning, classifiers like Logistic Regression, Support Vector Machines (SVM) and k-means Clustering work on numerical features computed from the input data. For example, an SVC classifier could take the presence (or absence) of a personal pronouns in the input text as one (among several) binary feature to help it classify the text as a claim or a premise. Similarly, embedding models like GloVe, ELMo and BERT generate vector representations of an input text which are then used by a classifier as the feature space of the input text. In a novel way, we use *textual representations of the numeric features* with the BERT model for the argument component classification task on three argumentative datasets.

The 'Pre-train, Prompt, Predict' paradigm has also been a game-changer in NLP [106]. In this paradigm, task-specific supervised fine-tuning is replaced by additional self-supervised training involving textual prompts designed for specific downstream tasks. For instance, the sentiment of the sentence 'I liked the movie!' is obtained by the output of the language model on the input 'I liked the movie! The movie was [MASK].' which includes the sentence and a task specific prompt. For argument component classification, however, the straightforward prompting approach would not capture the necessary contextual, structural and syntactic information.

Based on these considerations, we propose a novel approach, inspired by prompt engineering, which incorporates – in textual form – the contextual, structural and syntactic features necessary for argument component classification. Specifically, we introduce a novel model for argument component classification which is based on the popular BERT model. Our model incorporates contextual, structural and syntactic features *as text* to build a customized and enriched BERT-based representation of the argument component. We experiment with our model on three datasets: one written essays-based, one speech-based and one written social media-based. We show that: 1) our *features as text* sentence representation model improves upon the BERT-based component only representation, 2) our structural *features as text* representation outperforms the classical approach of numerically concatenating these features with BERT embedding, and 3) our model achieves state-of-art results on two datasets and 95% of the best results on the third. Overall, we situate our work within the 'better models vs better data' question by developing task-specific and customized data as opposed to designing more complex models. We make the code available on GitHub at: https://github.com/mohammadoumar/features_as_text

This paper is structured as follows. Section 9.2 describes the related literature that informs our work. Section 9.3 presents the datasets. In Section 9.4, we introduce our novel *features as text* model in detail. Section 9.5 presents the experimental setting, results and analysis of our work. Section 9.5 provides concluding remarks and future directions.

9.2 Related works

Stab and Gurevych [171] present a features-based approach for argument component classification in the *Persuasive Essays (PE)* dataset (see Section 9.3). They use hand-crafted features (lexical, structural, syntactic, etc.) with Support Vector Machines (SVMs) and Conditional Random Fields (CRFs). They show that structural features, which capture the position of the component in the full text, are most useful for component classification.

Hadaddan et al. [61] use both features-based and neural network-based approaches for argument component classification in the *Yes We Can (YWC)* political debates dataset (See Section 9.3). In the features-based approach, they use an SVM with both Bag of Words (BoW) and a custom features set (POS, syntactic, NER, etc). In the neural network-based setting, they use both a feed-forward neural network with the custom features set and an LSTM with FastText word embedding.

Potash et al. [138] present a Joint Neural Model for simultaneous learning of argument component classification and link extraction between argument components in the *PE* and *Micro-Text Corpus (MTC)* datasets. This model consists of a Bi-LSTM encoder, a fully connected layer for component classification and an LSTM decoder for link identification. They use three methods for textual representation: Bag of Words (BoW), GloVe embedding and structural features.

Kuribayashi et al. [89] introduce an extension to the LSTM-minus-based span representation [192] where they create separate representations of the argumentative markers

('I think', 'because', etc.) and argumentative component present in the argument unit. For textual/span representation, they use GloVe and ELMo embeddings concatenated with Bag of Words (BoW) and structural features. They experiment with the *PE* and *MTC* datasets.

Mayer et al. [112] use neural network-based architectures for argument mining in a dataset of abstracts of bio-chemical healthcare trials. They combine the boundary detection and component classification tasks into one sequence tagging task. They use several static and dynamic embeddings such as BERT, GloVe, ELMo, fastText, FlairPM, etc. with various combinations of LSTMs, GRUs and CRFs as well as BERT fine-tune. The authors introduce a new dataset consisting of abstracts of 500 Random Controlled Trials (RCTs) related to 5 diseases the PubMed/MEDILINE medical database. This dataset is annotated for both the argument component classification (claims, evidences/premises) task and the relation classification (support, attack) task. They propose an end-to-end Argument Mining pipeline and experiment with transformer based embeddings and Recurrent Neural Network architectures.

Madabushi, Kochkina et al. [177] also use BERT based model for sentence classification for propaganda detection in the Propaganda Techniques Corpus (PTC) dataset. They use BERT$_{BASE}$ embeddings for sentence representation.

We situate ourselves within the 'better models vs better data' question. We posit that the BERT model is powerful enough to achieve improved performance if provided with task-specific enriched input data. To that end, our work is the first to investigate and implement a *features as text*, BERT-based model for argument component classification.

9.3 Datasets

In our work, we use three datasets for argument classification: *Persuasive Essays (PE)* [171], *Yes We Can (YWC)* [61] and *Change My View (CMV)* [68]. In this section, we present and explain the datasets.

Persuasive Essays (PE):

The *PE* dataset was introduced by Stab and Gurevych [171]. It consists of 402 essays on diverse topics selected from the online portal *essayforum.com*. Each essay, which is divided into several paragraphs, consists of arguments (major claims, claims and premises) for or against a position on a controversial topic. A *MajorClaim* is a direct assertion of the author's position on the topic of the essay. A *Claim* is an assertion the author makes in support of his/her position on the topic. A *Premise* is a piece of evidence or warrant that the author presents to support his/her claim(s). For example, a snippet of the essay on the topic *'Should students be taught to compete or to cooperate?'* is given below with claim(s) in bold and premise(s) in italics:

> First of all, [**through cooperation, children can learn about interpersonal skills which are significant in the future life of all students.**]$_{claim_1}$ [*What*

> *we acquired from team work is not only how to achieve the same goal with others but more importantly, how to get along with others.*]$_{premise_1}$ [*During the process of cooperation, children can learn about how to listen to opinions of others, how to communicate with others, how to think comprehensively, and even how to compromise with other team members when conflicts occurred.*]$_{premise_2}$ [*All of these skills help them to get on well with other people and will benefit them for the whole life.*]$_{premise_3}$

Yes We Can (YWC):

The *YWC* dataset was introduced by Haddadan et al. [61]. It consists of presidential and vice presidential debates in the quadrennial US presidential elections from 1960 to 2016: a total of 39 debates. The dataset consists of transcripts of these debates with claims and premises made by the candidates. In every debate, a *Claim* is an assertion a candidate makes in support of his/her position on the issue under discussion. A *Premise* is a piece of evidence or warrant that the candidate presents to support his/her claim(s).

For example, this is a snippet from the 2004 debate between Vice President Dick Cheney and Democratic Vice Presidential Candidate John Edwards:

> Gwen, I want to thank you, and I want to thank the folks here at Case Western Reserve for hosting this tonight. It's a very important event, and they've done a superb job of putting it together. It's important to look at all of our developments in Iraq within the broader context of the global war on terror. [**And, after 9/11, it became clear that we had to do several things to have a successful strategy to win the global war on terror, specifically that we had to go after the terrorists wherever we might find them, that we also had to go after state sponsors of terror, those who might provide sanctuary or safe harbor for terror**]$_{claim_1}$. [*And we also then finally had to stand up democracies in their stead afterwards, because that was the only way to guarantee that these states would not again become safe harbors for terror or for the development of deadly weapons*]$_{premise_1}$.

Corpus statistics		Component Statistics	
Speech Turns	6,601	Claims	11,964
Sentences	34,013	Premises	10,316
Words	676,227	O	7,252
Debates	39	Total	29,621

Table 9.1: YWC dataset statistics

Change My Views (CMV)

The *CMV* dataset is presented by Tan et. al. [174, 68]. It is based on the "*r/changemyview*" subreddit from the social media platform *Reddit.com.* It consists of 113 threads containing argumentative conversations, made up of claims and premises, between internet users on 37 controversial topics. For example, here is a snippet of the discussion thread on the topic: *America is a better place because of the 55 million abortions its had*:

> CMV: America is a better place because of the 55 million abortions its had.
>
> [*There have been 55 million abortions in the US since 1973*]$_{premise_1}$. [*Something in the ballpark of 45% of women who have abortions have more than one abortion*]$_{premise_2}$. [**These people would have been raised by their incompetent parents to drain down society, increase crime rates, suck up resources, and generally screw things up**]$_{claim_1}$. [**Various ways to lower a resource negative population would have to be explored, if not because of this 55 million, then because of the next 55 million**]$_{claim_2}$. [**One possibility is that there would be wars waged to try to kill them all, perhaps even with other countries with similar problems**]$_{claim_3}$.

Corpus statistics		Component Statistics	
Words	75,078	Main Claims	116
Paragraphs/Sentences	3,869	Claims	1,589
Topics	37	Premises	2,059
Files	113	Total	3,764

Table 9.2: CMV dataset statistics

9.4 Model

In this section, we introduce our novel BERT-based model for argument component classification. Our model incorporates contextual, structural and syntactic features – represented *as text* – instead of the usual numerical form. This approach enables BERT to build an enriched representation of the argument component.

BERT

BERT models combine a feed-forward architecture with a self-attention mechanism, which allow to parallelize the computation and focus on any part of the input, respectively [37]. BERT architecture consists of twelve encoder blocks of the Transformer model stacked together and 12 self-attention heads [37]. The self-attention heads enable BERT to incorporate bidirectional context and focus on any part of the input sequence. BERT builds a

Dataset	*Corpus Statistics*		*Component Statistics*	
Persuasive Essays (PE)	Tokens	147,271	Major Claims	751
	Sentence	7,116	Claims	1,506
	Paragraphs	1,833	Premises	3,832
	Essays	402	Total	6,089
Yes We Can! (YWC)	Speech Turns	6,601	Claims	11,964
	Sentences	34,103	Premises	10,316
	Words	676,227	Other	7,252
	Debates	39	Total	29,621
Change My View (CMV)	Words	75,078	Main Claims	116
	Paragraphs	3,869	Claims	1,589
	Topics	37	Premises	2,059
	Files	113	Total	3,764

Table 9.3: Corpus and component statistics for *PE*, *YWC* and *CMV* datasets. In the *CMV* dataset, Major Claims are called Main Claims.

768 dimensional representation – or embedding – of the input text sequence. In this work, as opposed to current approaches, we enrich the BERT model with textual representation of contextual, structural and syntactic features. These features are described below. Its training is achieved in two stages: (i) an unsupervised pre-training process consisting of a masked language model (MLM) task followed by a next sentence prediction task (NSP); (ii) a supervised fine-tuning process on a downstream task that requires minimal modifications of the original architecture

Features

Contextual features:

Contextual features capture the full meaning of an argument component in its semantic and linguistic space. In our work, we use *full sentence* and *topic statement* as contextual features. The full sentence feature helps capture the presence of argumentative and/or discourse markers ('I think', 'In my opinion', etc.). These markers indicate that the component preceding or succeeding them in the sentence is more likely a claim than a premise. The topic statement feature helps discriminate between claims and premises because a claim is more likely to directly address the topic statement and, thus, be more semantically similar to it.

For both the *Persuasive Essays (PE)* and *Change My View (CMV)* datasets, the contextual features are the topic of the essay/discussion and the full sentence of the argument component. For the *Yes We Can (YWC)* dataset, in addition to the full sentence, we use candidate name and election year as topical information. We define the textual representation of contextual features as: The candidate identification is supposed to establish a pattern of speech about a certain topic that the candidate uses. The election year, on the other hand, is supposed to give information about the issues and topics that were to the fore in a particular presidential election. For the *Change My Views (CMV)* dataset, the contextual features are: topic and full sentence, using the same rationale as for the *Persuasive Essays* dataset. To demonstrate the textual representation of contextual features, consider the contextual features of the *PE* dataset: $topic$ and $sentence$.

`contextual_features_as_text` = 'Topic: t. Sentence: s.'

where t is the topic of the essay/discussion thread or the speaker and election year of the debate speech and s is the full sentence which contains the argument component (see Example 1).

Structural features:

Structural features incorporate the idea that argumentation follows a certain (perhaps fluid) pattern which can be used to discriminate between claims and premises. These features capture the location of the argument component in the whole essay and in the paragraph in which it appears. For example, claims are more likely to appear in the introductory and concluding paragraphs as well as in the beginning and towards the end of the paragraph. Premises, on the other hand, are more likely to follow a claim in the paragraph [171]. We define the textual representation of structural features as:

`structural_features_as_text` = 'Paragraph Number: n. Is in introduction: i. Is in conclusion: c. Is first in paragraph: f. Is last in paragraph: l.'

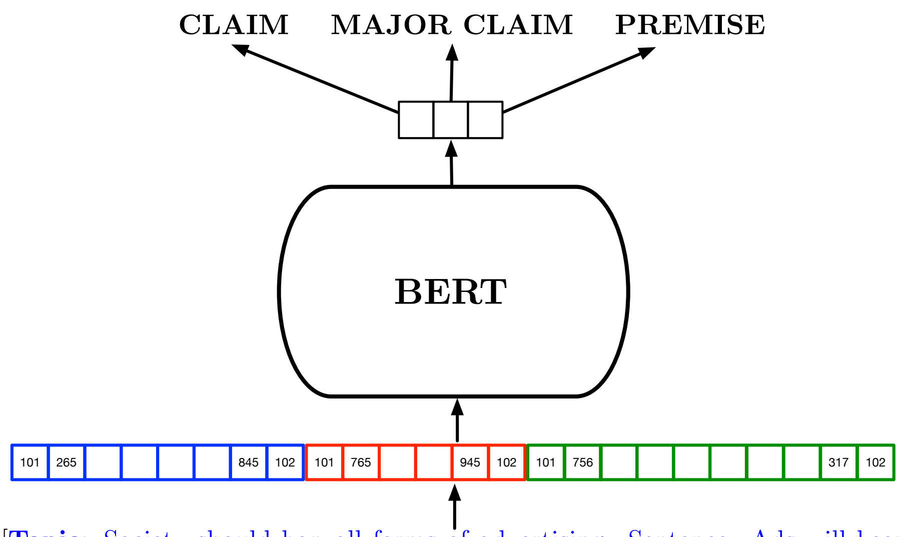
where n is the paragraph number in which the argument component is present, i is *Yes* if the argument component is in the introduction paragraph and *No* otherwise, c is *Yes* if the argument component is in the conclusion paragraph and *No* otherwise, f is *Yes* if the argument component is the first component in its paragraph and *No* otherwise, and l is *Yes* if the argument component in the last component in its paragraph and *No* otherwise (see Example 1).

Syntactic features:

Part-Of-Speech (POS) involves classification of English words into categories depending on their linguistic role in a sentence. These categories include noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection, numeral, article, or determiner [64]. We define the textual representation of syntactic features as:

`syntactic_features_as_text` = 'Part Of Speech tags: $t_1, t_2 ... t_n$'

where t_i represents the POS tag of the i-th word in the argument component.



[**Topic:** Society should ban all forms of advertising. Sentence: Ads will keep us well informed about new products and services, but we should also bear in mind that **advertising cigarettes and alcohol will definitely affect our children in negative way.**]$_{contextual}$ [**Paragraph Number:** Five. **Is in introduction:** No. **Is in conclusion:** Yes. **Is first in paragraph:** No. **Is last in paragraph:** Yes.]$_{structural}$ [**Part of Speech tags:** VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN]$_{syntactic}$

Figure 9.1: Illustration of the *features as text* BERT-based model.

Combined features as text

We combine the textual representations of the contextual, structural and syntactic features to build an enriched BERT-based representation of the argument component. The combined representation is defined as follows:

$$\begin{aligned}\texttt{combined\_features\_as\_text} = {} & \texttt{contextual\_features\_as\_text} + \\ & \texttt{structural\_features\_as\_text} + \\ & \texttt{syntactic\_features\_as\_text}\end{aligned}$$

where '+' denotes the string concatenation operation. Note that the argument component itself is included in the full sentence.

Commonly, features are used as numerical input to NLP and ML models. The novelty of our model is that we use features *as text* in our model. Since, BERT works with word embeddings and language models, our motivation was to see how BERT works when numerical features are given as text to it. We now demonstrate the Features as Text representation method.

Example 1:

We consider an example from the *Persuasive Essays (PE)* dataset: argument component 398 from essay 28:

$$\texttt{argument\_component} = \text{‘advertising cigarettes and alcohol will definitely affect our children in negative way’}$$

The contextual, structural and syntactic features of this argument component are given in Table 9.4.

Feature	*Value*
$essay\ topic$	‘Society should ban all forms of advertising.’
$full\ sentence$	‘Ads will keep us well informed about new products and services, but we should also bear in mind that **advertising cigarettes and alcohol will definitely affect our children in negative way.**’
$para\_nr$	5
is_in_intro	0
$is\_in\_conclusion$	1
$is\_first\_in\_para$	0
$is\_last\_in\_para$	1
pos_tags	VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN

Table 9.4: Features for argument component 398 of the *PE* dataset. The component itself is in bold.

The combined *features as text* representation of this argument component is:

‘[Topic: Society should ban all forms of advertising. Sentence: Ads will keep us well informed about new products and services, but we should also bear in mind that **advertising cigarettes and alcohol will definitely affect our children in negative way.**]$_{contextual}$ [Paragraph Number: Five. Is in introduction: No. Is in conclusion: Yes. Is first in paragraph: No. Is last in paragraph: Yes.]$_{structural}$ [Part of Speech tags: VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN]$_{syntactic}$’

where the argument component is in bold and the contextual, structural and syntactic features are contained in brackets. This combination of contextual features, structural features and argument component jointly form the enriched sentence representation that is input to the BERT model.

Experimentally, we also tried the following ‘abbreviated’ combined representation:

> ‘[Topic: Society should ban all forms of advertising. Sentence: Ads will keep us well informed about new products and services, but we should also bear in mind that **advertising cigarettes and alcohol will definitely affect our children in negative way.**]$_{contextual}$ [Structural features: Five. No. Yes. No. Yes.]$_{structural}$ [Part of Speech tags: VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN]$_{syntactic}$’

Example 2:

We consider an example from the *Yes We Can (YWC)* dataset: argument component 2 from the 2004 Vice Presidential debate:

`argument_component` = ‘What we did in Iraq was exactly the right thing to do.’
’

The contextual, structural and syntactic features of this argument component are given in Table 9.5.

Feature	*Value*
$Candidate$	Richard(Dick) B. Cheney
$full\ sentence$	‘What we did in Iraq was exactly the right thing to do.’
$para\_nr$	5
is_in_intro	0
$is\_in\_conclusion$	1
$is\_first\_in\_para$	0
$is\_last\_in\_para$	1
pos_tags	VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN

Table 9.5: Features for argument component 2 of the *YWC* dataset. The component itself is in bold.

The combined *features as text* representation of this argument component is:

'[Candidate: Richard(Dick) B. Cheney. US Presidential Elections 2004. Sentence: **What we did in Iraq was exactly the right thing to do.**]$_{contextual}$ [Paragraph number: Five. First in paragraph: Yes. Last in paragraph: No. Is in introduction: No. Is in conclusion: Yes. Is fist component: No. Is last component: No.]$_{structural}$ [Part of Speech tags: VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN]$_{syntactic}$'

where the argument component is in bold and the contextual, structural and syntactic features are contained in brackets. This combination of contextual features, structural features and argument component jointly form the enriched sentence representation that is input to the BERT model.

Example 3:

We consider an example from the *Change My View (CMV)* dataset: argument component 230 from debate thread 15:

`argument_component` = 'the still thinking criminals would change to some other safer crime as their chances of getting shot would also increase '

The contextual, structural and syntactic features of this argument component are given in Table 9.6.

The combined *features as text* representation of this argument component is:

'[Thread Topic: Criminal Justice System. Sentence: With scenario 1 I would expect the effect would be that **the still thinking criminals would change to some other safer crime as their chances of getting shot would also increase.**]$_{contextual}$ [Paragraph number: Five. First in paragraph: Yes. Last in paragraph: No. Is in introduction: No. Is in conclusion: Yes. Is fist component: No. Is last component: No.]$_{structural}$ [Part of Speech tags: VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN, ADP, ADJ, VERB, DET, NOUN, ADP]$_{syntactic}$'

9.5 Results and analysis

In this section, we present and analyse our results. We use our model for two tasks: 1) *BERT fine-tune:* We fine-tune BERT on the three datasets using our novel combined features as text sentence representation. 2) *Textual vs numerical features comparison:* We fine-tune BERT and compare results in two cases: first, with our structural features as text and second, with structural features numerically concatenated with BERT sentence embedding.

Feature	*Value*
$Thread\ Topic$	Criminal Justice System
$full\ sentence$	'With scenario 1 I would expect the effect would be that **the still thinking criminals would change to some other safer crime as their chances of getting shot would also increase.**'
$para\_nr$	5
is_in_intro	0
$is\_in\_conclusion$	1
$is\_first\_in\_para$	0
$is\_last\_in\_para$	1
pos_tags	VERB, NOUN, CCONJ, NOUN, VERB, ADV, VERB, DET, NOUN, ADP, ADJ, NOUN, ADP, ADJ, VERB, DET, NOUN, ADP

Table 9.6: Features for argument component 230 of the *CMV* dataset. The component itself is in bold.

Experimental setting

In the *Persuasive Essays* dataset, out of the 402 essays, some were repetition including two essays on the same topic and two essays which were exactly the same. We combined them into one. In the *Yes We Can* dataset, we excluded the components labeled *'O'* such as *'Thank you Gwen for arranging this debate'*. For the *Change My View* dataset, we excluded the components labeled *'Main Claim'* because they correspond exactly to the topic of the thread and gave nearly 100% accuracy, thus skewing up our results.

For the *PE* dataset, we use the original split: 322 essays in the train set (4,709 components) and 80 essays in the test set (1,258 components). For the *YWC* dataset, we also use the original split with 10,447 components in the train set, 6,567 components in the test set and 5,226 components in the validation set. For the *CMV* dataset, we randomly set aside 90 threads for the train set (2,720 components) and 23 threads for the test set (763 components). The implementation details of the model and experiment are presented in Table 9.7.

Task results

The results of Task 1 and Task 2 are presented in Tables 10.2 and 10.3, respectively. State-of-the-art results are also shown in Table 10.2: F1 score of 0.86 for *PE* [89] and 0.67 for *YWC* [61] datasets. The results can be summarized as follows:

- Our novel *features as text* sentence representation, which incorporates contextual, structural and syntactic features as text, improves upon the BERT-based component only representation.
- Our *features as text* representation outperforms the classical approach of numerically concatenating these features with BERT embedding.
- Our model achieves state-of-art results on two datasets and 95% of the best results on the third.

Analysis

The addition of contextual, structural and syntactic features *as text* enables BERT to relate the argument component to the linguistic and argumentative flow of the whole paragraph and essay.

In Task 1, for the *PE* dataset, the contextual, structural and syntactic parts of our combined representation improve the results compared to the BERT-based component only representation. The contextual representation improves the F1 score from 0.57 to 0.68. The combined contextual, structural and syntactic representation improves the F1 score from 0.68 to 0.82 which is 95% of the state-of-the-art result (0.86) [89]. However, the state-of-the-art approach works on paragraphs which are chunked into segmented discourse units and require argumentative marker (AM) versus argumentative component (AC) distinction in sentences. In contrast, our model simply works on the sentence level and requires no AM/AC distinction to be made. Overall, the improvement achieved by structural features emphasizes the importance of the position of the argument component in written argumentative texts, like persuasive essays.

Name	*Values*
Model	‘bert-base-uncased’
Embedding dimension	768
Batch size	[16, 24, 32, 48]
Epochs	[3, 6, 8, 12]
Learning rate	[1e-5, 2e-5, 1e-3, 5e-3, 5e-5]
Warmup ratio, Weight decay, Dropout	0.1, 0.01, 0.1
Loss function	Cross Entropy Loss

Table 9.7: Model implementation details. We experimented with several parameter values. For each experiment, the best parameter values are available on the GitHub repository.

Sentence representation	*PE*				*YWC*			*CMV*		
	MC	*C*	*P*	*F1*	*C*	*P*	*F1*	*C*	*P*	*F1*
component only	0.49	0.41	0.81	0.57	0.71	0.68	**0.69**	0.74	0.79	0.76
sentence	0.69	0.48	0.82	0.66	0.71	0.68	**0.69**	0.70	0.77	0.74
topic + sent	0.70	0.70	0.84	0.68	0.69	0.65	0.67	0.70	0.75	0.73
sent + strct	0.85	0.68	0.91	0.81	0.70	0.68	**0.69**	0.75	0.84	**0.79**
topic + sent + strct	0.86	0.68	0.91	0.81	0.69	0.65	0.67	0.76	0.80	0.78
topic + sent + strct + synt	0.86	0.71	0.91	**0.82**	0.71	0.62	0.67	0.76	0.78	0.77
LSTM + dist [89]	0.92	0.73	0.92	**0.86**	-	-	-	-	-	-
LSTM + word emb [61]	-	-	-	-	0.70	0.68	0.67	-	-	-

Table 9.8: Task 1 results. Performance of our *features as text* BERT-based model on the three datasets. We report results of different combinations of features as text. *MC*, *C* and *P* represents the F1 scores for *MajorClaim*, *Claim* and *Premise*, respectively. *F1* represents the macro F1 score. The abbreviations 'strct' and 'synt' stand for structural features and syntactic features respectively. The last two rows represent the state-of-the-art results for the *PE* and *YWC* datasets.

Dataset	*Features concatenated*				*Features as text*			
	MC	*C*	*P*	*F1*	*MC*	*C*	*P*	*F1*
Persuasive Essays (PE)	0.82	0.57	0.90	0.76	0.86	0.68	0.91	**0.81**
Yes We Can! (YWC)	-	0.70	0.65	0.67	-	0.69	0.65	**0.69**
Change My View (CMV)	-	0.70	0.76	0.73	-	0.76	0.80	**0.78**

Table 9.9: Task 2 results. Comparison between structural features numerically concatenated to BERT embedding and our *features as text* sentence representation.

For the *CMV* dataset, our combined contextual and structural representation improves the F1 score from 0.76 to 0.79. Here, the contextual only part does not improve the results because the argument component and full sentence boundaries almost always coincide. By contrast, the structural features do improve the results, but to a lesser extent than in the *PE* dataset. This difference is explained by the fact that written text on social media platforms is less structured than written text in academic essays.

In contrast with the other datasets, for *YWC*, the combined contextual, structural and syntactic representation does not show improvement. Nevertheless, our model outperforms the state-of-the-art results in the literature (0.69 vs 0.67) [61]. These results show that the somewhat concrete linguistic and structural flow present in the written *PE* dataset and (to a lesser extent) in the *CMV* dataset is lacking in the spoken *YWC* dataset because of its extemporaneous and fluid nature.

Overall, we see that our *features as text* sentence representation, which incorporates contextual, structural and syntactic features *as text*, improves upon the BERT-based component only representation. In fact, the latter representation is unable to capture two significant classification clues: the context and the structure. The context includes argumentative markers ('In my opinion', 'I think', etc.) while the structure captures the position of the argument component in argumentative text.

The results from Task 2 show that our features as text representation outperforms the classical representation where structural features are numerically concatenated with BERT embedding. For the *PE* and *CMV* datasets, the improvement in F1 scores is significant: from 0.76 to 0.82 and from 0.73 to 0.78, respectively. For the *YWC* dataset, on the other hand, the improvement is less significant: from 0.67 to 0.69. These results support our contention that the datasets for which the contextual and structural features provide the most significant information (Task 1) correspond precisely to those where the *features as text* representation performs the best (Task 2). In other words, the more significant the contextual and structural features, the better the *features as text* representation. Overall, our approach shows that BERT performs better when non-textual information is given to it as text.

Moreover, we think that the ratio of claims and premises in the datasets is also important. In *PE*, the number of premises is almost three times the number of claims and in *CMV*, it is almost one half times the claims. But in *YWC*, claims and premises almost equal in number (See Table 10.1). This contradicts the general pattern of structured argumentation where a claim is followed by several (at least one) premises in support of that claim. The fact that structural features do not improve the results for the *YWC* dataset reinforces the observation that the spoken *YWC* dataset lacks the structure present in the written datasets.

9.6 Conclusion

In this work, we introduce a novel model for argument component classification which is based on the popular BERT model and inspired by the game-changing prompting paradigm. Our model incorporates contextual, structural and syntactic features *as text* to build an enriched BERT-based representation of the argument component.

We experiment with our model on three datasets: two written and one spoken. We obtain three main results: 1) our *features as text* sentence representation model improves upon the BERT-based component only representation, 2) our structural *features as text* representation outperforms the classical approach of numerically concatenating these features with BERT embedding and 3) our model achieves state-of-art results on two datasets and 95% of the best results on the third. To the best of our knowledge, our work is the first to investigate and implement a model based on *features as text* sentence representation.

Based on our results and analysis, we think that a systematic study to compare Argument Mining dynamics in written and spoken datasets would be of great benefit to the AM community. In terms of prospective research directions, we plan to merge our *features as text* technique with the LSTM-minus-based span representation model of Kuribayashi et al. [89]. We also intend to extend our *features as text* technique to other features such as syntactic and lexical [171].

We see our work as a first step towards a hybrid BERT-PROMPT end-to-end AM pipeline, thereby combining two dominant NLP paradigms. We think that our features as text approach opens up exciting new possibilities both for Argument Mining as well as any NLP tasks which require feature engineering. More generally, our approach can be used in other ML settings where the features can be described as text.

Chapter 10

Argument Mining with Modular BERT and Transfer Learning

10.1 Introduction

In Natural Language Processing (NLP), Argument Mining is concerned with identification and analysis of argumentative and discursive structures in texts [60]. This field is gaining increasing importance with the growing amount of textual data involving argumentative discourse from different sources and domains. For instance, legal texts contain law-based reasoning with a complex underlying argumentative structure [121]. Essays and articles consist of ordered presentation of claims and premises on a certain topic [121, 170]. Organized political debates involve argumentative dialogues between candidates on different issues [114, 102]. Social media platforms provide an avenue for users to debate and discuss contentious issues [168].

A complete end-to-end Argument Mining pipeline consists of the following related sub-tasks [24, 132]: 1) Argument Component Detection (ACD): given a token, classify whether it is part of an argument component or not; 2) Argument Type Classification (ATC): given an argument component, classify it as a *Major Claim*, *Claim* or *Premise*; 3) Link Identification (LI): given an argument component, classify it as either *Linked* or *Not Linked* to another argument component and 4) Link Type Classification (LTC): given a linked argument component, classify whether the link is of a *Support* or of an *Attack* type. The end output of the Argument Mining pipeline is a tree-like structure of the argumentative text [171] where the classified argument components are the nodes and links between argument components are the edges. This structure can then be utilized for downstream reasoning-based applications, like Text Summarization and Question Answering. The Argument Mining sub-tasks have been approached from both single-task and joint-task learning perspectives, using model architectures of varying complexity and with or without additional features [171, 138, 61, 112, 89] (see Section 10.2 Related Works for more details).

Transformer models [184], like Bidirectional Encoder Representations from Transformers (BERT) [37], have revolutionized NLP. The BERT model, composed of stacked encoder blocks of the Transformer, combines the advantages of the powerful attention mechanism [184] with a fast and parallelizable feed-forward architecture. BERT is trained in a two stage process: a self-supervised stage where the model is pre-trained on a huge textual corpus, followed by a supervised stage in which the pre-trained model is fine-tuned on a downstream task. BERT and its distilled versions have been successfully used for several NLP tasks [184, 156]. When used as sentence representations, BERT outperforms earlier embeddings like GloVe, ELMo, FastText, etc.

Despite its high efficiency, a standalone BERT fine-tuned on isolated argument components suffers from performance limitations [122]. This is due to the complicated and nuanced nature of argumentative texts, where the text of an argument component alone does not provide sufficient information for its accurate classification. In fact, the role of an argument component depends strongly, among other factors, on the presence of argument markers (‘Consequently,’, ‘However,’ etc.). Additionally, accurate classification of an argument component also requires positional and structural information about the component: its position in the paragraph and the complete essay, etc. [171]. Therefore, it is crucial for a BERT-based model for Argument Mining to have the ability to capture the contextual, structural and syntactic features which are essential for accurate classification. Accordingly, our approach in this paper seeks to address these dynamics exactly: we first embed the complete paragraph, allowing for connective clues and structural flow between components to be captured. Then, we contextualize the three essential feature groups (contextual, structural and syntactic) in parallel. Finally, we combine the separate contextualized feature groups to form a targeted and enriched representation of the ADU.

In Argument Mining, transfer learning between the Argument Type Classification (ATC) and the Link Identification (LI) tasks is of particular relevance [89]. For example, in the ATC task, the classifier learns that the first component of a paragraph has a higher probability of being a claim. Then, via transfer learning, the classifier can use this information in the LI task to deduce that the first component in a paragraph is most likely linked to some other component, since claims are almost always linked, either by outgoing links to major claims or by incoming links from premises in the paragraph.

This work focuses on Argument Mining in the Persuasive Essays (PE) dataset which consists of written essays on various topics. We introduce a modular BERT-based model, called *BERT–MINUS*, which consists of four BERT models, a custom *Features as Text (FeaTxt)* sentence representation, and a *Selective Fine-tuning* process for transfer learning. The architecture of this model is a generalization of the LSTM–Minus model of Kuribayashi et al. [89]. The Features as Text (FeaTxt) enhancement is inspired by the cutting-edge Prompt Engineering approach [107] and is also in line with the work of Mushtaq and Cabessa [122].

The BERT–MINUS model works as follows: the Joint Module embeds a complete input paragraph which consists of several Argumentative Discourse Units (ADU) to be classified. Taking this paragraph embedding as input, the Span Representation Function computes span-based representations for argument markers (AM), argument components (AC), and additional features – *given as text* (FeaTxt). Subsequently, the Dedicated Module, composed of three BERT models, contextualizes these span representations separately to better capture the flow between them. These contextualized representations are then concatenated to obtain a combined representation of the ADU which is finally fed to a classification layer.

To exploit transfer learning between or across LI and ATC tasks, we endow the BERT–MINUS model with both intra-task and inter-task (classical) transfer learning capabilities through the *Selective Fine-tuning* mechanism.

The BERT–MINUS model achieves state-of-the-art results on the LI task and competitive results on the ATC task. Moreover, the synergy between the Features as Text and the selective fine-tuning mechanisms significantly improve the performance of BERT–MINUS. More generally, our study reveals the importance of careful fine-tuning for modular language models. It also naturally dovetails into the Prompt Engineering paradigm in NLP. We make the code available on GitHub at the following address:
https://github.com/mohammadoumar/BERT–MINUS–FeaTxt.

The main contributions of this paper are as follows:

- We introduce a modular BERT-based model, called BERT–MINUS, which consists of four BERT models which separately, and in parallel, contextualize AMs, ACs and FeaTxt of an ADU to form a targeted and enriched embedding of the ADU.

- We introduce a two-mode Selective Fine-tuning process for transfer learning between Link Identification (LI) and Argument Type Classification (ATC).
- BERT–MINUS achieves state-of-the-art results on the LI task and competitive results on the ATC task. The Features as Text and Selective Fine-tuning mechanisms significantly improves the performance of the model.

This paper is structured as follows. Section 10.2 presents the literature related to our work. Section 10.3 introduces the BERT–MINUS model and the selective fine-tuning mechanism in detail. Section 10.4 describes the experimental setup of our work. In Section 10.5, we present our results and analyse them. We conclude and propose future directions in Section 10.6.

10.2 Related Works

In the literature, several distinct approaches have been proposed for Argument Type Classification (ATC) and Link Identification (LI) in structured texts. For both tasks, different architectures and feature sets have been studied and analyzed.

Stab and Gurevych [171] investigated ATC, LI and Link Type Classification (LTC) in the Persuasive Essays (PE) dataset. They used Support Vector Machines (SVM) and Conditional Random Fields (CRF) with hand-crafted feature sets consisting of lexical, structural, syntactic, contextual and discursive features. For ATC, they report that structural features produce the best results. For LI, a combination of features yields the best performance. Their work reveals the importance of well-designed feature groups for Argument Mining sub-tasks. Our work incorporates their feature groups approach into transformer-based language models.

Hadaddan et al. [61] focused on the ACD and ATC tasks. They introduced the Yes We Can! (YWC) dataset which consists of transcribed political speeches. They present both feature-based and recurrent neural network-based approaches. The former involves simple feed-forward networks with features consisting of Bag of Words (BoW), N-Grams, Part of Speech (POS) tags, Named Entity Recognition (NER) tags, etc. The latter approach involves Feed-Forward and LSTM architectures with FastText word embedding. Their work posits the importance of syntactic and grammatical features for Argument Mining.

Potash et al. [138] approached both ATC and LI as a joint learning task. They introduced a custom Joint Neural Model for the Persuasive Essays (PE) and Micro-Text Corpus (MTC) datasets. This model consists of a Bi-LSTM encoder combined with a fully connected layer for ATC and an LSTM decoder for LI. For textual representation, they use Bag of Words (BoW), GloVe embedding and structural features. This approach combines the advantages of additional features and embeddings when used in conjunction with recurrent neural networks.

Mayer et al. [112] combined the ACD and ATC tasks into one sequence tagging task. They use a dataset based on abstracts of Randomized Controlled Trials (RCT) from the MEDLINE database. They use combinations of static and dynamic embeddings as textual representations together with LSTMs, GRUs and BERT fine-tune for an end-to-end Argument Mining pipeline.

Kuribayashi et al. [89] present a model which builds upon the *LSTM–Minus* span representations of Wang and Chang [193] and Li et al. [100]. The LSTM–Minus span representation of a text span (i, j) is defined as the subtraction ('Minus') of the hidden layer outputs of the LSTM model at indices j and i. Based on this definition, Kuribayashi et al. presented two cases: (i) a *joint span model* where an argumentative discourse unit (ADU) is considered as a single span (i, j) and (ii) a *distinction model* where the ADU span (i, j) is separated into an argument marker span (i, k) and an argument component span $(k + 1, j)$. The motivation for the latter model is to better capture the flow between the argument markers ('I think', 'because', etc.) and the argument components ('we should limit immigration', 'tertiary education is more important than secondary', etc.) [89].

In the Kuribayashi et al. distinction model, the span representation of both the argument marker and the argument component is computed according to the *LSTM–Minus representation formula*. Then, these two representations are contextualized using two separate Bi-LSTMs. Finally, these contextualized representations are concatenated, optionally with BoW and structural features, to obtain the representation of the complete ADU. Kuribayashi et al. considered three tasks: ATC, LI and LTC, both separately and jointly. In the joint learning setting, they used a custom loss function consisting of weighted combination of loss functions for all three tasks. They experiment with both the PE and MTC datasets.

Finally, Mushtaq and Cabessa [122] introduced the *BERT with Features as Text (BERT–FeaTxt)* model for ATC. They present a combined features as text sentence representation which incorporates contextual, structural and syntactic features along with the argument component. The contextual features are the topic and the full sentence, while the structural features relate to the position of the component in the essay and the paragraph. As syntactic features, Part Of Speech (POS) tags of the component are used. This enriched sentence representation is then utilized to fine-tune a BERT model for the ATC task.

Mushtaq and Cabessa [122] experiment with the PE, Change My View (CMV) and Yes We Can! (YWC) datasets. They report two important results: firstly, the BERT–FeaTxt model outperforms standalone BERT, and secondly, BERT–FeaTxt outperforms the classical case where structural features are concatenated numerically to the BERT embedding.

In this paper, we combine our previous work [122] with the Kuribayashi span-representation approach [89]. We seek to leverage the features as text capabilities of our BERT–FeaTxt model and the enhanced span-representation capabilities of the Kuribayashi model.

10.3 Model

In this section, we first recall the *BERT with Features as Text (BERT–FeaTxt)* model [122]. We then introduce our modular *BERT–MINUS* model for Argument Type Classification (ATC) and Link Identification (LI). Finally, we explain how the BERT–MINUS model can leverage transfer learning via the *Selective Fine-tuning* mechanism.

10.3.1 BERT–FeaTxt

Contextual, structural and syntactic features are crucial for building meaningful representations of argument components [171]. Accordingly, Mushtaq and Cabessa [122] introduced the *BERT with Features as Text (BERT–FeaTxt)* model. In addition to the argument component itself, this model incorporates in its input hand-crafted features – *given in textual form* – rather than in numerical form. This approach leverages the bidirectional contextual and linguistic capabilities of BERT, enabling it to create an enriched representation of the whole input text. The BERT–FeaTxt model and the textual representations of its features are described in more detail below.

Contextual Features The full meaning of an argument component depends inherently on the linguistic and semantic context in which it occurs. Therefore, contextual information is an important factor in the classification of an argument component. Accordingly, BERT–FeaTxt utilizes: 1) the full sentence in which the argument component occurs and 2) the topic of the essay as the contextual features for an argument component. Formally, the textual representation of contextual features is given as follows:

`contextual_fts` = 'Topic: t. Sentence: s.'

Structural Features Written essays naturally follow a structured argumentative pattern. The essay usually begins with a statement of the writer's stance on the topic. Thereafter, claims in support of the stance and premises to support these claims are presented in successive paragraphs. Therefore, the position of the argument component in the essay and paragraph contains vital information for its classification. As structural features, BERT–FeaTxt utilizes: 1) the paragraph number in which the argument component appears, 2) whether it is in the introductory or 3) concluding paragraph, and 4) if it is the first or 5) last component in the paragraph. Formally, the textual representation of structural features is given as follows:

`structural_fts` = 'Paragraph Number: n. Is in introduction: i. Is in conclusion: c. Is first in paragraph: f. Is last in paragraph: l.'

Syntactic Features The linguistic and grammatical characteristics of an argument component are also a factor in determining its argumentative role. Accordingly, BERT–FeaTxt incorporates Part of Speech (POS) tags of the argument component as its syntactic features. POS tags determine whether each token is a noun, a verb, an adjective, and so on. Formally, the textual representation of syntactic features is given as follows:

`syntactic_fts` = 'Part Of Speech tags: $t_1, t_2, \ldots, t_n$'

where t_i represents the POS tag of the i-th token in the argument component.

Combined Features as Text As its input, the BERT–FeaTxt model combines the contextual, structural and syntactic features as follows:

```
combined_fts = contextual_fts +
               structural_fts +
               syntactic_fts
```

where '+' denotes the string concatenation operation. Note that the argument component itself is, by definition, included in the contextual features.

10.3.2 BERT–MINUS

We now introduce our modular *BERT–MINUS* model in detail. BERT–MINUS contextualizes AM, AC and FeaTxt of an ADU in parallel to form a targeted and enriched embedding of the ADU. In the main, BERT–MINUS consists of three parts: 1) a joint BERT module, 2) a span representation function and 3) a dedicated module consisting of three BERT models with customized input embeddings. In addition to these parts, the BERT–MINUS model also has intermediate layers and an output layer (see Figure 10.1).

Input The input to the BERT–MINUS model consists of a paragraph from an essay and the spans tensor of the paragraph (see Figure 10.1, Paragraph and Spans). Each paragraph contains a number of Argumentative Discourse Units (ADU). Each ADU consists of an Argument Marker (AM) (blue text in Figure 10.1) and an Argument Component (AC) (red text in Figure 10.1). For a text sequence, its span is the pair of indices, (i, j), of its first and last token in the tokenized paragraph.

The BERT–MINUS model can be utilized both without or with features as text (FeaTxt). In the former case, the spans tensor consists of the AM spans (i_{am}, j_{am}) and the AC spans (i_{ac}, j_{ac}) of all ADUs in the paragraph. In the latter case, the spans tensor also includes the spans (i_{fts}, j_{fts}) of the features as text (cf. Section 10.3.1) of all ADUs. The spans tensor for the paragraph, then, consists of the list of spans

$$\big[(i_{am_1}, j_{am_1}), (i_{ac_1}, j_{ac_1}), (i_{fts_1}, j_{fts_1}),\\ (i_{am_2}, j_{am_2}), (i_{ac_2}, j_{ac_2}), (i_{fts_2}, j_{fts_2}), \dots\big]$$

of all the ADUs ($i = 1, 2, \dots$) in the paragraph.

Joint Module The first module of BERT–MINUS is a standalone pre-trained BERT model (see Figure 10.1, $\mathrm{BERT}_{\mathrm{joint}}$). We use this model to contextualize and embed the input paragraph.

Span Representation Function The span representation function takes two objects as input: the output of the Joint Module, which is a sequence of 768 dim vectors whose length equals the number of tokens in the paragraph, and the spans tensor of the paragraph. This function computes three span representations: one each for the AM, AC and FeaTxt of every ADU in the paragraph (see Figure 10.1, Span Representation Function). For a text sequence (AM, AC or FeaTxt of an ADU) with span (i, j), its BERT–MINUS span representation is computed as follows:

$$\left[\mathbf{h_j} - \mathbf{h_{i-1}}\ ;\ \mathbf{h_i} - \mathbf{h_{j+1}}\ ;\ \mathbf{h_{i-1}}\ ;\ \mathbf{h_{j+1}}\right]$$

where $\mathbf{h_i}$ is the output of the Joint Module at the i-th index and ';' represents tensor concatenation. In this computation, the first and second term represents the embedding of the text in the forward and backward direction, respectively. The last two terms capture the preceding and succeeding context of the text sequence (span). These representations are based on the LSTM–Minus representation of Kuribayashi et al. [89].

Each span representation is of dimension $4 * 768 = 3072$. Before they are input to the next module (Dedicated Module), these span representations are reshaped using three parallel intermediate linear layers: $\mathrm{LINEAR_{am}}$, $\mathrm{LINEAR_{ac}}$ and $\mathrm{LINEAR_{fts}}$, respectively, each of input dimension 3072 and output dimension 768.

Dedicated Module This module consists of three dedicated BERT models, $\mathrm{BERT_{am}}$, $\mathrm{BERT_{ac}}$, $\mathrm{BERT_{fts}}$ (see Figure 10.1), which process the AM, AC and FeaTxt span representations, respectively. The embedding layer of these models are customized so that they can take sequences of vectors (span representations) instead of token ids as inputs. In this way, each of the AM, AC and FeaTxt span representation is contextualized separately by a dedicated customized BERT model. The outputs of these dedicated models are then used to obtain a combined representation of the whole ADU as follows:

$$\begin{aligned}\mathrm{REP_{ADU}} \;=\; \big[&\mathrm{BERT_{am}}(\mathrm{am\_span\_representation});\\ &\mathrm{BERT_{ac}}(\mathrm{ac\_span\_representation});\\ &\mathrm{BERT_{fts}}(\mathrm{fts\_span\_representation})\big]\end{aligned}$$

This BERT–MINUS ADU representation is finally fed to a fully connected layer for classification into the respective classes for the two tasks.

10.3.3 Selective Fine-Tuning

To enable transfer learning between Link Identification (LI) and Argument Type Classification (ATC) tasks, we adjoin a three-step, two-mode *Selective Fine-tuning* mechanism to our BERT–MINUS model:

1. A pre-trained BERT model is fine-tuned on one task, ATC or LI.
2. This fine-tuned model is instantiated as the Joint BERT module of the BERT–MINUS model.
3. BERT–MINUS is fine-tuned, either on the same task as Step 1 (*auto-transfer* mode) or on the other task (*cross-transfer* mode).

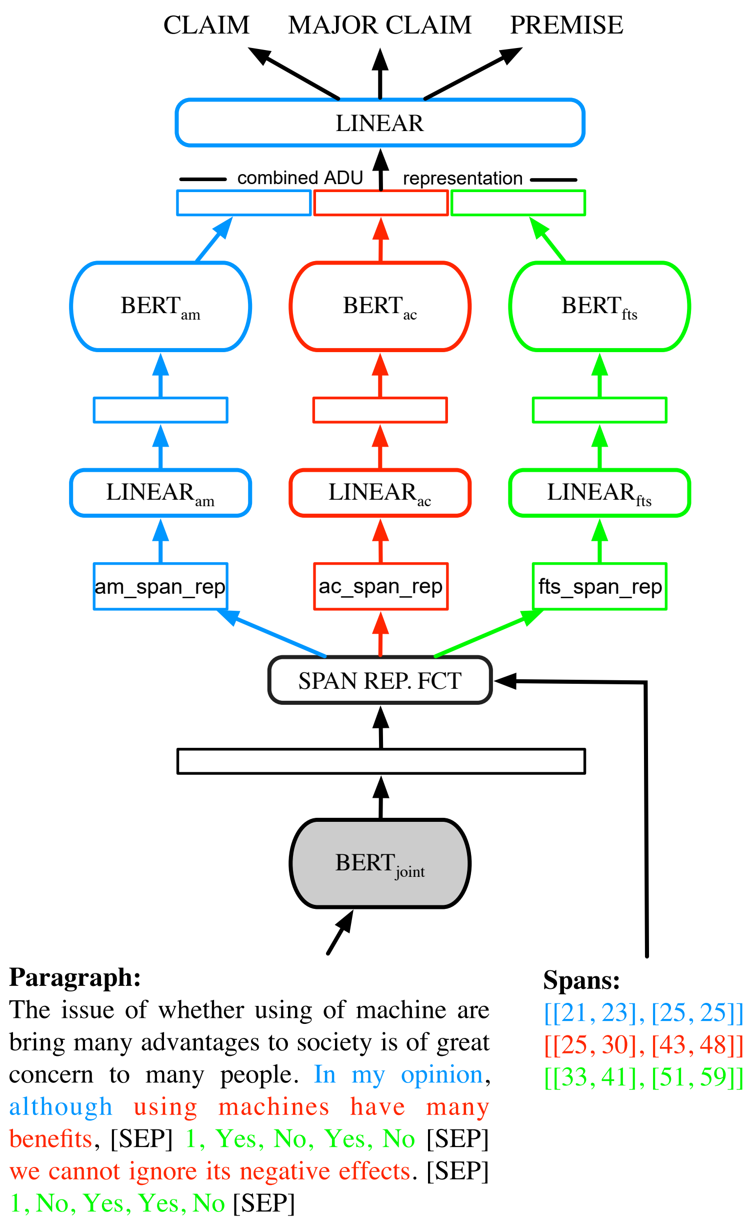

CLAIM
MAJOR CLAIM
PREMISE
LINEAR
combined ADU representation
BERT$_{am}$
BERT$_{ac}$
BERT$_{fts}$
LINEAR$_{am}$
LINEAR$_{ac}$
LINEAR$_{fts}$
am_span_rep
ac_span_rep
fts_span_rep
SPAN REP. FCT
BERT$_{joint}$
Paragraph:
The issue of whether using of machine are bring many advantages to society is of great concern to many people. In my opinion, although using machines have many benefits, [SEP] 1, Yes, No, Yes, No [SEP] we cannot ignore its negative effects. [SEP] 1, No, Yes, Yes, No [SEP]
Spans:
[[21, 23], [25, 25]]
[[25, 30], [43, 48]]
[[33, 41], [51, 59]]

Figure 10.1: Architecture of the BERT–MINUS model. The paragraph and the spans tensor are input to the model. The paragraph contains AMs (blue text), ACs (red text), and additional features as text (green text, only abbreviated form shown for brevity's sake), separated by the [SEP] tokens. The spans tensors consists of the span indices of AMs, ACs and features as text of the ADUs in the paragraph. The paragraph is fed to a joint BERT model. The output of this model, together with the spans tensor, are fed to the spans representation function. The AM, AC and FeaTxt BERT–MINUS representations obtained from this function are reshaped via three linear layers. These reshaped representations are fed to three dedicated BERT models. The outputs of these models are then concatenated to construct an enriched representation of the whole ADU. This ADU representation is then fed to a final fully connected layer for classification. The selective fine-tuning of the joint BERT module is represented by a gray coloring.

Instead of a generic pre-trained BERT model, the selective fine-tuning mechanism uses a BERT model already fine-tuned on one of the two tasks. By means of transfer learning, the paragraph embedding computed by the joint BERT module is more targeted towards the particular task. In addition, the selective fine-tuning mechanism is also motivated by Wieting and Kiela [199] who emphasize the importance of the embedding layer over the complexity of the subsequent encoder block.

10.4 Experiments

10.4.1 Dataset

We use the Persuasive Essays (PE) dataset introduced by Stab and Gurevych [171]. The PE dataset consists of 402 structured essays on various controversial topics such as *'Businesses should be only concerned about making profits'* and *'Spending money on supporting art or protecting environment'*. Of the 402 essays, 322 are set aside for the train set and 80 for the test set. The statistics of the the PE dataset are given in Table 10.1.

For our BERT–MINUS model, we separated each Argumentative Discourse Unit (ADU) of the dataset into an argument marker (AM) and an argument component (AC). To that end, we used the four types of argument markers of Stab and Gurevych: forward, backward, thesis and rebuttal [171].

Corpus Statistics		**Component Statistics**	
Tokens	147,271	Major Claims	751
Sentence	7,116	Claims	1,506
Paragraphs	1,833	Premises	3,832
Essays	402	Total	6,089

Table 10.1: Corpus and component statistics for the PE dataset.

10.4.2 Tasks

We focus on the two following Argument Mining sub-tasks:

1. *Link Identification (LI):* Given an argument component (AC), classify it as either *Linked* or *Not Linked*. Here, we approach LI as the task of classifying single argument components, as opposed to pairs of components as in [171, 89]. Since linked claims and linked premises are, by and large, linked to the major claims and the claims at the beginning of the paragraph, respectively, the essay tree structure can be properly reconstructed from the classification of separate components [171].

2. *Argument Type Classification (ATC):* Given an argument component (AC), classify it as either a *Major Claim*, a *Claim* or a *Premise*.

10.4.3 Models

In our work, we consider the following models:

- **BERT:** a standalone BERT model fine-tuned on argument components alone, without features as text (FeaTxt).
- **BERT–FeaTxt:** a BERT model fine-tuned on the combined features as text representation, as described in Section 10.3.1. The standalone BERT and BERT–FeaTxt models represent our baselines.
- **BERT–MINUS:** a BERT–MINUS model fine-tuned on paragraph texts as described in Section 10.3.2. This model takes no additional features as text (FeaTxt) as inputs (green features and modules in Figure 10.1) and has no selective fine-tuning.
- **BERT–MINUS–Auto:** a BERT–MINUS model where the joint BERT module is selectively fine-tuned on the same task (LI or ATC) as the one on which the BERT–MINUS model is being trained, as described in Section 10.3.3. We call this mode *auto-transfer learning*, i.e., transfer from one task to itself.
- **BERT–MINUS–Cross:** a BERT–MINUS model where the joint BERT module is selectively fine-tuned on the opposite task (LI $\rightarrow$ ATC, and vice-versa) as the BERT–MINUS model. We call this mode (classical) *cross-transfer learning*, i.e., transfer from one task to another.
- **BERT–MINUS–FeaTxt:** a BERT–MINUS model augmented with features given as text (FeaTxt), as described in Section 10.3.1 and illustrated in Figure 10.1, and with no selective fine-tuning.
- **BERT–MINUS–FeaTxt–Auto:** a BERT–MINUS–FeaTxt model with selective fine-tuning in the *auto-transfer* mode.
- **BERT–MINUS–FeaTxt–Cross:** a BERT–MINUS–FeaTxt model with selective fine-tuning in the *cross-transfer* mode.

10.5 Results and Analysis

We present and analyze the results of the various BERT–MINUS models on the Link Identification (LI) task and the Argument Type Classification (ATC) tasks. We also present the result of the Link Type Classification (LTC) task with the BERT–FeaTxt model [122].

10.5.1 Link Identification Task

The results for the LI task are given in Table 10.2. The analysis of these results reveals several important insights and patterns.

Models	**L**	**NL**	**F1**
BERT	0.216	0.833	0.524
BERT–FeaTxt	0.585	0.877	0.731
BERT–MINUS	0.721	0.826	0.773
BERT–MINUS–Auto	0.760	0.830	0.795
BERT–MINUS–Cross	0.750	0.835	0.793
BERT–MINUS–FeaTxt	0.709	0.800	0.755
BERT–MINUS–FeaTxt–Auto	0.763	0.841	0.802
BERT–MINUS–FeaTxt–Cross	0.778	0.850	**0.814**
Stab and Gurevych [171]	0.585	0.918	0.751
Niculae et al. [127]			0.601
Kuribayashi et al. [89]			0.783

Table 10.2: Results for the LI task. The performance of the different BERT and BERT–MINUS models described in Section 10.4.3 are reported. L and NL represents the F1 scores for *Linked*, and *NotLinked*, respectively. F1 stands for the macro F1 score. The empty cells come from the fact that in the literature, only the macro F1 score was given. The 2 first rows concern the BERT model, the 3 next ones the BERT–MINUS model, and the 3 following ones the BERT–MINUS–FeaTxt model.

First, we see that the BERT–FeaTxt model drastically improves on the standalone BERT performance (Table 10.2, rows 1 and 2). This improvement is due to the addition of contextual, structural and syntactic features which allows the model to build richer embeddings of the argument components. This observation comports exactly with the results of Mushtaq and Cabessa [122]. Indeed, they further show that the additional features are better exploited when given in a textual rather than numerical form.

Secondly, we see that the BERT–MINUS model significantly improves over both standalone BERT and BERT–FeaTxt (Table 10.2, rows 1, 2 and 3). Recall that BERT–MINUS takes complete paragraphs as input whereas both BERT and BERT–FeaTxt take single components only. Consequently, BERT–MINUS is better able to capture the contextual and argumentative flow between successive components. As a result, the BERT–MINUS component representations are more contextually enriched, leading to improved accuracy. This suggests that, for some tasks, it is more efficient for a model to build contextualized representations from raw texts (BERT–MINUS) than from descriptive features (BERT–FeaTxt). We will see, however, that this does not apply to the ATC task.

Thirdly, for both BERT–MINUS and BERT–MINUS–FeaTxt models, the selective fine-tuning mechanism improves the results (Table 10.2, rows 3–5 and 6–8). We believe that this phenomenon is due to two important reasons: firstly, when selectively fine-tuned, BERT–MINUS is placed in a more 'informative' initial configuration from which it can reach a lower local minimum during training. Secondly, selective fine-tuning improves the quality of the paragraph embedding which, in turn, positively impacts the whole training process. These results are in line with those of Wieting and Kiela [199], who show the importance of the embedding over the complexity of the subsequent encoder.

Furthermore, note that for BERT–MINUS, both *auto-transfer* and *cross-transfer* modes achieve comparable results (Table 10.2, rows 4–5). By contrast, for BERT–MINUS–FeaTxt, *cross-transfer* significantly outperforms *auto-transfer* (Table 10.2, rows 7–8). In fact, the results achieved by BERT–MINUS with FeaTxt and *cross-transfer* are state-of-the-art.

Finally and surprisingly, BERT–MINUS outperforms BERT–MINUS–FeaTxt (Table 10.2, rows 3 and 6). This shows that, when no transfer-learning is involved, it is actually more efficient for BERT–MINUS to build contextualized representations from raw texts than from descriptive features. By contrast, when transfer-learning come into play, BERT–MINUS–FeaTxt outperforms its BERT–MINUS counterpart (Table 10.2, rows 4–5 and 7–8). In fact, performing both the first and third steps of selective fine-tuning with same features as text leverages and enables transfer learning between the tasks.

10.5.2 Argument Type Classification Task

The results of the ATC task are presented in Table 10.3.

Models	**MC**	**C**	**P**	**F1**
BERT	0.703	0.507	0.841	0.686
BERT–FeaTxt	0.855	0.678	0.909	0.814
BERT–MINUS	0.784	0.602	0.865	0.750
BERT–MINUS–Auto	0.847	0.617	0.888	0.784
BERT–MINUS–Cross	0.813	0.633	0.888	0.778
BERT–MINUS–FeaTxt	0.746	0.537	0.863	0.715
BERT–MINUS–FeaTxt–Auto	0.900	0.687	0.903	**0.831**
BERT–MINUS–FeaTxt–Cross	0.869	0.618	0.890	0.792
Stab and Gurevych [171]	0.891	0.682	0.903	0.826
Niculae et al. [127]	0.782	0.645	0.902	0.776
Kuribayashi et al. [89]				**0.856**

Table 10.3: Results for the ATC task. The performance of the different BERT and BERT–MINUS models described in Section 10.4.3 are reported. MC, C and P represents the F1 scores for *Major Claim*, *Claim* and *Premise*, respectively. F1 stands for the macro F1 score.

As for the LI task, we see that BERT–FeaTxt significantly outperforms standalone BERT (Table 10.3, rows 1 and 2). This shows that contextual, structural and syntactic features – *given as text (FeaTxt)* – capture important information necessary for determining the argumentative role of a component [122].

Secondly, BERT–MINUS also improves upon standalone BERT (Table 10.3, rows 1 and 3). As already explained for the LI task, the contextualized representations built by BERT–MINUS capture argumentative flow from complete paragraphs as opposed to individual components. However, in contrast with the LI task, BERT–FeaTxt outperforms BERT–MINUS (Tables 10.2 and 10.3, rows 1–3). This means that, for this task, the component representations built from descriptive features are more useful than those obtained from full markers and components.

Thirdly, our selective fine-tuning mechanism improves classification accuracy for both BERT–MINUS and BERT–MINUS–FeaTxt (Table 10.3, rows 3–5 and 6–8). As with the LI task, we conjecture that transfer learning yields an improved initial configuration of the BERT–MINUS model as well as an improved embedding of the paragraph text.

Moreover, the *cross-transfer* mode under-performs the *auto-transfer* mode for both BERT–MINUS and BERT–MINUS–FeatTxt (Table 10.2, rows 4–5 and rows 7–8). By comparing these results for the two tasks, we conclude that transfer learning from ATC to LI is more successful than that from LI to ATC. This is explained by the fact that the argumentative role of a component is more useful for inferring its linked or not linked type, than vice versa.

Furthermore, BERT–MINUS outperforms BERT–MINUS–FeaTxt (Table 10.3, rows 3 and 6) for the ATC task as well. However, with selective fine-tuning, BERT–MINUS–FeaTxt outperforms BERT–MINUS (Table 10.2, rows 4–5 and 7–8). As with the LI task, this shows that transfer learning happens properly when both joint BERT module and BERT–MINUS model are fine-tuned with features as text.

Finally, we observe that the combination of the features as text and selective fine-tuning process in *cross-transfer* mode leads to the best results. The synergy of the two mechanisms generates a combined effect that surpasses the sum of its parts. For this task, we improve above Stab and Gurevych's Joint ILP Model [171], but unfortunately, remain below the Kuribayashi LSTM-Minus model [89].

10.5.3 Link Type Classification

In addition, to reinforce the results of Mushtaq and Cabessa [122], we also trained BERT–FeaTxt on the Link Type Classification (LTC) task. The results are given in in Table 10.4.

Models	**Attack**	**Support**	**F1**
BERT–FeaTxt	0.506	0.960	0.733
Stab and Gurevych [171]	0.413	0.947	0.680
Kuribayashi et al. [89]			**0.796**

Table 10.4: Results for the Link Type Classification task. The Stab and Gurevych results are for the full features set and an SVM classifier [171]. The BERT–FeaTxt results are from Mushtaq and Cabessa [122].

In the LTC task, BERT–FeaTxt improves the performance of Stab and Gurevych [171]. Once again, this shows that the features as text yield to enriched and improved representations of argument components, leading to better classification accuracy. However, we remain below Kuribayashi et al. [89] which we plan to investigate from the BERT–MINUS perspective in a future paper.

10.6 Conclusion

In this paper, we focus on two Argument Mining sub-tasks: Link Identification (LI) and Argument Type Classification (ATC) for the Persuasive Essays (PE) dataset. More precisely, we introduce the modular BERT–MINUS model with Features as Text (FeaTxt) and Selective Fine-tuning mechanisms. The model works by constructing an enriched embedding for the whole paragraph text via a joint BERT module and then contextualizing the argument marker, component and additional features as text of the argument discourse unit (ADU) separately via a dedicated module consisting of three customized BERT models. The aggregation of these contextualized representations yields an enriched representation of the ADU. We endow our model with transfer learning capabilities via selective fine-tuning which comes in two modes: *auto-transfer* which implements intra-task transfer, and *cross-transfer* which implements inter-task/classical transfer.

Our experiments show that the BERT–MINUS model with features as text and selective fine-tuning improves over standalone BERT and BERT–FeaTxt for both LI and ATC tasks. The combination of features as text and selective fine-tuning mechanisms significantly augment the capabilities of the BERT–MINUS model. With this enhanced combination, we achieve state-of-the-art results on the LI task and competitive results for the ATC task.

We believe that our work opens up several interesting research directions. For example, an end-to-end Argument Mining pipeline based on our BERT–MINUS-FeaTxt model is the natural next step. Furthermore, we think that selective fine-tuning, both in the *auto-transfer* and the *cross-transfer* modes, can be used to investigate transfer learning between Argument Mining sub-tasks in various architectures and models like Potash et al. [138] and Kuribayashi et al. [89]. Moreover, our BERT–MINUS model is a generalization of LSTM–Minus span representation-based model of Kuribayashi et al. [89]. We think that span representation computations can be enhanced using BERT's particular attention-based contextualization capabilities instead of the LSTM–Minus construction. In addition, following Kuribayashi et al. [89] who report improvements in the joint-task learning setting, we plan to investigate joint-task learning for the BERT–MINUS model.

More generally, we believe that our selective fine-tuning mechanism opens possibilities for exploration and implementation in other modular Language Models. Finally, our work also dovetails naturally into the cutting-edge Prompt Engineering paradigm in NLP.

Chapter 11

Explainable Decisions under Incomplete Knowledge with Weights

11.1 Introduction

An individual decision analysis process involves an agent (the decision maker) taking account of the decision situation and then evaluating different courses of action (decision alternatives). In order to be able to do so, the decision maker must characterize the decision-making situation with respect to two distinct components: a formulation of the decision goals and a characterization of the decision alternatives [178]. Usually, the evaluation is based on an associated utility function (see e.g. the introductory book of [147]) which encodes the satisfaction degrees reached by choosing each decision alternative. Despite a lot of works on decision theory, two issues are often not easy to solve. The information of the agent about the decision situation is often uncertain, incomplete and distributed. Hence the first issue is to deal with imperfect information (uncertainty, incomplete and distributed knowledge). The second issue is to be able to explain and justify the decisions that are made. It is also a desirable goal to enable the decision makers to have a broader view of the principles that govern the decision and to enable them to participate in their elaboration. These issues are even more important when the decision to be made concerns a group of autonomous agents that have their own knowledge and preferences.

Classical qualitative decision-making approaches use aggregation criteria that combine the measurement of uncertainty with utility (see e.g. [44]). Roughly speaking we can sum up the standard approaches of decision under uncertainty as follows: the decision maker defines a utility function $f(d, s)$ which associates a value to a decision d in a given scenario s. The second step consists in defining an aggregation function on all the possible scenarios given the uncertain current knowledge about the real situation. We propose an approach which allows the user to choose the best decision and also to explain it. Indeed, instead of using a utility function over all possible states, we propose to use a set of decision principles (DP). A DP relates some characteristic features of the situation to the achievement of a tangible result (which has a utility level). For instance, if an agent wants to find a hotel, we can enunciate a decision principle saying that "a priori, a hotel with a pool gives the opportunity to swim" (where "swimming" is a tangible result with a good level of utility for our agent). Our approach is a two step process: the first step computes the certainty of the achievement of some tangible results, leading to compute the function $N(r|S)$ that gives the necessity of having the result r given a set of situations S (obtained by a decision made on an uncertain scenario). Then the second step consists in aggregating (taking into account their importance, polarities and certainty) the possible results that can be obtained in a given situation in order to compare the different situations. For this step, we use an extension of the qualitative bipolar approach of Dubois and Fargier [41] where positive arguments (pros) and negative arguments (cons) are uncertain.

The particular originality of the BLFSW is mainly its first step process which is done by using several argumentation graphs, each argumentation graph enables us to assess about the achievement of one tangible result. In argumentation theory, there are two kinds of actions on arguments: *attacks* that tend to say that the conclusion of the argument (here the tangible result) is not achieved in a given situation and *supports* that increase the belief degree that the result is achieved. Principles in BLFSWs are akin to arguments in that they state a reason for believing that a tangible result is obtained. The notion of *argument* in favor or against a decision has been developed in practical argumentation domain which has been widely studied (see e.g. [201, 7]) since the initial proposal of Raz [149] and the philosophical justification provided by Walton [187]. Practical argumentation aims at answering the question *what is the right thing to do in a given situation* which is clearly related to a decision problem. Several works are using argumentative approaches to tackle it: Amgoud and Prade [8] propose a bipolar argumentation-based approach distinguishing epistemic and practical arguments. Argumentation has also been proposed to govern decision making in a negotiation context (see for instance [9] and [146] for a survey). However in all the argumentative approaches mentioned above, it is difficult to obtain a precise explanation of the decision: either because the arguments are abstract and only the attack relation between them is informative, or because there is no clear explanation of how and why the content of the argument justifies the final choices. BLFSW is a new method for reasoning about decision arguments in which more place is given to explanation by making explicit both the decision principles and their supports and inhibitors.

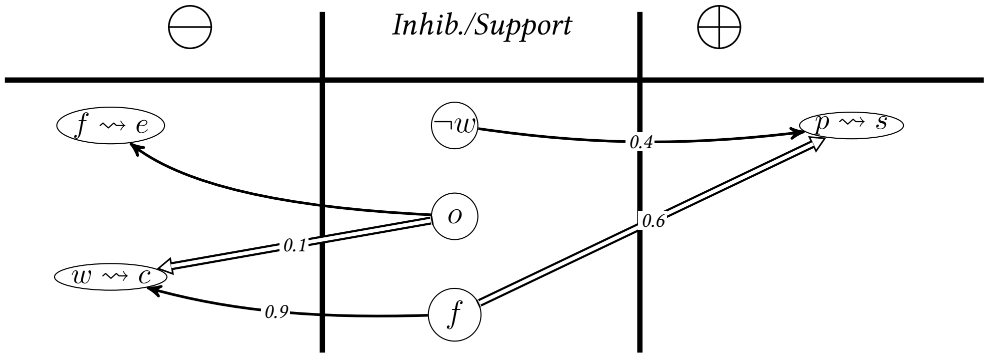
11.2 BLF with supports and weights

A BLFSW is a visual bipolar framework that represents all explicit information known about a decision domain. Hence, it contains both the knowledge for reasoning about the achievement of tangible results (called goals) and the preference information associated to these goals: namely their polarities and their importance level. The polarity of a goal is positive if it is a desirable result, it is negative when this result should be avoided.

We consider a set d of alternatives about which some information is available and two languages $\mathscr{L}_F$ (a propositional language based on a vocabulary $\mathcal{V}_F$) representing information about some features that are believed to hold for an alternative and $\mathscr{L}_G$ (another propositional language based on a distinct vocabulary $\mathcal{V}_G$) representing information about the achievement of some goals when an alternative is selected. In the propositional languages used here, the logical connectors *or*, *and*, *not* are denoted respectively by $\vee$, $\wedge$, and $\neg$. A *literal* is a propositional symbol x or its negation $\neg x$, the set of literals of $\mathscr{L}_G$ are denoted by LIT_G. Classical inference, logical equivalence and contradiction are denoted respectively by $\models$, $\equiv$, $\bot$. We use a special symbol $\rightsquigarrow$ to encode an a priori deduction, called Decision Principle, from some observations to a goal.

Example 1. *Let us imagine an agent who wants to find a hotel which is not expensive (e) and in which he can swim (s). This agent prefers to avoid crowded hotels (c). The possible pieces of information concern the following attributes: $\mathcal{V}_F = \{p, f, w, o\}$ that describes the respective features of the hotel "to have a pool", "to be a four star hotel", "to be in a place where the weather is fine", "to propose special offers". The agent may consider the following principles: $\mathcal{P} = \{p \rightsquigarrow s, f \rightsquigarrow e, w \rightsquigarrow c\}$. They respectively express that "a priori when there is a pool the agent can swim", "a priori if the hotel is four star then it is expensive" and "if the weather is fine in this area then the hotel is a priori crowded".*

The principles can be supported or inhibited, this is represented by double and single arcs *that are weighted accordingly to the strength of the support/inhibition*[1]*. For instance, a special offer increases the certainty to have a crowded hotel when the weather is fine. Four star hotels are expensive but a special offer may inhibit this deduction. The following picture is a graphical representation of this example by a BLFSW: it is a tripartite graph represented in three columns, the DPs with a positive goal are situated on the left column, the inhibitors and supports are in the middle, and the DPs with a negative polarity are situated on the right. The more important (positive and negative) DPs are in the higher part of the graph, equally important DPs are drawn at the same horizontal level. Hence the highest positive level is at the top left of the figure, the bottom right contains DPs with negative goals of low importance. The heights of the inhibitors and supports are not significant; only their existence is meaningful.*



More formally, a BLFSW is defined as follows:

Definition 1 (BLFSW)**.** *A Bipolar Layered Framework with Supports and Weights is a tuple* $(\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$*.* $\mathcal{P}$ *is a set of decision principles:* $\mathcal{P} = \{\varphi \rightsquigarrow g | \varphi \in \mathscr{L}_F, g \in LIT_G\}$*.* $\mathcal{I} \subseteq (\mathcal{L}_F \times \mathcal{P})$ *is a set of inhibitors.* $\mathcal{S} \subseteq (\mathcal{L}_F \times \mathcal{P})$ *is a set of supports.* pol *is a function* $pol : \mathcal{V}_G \to \{\oplus, \ominus\}$ *which gives the polarity of a goal* $g \in \mathcal{V}_G$*, this function is extended to goal literals by* $pol(\neg g) = -pol(g)$ *with* $-\oplus = \ominus$ *and* $-\ominus = \oplus$ *and to DPs accordingly:* $pol(\varphi, g) = pol(g)$*.* LIT_G *is totally ordered by the relation* $\preceq$ *("less or equally important than") and DPs are ordered accordingly:* $(\varphi \rightsquigarrow g) \preceq (\psi \rightsquigarrow g')$ *iff* $g \preceq g'$*.* $w : \mathcal{I} \cup \mathcal{S} \to]0, 1]$ *is a weight function on inhibitors and supports.*

The weight on a support/inhibitor of a DP is expressing an increased/decreased certainty degree about the fact that triggering this DP will lead to the achievement of its conclusion. We do not allow for supports or inhibitors of weight 0, since it would mean that there is no information about the supporting/inhibiting effects.

Reasoning about goal achievements

The first part of the process is a reasoning part: it consists in reasoning with the argumentation graphs that concern each goal in order to check what are the realized goals. This is

[1]For a simpler representation, the drawing of a BLFSW obeys the convention that *if no weight is given for a set of supports and inhibitors concerning the same DP then all weights are equal to* 1.

done by considering what is known: given a consistent knowledge base K, we first define a K-BLFSW as the BLFSW that is obtained when all what is known is K. More formally,

Definition 2 (K-BLFSW). *Given a consistent knowledge base K and a BLFSW $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$, a K-BLFSW associated to B is a tuple $(\mathcal{P}_K, \mathcal{I}_K, \mathcal{S}_K, pol, \preceq, w_K)$ where*

- $\mathcal{P}_K = \{(\varphi, g) \in \mathcal{P}$, *s.t.* $K \models \varphi\}$ *is the set of valid DPs in $\mathcal{P}$ (i.e., those whose reason φ holds in K).*
- $\mathcal{I}_K = \{(\varphi, p) \in \mathcal{I}$, *s.t.* $K \models \varphi$ *and* $p \in \mathcal{P}_K\}$ *is the set of* valid inhibitions *according to* K.
- $\mathcal{S}_K = \{(\varphi, p) \in \mathcal{S}$, *s.t.* $K \models \varphi$ *and* $p \in \mathcal{P}_K\}$ *is the set of* valid supports *according to* K.
- w_K *is the restriction of* w *on* $\mathcal{I}_K \cup \mathcal{S}_K$.

The DPs that are not inhibited in the K-BLFSW are the ones that are trusted, in order to know if a DP is inhibited, we have to compare the weights of its inhibitors and supports.

Definition 3. *Given a K-BLFSW $(\mathcal{P}_K, \mathcal{I}_K, \mathcal{S}_K, pol, \preceq, w_K)$, we define the activation level of $p \in \mathcal{P}_K$ as follows:*

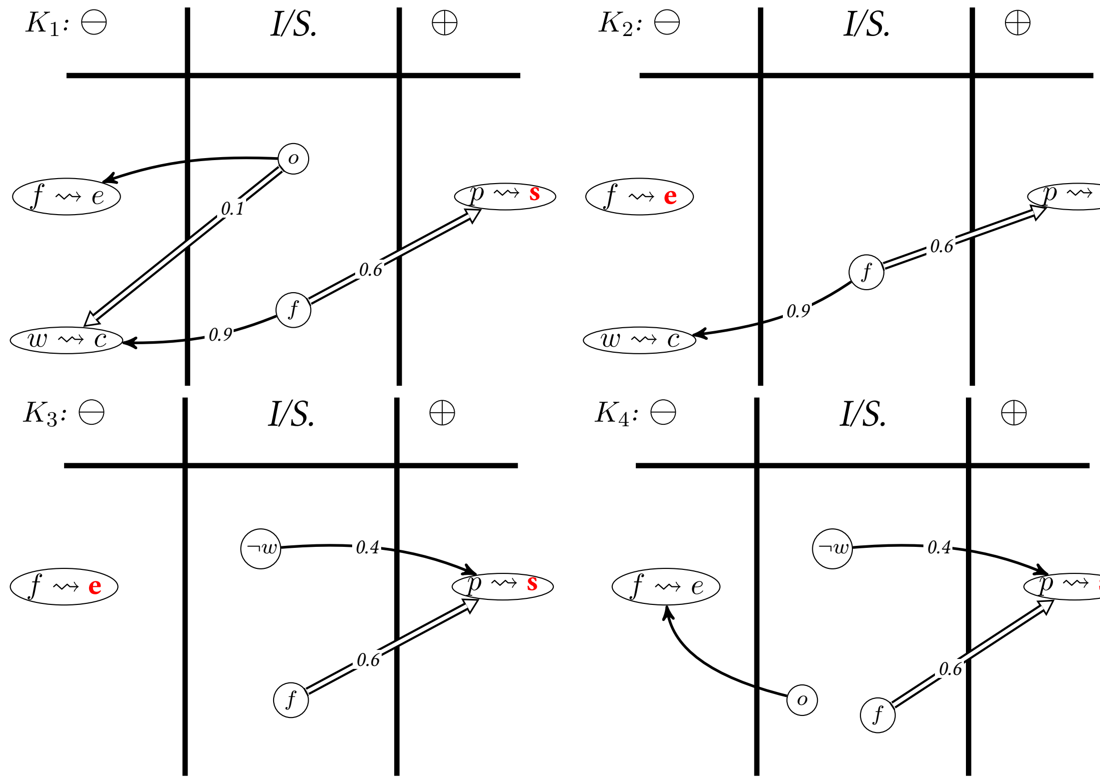
$$\alpha(p) = \sum_{s \in \mathcal{S}_K(p)} w_K(s,p) - \sum_{i \in \mathcal{I}_K(p)} w_K(i,p)$$

with $\mathcal{S}_K(p) = \{\psi \in \mathscr{L}_F | (\psi, p) \in \mathcal{S}_K\}, \mathcal{I}_K(p) = \{\psi \in \mathscr{L}_F | (\psi, p) \in \mathcal{I}_K\}$. *According to* $\alpha(p)$, *the DP p is either* inhibited *iff* $\alpha(p) < 0$, supported *iff* $\alpha(p) > 0$ *or* unaffected *iff* $\alpha(p) = 0$.

In other words, the weights of supports and inhibitors that concerns a given DP p are used to determine whether p is globally supported or inhibited or unaffected. The DP is said supported when supports are stronger than inhibitors, it is inhibited in the opposite case. When inhibitors and supports are equal they cancel each other.

Definition 4 (realized goals). *Given a K-BLFSW $B_K = (\mathcal{P}_K, \mathcal{I}_K, \mathcal{S}_K, pol, \preceq, w_K)$, a goal g in LIT_G is said to be* realized *wrt B_K if there is a DP in $\mathcal{P}_K$ that concludes g and that is not inhibited.*

Example 1 (cont.): *Let us consider* $K_1 = \{p \wedge w \wedge f \wedge o\}$, $K_2 = \{p \wedge w \wedge f\}$, $K_3 = \{p \wedge \neg w \wedge f\}$, $K_4 = \{p \wedge \neg w \wedge f \wedge o\}$. *The four corresponding BLFSWs are:*



In K_1, $f \rightsquigarrow e$ is inhibited (since it has only one inhibitor with a default weight of 1), $w \rightsquigarrow c$ is inhibited (since it has an inhibitor of weight 0.9 which is heavier than the weight 0.1 of its support) and $p \rightsquigarrow s$ is supported. Hence the only realized goals of K_1 is s. Similarly, we compute the realized goals of the other hotels: K_2 and K_3 have the same realized goals: e and s, K_4 has only one realized goal: s (since the $p \rightsquigarrow s$ is supported by a support that is heavier than its inhibitor).

Handling preferences

Once the reasoning step is done, we know what goals are realized in what alternative situation, then the second step consists in taking into account the preferences expressed in terms of importance and polarities of the goals. Hence, the definition of realized goals wrt a K-BLFSW allows us to compare alternatives according to the goals they achieve.

In [19], three decision rules called Pareto, Bipolar Possibility and Bipolar Leximin have been introduced. We have chosen to only translate the Bipolar Leximin rule in order to compare two alternatives.

Definition 5 (BiLexi decision rule). *Given a BLFSW $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$ and two alternatives described respectively by K and K' with their associated realized goals $\mathtt{R}$ and $\mathtt{R}'$, the Bipolar Leximin dominance relation $\succeq_{BiLexi}$ ($BiLexi$-*preferred to*) is s.t.: let $X_g^{pol} = \{g' \in X$ s.t. $g \simeq g'$ and $pol(g') = pol\}$ and $M = \max(\{g \in \mathtt{R} \cup \mathtt{R}'$ s.t. $|\mathtt{R}_g^{\oplus}| \neq |\mathtt{R}_g'^{\oplus}|$ or*

$|\mathtt{R}_g^{\ominus}| \neq |\mathtt{R}_g'^{\ominus}|\}, \preceq)$

$$\begin{array}{lll} K \succ_{BiLexi} K' & \textit{iff} & \left| \begin{array}{l} M \textit{ exists and} \\ |\mathtt{R}_M^{\oplus}| \geq |\mathtt{R}_M'^{\oplus}| \textit{ and } |\mathtt{R}_M^{\ominus}| \leq |\mathtt{R}_M'^{\ominus}| \end{array} \right. \\ K \simeq_{BiLexi} K' & \textit{iff} & M \textit{ does not exist} \end{array}$$

In other words, an alternative described by K is $BiLexi$-preferred to another one described by K' if there is a goal M such that the number of realized positive and negative goals at levels strictly more important than M are the same for K', but at the level M either the number of positive goals of K is greater than those of K' or the number of negative goals of K is lower than those of K'.

Example 1 (cont.): *The ranks of the hotels are:*

$$(K_1 \simeq_{BiLexi} K_4) \succ_{BiLexi} (K_2 \simeq_{BiLexi} K_3)$$

Note that in case of equality between two alternatives, the activation levels of the DPs that are justifying the goals achieved by the alternatives, can be used to choose between them.This means that goals are associated with two evaluations, one concerning their importance (that can be called utility when the goal is positive and disutility when it is negative, in our framework it is characterized by $\preceq$ and pol) and one concerning the certainty α_K about their realization in K, as defined below.

Definition 6 (Certainty of a realized goal)**.** *Given a BLFSW $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$ and an alternative described by K, for all goal g realized in B_K, the certainty associated to g is:*

$$\alpha_K(g) = \max_{\varphi \in \mathscr{L}_F, p = \varphi \rightsquigarrow g \in \mathcal{P}_K} \alpha(p)$$

In other words, the certainty associated to g corresponds to the maximum activation level of a DP concluding g.

Definition 7 (BW decision rule)**.** *Given a BLFSW $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$ and two alternatives described respectively by K and K' with their associated realized goals $\mathtt{R}$ and $\mathtt{R}'$, the BW dominance relation $\succeq_{BW}$ ($BiLexi\&Weight$-preferred to) is defined by: $K \succ_{BW} K'$ iff $K \succ_{BiLexi} K'$ or ($K \simeq_{BiLexi} K'$ and*

$$\exists M = \max(\{g \in \mathtt{R} \cup \mathtt{R}' \left| \begin{array}{l} \max_{g_1 \in \mathtt{R}_g^{\oplus}} \alpha_K(g_1) > \max_{g_2 \in \mathtt{R}_g'^{\oplus}} \alpha_{K'}(g_2) \textit{ or} \\ \max_{g_1 \in \mathtt{R}_g^{\ominus}} \alpha_K(g_1) < \max_{g_2 \in \mathtt{R}_g'^{\ominus}} \alpha_{K'}(g_2) \end{array} \right. \}, \preceq) \,)$$

In the previous definition, M is the highest important goal s.t. the maximum weight of a positive or a negative achieved goal of same priority for K and K' differs in favor of K i.e., either the maximum weight of positive achieved goals for K is strictly greater than the one for K' or the maximum weight of negative achieved goals for K is strictly lower than the one for K'.

Example 1 (cont.): *We get: $K_1 \succ_{BW} K_4 \succ_{BW} K_2 \succ_{BW} K_3$. Since in K_1, $p \rightsquigarrow s$ is supported and not attacked hence the activation level of $p \rightsquigarrow s$ is 0.6, while in K_4, $p \rightsquigarrow s$*

has an activation level of $0.6 - 0.4 = 0.2$ *which means that the achievement of swim is more certain in the situation described by* K_1 *than in the situation described by* K_4*. The same refinement is done to differentiate* K_2 *and* K_3*, their negative goal* e *is achieved with the same certainty while the positive goal* s *is more certainly achieved in* K_2 *than in* K_3*.*

11.3 Towards an automatic Explanation of a Possibilistic decision setting

As seen above, a BLFSW is a tool that enables the user to make explicit the decision setting. This paper aims at translating classical decision settings into BLFSW in order to give an automatic explanation to the utilities attached to decisions. In this section, we first write a reminder about Possibility Theory and Defaults, then we show how the decision principles, inhibitors and weights of a BLFSW can be interpreted in terms of possibility theory. This is done by following up the work of [46] in order to build DPs from uncertain knowledge expressed under the form of a possibility distribution on worlds and from preferences expressed as utilities associated to goals. In this process, a DP $\varphi \rightsquigarrow g$ is viewed as a defeasible rule saying that *if* φ *holds then a priori* g *is achieved*, and we explain how weighted inhibitions and supports can be defined according to this view. The third subsection show how to automatically build a BLFSW from possibilistic data.

Background on Possibility Theory and Defaults

In [43], possibility theory is introduced as a basis for qualitative decision theory. The author relate the expected pay-off $u(x)$ of a situation x to a preference relation $\preceq$ over situations s.t. $x \preceq y$ iff $u(x) \geq u(y)$. In presence of uncertainty, i.e., when situations are not precisely known, the belief state about what is the actual situation is represented by a possibility distribution π. The theory of possibility is a qualitative setting first introduced by Zadeh [209] and further developed by Dubois and Prade in [42]. It is qualitative in the sense that the only operations required are $\max$, $\min$ and order-reversing operations. However, numbers in the scale [0,1] are often used for convenience but the exact values of the numbers are not meaningful, it is only their order in the scale that is taken into account.

A possibility distribution π is used to compare the plausibility of situations: $\pi(x) \leq \pi(x')$ means that it is at least as plausible for x' to be the actual situation as for x to be it. $\pi(x) = 0$ means impossibility, $\pi(x) = 1$ means that x is unsurprising or normal. The state of total ignorance is represented by a possibility distribution where any situation is totally possible ($\forall x, \pi(x) = 1$). In order to reason on formula (hence sets of situations), two measures Π and N are defined: the possibility measure Π evaluates how unsurprising a formula is, hence $\Pi(\varphi) = 0$ means that φ is bound to be false. The necessity measure is its dual defined by $N(\varphi) = 1 - \Pi(\neg\varphi)$: $N(\varphi) = 1$ means that φ is bound to be true. N is defined from a possibility distribution π by: $N(\varphi) = min_{\omega \models \neg\varphi}(1 - \pi(\omega))$: a formula is all the more necessary as its counter models are less plausible.

In [44], the authors show that the utility of a decision d can be evaluated by combining the plausibilities $\pi(x)$ of the states x in which d is made and the utility $u(d(x))$ of the possible resulting state $d(x)$ after d, where $u(d(x))$ represents the satisfaction to be in the precise situation $d(x)$ (it is equal to the membership degree to the fuzzy set of preferred situations). The pessimistic criterion has been first introduced by Whalen [198] and leads to a pessimistic utility level of a decision d defined as follows: $u_{pes}(d) = \inf_{x \in X} \max(1 - \pi(x), u(d(x)))$. The optimistic criterion has been first proposed by Yager [205] and is defined by: $u_{op}(d) = \sup_{x \in X} \min(\pi(x), u(d(x)))$.

In possibilistic decision theory, the scales for possibilities and utilities are the same, hence, commensurable. In our proposal the commensurability of the two scales is not required: we do not aggregate possibilities and utilities, we rather use a kind of chance constrained approach [28, 99] in which they are dealt with separately.

Since a decision principle represents a defeasible reason to believe that some goal is achieved, we also need to recall some basics about handling defeasible rules in a possibilistic setting. A defeasible rule is a compact way to express a general rule without mentioning every exception to it. In a BLFSW the exceptions to a decision principle are its inhibitors. The conditional possibility measure denoted $\Pi(\varphi|\psi)$ is the possibility that φ holds in the worlds where ψ holds. It is related to the conditional possibility distribution as follows: $\Pi(\varphi|\psi) = \max_\omega \min(\Pi(\varphi|\omega), \pi(\omega|\psi))$. A default rule $a \rightsquigarrow b$ translates, in the possibility theory framework, into the constraint $\Pi(a \wedge b) > \Pi(a \wedge \neg b)$ which expresses that having b true is strictly more possible than having it false when a is true [15]. Note that the constraint $\Pi(a \wedge b) > \Pi(a \wedge \neg b)$ is equivalent to $N(b|a) > 0$. Hence, if we know a and we search for a conclusion which satisfies the constraint $N() > 0$ then a solution is b. In this sense, decision principles are related to chance constraints in quantitative optimization problem. In this article, we will use the min conditioning ($|_{min}$) since we are interested in qualitative decision problems, i.e.,

$$\pi(\omega \,|_{min}\, \varphi) = \left| \begin{array}{ll} 1 & \text{if } \omega \models \varphi \text{ and } \pi(\omega) = \Pi(\varphi); \\ \pi(\omega) & \text{if } \omega \models \varphi \text{ and } \pi(\omega) < \Pi(\varphi); \\ 0 & \text{if } \omega \not\models \varphi \end{array} \right.$$

Interpreting a BLFSW in Possibility Theory

This section is devoted to give an interpretation of Support/Inhibitor and strength of a DP in a possibilistic setting. This will allow the designer of a Decision System to move from one formalism to another in order to check the accuracy of his proposed model. In addition, Possibility theory is recognized as a theory taking into account uncertainty and qualitative reasoning, so showing that there is a translation from a possibilistic representation of uncertainty and preferences to a BLFSW increases the validity of this framework. The BLFSW is able to take into account the degree of certainty of a DP which is not possible in a plain BLF. Nevertheless the possibilistic meaning of a DP and an inhibitor in a BLFSW are the same as those found for a BLF in [46]. First, we restate Π-DP and Π-inhibitor definitions: in order to be well-defined a DP has to be informative, i.e., the DP $\varphi \rightsquigarrow g$ is well-defined if the necessity of the goal g increases when φ holds:

Definition 8 (Π-DP [46])**.** *Given a possibility measure* Π, *a* Π-*DP* $\varphi \rightsquigarrow g$ *is s.t.* $N(g|\varphi) > 0$

In other words, the DP is the piece of knowledge which increases the certainty that the goal is realized. In the same way we can interpret the notion of inhibitor and support in possibility theory: an inhibitor ψ makes the default rule $\varphi \rightsquigarrow g$ no more valid in such a way that we are no longer sure that g will be realized when φ and ψ hold together. More precisely, ψ can be defined as an inhibitor of $\varphi \rightsquigarrow g$ if when ψ holds, the necessity of g being achieved (which was previously > 0) is reduced to zero.

Definition 9 (Π-Inhibitor [46]). *Given a possibility measure Π, the pair (ψ, p) is a Π-Inhibitor of the DP $p = \varphi \rightsquigarrow g$ if $N(g|\varphi \wedge \psi) = 0$*

In contrast, the support increases the certainty of the default rule. So when the support ψ holds, we are more sure that g will be realized.

Definition 10 (Π-Support). *Given a possibility measure Π, the pair (ψ, p) is a Π-Support of the DP $p = \varphi \rightsquigarrow g$ if $N(g|\varphi \wedge \psi) > N(g|\varphi)$*

Moreover to complete the interpretation of a BLFSW in possibility theory we need to define the global strength $\alpha(p)$ of a DP p in possibilistic terms.

Definition 11 (Π-weight). *Given a possibility measure Π and a weight function w. w is a Π-weight function iff for all possible K-BLFSW ($\mathcal{P}_K$, $\mathcal{I}_K$, $\mathcal{S}_K$, pol, $\preceq$, w_K) where w_K is the restriction of w on $\mathcal{I}_K \cup \mathcal{S}_K$ and for all decision principles $p = \varphi \rightsquigarrow g, p' = \varphi' \rightsquigarrow g' \in \mathcal{P}_K$*

- $\alpha(p) < 0$ *iff* $N(g \,|\, \varphi \bigwedge_{\psi \in \mathcal{I}_K(p) \cup \mathcal{S}_K(p)} \psi) = 0$
- $\alpha(p) \geq 0$ *iff* $N(g \,|\, \varphi \bigwedge_{\psi \in \mathcal{I}_K(p) \cup \mathcal{S}_K(p)} \psi) > 0$
- $\alpha(p) \geq \alpha(p') \geq 0$ *iff* $N(g \,|\, \varphi \bigwedge_{\psi \in \mathcal{I}_K(p) \cup \mathcal{S}_K(p)} \psi) \geq N(g' \,|\, \varphi' \bigwedge_{\psi \in \mathcal{I}_K(p') \cup \mathcal{S}_K(p')} \psi) > 0$

with $\alpha(p)$ defined from w_K according to Definition 3.

In other words, the activation level $\alpha(p)$ of $p = \varphi \rightsquigarrow g$ defined in Definition 3 should reflect the certainty about the default rule $\varphi \rightsquigarrow g$ and should behave as stated in Definition 3: a negative activation level means that the default rule does not hold in presence of all its supports and inhibitors, a strictly positive one means that the goal is all the more likely to be achieved that the level is high, two distinct positive activation levels should be ranked according to the two necessities of the DPs. This last point will allow us to rank order alternatives more precisely. Using the definitions above we are now in position to define a Π-BLFSW.

Definition 12 (Π-BLFSW). *A BLFSW $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$ is a Π-BLFSW iff there exists a possibility distribution π over Ω and a utility function u on the set of goals LIT_G, s.t.*

- $\forall p = \varphi \rightsquigarrow g \in \mathcal{P}, u(g) \neq 0$

- $\forall g \in LIT_G, pol(g) = \oplus$ *iff* $u(g) > 0$
- $\forall g, g' \in LIT_G, g \preceq g'$ *iff* $u(g) \leq u(g')$
- *for all consistent knowledge base* K*,* $\forall g \in LIT_G$ *s.t.* $u(g) \neq 0, \forall \omega \in \Omega$*,* $\pi(g|\omega)$ *satisfies the constraints of Definitions 8, 9, 10 and 11*

Intuitively, in a Π-BLFSW the polarities and importances of the goals are based on a utility function and the weights on supports and inhibitors of DPs are consistent with the necessities of the default rules associated to DPs.

Thanks to this last definition, the designer can check whether her BLFSW is a Π-BLFSW hence whether it is consistent wrt a classical qualitative theory of uncertainty. If the possibility distribution does not seem realistic to the designer, she should modify the BLFSW (which summarizes it).

Remark 1. *It may happen that the agents want to distinguish between the strengths of two DPs* $p_1 = (\varphi_1 \rightsquigarrow g_1)$ *and* $p_2 = (\varphi_2 \rightsquigarrow g_2)$ *because she knows that* $N(g_1|\varphi_1) > N(g_2|\varphi_2)$*. In order to do that, she may use the notion of support, by adding a support* $s_1 = (\varphi_1, p_1)$*. In that case,* $\alpha_{p_1} = w(s_1)$ *is necessarily greater than* $\alpha_{p_2} = 0$*.*

The following proposition shows the relation between the weight associated to a goal in a K-BLFSW and its necessity to hold wrt this knowledge base K. The ranking on alternatives described by K based on the goals achieved in K is the same as the one obtained in a Π-BLFSW based on K.[2]

Proposition 1. *Given a* Π*-BLFSW* $B = (\mathcal{P}, \mathcal{I}, \mathcal{S}, pol, \preceq, w)$ *built on a possibility distribution* π *on the set of worlds* Ω *and on a utility function* u *on* LIT_G*.* B *is s.t. for all consistent knowledge bases* K, K' *and for any goals* g, g'*:*

- g *not realized wrt* B_K *iff* $N(g|K) = 0$ *or* $u(g) = 0$
- *if* g *realized wrt* B_K *and* g' *realized wrt* $B_{K'}$ *then* $\alpha_K(g) > \alpha_{K'}(g')$ *iff* $N(g|K) > N(g'|K')$

Proof (sketch) The first item follows from definitions 8 and 12, the second one from definitions 3 and 11. □

From Possibility theory to BLFSW: An Example

Building a BLFSW can be done in two independent steps: first define the DPs and their weighted relationships from a given possibility distribution, second use the utilities of goals in order to filter out DPs with goals of null utility and to rank the DPs in the BLFSW. Here,

ω	$\pi(\omega)$	$\Pi(s\|\omega)$	$\Pi(\neg s\|\omega)$	$\Pi(c\|\omega)$	$\Pi(\neg c\|\omega)$	$\Pi(e\|\omega)$	$\Pi(\neg e\|\omega)$
ω_1: $pwfo$	0.3	1	0	1	1	0.2	1
ω_2: $pwf\neg o$	0.3	1	0	0.8	1	1	0.4
ω_3: $pw\neg fo$	1	1	0	1	0.3	0	1
ω_4: $pw\neg f\neg o$	1	1	0	1	0.4	0	1
ω_5: $p\neg wfo$	0.2	1	0.3	0.2	1	0.2	1
ω_6: $p\neg wf\neg o$	0.2	1	0.3	0.1	1	1	0.4
ω_7: $p\neg w\neg fo$	0.4	1	1	0.8	1	0	1
ω_8: $p\neg w\neg f\neg o$	0.4	1	1	0	1	0	1
ω_9: $\neg pwfo$	0.3	0	1	1	1	0	1
ω_{10}: $\neg pwf\neg o$	0.3	0	1	0.8	1	1	0.4
ω_{11}: $\neg pw\neg fo$	1	0	1	1	0.3	0	1
ω_{12}: $\neg pw\neg f\neg o$	1	0	1	1	0.4	0	1
ω_{13}: $\neg p\neg wfo$	0.3	0	1	0.2	1	0	1
ω_{14}: $\neg p\neg wf\neg o$	1	0	1	0.1	1	1	0.4
ω_{15}: $\neg p\neg w\neg fo$	1	0	1	0.2	1	0	1
ω_{16}: $\neg p\neg w\neg f\neg o$	0.4	0	1	0	1	0	1

Table 11.1: Possibility distributions on worlds and goals

we only present the first step based on the knowledge of a possibility distribution over all possible worlds $\omega \in \Omega$ and the possibility for each goal to be true in each world (Table.11.1).

For each goal in LIT_G (here in $\{s, \neg s, c, \neg c, e, \neg e\}$), we check whether we can generate a DP concluding it by checking Definition 8, on all the conjunctive formulas that can be built, starting from formulas restricted to a single literal and adding new literals progressively. Let us consider the goal s when we know p, we have $N(s|p) = 1 - \Pi(\neg s|p) = 1 - \max_\omega(\min(\pi(\omega|p), \Pi(\neg s|\omega))) = 1 - 0.4 = 0.6 > 0$ hence $p_1 = p \rightsquigarrow s$ is a DP. If we suppose that we know w, $N(s|w) = 1 - \Pi(\neg s|w) = 1 - 1 = 0$ due to the world ω_{12} hence $s \rightsquigarrow w$ is not a DP. Let us look for supports and inhibitors, we have $N(s|p \wedge f) = 0.8 > N(s|p)$, so due to definition 10, $s_1 = (f, p_1)$ is a support. p_1 has also an inhibitor since adding $\neg w$ we get $N(s|p \wedge \neg w) = 0$. $N(s|p \wedge f \wedge \neg w) = 0.7$ thus $s_2 = (f \wedge \neg w, p_1)$ is also a support. Using the same process on the other two goals, we obtain $p_2 = w \rightsquigarrow c$, $N(c|w) = 0.6$, $i_2 = (f, p_2)$, $N(c|w \wedge f \wedge o) = 0$, $s_3 = (o, p_2)$, $N(c|w \wedge o) = 0.7$, $i_3 = (f \wedge o, p_2)$, $N(c|w \wedge f \wedge o) = 0$, $p_3 = (f, e)$, $N(e|f) = 0.6$ and $i_4 = (o, p_3)$, $N(e|f \wedge o) = 0$. Let us now focus on the weight assignments. The weights must satisfy all the constraints entailed by Definition 11. For instance, $w_{s_1} > w_{s_1} + w_{s_2} - w_{i_1} > 0$ and $0 \geq -w_{i_1}$. Note that if $N(s|p \wedge f \wedge \neg w) = N(s|p \wedge f) = 0.6$ then $w(f \wedge \neg w, P_1) = w(s|p \wedge \neg w)$. In that case the inhibitor $\neg w$ is cancelled by the support $f \wedge \neg w$. The possible assignments of weights are infinite, for instance the one given in Example 1: $w_{p_1} = 0.4$, $w_{s_1} = 0.6$, $w_{s_2} = 0$, $w_{i_1} = 0.4$ satisfies the constraints. Using the same process on the other two goals, for c we have $w_{p_2} + w_{s_3} - w_{i_3} \leq 0$, $w_{p_2} - w_{i_3} \leq 0$ and $w_{s_3} > 0$ for instance the one given in Example 1: $w_{i_3} = 0.9$, $w_{s_3} = w_{p_2} = 0.1$. So the decision problem defined by Table.11.1 is equivalent to the BLFSW of Example 1.

[2]This ranking can be obtained with the relations $\preceq_{BiLexi}$ or $\preceq_{BW}$.

11.4 Conclusion

This paper proposes an extension of the BLF of [47] in order to deal with supports and weights.The framework is a visual way to encode and explain a qualitative decision theory. The BLFSW's main benefit is to provide a representation that allows a user to clearly express the principles and utility levels which govern the decision process. In BLFSW, the decision is justified by the importance and polarities of the tangible results that are realized if the alternative is chosen, these results are also explained by the valid principles (not inhibited DPs) that apply in the situation. The ability to explain how the weights of inhibitors and supports of DPs are computed is one of the main result of this paper. This result is based on a procedure that builds a BLFSW from utilities and uncertain knowledge expressed in possibilistic terms.

Bibliography

[1] URL: https://openai.com/blog/openai-codex.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. “Contextual String Embeddings for Sequence Labeling”. In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018.* Ed. by Emily M. Bender, Leon Derczynski, and Pierre Isabelle. Association for Computational Linguistics, 2018, pp. 1638–1649. URL: https://www.aclweb.org/anthology/C18-1139/.

[3] Khalid Al-Khatib et al. “Cross-Domain Mining of Argumentative Text through Distant Supervision”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* San Diego, California: Association for Computational Linguistics, June 2016, pp. 1395–1404. DOI: 10.18653/v1/N16-1165. URL: https://aclanthology.org/N16-1165.

[4] Ahmed Alajrami and Nikolaos Aletras. “How does the pre-training objective affect what large language models learn about linguistic properties?” In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers).* Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 131–147. DOI: 10.18653/v1/2022.acl-short.16. URL: https://aclanthology.org/2022.acl-short.16.

[5] Jay Alammar. *Visualizing machine learning one concept at a time.* URL: https://jalammar.github.io/.

[6] Zeyuan Allen-Zhu and Yuanzhi Li. *Towards Understanding Ensemble, Knowledge Distillation and Self-Distillation in Deep Learning.* 2023. arXiv: 2012.09816 [cs.LG].

[7] L Amgoud, C Devred, and M.-C. Lagasquie-Schiex. “A constrained argumentation system for practical reasoning”. In: *Argumentation in Multi Agent Systems.* Springer, 2009, pp. 37–56.

[8] L. Amgoud and H. Prade. “Comparing decisions on the basis of a bipolar typology of arguments”. anglais. In: *Preferences and Similarities.* Ed. by Giacomo Della Riccia et al. Springer, 2008, pp. 249–264.

[9] L. Amgoud and S. Vesic. “A formal analysis of the role of argumentation in negotiation dialogues”. anglais. In: *Journal of Logic and Computation* 22 (2012), pp. 957–978.

[10] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: *CoRR* abs/1607.06450 (2016). arXiv: 1607.06450. URL: http://arxiv.org/abs/1607.06450.

[11] L.R. Bahl et al. “A tree-based statistical language model for natural language speech recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.7 (1989), pp. 1001–1008. doi: 10.1109/29.32278.

[12] Roy Bar-Haim et al. “Stance Classification of Context-Dependent Claims”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 251–261. url: https://aclanthology.org/E17-1024.

[13] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. “An introduction to argumentation semantics”. In: *Knowledge Eng. Review* 26 (Dec. 2011), pp. 365–410. doi: 10.1017/S0269888911000166.

[14] Eyal Ben-David, Nadav Oved, and Roi Reichart. *PADA: Example-based Prompt Learning for on-the-fly Adaptation to Unseen Domains*. 2022. arXiv: 2102.12206 [cs.CL].

[15] S. Benferhat, D. Dubois, and H. Prade. “Representing default rules in possibilistic logic”. In: *KR*. 1992, pp. 673–684.

[16] Luisa Bentivogli, Ido Dagan, and Danilo Giampiccolo. “The Fifth PASCAL Recognizing Textual Entailment Challenge”. In: *Proceedings of the Fifth PASCAL Recognizing Textual Entailment Challenge*. Association for Computational Linguistics. 2009, pp. 1–9.

[17] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. 2017. arXiv: 1607.04606 [cs.CL].

[18] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* (2017). doi: 10.1162/tacl_a_00051.

[19] JF. Bonnefon, D. Dubois, and H. Fargier. “An overview of bipolar qualitative decision rules”. In: *Preferences and Similarities*. vol 504, CISM. Springer, 2008, pp. 47–73.

[20] Thorsten Brants et al. “Large Language Models in Machine Translation”. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 858–867. url: https://aclanthology.org/D07-1090.

[21] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. url: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[22] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[23] Elena Cabrio and Serena Villata. “Detecting Bipolar Semantic Relations among Natural Language Arguments with Textual Entailment: a Study.” In: *Proceedings of the Joint Symposium on Semantic Processing. Textual Inference and Structures in Corpora.* Trento, Italy, Nov. 2013, pp. 24–32. URL: https://aclanthology.org/W13-3815.

[24] Elena Cabrio and Serena Villata. “Five Years of Argument Mining: A Data-Driven Analysis”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence.* IJCAI’18. Stockholm, Sweden: AAAI Press, 2018, pp. 5427–5433. ISBN: 9780999241127.

[25] Nicholas Carlini et al. *Extracting Training Data from Large Language Models.* 2021. arXiv: 2012.07805 [cs.CR].

[26] Rich Caruana. “Multitask Learning”. In: *Machine Learning* 28 (July 1997). DOI: 10.1023/A:1007379606734.

[27] Daniel Cer et al. “SemEval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation”. In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017).* 2017, pp. 1–14.

[28] A. Charnes and W. Cooper. “Chance-constrained programming”. In: *Management science* 6.1 (1959), pp. 73–79.

[29] Xiang Chen et al. “KnowPrompt: Knowledge-aware Prompt-tuning with Synergistic Optimization for Relation Extraction”. In: *Proceedings of the ACM Web Conference 2022.* ACM, 2022. DOI: 10.1145/3485447.3511998. URL: https://doi.org/10.1145%2F3485447.3511998.

[30] Zihang Chen et al. “Quora Question Pairs”. In: 2017.

[31] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL.* Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, 2014, pp. 1724–1734. DOI: 10.3115/v1/d14-1179. URL: https://doi.org/10.3115/v1/d14-1179.

[32] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways.* 2022. arXiv: 2204.02311 [cs.CL].

[33] Kevin Clark et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.* 2020. arXiv: 2003.10555 [cs.CL].

[34] Ronan Collobert et al. *Natural Language Processing (almost) from Scratch.* 2011. arXiv: 1103.0398 [cs.LG].

[35] Leyang Cui et al. *Template-Based Named Entity Recognition Using BART.* 2021. arXiv: 2106.01760 [cs.CL].

[36] Joe Davison, Joshua Feldman, and Alexander Rush. "Commonsense Knowledge Mining from Pretrained Models". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1173–1178. doi: 10.18653/v1/D19-1109. url: https://aclanthology.org/D19-1109.

[37] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of NAACL-HLT 2019*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. ACL, 2019, pp. 4171–4186.

[38] William B Dolan and Chris Brockett. "Automatically constructing a corpus of sentential paraphrases". In: *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. 2005, pp. 9–16.

[39] Mengnan Du et al. *Shortcut Learning of Large Language Models in Natural Language Understanding*. 2023. arXiv: 2208.11857 [cs.CL].

[40] Nan Du et al. *GLaM: Efficient Scaling of Language Models with Mixture-of-Experts*. 2022. arXiv: 2112.06905 [cs.CL].

[41] D. Dubois and H. Fargier. "Qualitative Bipolar Decision Rules: Toward More Expressive Settings". In: *Preferences and Decisions*. Springer, 2010, pp. 139–158.

[42] D. Dubois and H. Prade. *Possibility theory*. Plenum Press, New York, 1988.

[43] D. Dubois and H. Prade. "Possibility theory: qualitative and quantitative aspects". In: *Quantified Representation of Uncertainty and Imprecision*. Kluwer Academic Publishers, 1998, pp. 169–226.

[44] D. Dubois, H. Prade, and R. Sabbadin. "Decision-theoretic foundations of qualitative possibility theory". In: *Euro. J. of Operational Research* 128.3 (2001), pp. 459–478.

[45] Phan Minh Dung. "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games". In: *Artif. Intell.* 77 (1995), pp. 321–358.

[46] Florence Dupin De Saint Cyr and Romain Guillaume. "Analyzing a Bipolar Decision Structure through Qualitative Decision Theory". In: *KI - Künstliche Intelligenz* 31.1 (2017), pp. 53–62.

[47] Florence Dupin De Saint Cyr and Romain Guillaume. "Group Decision Making in a Bipolar Leveled Framework". anglais. In: *PRIMA*. 2017, pp. 34–52.

[48] Mihai Dusmanu, Elena Cabrio, and Serena Villata. "Argument Mining on Twitter: Arguments, Facts and Sources". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2317–2322. doi: 10.18653/v1/D17-1245. url: https://aclanthology.org/D17-1245.

[49] Rory Duthie, Katarzyna Budzynska, and Chris Reed. "Mining Ethos in Political Debate". English. In: *Computational Models of Argument.* Ed. by Pietro Baroni et al. Vol. 287. Frontiers in Artificial Intelligence and Applications. This research was supported in part by EPSRC in the UK under grant EP/M506497/1 and in part by the Polish National Science Centre under grant 2015/18/M/HS1/00620. Netherlands: IOS Press, 2016, pp. 299–310. ISBN: 9781614996859. DOI: 10.3233/978-1-61499-686-6-299.

[50] Steffen Eger, Johannes Daxenberger, and Iryna Gurevych. *Neural End-to-End Learning for Computational Argumentation Mining.* 2017. arXiv: 1704.06104 [cs.CL].

[51] William Fedus, Barret Zoph, and Noam Shazeer. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.* 2022. arXiv: 2101.03961 [cs.LG].

[52] William A. Gale and Geoffrey Sampson. "Good-Turing Frequency Estimation Without Tears". In: *J. Quant. Linguistics* 2 (1995), pp. 217–237. URL: https://api.semanticscholar.org/CorpusID:46217277.

[53] Jianfeng Gao and Chin-Yew Lin. "Introduction to the Special Issue on Statistical Language Modeling". In: *ACM Transactions on Asian Language Information Processing* 3.2 (2004), 87–93. ISSN: 1530-0226. DOI: 10.1145/1034780.1034781. URL: https://doi.org/10.1145/1034780.1034781.

[54] Tianyu Gao, Adam Fisch, and Danqi Chen. "Making Pre-trained Language Models Better Few-shot Learners". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).* Online: Association for Computational Linguistics, Aug. 2021, pp. 3816–3830. DOI: 10.18653/v1/2021.acl-long.295. URL: https://aclanthology.org/2021.acl-long.295.

[55] Jean-Luc Gauvain et al. "Estimation of probabilities from Sparse data for the language model component of a speech recognizer". In: (Feb. 2003).

[56] F.A. Gers and J. Schmidhuber. "Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium.* Vol. 3. 2000, 189–194 vol.3. DOI: 10.1109/IJCNN.2000.861302.

[57] Felix Alexander Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning Precise Timing with LSTM Recurrent Networks". In: *J. Mach. Learn. Res.* 3 (2003), pp. 115–143.

[58] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. *Speech Recognition with Deep Recurrent Neural Networks.* 2013. arXiv: 1303.5778 [cs.NE].

[59] Ivan Habernal and Iryna Gurevych. "Argumentation Mining in User-Generated Web Discourse". In: *Computational Linguistics* 43.1 (Apr. 2017), pp. 125–179. doi: 10.1162/COLI_a_00276. url: https://aclanthology.org/J17-1004.

[60] Ivan Habernal et al. "The Argument Reasoning Comprehension Task: Identification and Reconstruction of Implicit Warrants". In: *Proceedings of NAACL-HLT 2018*. ACL, 2018, pp. 1930–1940.

[61] Shohreh Haddadan, Elena Cabrio, and Serena Villata. "Yes, we can! Mining Arguments in 50 Years of US Presidential Campaign Debates". In: *Proceedings of ACL 2019*. ACL, 2019, pp. 4684–4690.

[62] Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. *WARP: Word-level Adversarial ReProgramming*. 2021. arXiv: 2101.00121 [cs.CL].

[63] Xu Han et al. *PTR: Prompt Tuning with Rules for Text Classification*. 2021. arXiv: 2105.11259 [cs.CL].

[64] Martin Haspelmath. "Word Classes and Parts of Speech". In: Dec. 2001, pp. 16538–16545. isbn: 9780080430768. doi: 10.1016/B0-08-043076-7/02959-4.

[65] Adi Haviv, Jonathan Berant, and Amir Globerson. "BERTese: Learning to Speak to BERT". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Apr. 2021, pp. 3618–3623. doi: 10.18653/v1/2021.eacl-main.316. url: https://aclanthology.org/2021.eacl-main.316.

[66] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. url: http://arxiv.org/abs/1512.03385.

[67] Danny Hernandez et al. *Scaling Laws and Interpretability of Learning from Repeated Data*. 2022. arXiv: 2205.10487 [cs.LG].

[68] Christopher Hidey et al. "Analyzing the Semantic Types of Claims and Premises in an Online Persuasive Forum". In: *Proceedings of ArgMining@EMNLP 2017*. ACL, 2017, pp. 11–21.

[69] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[70] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL].

[71] Ari Holtzman et al. *The Curious Case of Neural Text Degeneration*. 2020. arXiv: 1904.09751 [cs.CL].

[72] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. doi: 10.1073/pnas.79.8.2554.

[73] Ebuka Ibeke et al. “Extracting and Understanding Contrastive Opinion through Topic Relevant Sentences”. In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, Nov. 2017, pp. 395–400. URL: https://aclanthology.org/I17-2067.

[74] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[75] Frederick Jelinek. *Statistical Methods for Speech Recognition*. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262100665.

[76] Zhengbao Jiang et al. “How Can We Know What Language Models Know?” In: *Transactions of the Association for Computational Linguistics* 8 (July 2020), pp. 423–438. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00324. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf. URL: https://doi.org/10.1162/tacl_a_00324.

[77] Zhengbao Jiang et al. “How Can We Know What Language Models Know?” In: *Transactions of the Association for Computational Linguistics* 8 (July 2020), pp. 423–438. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00324. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf. URL: https://doi.org/10.1162/tacl_a_00324.

[78] Mandar Joshi et al. *SpanBERT: Improving Pre-training by Representing and Predicting Spans*. 2020. arXiv: 1907.10529 [cs.CL].

[79] Biing-Hwang Juang and Lawrence R. Rabiner. “Mixture autoregressive hidden Markov models for speaker independent isolated word recognition”. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 1986, Tokyo, Japan, April 7-11, 1986*. IEEE, 1986, pp. 41–44.

[80] Vijay Kanade. *What is reinforcement learning? working, algorithms, and uses*. 2022. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/.

[81] Nikhil Kandpal, Eric Wallace, and Colin Raffel. *Deduplicating Training Data Mitigates Privacy Risks in Language Models*. 2022. arXiv: 2202.06539 [cs.CR].

[82] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG].

[83] Daniel Khashabi et al. “UNIFIEDQA: Crossing Format Boundaries with a Single QA System”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1896–1907. DOI: 10.18653/v1/2020.findings-emnlp.171. URL: https://aclanthology.org/2020.findings-emnlp.171.

[84] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. doi: 10.1073/pnas.1611835114. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114. url: https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

[85] Stefan Kombrink et al. “Recurrent Neural Network Based Language Modeling in Meeting Recognition.” In: Aug. 2011, pp. 2877–2880. doi: 10.21437/Interspeech.2011-720.

[86] Kowsari et al. “Text Classification Algorithms: A Survey”. In: *Information* 10.4 (2019), p. 150. doi: 10.3390/info10040150. url: https://doi.org/10.3390%2Finfo10040150.

[87] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. arXiv: 1808.06226 [cs.CL].

[88] Sawan Kumar and Partha Talukdar. *Reordering Examples Helps during Priming-based Few-Shot Learning*. 2021. arXiv: 2106.01751 [cs.CL].

[89] Tatsuki Kuribayashi et al. “An Empirical Study of Span Representation in Argumentation Structure Parsing”. In: *Proceedings of ACL*. ACL, 2019, pp. 4691–4698.

[90] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: 1909.11942 [cs.CL].

[91] Hugo Laurençon et al. *The BigScience ROOTS Corpus: A 1.6TB Composite Multilingual Dataset*. 2023. arXiv: 2303.03915 [cs.CL].

[92] Hung-yi Lee, Shang-Wen Li, and Thang Vu. “Meta Learning for Natural Language Processing: A Survey”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 666–684. doi: 10.18653/v1/2022.naacl-main.49. url: https://aclanthology.org/2022.naacl-main.49.

[93] Katherine Lee et al. “Deduplicating Training Data Makes Language Models Better”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8424–8445. doi: 10.18653/v1/2022.acl-long.577. url: https://aclanthology.org/2022.acl-long.577.

[94] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL].

[95] Hector Levesque, Ernest Davis, and Leora Morgenstern. “The winograd schema challenge”. In: *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer. 2011.

[96] Ran Levy et al. “Context Dependent Claim Detection”. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University and ACL, Aug. 2014, pp. 1489–1500.

[97] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].

[98] Irene Li. *Elmo in practice*. 2019. URL: https://ireneli.eu/2018/12/17/elmo-in-practice/.

[99] Pu Li, Harvey Arellano-Garcia, and Günter Wozny. “Chance constrained programming approach to process optimization under uncertainty”. In: *Computers & chemical engineering* 32.1-2 (2008), pp. 25–45.

[100] Qi Li, Tianshi Li, and Baobao Chang. “Discourse Parsing with Attention-based Hierarchical Neural Networks”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: ACL, 2016, pp. 362–371. DOI: 10.18653/v1/D16-1035. URL: https://aclanthology.org/D16-1035.

[101] Xiaoya Li et al. “A Unified MRC Framework for Named Entity Recognition”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 5849–5859. DOI: 10.18653/v1/2020.acl-main.519. URL: https://aclanthology.org/2020.acl-main.519.

[102] Marco Lippi and Paolo Torroni. “Argument Mining from Speech: Detecting Claims in Political Debates”. In: *Proceedings of AAAI 2016*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 2979–2985.

[103] Marco Lippi and Paolo Torroni. “MARGOT: A web server for argumentation mining”. In: *Expert Syst. Appl.* 65 (2016), pp. 292–303.

[104] Jiachang Liu et al. *What Makes Good In-Context Examples for GPT-3?* 2021. arXiv: 2101.06804 [cs.CL].

[105] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. 2021. arXiv: 2107.13586 [cs.CL].

[106] Pengfei Liu et al. “Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. In: *CoRR* abs/2107.13586 (2021).

[107] Pengfei Liu et al. “Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. In: *ACM Comput. Surv.* 55.9 (2023). ISSN: 0360-0300. DOI: 10.1145/3560815. URL: https://doi.org/10.1145/3560815.

[108] Xiao Liu et al. *GPT Understands, Too*. 2021. arXiv: 2103.10385 [cs.CL].

[109] Xiao Liu et al. “Self-Supervised Learning: Generative or Contrastive”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2023), pp. 857–876. doi: 10.1109/TKDE.2021.3090866.

[110] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019. arXiv: 1907.11692 [cs.CL].

[111] Yao Lu et al. *Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity.* 2022. arXiv: 2104.08786 [cs.CL].

[112] Tobias Mayer, Elena Cabrio, and Serena Villata. “Transformer-Based Argument Mining for Healthcare Applications”. In: *Proceedings of ECAI 2020.* IOS Press, 2020, pp. 2108–2115.

[113] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133.

[114] Stefano Menini et al. “Never Retreat, Never Retract: Argumentation Analysis for Political Speeches”. In: *Proceedings of AAAI 2018* 32.1 (2018).

[115] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality.* 2013. arXiv: 1310.4546 [cs.CL].

[116] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2.* NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119.

[117] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 [cs.CL].

[118] Tomas Mikolov et al. “Recurrent neural network based language model”. In: *Interspeech.* 2010. url: https://api.semanticscholar.org/CorpusID:17048224.

[119] Swaroop Mishra et al. *Cross-Task Generalization via Natural Language Crowdsourcing Instructions.* 2022. arXiv: 2104.08773 [cs.CL].

[120] Raquel Mochales and Marie-Francine Moens. “Argumentation Mining”. In: *Artificial Intelligence and Law* 19.1 (2011), pp. 1–22. doi: 10.1007/s10506-010-9104-x.

[121] Marie-Francine Moens et al. “Automatic Detection of Arguments in Legal Texts”. In: *Proceedings of ICAIL 2007.* Stanford, California: ACM, 2007, pp. 225–230. isbn: 9781595936806.

[122] Umer Mushtaq and Jérémie Cabessa. “Argument Classification with BERT Plus Contextual, Structural and Syntactic Features as Text”. In: *Neural Information Processing.* Ed. by Mohammad Tanveer et al. Singapore: Springer Nature Singapore, 2023, pp. 622–633. isbn: 978-981-99-1639-9.

[123] Nona Naderi and Graeme Hirst. "Argumentation Mining in Parliamentary Discourse". In: *Principles and Practice of Multi-Agent Systems - International Workshops: IWEC 2014, Gold Coast, QLD, Australia, December 1-5, 2014, and CMNA XV and IWEC 2015, Bertinoro, Italy, October 26, 2015, Revised Selected Papers*. Ed. by Matteo Baldoni et al. Vol. 9935. Lecture Notes in Computer Science. Springer, 2015, pp. 16–25. DOI: 10.1007/978-3-319-46218-9_2. URL: https://doi.org/10.1007/978-3-319-46218-9_2.

[124] Anindya Naskar. *Naive Bayes algorithm in machine learning with python*. 2022. URL: https://thinkinfi.com/naive-bayes-algorithm-in-machine-learning-with-python/.

[125] Huy V. Nguyen and Diane J. Litman. "Argument Mining for Improving the Automated Scoring of Persuasive Essays". In: *AAAI Conference on Artificial Intelligence*. 2018.

[126] Vlad Niculae, Joonsuk Park, and Claire Cardie. "Argument Mining with Structured SVMs and RNNs". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 985–995. DOI: 10.18653/v1/P17-1091. URL: https://aclanthology.org/P17-1091.

[127] Vlad Niculae, Joonsuk Park, and Claire Cardie. "Argument Mining with Structured SVMs and RNNs". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Ed. by Regina Barzilay and Min-Yen Kan. ACL, 2017, pp. 985–995. DOI: 10.18653/v1/P17-1091. URL: https://doi.org/10.18653/v1/P17-1091.

[128] Christopher Olah. *Understanding LSTM networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[129] Departement Operationnelle et al. "A Neural Probabilistic Language Model". In: (Oct. 2001).

[130] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[131] Jungyeul Park et al. "Second-Order Belief Hidden Markov Models". In: *Belief Functions: Theory and Applications*. Ed. by Fabio Cuzzolin. Cham: Springer International Publishing, 2014, pp. 284–293. ISBN: 978-3-319-11191-9.

[132] Andreas Peldszus and Manfred Stede. "From Argument Diagrams to Argumentation Mining in Texts: A Survey". In: *Int. J. Cogn. Informatics Nat. Intell.* 7 (2013), pp. 1–31.

[133] Andreas Peldszus and Manfred Stede. "Joint prediction in MST-style discourse parsing for argumentation mining". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 938–948. DOI: 10.18653/v1/D15-1110. URL: https://aclanthology.org/D15-1110.

[134] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[135] Matthew E. Peters et al. “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: ACL, June 2018, pp. 2227–2237.

[136] Fabio Petroni et al. *Language Models as Knowledge Bases?* 2019. arXiv: 1909.01066 [cs.CL].

[137] Fabio Petroni et al. “Language Models as Knowledge Bases?” In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2463–2473. doi: 10.18653/v1/D19-1250. url: https://aclanthology.org/D19-1250.

[138] Peter Potash, Alexey Romanov, and Anna Rumshisky. “Here’s My Point: Joint Pointer Architecture for Argument Mining”. In: *Proceedings of EMNLP 2017*. ACL, 2017, pp. 1364–1373.

[139] Guanghui Qin and Jason Eisner. “Learning How to Ask: Querying LMs with Mixtures of Soft Prompts”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 5203–5212. doi: 10.18653/v1/2021.naacl-main.410. url: https://aclanthology.org/2021.naacl-main.410.

[140] Shilin Qiu et al. “Adversarial attack and defense technologies in natural language processing: A survey”. In: *Neurocomputing* 492 (2022), pp. 278–307. issn: 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.04.020. url: https://www.sciencedirect.com/science/article/pii/S0925231222003861.

[141] Lawrence R. Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proc. IEEE* 77.2 (1989), pp. 257–286.

[142] Alec Radford et al. “Improving language understanding by generative pre-training”. In: *OpenAI* (2018).

[143] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: *OpenAI* (2019).

[144] Jack W. Rae et al. *Scaling Language Models: Methods, Analysis & Insights from Training Gopher*. 2022. arXiv: 2112.11446 [cs.CL].

[145] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG].

[146] I. Rahwan et al. “Argumentation-based negotiation”. In: *Knowledge Engineering Review* 18.4 (2003), pp. 343–375.

[147] H. Raiffa. *Decision Analysis: Introductory lectures on choices under uncertainty.* Reading, Massachusetts: Addison-Wesley, 1970.

[148] Ravish Raj. *Supervised, unsupervised and semi-supervised learning with real-life USE-CASE.* 2021. URL: https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning.

[149] J. Raz. *Practical reasoning.* Oxford University Press, 1978.

[150] Ruty Rinott et al. “Show Me Your Evidence - an Automatic Method for Context Dependent Evidence Detection”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 440–450. DOI: 10.18653/v1/D15-1050. URL: https://aclanthology.org/D15-1050.

[151] A. J. Robinson and Frank Fallside. *The Utility Driven Dynamic Error Propagation Network.* Tech. rep. CUED/F-INFENG/TR.1. Cambridge, UK: Engineering Department, Cambridge University, 1987.

[152] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[153] R. Rosenfeld. “Two decades of statistical language modeling: where do we go from here?” In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278. DOI: 10.1109/5.880083.

[154] Sebastian Ruder et al. “Transfer Learning in Natural Language Processing”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials.* Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 15–18. DOI: 10.18653/v1/N19-5004. URL: https://aclanthology.org/N19-5004.

[155] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[156] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108. URL: http://arxiv.org/abs/1910.01108.

[157] Victor Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization.* 2022. arXiv: 2110.08207 [cs.LG].

[158] Timo Schick, Helmut Schmid, and Hinrich Schütze. "Automatically Identifying Words That Can Serve as Labels for Few-Shot Text Classification". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 5569–5578. DOI: 10.18653/v1/2020.coling-main.488. URL: https://aclanthology.org/2020.coling-main.488.

[159] Timo Schick and Hinrich Schütze. *Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference*. 2021. arXiv: 2001.07676 [cs.CL].

[160] Timo Schick and Hinrich Schütze. *Few-Shot Text Generation with Pattern-Exploiting Training*. 2021. arXiv: 2012.11926 [cs.CL].

[161] Timo Schick and Hinrich Schütze. *It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners*. 2021. arXiv: 2009.07118 [cs.CL].

[162] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. DOI: https://doi.org/10.1016/j.neunet.2014.09.003.

[163] Catherine D. Schuman et al. "A Survey of Neuromorphic Computing and Neural Networks in Hardware". In: *CoRR* abs/1705.06963 (2017). arXiv: 1705.06963. URL: http://arxiv.org/abs/1705.06963.

[164] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://aclanthology.org/P16-1162.

[165] Murray Shanahan. *Talking About Large Language Models*. 2023. arXiv: 2212.03551 [cs.CL].

[166] Taylor Shin et al. *AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts*. 2020. arXiv: 2010.15980 [cs.CL].

[167] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. 2013, pp. 1631–1642.

[168] Swapna Somasundaran and Janyce Wiebe. "Recognizing Stances in Online Debates". In: *Proceedings of ACL/IJCNLP 2009*. ACL, 2009, pp. 226–234.

[169] Kaitao Song et al. "MPNet: Masked and Permuted Pre-training for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 16857–16867. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/c3a690be93aa602ee2dc0ccab5b7b67e-Paper.pdf.

[170] Yi Song et al. “Applying Argumentation Schemes for Essay Scoring”. In: *Proceedings of ArgMining@ACL 2014*. ACL, 2014, pp. 69–78.

[171] Christian Stab and Iryna Gurevych. “Parsing Argumentation Structures in Persuasive Essays”. In: *Computational Linguistics* 43.3 (2017), pp. 619–659.

[172] Andreas Stolcke. “SRILM - an extensible language modeling toolkit”. In: *Interspeech*. 2002. URL: https://api.semanticscholar.org/CorpusID:1988103.

[173] Zhiqing Sun et al. “MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Ed. by Dan Jurafsky et al. Association for Computational Linguistics, 2020, pp. 2158–2170. DOI: 10.18653/v1/2020.acl-main.195. URL: https://doi.org/10.18653/v1/2020.acl-main.195.

[174] Chenhao Tan et al. “Winning Arguments: Interaction Dynamics and Persuasion Strategies in Good-Faith Online Discussions”. In: *Proceedings of WWW 2016*. ACM, 2016, pp. 613–624. ISBN: 9781450341431.

[175] Ruixiang Tang et al. *Does Synthetic Data Generation of LLMs Help Clinical Text Mining?* 2023. arXiv: 2303.04360 [cs.CL].

[176] Ross Taylor et al. *Galactica: A Large Language Model for Science*. 2022. arXiv: 2211.09085 [cs.CL].

[177] Harish Tayyar Madabushi, Elena Kochkina, and Michael Castelle. “Cost-Sensitive BERT for Generalisable Sentence Classification on Imbalanced Data”. In: *Proceedings of the Second Workshop on Natural Language Processing for Internet Freedom: Censorship, Disinformation, and Propaganda*. Hong Kong, China: ACL, Nov. 2019, pp. 125–134.

[178] A. Tchangani, Y. Bouzarour-Amokrane, and F. Pérès. “Evaluation Model in Decision Analysis: Bipolar Approach”. In: *Informatica* 23.3 (2012), pp. 461–485.

[179] Arsene Fansi Tchango et al. *DDXPlus: A New Dataset For Automatic Medical Diagnosis*. 2022. arXiv: 2205.09148 [cs.CL].

[180] Milagro Teruel et al. “Increasing Argument Annotation Reproducibility by Using Inter-annotator Agreement to Improve Guidelines”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://aclanthology.org/L18-1640.

[181] Simone Teufel. “Towards Discipline-Independent Argumentative Zoning : Evidence from Chemistry and Computational Linguistics”. In: 2009.

[182] Trias Thireou and Martin Reczko. “Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4.3 (2007), pp. 441–446. DOI: 10.1109/tcbb.2007.1015.

[183] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models.* 2023. arXiv: 2302.13971 [cs.CL].

[184] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of NIPS 2017.* Curran Associates Inc., 2017, pp. 6000–6010. isbn: 9781510860964.

[185] Maria Paz Garcia Villalba and Patrick Saint-Dizier. “Some Facets of Argument Mining for Opinion Analysis”. In: *Comma.* 2012.

[186] Marilyn Walker et al. “A Corpus for Research on Deliberation and Debate”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12).* Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 812–817. url: http://www.lrec-conf.org/proceedings/lrec2012/pdf/1078_Paper.pdf.

[187] D. Walton. *Argumentation schemes for presumptive reasoning.* (first edition 1996) Routledge, 2013.

[188] Alex Wang et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP.* 2018, pp. 353–355.

[189] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.* 2019. arXiv: 1804.07461 [cs.CL].

[190] Jindong Wang et al. *On the Robustness of ChatGPT: An Adversarial and Out-of-distribution Perspective.* 2023. arXiv: 2302.12095 [cs.AI].

[191] Thomas Wang et al. *What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization?* 2022. arXiv: 2204.05832 [cs.CL].

[192] Wenhui Wang and Baobao Chang. “Graph-based Dependency Parsing with Bidirectional LSTM”. In: *Proceedings of ACL 2016.* ACL, 2016, pp. 2306–2315.

[193] Wenhui Wang and Baobao Chang. “Graph-based Dependency Parsing with Bidirectional LSTM”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Berlin, Germany: ACL, 2016, pp. 2306–2315. doi: 10.18653/v1/P16-1218. url: https://aclanthology.org/P16-1218.

[194] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. “Neural Network Acceptability Judgments”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers).* 2018, pp. 567–574.

[195] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* 2023. arXiv: 2201.11903 [cs.CL].

[196] Jason Wei et al. *Emergent Abilities of Large Language Models.* 2022. arXiv: 2206.07682 [cs.CL].

[197] Paul J. Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural Networks* 1.4 (1988), pp. 339–356. DOI: https://doi.org/10.1016/0893-6080(88)90007-X.

[198] T. Whalen. “Decision making under uncertainty with various assumptions about available information”. In: *IEEE Trans. on Systems, Man and Cybern.* 14.6 (1984), pp. 888–900.

[199] John Wieting and Douwe Kiela. “No Training Required: Exploring Random Encoders for Sentence Classification”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net, 2019. URL: https://openreview.net/forum?id=BkgPajAcY7.

[200] Adina Williams, Nikita Nangia, and Samuel Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers).* New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: http://aclweb.org/anthology/N18-1101.

[201] Michael J Wooldridge. *Reasoning about rational agents.* MIT press, 2000.

[202] BigScience Workshop et al. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model.* 2023. arXiv: 2211.05100 [cs.CL].

[203] Shijie Wu et al. *BloombergGPT: A Large Language Model for Finance.* 2023. arXiv: 2303.17564 [cs.LG].

[204] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *ArXiv* abs/1609.08144 (2016).

[205] R. Yager. “Possibilistic decision making”. In: *IEEE Transactions on Systems, Man and Cybernetics* 9 (1979), pp. 388–392.

[206] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding.* 2020. arXiv: 1906.08237 [cs.CL].

[207] Wenpeng Yin, Jamaal Hay, and Dan Roth. “Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3914–3923. DOI: 10.18653/v1/D19-1404. URL: https://aclanthology.org/D19-1404.

[208] Weizhe Yuan, Graham Neubig, and Pengfei Liu. *BARTScore: Evaluating Generated Text as Text Generation.* 2021. arXiv: 2106.11520 [cs.CL].

[209] L.A. Zadeh. *Fuzzy Sets as a Basis for a Theory of Possibility.* Memorandum: Electronics Research Laboratory. U. of California, 1977.

[210] Daochen Zha et al. *Data-centric Artificial Intelligence: A Survey*. 2023. arXiv: 2303.10158 [cs.LG].

[211] ChengXiang Zhai. "Statistical Language Models for Information Retrieval". In: *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Tutorial Abstracts*. Rochester, New York: Association for Computational Linguistics, Apr. 2007, pp. 3–4. URL: https://aclanthology.org/N07-5002.

[212] ChengXiang Zhai. "Statistical Language Models for Information Retrieval A Critical Review". In: *Found. Trends Inf. Retr.* 2.3 (2008), 137–213. ISSN: 1554-0669. DOI: 10.1561/1500000008. URL: https://doi.org/10.1561/1500000008.

[213] Susan Zhang et al. *OPT: Open Pre-trained Transformer Language Models*. 2022. arXiv: 2205.01068 [cs.CL].

[214] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].

[215] Zexuan Zhong, Dan Friedman, and Danqi Chen. *Factual Probing Is [MASK]: Learning vs. Learning to Recall*. 2021. arXiv: 2104.05240 [cs.CL].

[216] Ce Zhou et al. *A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT*. 2023. arXiv: 2302.09419 [cs.AI].

[217] Kaiyang Zhou et al. "Domain Generalization: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), pp. 1–20. DOI: 10.1109/tpami.2022.3195549. URL: https://doi.org/10.1109%2Ftpami.2022.3195549.

[218] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. arXiv: 1506.06724 [cs.CV].