DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

French AXA Insurance Word Embeddings

Effects of Fine-tuning BERT and Camembert on AXA France's data

HEND ZOUARI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Title

French AXA Insurance Word Embeddings: Effects of Fine-tuning BERT and Camembert on AXA France's data

Authors

Hend ZOUARI
55 Rue Navier 75017
Paris
France
KTH Royal Institute of Technology
Double Degree with Telecom Paris
zouari@kth.se

Axa France: Insurance Company

Paris, France
313 Terrasse de l'Arche, 92000 Nanterre

Degree project subject and program

Degree Project in Computer Science and Engineering, specializing in Machine Learning, Second Cycle
Double Degree : Master's program in Computer Science)

Examiner

Pontus Johnson
TEKNIKRINGEN 33
Stockholm 10044 Stockholm
Sweden
KTH Royal Institute of Technology
pontusj@kth.se

Supervisor

Robert Lagerström
TEKNIKRINGEN 33
Stockholm 10044 Stockholm
Sweden
KTH Royal Institute of Technology
robertl@kth.se

Date

25/09/2020

Abstract

We explore in this study the different Natural Language Processing state-of-the art technologies that allow transforming textual data into numerical representation. We go through the theory of the existing traditional methods as well as the most recent ones. This thesis focuses on the recent advances in Natural Language processing being developed upon the Transfer model. One of the most relevant innovations was the release of a deep bidirectional encoder called BERT that broke several state of the art results. BERT utilises Transfer Learning to improve modelling language dependencies in text. BERT is used for several different languages, other specialized model were released like the french BERT: Camembert. This thesis compares the language models of these different pre-trained models and compares their capability to insure a domain adaptation. Using the multilingual and the french pre-trained version of BERT and a dataset from AXA France's emails, clients' messages, legal documents, insurance documents containing over 60 million words. We fine-tuned the language models in order to adapt them on the Axa insurance's french context to create a French AXA Insurance BERT model. We evaluate the performance of this model on the capability of the language model of predicting a masked token based on the context. BERT proves to perform better : modelling better the french AXA's insurance text without fine-tuning than Camembert. However, with this small amount of data, Camembert is more capable of adaptation to this specific domain of insurance.

Keywords

NLP, Language model, Word embedding, BERT, camemBERT

Svenskt abstract

I denna studie undersöker vi de senaste teknologierna för Natural Language Processing, som gör det möjligt att omvandla textdata till numerisk representation. Vi går igenom teorin om befintliga traditionella metoder såväl som de senaste. Denna avhandling fokuserar på de senaste framstegen inom bearbetning av naturliga språk som utvecklats med hjälp av överföringsmodellen. En av de mest relevanta innovationerna var lanseringen av en djup dubbelriktad kodare som heter BERT som bröt flera toppmoderna resultat. BERT använder Transfer Learning för att förbättra modelleringsspråkberoenden i text. BERT används för flera olika språk, andra specialmodeller släpptes som den franska BERT: Camembert. Denna avhandling jämför språkmodellerna för dessa olika förutbildade modeller och jämför deras förmåga att säkerställa en domänanpassning. Med den flerspråkiga och franska förutbildade versionen av BERT och en dataset från AXA Frankrikes e-postmeddelanden, kundmeddelanden, juridiska dokument, försäkringsdokument som innehåller över 60 miljoner ord. Vi finjusterade språkmodellerna för att anpassa dem till Axas försäkrings franska sammanhang för att skapa en fransk AXA Insurance BERT-modell. Vi utvärderar prestandan för denna modell på förmågan hos språkmodellen att förutsäga en maskerad token baserat på sammanhanget. BERTpresterar bättre: modellerar bättre den franska AXA-försäkringstexten utan finjustering än Camembert. Men med denna lilla mängd data är Camembert mer kapabel att anpassa sig till denna specifika försäkringsdomän.

Nyckelord

NLP, Language model, Word embedding, BERT, camemBERT

Abbreviations and Acronyms

AF Activation Function

AP Average Precision

ANN Artificial Neural Network

BERT Bi-directional Encoder Representations using Transformers

BOW Bag of Words

CBOW Continuous Bag-OF-Words

CNN Convolutional Neural Network

CRF Conditional Random Field

CV Computer Vision

i.i.d. independent and identically distributed

LASSO Least Absolute Shrinkage and Selection Operator

LDA Latent Dirichlet Allocation

LM Language Model

LSA Latent Semantic Analysis

LSTM Long Short-Term Memory

ML Machine Learning

MLP Multi Layer Perceptron

MT Machine Translation

MTL Multi-Task Learning

MSE Mean Squared Error

NER Named Entity Recognition

NLP Natural Language Processing

NN Neural Networks

OOB Out of the Box

OOV Out of Vocabulary

POS Part-OF-Speech

QA Question Answering

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SGNS Skip-Gram with Negative Sampling

SGD Stochastic Gradient Descent

SRL Semantic Role Labelling

SSL Semi-Supervised Learning

TFIDF Term Frequency-Inverse Document Frequency

Contents

Chapter 1

Introduction

Language is the scaffold of our minds. people build their thoughts through language and it conditions how they experience and interact with the world. It is the main communication tool used to express opinions, expectations, needs and answers. However, the social nature of the human being makes us dependent on each other for our most crucial needs. In order to achieve fluent interaction, natural language is the principal communication tool to express our intents and expectations. From its primitive form including vocal and body cues to digital text representations, language has enabled but also evolved together with the technological progress.

Natural Language Processing (NLP) is the discipline within the field of Artificial Intelligence (AI) that intends to equip machines with the same comprehension capability of natural language as humans do. This field has the goal of extracting knowledge from a text corpus and processing it for a wide array of tasks that provide valuable insights on the analyzed data. Commonly, computers are well suited to process formal language. This entails structured data, organized rules and commands without ambiguity. Examples of such are programming languages or mathematical expressions.

Natural language comes with its own set of challenges. Not only the content is unstructured, but the language itself is ambiguous and inconsistent. Metaphors, polysemy, rhetoric such as sarcasm or irony and a vast collection of ambiguities are even hard to grasp for humans when reading. These nuances and sources of difficulties to proper understanding are exacerbated by the variety of national languages (English,

French, German, etc.). At the same time, the technical domains where it is being used (scientific, administrative, insurance language to name a few) play an essential role defining the meaning of the words. Finally, the context and the implied information from world knowledge are important to the correct interpretation. So, how does NLP deal with these barriers?

Traditionally, methods employed by NLP practitioners have been based on complex sets of hand-written rules. The design and implementation of rules that try to model the complexity of a language needed to take into account all the linguistic elements and nuances. Needless to say, these systems are hard to implement, maintain, scale and transfer. They are generally not flexible enough as they cannot be extended to unknown words and infer their lexical nature. The linguist Noam Chomsky gave another excellent example of the challenge with his sentence: ”Colorless green ideas sleep furiously”. Despite of the correct syntax, the sentence is incoherent due to the inherent properties of the entities and their possible attributes. Moreover, considering language as an ever evolving instrument that mutates with the time, adapting these rules would be infeasible. Rule based systems were the norm until late 80s. Then, research increasingly turned to machine learning and statistical methods. The machine learning approaches have ever since been gaining traction. This is because of their capability to produce probability based predictions that can reliably solve multiple tasks and sub-tasks. These methods have attained remarkable results and have proven themselves robust when extrapolated to new data. Another factor that pushed forward the trend is the continuous progress of hardware performance. Deep neural networks are computationally expensive and it is only with the nowadays wide availability of GPUs that the processing power meets the required demand.

1.1 Motivation

A New Milestone in NLP

In the late 2018, the research community in Artificial Intelligence saw a significant advance in the development of deep learning based NLP techniques. This is due to the publication of the paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” by the Google AI team [45]. As the title suggests, the work takes a twist on the recent Transformer architecture [188] which is solely based on the attention mechanism and defines a novel type of deep neural network arrangement.

Their bidirectional learning approach managed to achieve unprecedented performance and pushed the state-of-the-art in 11 downstream tasks such as Named Entity Recognition, classification, question-answering, language inference among others. Followed by the open sourcing of their model, academics working with deep learning methods for NLP [195] were able to reproduce such results, as well as fine-tuning the model for their own research tasks.

BERT is an extremely large neural network model pre-trained over a 3.3 billion words English corpus extracted from Wikipedia and the BookCorpus [205] as training dataset. The model has been influenced by the new movement in NLP initiated by ELMo [144] and ULMFiT [72], that is transfer learning. The main idea of this technique is to allow the reuse of existing deep learning models that have been trained from scratch, saving costly computation power by adapting them across different domains, languages and/or tasks [160]. Research data scientist at Deepmind Sebastian Ruder, compares the impact of BERT for the NLP community with the acceleration that pre-trained models for images ImageNet brought to the computer vision field. Transfer learning in the industry

For businesses specialized in providing technical solutions based in text mining, the introduction of transfer learning in NLP represents a major paradigm shift in the development and training of deep learning models for NLP. AXA, the insurance company that supports this current thesis, is highly interested in evaluating the viability and cost-opportunity derived from this approach. Transfer learning and, in particular, domain adaptation would in theory reduce drastically the time required for producing a new model. With the means of adapting a general model to different industry domains in a time and cost optimized manner, transfer learning would reshape the way deep learning solutions are delivered to improve the client's experience and accelerate proccesses.

1.2 French Language Challenge

Since Axa France is based in France, French is a language of interest because of their portfolio of clients. If the linguistic diversity on the Internet is to be considered, French has been estimated to be the sixth most common online language after English and Russian, Spanish, Turkish and Persian. Despite of this, French would represent, in relative value, just 2.6% of the global content. According to W3Techs [1], this is almost

30 times less than English, which is the international vehicular language sitting in the first position covering 54% of all online content.

The situation is analog in the field of NLP research, primarily due to the fact that French corpora collections suitable for NLP are far less abundant than in English. Secondly, the Internet has become one of the main sources of data for many studies because of its accessibility as well as its exponentially increasing volume. Additionally, English being the lingua franca in academia, the most renowned benchmarks for NLP tasks are, therefore, also aimed to evaluate language models and tasks using text corpora in English. French, despite of being widespread, can be considered a relatively low resource language in task-specific datasets and this turns it into an ideal candidate for the application of transfer learning.

1.3 Insurance and NLP

The insurance text has specific vocabulary. There are some situations where the words used for a specific product like ”santé”, ”auto”, or some abbreviations like ”AN”:”Affaire Nouvelle”. These words exist in french but they have a very different meaning. ”auto” in French is a prefix that is related to automation while it is mainly car insurance in AXA’s jargon. This adds a challenge to the NLP tasks. Insurance context adds a challenge to the NLP tasks but can make a big use of the different NLP tasks to accelerate the different processes, to improve the client’s experience.

1.4 Problem

The important increase in the volume of the clients’ messages, emails, comments on the social media has made NLP an essential tool to consider for large-scale knowledge extraction and machine reading of this textual data. Recent progress in NLP has been driven by the adoption of deep neural models, but training such models often requires large amounts of labeled data. In general domains and for english language, large-scale training data is often possible to obtain through crowdsourcing, but for French natural Language written by axa clients in a specific insurance domain, annotated data is difficult and expensive to collect due to the expertise required for quality annotation. The first need detected while dealing with the huge number of comments is the need of a Named Entity Recognition tool that detects the relevant information in the clients

messages (Name, surname, Contract Number, the phone number, delay, dates...). The clients, while complainig or describing their experience would generally provide a lot of relevant information hoping to be called back or asking for a service. The number of messages, comments and mails through the different channels of communication is estimated to be almost 100000 a day. This makes the humain treatment of every message very difficult. Thus, the main problems encountered are the detection of the named entities of these messages, the classification of the messages, the indexation of the message inorder to enrich the data base about the clients with the added important information mentioned in these messages and finally the anonymisation. These are different types of NLP tasks needed to respond to these problems : NER(Named Entity Recognition), Classification. NLp tasks would always rely on the representation of the input textual data and add the different needed bricks to answer the provide the wanted task.

1.5 Research Question

Inspired by these latest developments, the goal of this research project consists in determining whether transfer learning, domain adaptation in particular, is a promising technique ready to be adopted by NLP professionals or not. The chosen method to evaluate this is by measuring the effects of using domain vocabulary and training a new specific-domain langue model. The current language domain being considered is the insurance field in French. As BERT has been pre-trained using Wikipedia, a multilingual model “BERTBase, Multilingual Cased” supporting 104 languages is available. Nonetheless, a multilingual model presents possible shortcomings in performance since the number of articles on Wikipedia varies greatly per language. Therefore, a specific BERT model pre-trained and retrained in french would be choosen to ensure more robust representations and avoid interference from other languages. The second operation will be with the camemBert model pre-trained on french data and retrained on insurance specific domain data. The main purpose is to get the best representation of insurance-related text in Axa: The word embedding. This embedding will be an Axa-specific representation of text that will be the main block to start with all the other NLP tasks.

The configuration of different types of texts, comments on social media and mails in Axa’s canals is an opportunity for the implementation of several downstream tasks.

For example, a classifier: given a transcription of a client's feedback, the model should be able to classify to which type of problem the client is referring to. A Named Entity Recognition system should get as input the best representation of the text taking into account the context of the sentence, the meaning and the relation between the different words. The better is the embedding of the input text the better detection of relevant information the NER systems yields.

The project aims to answer the main research question:

The main Research Question that was guiding the research is : "How can the state-of-the art in NLP used to have a better representation of French insurance AXA-related data? " This splits the thesis into two parts: Studying the state-of-the-art and adapting it to this specific type of data. After the state-of-art study, BERT model and its different versions were chosen as the main research baseline. Thus the main Research Question would be: How to get a better representation (word embedding) of French Insurance AXA-related text using the existing pre-trained BERT models?

This main question can be subsequently divided into sub-questions to help us underpin the different aspects that lead to a complete and thorough answer: 1. How does BERT models deal with the specific vocabulary of AXA. 2.What are the requirements for domain adaptation of BERT model ? 3. What is the impact of fine-tuning the existing pre-trained models on the word embeddings and text representation?

1.6 Thesis Outline

The thesis is organized as follows: Chapter 2 reviews the background theories that set the foundational knowledge for this research. It analyses the existing related work. Chapter 3 gives an overview of the methodology. The experiments implemented using Axa's data and their results are presented in chapter 5. Finally the thesis closes with the conclusions and a discussions on further work in Chapter 6.

Chapter 2

Machine Learning and Deep Learning

Artificial intelligence is an emerging field which has been actively attracting attention for several years. Machine Learning and Deep learning are 2 subsets of artificial Intelligence (AI) which try to simulate the human intelligence by programming machines and algorithms in order to think like humans and mimic their decisions. In this chapter, a detailed description about background of machine learning and deep learning is presented .

2.1 Machine Learning

In this section, the reader is introduced to machine learning, which builds mathematical models from data. many concepts described in this section will be useful to understand throughout the thesis, either forming the building blocks used in more advanced neural network-based methods(section) or supplying the theory that underpins many of the proposed models.

One famous definition of Machine Learning (ML) is based on the idea of learning from experience. ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” [130]

The following example should help understanding this definition. Let’s say there is a sunny forecasting system that predicts whether it will be sunny today or not (T). The system is a binary classifier (sunny, not sunny) and its performance will be

measured by its accuracy (P): among all the predictions how much are correct. It learns from time, historic weather data, temperature, season ... (E) to predict the right outcome.

Thus, Machine learning is a modelling approach where algorithms are used to implicitly find underlying patterns in a data set instead of specifying what features are important to solve a problem. This is accomplished by providing the algorithm with a learning function to optimize and a rich data set to find features in. From there, the algorithm tunes the parameters of the model to optimize the model's predictive performance on unseen data which is used to evaluate the model.

In addition, ML is one central subject in AI and splits into four subcategories: Supervised Learning, Unsupervised Learning, Reinforcement Learning and Deep Learning. Supervised and Unsupervised Learning are the two main types of data mining problems.

Unsupervised problems do not have a known outcome and the solution is often based on instance similarity patterns and groups that have to be found in the data.Lacking knowledge about the output, unsupervised learning is more commonly used for finding structures in data making it suitable for, e.g., clustering data.

Machine learning problems are called supervised learning problems if the dataset also contains the true labels to compare predictions against: The target variable is know for a certain dataset and can be used for model learning. For example, based on a text sentence x_i, a learning model H should classify the emotional undertone y_i in the sentence (sad, happy, angry...). Then, a comparison of the predicted emotion labels $\hat{y}_i$ with the actual emotion labels is conducted to evaluate our model. Supervised learning utilizes labeled data such that for every input x there exists an output y, i.e., a ground truth or target. This makes supervised learning suitable for tasks such as classification and regression. Classification problems aim to classify data into a finite number of categories, whereas regression problems predict a continuous number as their output. Since it is possible to quantify how well our model does at predicting the correct output values, one can see how changing the parameters of the model changes our predictions. Therefore it is optimizing a function that punishes worsened predictions and rewards improvements. In a machine learning setting, such a measure is called a cost function. The way this function is minimized is through an iterative algorithm called gradient descent.

2.1.1 Definition

In machine learning, each input is typically represented as a vector x $\in R^d$ of d features, where each feature contains the value for a particular attribute of the data and each example is assumed to be drawn independently from the data generating distribution $\hat{p}_{data}$, the true distribution p which is different from the model distribution p_{model}. An entire dataset can be seen as a matrix X $\in R^{n \gg d}$ containing n examples, one example in each row.

In supervised learning, for every input x_i , the output is typically a separate label y_i, which can be arranged as a vector of labels y for the entire dataset. In unsupervised learning, no designated labels are available. Two common categories of machine learning tasks are classification and regression: In classification, the label y_i belongs to one of a predefined number of classes or categories. In regression, y_i is a continuous number.

Classification further subsumes binary classification, multi-class classification, and multilabel classification. Binary classification only deals with two classes, while multi-class classification deals with more than two classes. Typically, every example x_i only has one correct label y_i. In multi-label classification, every x_i may be associated with multiple labels.

In many scenarios throughout this thesis, the output may be more than a single number. Tasks with more complex outputs, known as structured prediction, are common in natural language processing and will be discussed in Section 3.0.3.

2.1.2 Maximum Likelihood Estimation

The most common way to design a machine learning algorithm is to use the principle of maximum likelihood estimation (MLE). An MLE model is defined as a function p_{model}(x; θ) that maps an input x to a probability using a set of parameters θ. As the true probability p(x) of an example x is unknown, the true probability p(x) is approximated with the probability $\hat{p}_{data}$(x) under the empirical or data generating distribution.
The objective of MLE then is to bring the probability of our model p_{model}(x; θ) as close as possible to the empirical probability of the input $\hat{p}_{data}$(x). In other words, MLE seeks to maximize the likelihood or probability of the data under the configuration of the model.

The maximum likelihood estimator is defined as:

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}} p_{model}(x;\theta) \tag{2.1}$$

$$= \underset{\theta}{\text{argmax}} \prod_{i=1}^{n} p_{model}(x_i;\theta) \tag{2.2}$$

In practice, many of the probabilities in the product can be small, leading to underflow. Taking the logarithm does not change the arg max, but transforms the product into a sum, which results in a more convenient optimization problem [57].

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log p_{model}(x_i;\theta) \tag{2.3}$$

As the arg max also does not change under division by a constant value, divided by n to obtain an expectation with respect to the empirical distribution of the data $pdata$:

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}} \mathbb{E}_{x\sim\hat{p}_{data}} [\log p_{model}(x_i;\theta)] \tag{2.4}$$

Rather than maximizing the likelihood of the data under the model, MLE can also be seen as minimizing the dissimilarity between the empirical distribution $\hat{p}_{data}$ and the model distribution p_{model} as measured by the KL divergence:

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}} [\log p_{data}(x) - \log p_{model}(x_i;\theta)] \tag{2.5}$$

As the term on the left, $\log p_{data}(x)$ is only a function of the data generating distribution and not the model, one can train the model to minimize the KL divergence by only minimizing the term on the right-hand side, $\log p_{model}(x_i;\theta)$. Minimizing a negative term is the same as maximizing the term, so this objective is the same as the MLE objective in Equation (2.4)

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmin}} - \mathbb{E}_{x\sim\hat{p}_{data}} [\log p_{model}(x_i;\theta)] \tag{2.6}$$

Furthermore, this objective is also the same as minimizing the cross-entropy defined in Equation 2.67 between the empirical distribution $\hat{p}_{data}$ and the model distribution

p_{model}:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmin}} H(\hat{p}_{data}, p_{model}) \tag{2.7}$$

Cross-entropy is a common loss term in machine learning and the objective function that is most commonly used in neural networks. Consequently, frequent use of it will be seen throughout this thesis.

Conditional maximum likelihood The MLE estimator p_{model}(x; θ) discussed so far essentially does unsupervised learning as it only seeks to estimate the likelihood of the data. For supervised learning, one instead need to estimate the conditional probability P(y| x; θ) in order to predict the label y given x. The conditional maximum likelihood estimator is:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} P(y|X;\theta) \tag{2.8}$$

This can again be decomposed into:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log P(y|x_i;\theta) \tag{2.9}$$

Point estimation The conditional maximum likelihood estimator is a point estimator : It provides the single 'best' prediction $\hat{y}$ for the true label y. A point estimator $\hat{\theta}$ is any function of the data that seeks to model the true underlying parameter θ^* of the data:

$$\hat{\theta} = g(X) \tag{2.10}$$

As the data is assumed to be generated from a random process and $\hat{\theta}$ is a function of the data, $\hat{\theta}$ is itself a random variable.

2.1.3 Linear regression

The simplest example of a point estimator that maps from inputs to outputs is linear regression, which solves a regression problem. The Linear regression models a conditional probability distribution $p(y|x)$: it takes as input a vector $x \in \mathbb{R}^d$ and aims to predict the value of a scalar $y \in \mathbb{R}$ using a vector $\theta \in \mathbb{R}^d$ of weights or parameters and an intercept or bias term $b \in \mathbb{R}$

$$\hat{y}(x;\theta) = \theta^T x + b \tag{2.11}$$

where $\hat{y}$ is the predicted value of y. The mapping from features to prediction is an affine function, i.e. a linear function plus a contant.

Mean squared error: In order to learn the weights θ, the model's error can be minimized, a task-specific measure of haw far the model's prediction $\hat{y}$ differs from the tru value y. A common error measure is mean squared error, which is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.12}$$

Other commonly used terms for such an error measure are objective functions, cost function and loss. One can view mean squared error also as maximum likelihood estimation, especially as the cross-entropy between the empirical distribution and a Gaussian model. Let the conditional distribution $p(y|x)$ modelled by a linear regression be parameterized by a Gaussian. The conditional maximum likelihood estimator as defined in Equation 2.9 for linear regression is then:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} logp(y|x_i;\theta) \tag{2.13}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} logN(y;\hat{y}(x;\theta),\sigma^2) \tag{2.14}$$

where $\hat{y}(x;\theta)$ predicts the mean of the Gaussian and σ^2 is a constant. Substituting the defintion of the Gaussian distribution from the previous equations obtained:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} log[\sqrt{\frac{1}{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(\hat{y}_i - y_i)^2\right\}] \tag{2.15}$$

Taking the logarithm of a product and as $log(e^b) = b$, one should get:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \frac{1}{2} log(\frac{1}{2\pi\sigma^2}) - \frac{1}{2\sigma^2}(\hat{y}_i - y_i)^2 \tag{2.16}$$

Applying the linearity of summation yields:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \frac{n}{2} log(\frac{1}{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.17}$$

The right-most term is just the mean squarred error. thus you have:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \frac{n}{2} log(\frac{1}{2\pi\sigma^2}) - \frac{1}{2\sigma^2} MSE \tag{2.18}$$

Since n, π, and σ^2 are constants, MLE requires only to maximize the negative MSE term which is the same as minimizing the MSE.

Linear regression with mean squared error is also known as linear least squares. A common way to find a solution is to view the problem as a matrix equation (omiiting the bias term):

$$X\theta = y \tag{2.19}$$

The normal equation then minimized the sum of the squared differences between the left and the roght side and yields the desired parameters θ:

$$X^T X\hat{\theta} = X^T y\hat{\theta} \qquad = (X^T X)^{-1} X^T y \tag{2.20}$$

$X^T X$ is also known as normal matrix.

2.1.4 Logistic Regression

One can apply linear regression in a way to have a classification. In the case of binary classification, there are two classes, class 0 and class 1. The output of linear regression can be transformed into a probability by ”squashing” it to be in the interval (0,1) using the sigmoid or logistic regressino function σ, which is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.21}$$

The probability produced by logistic regression is then calculated as follows:

$$\hat{p}(y = 1|x; \theta) = \hat{y} = \sigma(\theta^T x) \tag{2.22}$$

Specifying the probability of one of these classes determines the probability of the other class, as the output random variable follows a Bernoulli distribution. For multi-class classification, a separate set of weights $\theta_i \in \theta$ is learnt for the label y_i of the i-th class. The softmax function is used to squash the values to obtain a categorical distribution:

$$\hat{p}(y_i|x; \theta) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{C} e^{\theta_j^T x}} \tag{2.23}$$

where the denominator is the so-called partition function that normalized the distribution by summing over the scores for all C classes.

The cross-entropy is calculated between the empirical conditional probability $p(y|x)$ and the probability of our model $\hat{p}(y|x; \theta)$ for each example x:

$$H(p, \hat{p}; x) = -\sum_{i=1}^{C} p(y_i|x) log \hat{p}(y_i|x; \theta) \tag{2.24}$$

For binary classification, this simplifies to

$$H(p, \hat{p}; x) = -(1 - y) log(1 - \hat{y}) - y log \hat{y} \tag{2.25}$$

As our cost function $J(\theta)$, the average cross-entropy is minimized over all examples in our data:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} H(p, \hat{p}; x_i) \tag{2.26}$$

In contrast to linear regression with mean squared error, there is typically no closed-form solution to obtain the optimal weights for most loss functions. Instead, the error of our model is iteratively minimized using an algorithm known as gradient

descent.

2.1.5 Gradient descent

Gradient descent is an efficient method to minimize an objective function J(θ). It updates the model's parameters $\theta \in R^d$ in the opposite direction of the gradient $\nabla_\theta J(\theta)$ of the function. The gradient is the vector containing all the partial derivatives $\frac{\partial J(\theta)}{\partial \theta_i}$. The i-th element of the gradient is the partial derivative of J(θ) with respect to θ_i. Gradient descent then updates the parameters:

$$\theta = \theta - \eta . \nabla_\theta J(\theta) \tag{2.27}$$

where η is the learning rate that determines the magnitude of an update to our parameters. In practice, η is one of the most important settings when training a model. To guarantee convergence of the algorithm, the learning rate is often reduced or annealed over the course of training. As seen previously, the expected value or average of an error function is typically minimized over the empirical distribution of our data:

$$J(\theta) = \mathbb{E}_{x,y \sim p_{data}} \mathcal{L}(x, y, \hat{y}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, y_i, \hat{y}_i, \theta) \tag{2.28}$$

The gradient $\nabla_\theta J(\theta)$ is thus:

$$\nabla_\theta J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x_i, y_i, \hat{y}_i, \theta) \tag{2.29}$$

This is known as batch gradient descent and is expensive as for each update the gradient needs to be computed for all examples in the data. Alternatively, stochastic gradient descent iterates trough the data, computes the gradient, and performs an update for each example i:

$$\nabla_\theta J(\theta) \approx \nabla_\theta \mathcal{L}(x_i, y_i, \hat{y}_i, \theta) \tag{2.30}$$

While this is cheaper, the resulting gradient estimate is a lot more noisy. The most common approach is to choose a middle ground and compute the gradient over a minibatch of m examples, which is commonly known as mini-batch gradient descent or stochastic gradient descent with mini-batches:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{L}(x_i, y_i, \hat{y}_i, \theta) \tag{2.31}$$

The mini-batch size m typically ranges from 2 to a few hundred and enables training of large models on datasets with hundreds of thousands or millions of examples. In practice, mini-batch gradient descent is the default setting and is often referred to as stochastic gradient descent as well.

While stochastic gradient descent works surprisingly well in practice and is the main way to train neural networks, it has a few weaknesses: It does not remember its previous steps and uses the same learning rate for all its parameters. We direct the reader to [162] for an overview of momentum-based and adaptive learning rate techniques that seek to ameliorate these deficiencies.

2.1.6 Generalization

The goal of machine learning is generalization, training a model that performs well on new and previously unseen inputs. To this end, the available data X is typically split into a part that is used for training, the training set and a second part reserved for evaluating the model, the test set. Performance on the test set is then used as a proxy for the model's ability to generalize to new inputs.

This measure is responsible for the main tension in machine learning: During training, the training error is computed, the error of the model on the training set, which is intended to be minimized. The actual measure of interest, however, is the generalization error or test error, the model's performance on the test set, which it has never seen before. This is also the main difference to optimization: While optimization seeks to find the minimum that minimizes the training error, machine learning aims to minimize generalization error Train and test sets are typically assumed to be i.i.d.: Examples in each dataset are independent from each other and train and tests sets are identically distributed, i.e. drawn from the same probability distribution.

Bias-variance trade-off The goal to minimize a model's generalization error gives rise to

two desiderata: 1. to minimize the training error; 2. and to minimize the gap between training and test error. This dichotomy is also known as bias-variance trade-off. If the model is not able to obtain a low error on the training set, it is said to have high bias. This is typically the result of erroneous assumptions in the learning algorithm that cause it to miss relevant relations in the data. On the other hand, if the gap between the training error and test error is too large, the model has high variance. It is sensitive to small fluctuations and models random noise in the training data rather than the true underlying distribution.

More formally, the bias of an estimator $\hat{\theta}$ is the expected difference between the value of the parameter $\hat{\theta}$ and the true underlying value of the parameter θ^* with regard to the data generating distribution:

$$Bias(\hat{\theta}) = \mathbb{E}[\hat{\theta} - \theta^*] \tag{2.32}$$

The estimator $\hat{\theta}$ is unbiased if bias($\hat{\theta}$) = 0. For instance, the sample mean is an unbiased estimator of the mean of a distribution. The variance of an estimator is simply its variance:

$$Var(\hat{\theta}) = \mathbb{E}[\hat{\theta}^2] - \mathbb{E}[\hat{\theta}]^2 \tag{2.33}$$

By rearranging:

$$\mathbb{E}[\hat{\theta}^2] = Var(\hat{\theta}) + \mathbb{E}[\hat{\theta}]^2 \tag{2.34}$$

The square root of the variance of an estimator is called the standard error SE($\hat{\theta}$). To measure an estimator's performance, the mean squared error of the estimator $\hat{\theta}$ can be compared to the true parameter value θ^*:

$$MSE = \mathbb{E}[(\hat{\theta} - \theta^*)^2] \tag{2.35}$$

Expanding the binomial:

$$MSE = \mathbb{E}[\hat{\theta}^2] - \mathbb{E}[2\hat{\theta}\theta^*] + \mathbb{E}[\theta^{*2}] \tag{2.36}$$

By replacing $\mathbb{E}[\hat{\theta}^2]$ and $\mathbb{E}[\theta^{*2}]$ with the right-hand side of Equation (2.34) respectively:

$$MSE = Var(\hat{\theta}) + \mathbb{E}[\hat{\theta}]^2 - \mathbb{E}[2\hat{\theta}\theta^*] + Var(\theta^*) + \mathbb{E}[\theta^*]^2 \tag{2.37}$$

One can now form another binomial expansion, reduce it to a binomial, and using the linearity of expectation:

$$\begin{aligned} MSE &= Var(\hat{\theta}) + Var(\theta^*) + (\mathbb{E}[\hat{\theta}]^2 - \mathbb{E}[2\hat{\theta}\theta^*] + \mathbb{E}[\theta^*]^2) && (2.38)\\ &= Var(\hat{\theta}) + Var(\theta^*) + (\mathbb{E}[\hat{\theta}] - \mathbb{E}[\theta^*])^2 && (2.39)\\ &= Var(\hat{\theta}) + Var(\theta^*) + (\mathbb{E}[\hat{\theta} - \theta^*])^2 && (2.40) \end{aligned}$$

Var(θ) is the true variance σ^2 of the parameter θ and the right-most term under the square is the definition of the bias in Equation (2.32). Replacing both yields the bias-variance decomposition for squared error:

$$MSE = Var(\hat{\theta}) + \sigma^2 + Bias(\hat{\theta})^2 \tag{2.41}$$

This decomposition sheds more light on the trade-off between bias and variance in machine learning. The expected error of a model trained with mean squared error is thus lower bounded by the sum of three terms:

• The square of the bias of the method, i.e. the error caused by the simplifying assumptions inherent in the model. • The variance of the method, i.e. how much its results vary across the mean. • The variance of the true underlying distribution.
If a model has high bias it is also said to be *underfitting*. If a model has high variance, it is known to be *overfitting*. A key factor that determines whether a model underfits or

overfits is its capacity, which is its ability to fit a variety of functions. One way to control a model's capacity is to choose an appropriate hypothesis space, the set of functions it can choose from to find the solution. The *hypothesis space* of linear regression is the set of all linear functions of its input. One can increase the capacity of linear regression by generalizing it to include polynomials of degree k:

$$\hat{y} = b + \sum_{i=1}^{k} {\theta_i}^T x^i \tag{2.42}$$

where $\theta_i \in \mathbb{R}^d$ are additional weight vectors for each polynomial. A machine learning model performs best when its capacity is appropriate for the task it is required to solve. A commonly used heuristic is expressed by *Occam's razor*, which states that among competing hypotheses that explain known observations equally well, one should choose the "simplest", which in this context refers to the model with the lowest capacity. However, while simpler functions are more likely to generalize, a hypothesis is still required that is sufficiently complex to achieve low training error.
In machine learning, the no free lunch theorem [196] states that no algorithm is universally better than any other. Specifically, averaged over all possible data generating distributions, every classification algorithm achieves the same error when classifying previously unknown points. Our goal in practice is thus to bias the algorithm towards distributions or relations in the data that are more likely to be encountered in the real world and to design algorithms that perform well on particular tasks.

Throughout this thesis, bias and inductive bias will be used interchangeably to describe assumptions that are encoded in a model about unseen data. The general aim is to develop models with an inductive bias that is useful to generalize to novel domains, tasks, and languages.

Statistical learning theory provides theoretical bounds on the generalization error: In particular, the difference between training error and generalisation error has been shown to grow with the capacity of the model but shrink as the number of training examples increases [187]. These bounds, however, are rarely used in practice as they are quite loose and it is difficult to determine the capacity of deep neural networks [57].

Nevertheless, the generalisation behaviour of deep neural networks is an active area of research. In practice, a validation set is often used in addition to tune different settings of the model, its hyper-parameters, such as the degree of the polynomial in logistic regression. If the test set is too small, another technique called cross-validation is typically used. Cross-validation repeats the training and test computations on different randomly chosen splits of the data and averages the test error over these splits. The most common variation is k-fold cross-validation, which splits the data into k subsets of equal size and repeats training and evaluation k times, using k – 1 splits for training and the remaining one for testing.

2.1.7 Regularization

Another way to modify a model's capacity is to encourage the model to prefer certain functions in its hypothesis space over others. The most common way to achieve this is by adding a regularization term $\sum(\theta)$ to the cost function J(θ):

$$J(\theta) = MSE + \lambda \sum(\theta) \tag{2.43}$$

where λ controls the strength of the regularization. If λ = 0, there is no restriction. As λ grows larger, the preference that we impose on the algorithm becomes more prominent. The most popular forms of regularization leverage common *vector norm*s. ℓ_1 regularization places a penalty on the ‘1 norm, i.e. the sum of the absolute values of the weights and is defined as follows:

$$\sum(\theta) = \|\theta\|_1 = \sum_i |\theta_i| \tag{2.44}$$

where $\theta_i \in \mathbb{R}$. ℓ_1 regularization is also known as lasso (least absolute shrinkage and selection operator) and is the most common way to induce sparsity in a solution as the ℓ_1 norm will encourage most weights to become 0. ℓ_2 regularization is defined as:

$$\sum(\theta) = \|theta\|_2^2 \tag{2.45}$$

where $\|\theta\|_2 = \sqrt{\sum_i {\theta_i}^2}$ is the Euclidean norm or ℓ_2 norm. Somewhat counter-intuitively, ℓ_2 regularization thus seeks to minimize the squared ℓ_2 norm as in practice, the squared ℓ_2 norm is often more computationally convenient to work with than the ℓ_2 norm. For instance, derivatives of the squared ℓ_2 norm with respect to each element of θ depend only on the corresponding element, while derivatives of the ℓ_2 norm depend on the entire vector [57]. ℓ_2 regularization is also known as *Tikhonov regularization*, *ridge regression*, and *weight decay*. ℓ_2 regularization expresses a preference for smaller weights in a model.

Different forms of regularization may also be combined. The combination of ℓ_1 and ℓ_2 regularization is also known as elastic net regularization. It uses an α parameter to balance the contributions of both regularizers:

$$\sum(\theta) = \alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2 \tag{2.46}$$

Besides the ℓ_1 and ℓ_2 norms, the only other norm that is used occasionally for regularization is the ℓ_{inf} norm or max norm, which penalizes only the maximum parameter value

$$\|\theta\|_{\text{inf}} = \max|\theta_i| \tag{2.47}$$

In some scenarios, one can be interested in imposing a norm on a weight matrix $W \in \mathbb{R}^{m \text{»} n}$. For this case, we use the matrix counterpart of the ℓ_2 norm, the Frobenius norm:

$$\|W\|_F = \sqrt{\sum_{i,j} {W_{i,j}}^2} \tag{2.48}$$

The Frobenius norm is useful for instance to express the preference that two weight matrices W_1 and W_2 should be orthogonal, i.e. $W_1^T W_2 = I$. This is achieved by placing the squared Frobenius norm on the matrix product:

$$\sum(W_1, W_2) = {\left\|W_1^T W_2\right\|_F}^2 \tag{2.49}$$

This orthogonality constraint is a common component of current approaches to domain adaptation, which is used to encourage non-redundancy of the representations of different layers. Another common matrix norm is the nuclear norm or trace norm, which applies the ℓ_1 norm to the vector of singular values σi of matrix W:

$$\|W\|_* = \sum_{i=1}^{min(m,n)} \sigma_i(W) \tag{2.50}$$

The trace norm is the tightest convex relaxation of the rank of a matrix [Recht et al.,2010], so can be useful to encourage a matrix to be low-rank. It has been frequently used in multi-task learning. While the focus was on vector and matrix norms in this section, any approach that implicitly or explicitly expresses a preference for particular solutions can be seen as regularization.

2.2 Artificial Neural Networks

This section's focus is on the concept of Artificial Neural Networks (ANNs) and FeedForward Neural Networks (FNN), as this is the first developed network type. ANNs take the human brain as an example and are based on the neurons of McCulloch and Pitts, which were established in 1943 [122].

2.2.1 Single Layer Perceptron

The McCulloch and Pitts' neuron takes vectors with length m as input and multiplies each value x_1, x_2, . . . , x_m by a corresponding weight w_1,w_2, . . . ,w_m. The neuron has the ability to activate when the overall sum is higher than a given threshold θ. An activation in this particular case means that the neuron outputs a 1, otherwise a 0 [119]. This is useful for predicting binary class labels that are suitable for the sunny day forecast task, which was discussed above. At first the neuron calculates the logit z of the input vector x_1, x_2, . . . , x_m:

$$z = \sum_{i=1}^{m} w_i x_i \tag{2.51}$$

Then the result y is determined by the activation function ϕ, which in this case is a Heaviside-Function. It takes the logit z as input and checks if it is below or above a

certain threshold θ:

$$o = \phi(z) = \begin{cases} 1, & \text{if } z > \theta \\ 0, & \text{otherwise} \end{cases} \tag{2.52}$$

Therefore, the output y of a McCulloch and Pitts' neuron is 0 or 1.

The Single Layer Perceptron is a collection of McCulloch and Pitts' neurons. The neurons are combined to create more complex ANNs. The model of a Single Layer Perceptron is shown in Figure 2.2.1, which is also known as a Single Layer Network. The network takes again m values as input. In addition, the network has more than one neuron in the output layer. Every input value is fully connected to each output neuron. Each connection is weighted to adjust the inputs received by the neurons. Therefore, the weights are stored in the form of a matrix and no longer in the form of a single vector.

For a Single Layer Perceptron only one weight matrix $W^{(1)}$ exists, which connects the m inputs to the n output nodes:

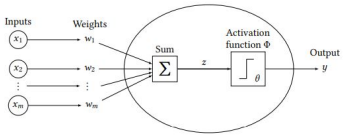


Figure 2.2.1: : Single Layer Neural Network with 3 outputs and m inputs [119]. It contains 3 Neurons in the output layer and a weight matrix W^1 that connects the inputs to the neurons.

The last row w_{b1},w_{b2}, . . . ,w_{bn} in the matrix indicates the weights for a bias value b. The bias is handled as a separate input and is used to shift the activation of a neuron. If a neuron receives a zero for input (x_1, x_2, . . . , x_m = 0), it cannot be activated. To implement this capability, a bias b is added, which must not be 0. In practice this value is often set to –1 or 1. Moreover, the j-th neuron in the output layer is denoted by hj. The weights and outputs are independent and a generalization of Equations (2.51) and (2.52) can be used to calculate the result y_j for each neuron's logit z_j in the output layer:

$$s_j = \sum_{i=1}^{m} {w_{ij}}^{(1)} x_i + w_{bj}^{(1)} \tag{2.53}$$

$$o_j = \phi(z_j) \tag{2.54}$$

In addition, it is possible to use another activation function ϕ to change the outputs of the neurons. Depending on the application, a certain output format is required. Moreover, the training algorithm requires a differentiable activation function and the step function does not meet this criterion [119]. Different activation functions such as Sigmoid, Rectified Linear Unit (ReLU) or softmax that will be explained in the next section.

2.2.2 Layers and models

In this section, there is an overview of the fundamental building blocks used in neural networks. We will now detail the layers and models, focusing more on those commonly applied to NLP tasks. ANNs are able to solve various types of classifiation and continuous variable prediction (regression) tasks. ANNs are the generic term for neural networks, such as a FNN, a CNN, autoencoders, RNN, which are used in AI.

Multilayer Perception MLP or Feedforward Neural network FNN

The Single Layer Perceptron, which was discussed above, is capable of solving linear separable problems, but cannot decide nonlinear issues such as the XOR problem [128]. To overcome this shortcoming more linear layers are added to the network to form a Multilayer Perceptron, such as shown in Figure 2.2.2

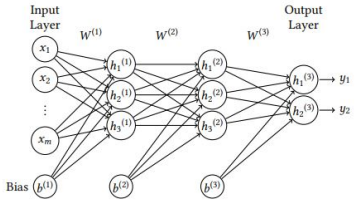


Figure 2.2.2: : Multilayer Perceptron with 2 hidden layers and 2 output neurons[119]. Three weight matrices connect the layers to pass the values through the network

A model with one hidden layer is known as a one-layer feed-forward neural network, which is also known as a multilayer perceptron (MLP):

$$h = \sigma_1(W_1 x + b_1) \tag{2.55}$$

$$y = softmax(W_2 h + b_2) \tag{2.56}$$

where σ_1 is the activation function of the first hidden layer. Note that each layer is parameterized with its own weight matrix W and bias vector b. Layers typically have separate parameters, but different layers can also set their parameters to be the same, which is referred to as tying or sharing of such parameters. Such parameter sharing induces an inductive bias that can often help with generalization. Many instances of parameter sharing is observec throughout this thesis, such as in multi-task learning. Computing the output of one layer, e.g. h that is fed as input to subsequent layers, which eventually produce the output of the entire network y is known as *forward propagation*.
forward propagation:

The first layer is called the input layer and has no neurons. It just receives the m input values. The output layer is the last layer of a Multilayer Perceptron and its neurons return the final result of the network. All intermediate layers are called hidden layers, because they perform the calculation within the network and are not visible to the network's external environment. The k-th layer of a network will be denoted with $h^{(k)}$. Every layer $h^{(k)}$ consists of a number of neurons and has its own bias $b^{(k)}$. Besides, it is connected to the previous layer with a weight matrix $W^{(k)}$. By adding more layers or neurons to a layer, a deep neural network (DNN) can be created. The number of neuron layers in a network will be labeled with K. The result of each neuron is calculated by using Equations 2.53 and 2.54. Each neuron's activation output from one layer is then forwarded as input to all neurons of the next layer. By applying this scheme in a row,

the network maps the input area to the output space. Therefore, an ANN can be seen as a function f(x, θ), which predicts the outcome $\hat{y}$ of an instance x = (x_1, . . . , x_m). To shorten and simplify notation the variable θ = (W , b) is introduced, which represents the parameters of an ANN. W indicates all weight matrices of the network ($W^{(1)}$, . . . ,$W^{(K)}$) and b is the vector of biases b = (b_1, . . . , b_K) [119]. For example, the network in Figure 2.2.2 could be a solution to the sunny day forecasting problem, which was briefly mentioned above. Date, temperature data, rain records of the previous days are suitable features for input. One of the output neurons returns the probability for a sunny day and another for a cloudy day.

FNNs are networks without cycles and are based on the Single Layer Perceptron 2.2.1 and Mulitlayer Perceptron 2.2.2 [167]. This chapter explains the basic theory they are based on. Neural networks can be seen as compositions of functions. In fact, the basic machine learning models described so far can be seen , linear regression and logistic regression, as simple instances of a neural network. Recall that multi-class logistic regression consists of the following functions:

$$f(x) = Wx + b \tag{2.57}$$

$$g(y) = softmax(y) \tag{2.58}$$

$$\tag{2.59}$$

where $W \in \mathbb{R}^{Cxd}$, $x \in \mathbb{R}^d$, $b \in \mathbb{R}^C$, $y \in \mathbb{R}^C$, C is the number of classes and d is the dimensionality of the input. In the following, W will be used to designate a matrix of weights, while θ, W, b is the set of parameters of the model. Logistic regression can be seen as a composition of the functions f and g: g(f(x)) where f(·) is an affine function and g(·) is an activation function, in this case the softmax function. A neural network is a composition of multiple such affine functions interleaved with non-linear activation functions.

To sum up, in the *feedforward neural network* or *multilayer perceptron*(MLP), information flows forward from the input layer through the intermediate layers and to the output layer. For an input vector x, the first hidden layer, $h^{(1)}$, in a network computes $h^{(1)}$ = g($w^{(1)}$x + b), where g is some activation function, $w^{(1)}$ is the weight vector in the first hidden layer and b is a bias term. These computations continue in the subsequent layers of a network until the output has been computed. An intermediate layer, such as $h^{(1)}$, can be seen as a vector representation of the input x to an MLP.

Neural word embeddings are an example of this.

Activation functions: As a composition of linear functions can be expressed as another linear function, the expressiveness of deep neural networks mainly comes from its non-linear activation functions. Activation functions have an influence on the model capacity and complexity of an ANN. They can be linear or non-linear. Furthermore, their use also depends on the actual data mining problem. The usage criterion mainly applies to the last layer of an ANN that generates the overall output. If a continuous result in $\mathbb{R}$ is required, e.g. in a regression task, a function that has the same value space is more useful. In contrast, the step function and other functions that map to a value between 0 and 1 are suitable for binary classification.

The simplest activation function is the linear or identity function that returns the weighted sum z Equation (2.60). The output is a continuous value:

$$\phi(z) = linear(z) = z \tag{2.60}$$

The sigmoid σ function is similar to the step function that was defined in Equation (2.61). Sigmoid can be derived, which is an important feature for training ANNs.

$$\phi(z) = \sigma(z) = \frac{1}{1 + \exp\{-z\}} \tag{2.61}$$

Furthermore, sigmoid is a continuous function, tends to become 0 if z ==> $-\inf$ and converges to 1 if z ==> $\inf$. In practice, it is often used in classification tasks due to its output. One drawback of sigmoid is its sensitivity to the vanishing gradient problem, which will be discussed further[57].

Another activation function that is less often used in practice is the *hyperbolic tangent or tanh* function, which outputs values in the range (−1, 1)

$$\sigma(x) = \frac{\exp\{x\} - \exp\{-x\}}{\exp\{x\} + \exp\{-x\}} \tag{2.62}$$

It has a similar shape and characteristics as the sigmoid function, but converges to -1 if $z \to -\infty$ and to 1 if $z \to \infty$. In addition, it is also differentiable and vulnerable to a vanishing gradient.

The Rectified Linear Unit (ReLU) is a state-of-the-art activation function, which is now

one of the most successful activation functions. Especially in Deep Neural Networks (DNNs), ReLUs lead to a faster learning time [105]. Besides, it is not susceptible to a vanishing gradient and therefore often used in practice. ReLU calculates the maximum between 0 and the linear outcome of a neuron:

$$\sigma(x) = max(0, x) \tag{2.63}$$

The Exponential Linear Unit (ELU) was introduced into Neural Networks in 2015 and outperforms ReLU [36]. It can have a negative outcome, does better generalize and even learns faster than ReLU. It is defined as:

$$\phi(z) = ELU(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha(\exp\{z\} - 1), & \text{otherwise} \end{cases} \tag{2.64}$$

softmax determines a probability value for each class. The outcome for neuron z is dependent on the outcome of the other neurons hi in the same layer, which makes it suitable for multi-class problems:

$$\phi(z) = softmax(z) = \frac{\exp\{z\}}{\sum_{i=1}^{N} \exp\{z_i\}} \tag{2.65}$$

The total value adds up to 1 and all class probabilities are between 0 and 1. This list of activation functions is by far not complete and other functions, such as Sigmoid-weighted Linear Unit (SiL) or Parametric ReLU (PReLU), exist [154]. Due to the fact that covering all activation functions would be too extensive for this thesis, only the most common ones were presented.

The softmax activation function is used for multi-class problems and uses the logits of all neurons in the same layer. It is often applied on top of the output layer. However, the hidden layers between input and output nodes may have other types of activation functions [119].

The softmax and sigmoid functions are common functions used at the final or output layer of a neural network to obtain a categorical and Bernoulli distribution respectively.

Non-output layers are referred to as hidden layers. Linear regression can be seen as a neural network without a hidden layer and a linear activation function—the identity function—while logistic regression employs a non-linear activation function. Neural networks are typically named according to the number of hidden layers.

Training

The training of an ANN is the basic part in this machine learning procedure. It puts the experience into the network. This section discusses details about the learning of a FNN. The training is similarly used for other ANN types. When training a neural network, the goal is to maximize or minimize some objective function. A loss function is a type of objective function that is to be minimized [57]. The purpose of a loss function is to measure how well a model predicts the expected outcome for any data point in the training set. Cost function is the term for the performance measure evaluated on the whole training set [57].

The section will start with the explanation of a cost function, which leads us to an optimization criterion. Then, for a more detailed overview, the algorithm is divided into smaller parts.

Cost Function:
Cost functions, denoted as J, determine the derivation of an instance's prediction $\hat{y}$ and its true value y, which is also known as error. The terms cost function, loss function and error function are often used interchangeably. The most common cost functions are the Mean Squared Error (MSE) and the Cross Entropy Loss. The first one determines the average deviation between prediction $\hat{y} = f(x, \theta)$ and true label y:

$$J(\hat{y}, y) = MSE(\hat{y}, y) = \frac{1}{N} \sum_{i=1} N(\hat{y}_i, y_i)^2 \tag{2.66}$$

N indicates the number of neurons in the output layer. This function could be used for any problem in general. In an environment with multiple class labels another cost metric has to be considered. The cross-entropy error function as described in

(2.67),

$$H(P,Q) = -\mathbb{E}_{x\sim P}[\log Q(x)] = -\sum_i P(i)\log Q(i) \tag{2.67}$$

where P is the target distribution and Q is the distribution of the network's prediction is commonly used objective function in neural networks. Consequently, minimizing a negative term is the same as maximizing a positive term. This means that minimizing cross-entropy corresponds to maximizing the likelihood of the data, since maximum likelihood estimation (MLE) is defined as equation (2.68).

$$\hat{\theta}_{MLE} = \mathbb{E}_{x\sim P}[\log Q(x)] \tag{2.68}$$

The Cross Entropy Loss (CE) function is calculated with:

$$J(\hat{y}, y) = CE(\hat{y}, y) = -\sum_{i=1} N y_i \ln \hat{y}_i \tag{2.69}$$

where each prediction $\hat{y}$ represents the probability for an independent class label. CE is often used in combination with softmax (Equation (2.65), as it calculates the deviation in a multi-class situation [119]. It is important that the cost function and activation function in the output layer are compatible. The goal of an ANN and the basic concept of the training is to reduce the cost function. Therefore, learning is an optimization problem with the following criterion:

$$\min_\theta min J(f(x,\theta), y) \tag{2.70}$$

θ = (W , b) are the parameters of the network that can be changed to reduce the error. Initialization and Forward Pass Before training an ANN the initialization phase takes place. Different approaches to initialize an ANN exist. A common method in practise is to set the weights W between the input nodes, hidden layers and the output layer to a small negative or positive random value. In addition to the weights, the bias $b^{(k)}$ for each layer k is set to a number, which is not 0. After this initial step the actual training starts. A training step encompasses the forward pass, where the values of the

first instance vector are passed to the input nodes. For each neuron the weighted sum is calculated with Formula 2.53 and inserted into its activation function Φ. Finally, the output is obtained by performing a set of computations at each layer k using the results of the previous layer as input. In a network with K layers the forward pass is calculated by:

$$h^{(1)} = \phi^{(1)}(W^{(1)} + x + b^{(1)}) \tag{2.71}$$

$$h^{(2)} = \phi^{(2)}(W^{(2)} + h^{(1)} + b^{(2)}) \tag{2.72}$$

$$\hat{y} = o = h^{(K)} = \phi^{(K)}(W^{(K)} + h^{(K-1)} + b^{(K)}) \tag{2.73}$$

$$\tag{2.74}$$

The input x is put into $h^{(1)}$ and the result of $h^{(1)}$ is passed forward. Finally, the output layer $h^{(1)}$ emits the prediction $\hat{y}$ [57].
Backword Pass In the backward pass, also known as back-propagation of error, the error is determined by a cost function. To actually update the network, function J(f (x, θ),y) needs to be differentiated with respect to θ [57]. The idea behind this process is to minimize the error by following the gradient of the cost function downwards (gradient descent), as illustrated by Plot 2.2.3

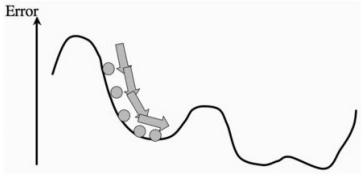
[h] 

Figure 2.2.3: Cost function going down the hill to find a local or global minimum [119].

The gradient of J with respect to θ is defined as:

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta} \tag{2.75}$$

This error signal obtained by the output layer is now passed backwards through the whole network. By applying the chain rule of differentiation, the partial derivative of the cost function J at the output layer $h^{(K)}$ is calculated as:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial J(\theta)}{\partial h^{K-1}} \frac{\partial h^{K-1}}{\partial \theta} \tag{2.76}$$

h^{K-1} is the result from the previous layer, which is passed to the output layer as input. Furthermore, $J(\hat{y}, y)$ includes the prediction of $\hat{y} = f(h^{(K1)}, \theta)$. Equation (2.76) indicates the change of error at output level when the parameters θ are varied. Now, the chain rule can be applied backwards through the whole network. This makes it possible to compute the gradients recursively and obtain all $\frac{\partial J(\theta)}{\partial W^k}$ for the weight matrices and $\frac{\partial J(\theta)}{\partial b^k}$ for the bias vectors in θ [57].

Gradient Descent: The gradient descent step happens after the backward pass. The parameters θ are updated with respect to the computed gradients. The weight matrices between the layers and the bias vectors are adjusted in the direction of the descending error. For a step wise adaption a learning rate η is multiplied with the gradient. This rate should be set appropriately, because a very low value would lead to slow learning and finding a local minimum only. A high learning rate could lead to an overshooting of the optimal minimum of the error. Three basic Gradient Descent manifestations exist: Batch Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent as described above.

The simplest one is Batch Gradient Descent, which is also known as Vanilla Gradient Descent. With this approach the weights and biases are updated by:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{2.77}$$

The term has to be subtracted from the actual parameters, since the error has to be reduced. Forward Pass, Backward Pass and Gradient Descent are repeated several times to improve the performance of the network [119]. Learning is stopped as soon as a criterion is met, e.g. if a certain number of training epochs has been completed or the training error has been reduced to a threshold value. An epoch is completed when all instances of the training dataset have been processed. Batch Gradient Descent updates the weights after processing all training instances. When learning on large data sets, it requires a lot of memory. In contrast, Stochastic Gradient Descent (SGD) updates the weights after each sample, which results in a high learning time. Therefore, the parameters are updated after each instance $x^{(i)}$ with its true label $y^{(i)}$ has been processed:

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^{(i)}, y^{(i)}) \tag{2.78}$$

Due to the fact that both methods do have some drawbacks, mini-batching is often used in practice to get the best out of both [159]

Mini-batch Gradient Descent Mini-batching aka Mini-Batch Gradient Descent improves the learning speed and saves memory. Without mini-batching, the weights of a neural network are not updated until all instances of the training data have been processed (Batch Gradient Descent) or the weights are updated after each instance (Stochastic Gradient Descent). To enable mini-batching the training data has to be split into groups of a batch size n, which is smaller than the size of the training set. In practice, batch sizes between 32 and 512 are commonly used [90]. The weights are updated after a batch has been processed [159]. The batch instances are passed into the network simultaneously:

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^{(i:+n)}, y^{(i:+n)}) \tag{2.79}$$

Shuffing the data and picking these batches randomly also improves the whole learning process. Furthermore, it is necessary to use an averaged version of the cost function. This method updates the weights into the direction in which the most instances of a batch are about to be optimized. Local minima are found faster and the estimation of the gradient error is better [119].

Regularization and Early stopping Neural networks, in particular DNNs, have many parameters and can be applied to various tasks of machine learning. In training, ANNs easily overfit and learn the training data by heart. Regularization is an important method to prevent this problem. Mainly two methods are applied for their regularization: Dropout and weight decay Dropout can be used at the input and output layer of a neural network. During training, it randomly drops neurons with a predefined dropout rate and their connections within the neural network. By doing this, dropout reduces the network's inner complexity and prevents overfitting on the training data. Furthermore, it improves the results in supervised learning tasks, such as document classiffcation and speech recognition [176].

Weight decay limits the growth of the network's parameters and improves the generalization of the network. The growth of the parameters is limited by a weight decay value. The weights are chosen so that the network continues to classify the instance correctly, but does not adopt it by heart. Thus, in each step of updating the

weights, all parameters are penalized with a term in the cost function:

$$J_{wd}(\theta) = J(\theta) + \frac{1}{2}\lambda \sum_{i}^{|\theta|} {w_i}^2 \tag{2.80}$$

Thereby, J(θ) represents a cost function like the the cross entropy loss or mean squared error. The actual weight decay factor is represented by λ and $|\theta|$ stands for the number of parameters in the neural network. In equation (2.80) one example for a simple penalty term is given. It is possible to take other forms of penalty terms [100]. Early stopping is another important technique to prevent overfitting of ANNs. It stops the training phase of the network after a certain number of epochs and stops further adoption on the training data [149]. It is not a regularization technique, but has the same purpose.

Neural networks have become the tool of choice in natural language processing in recent years.

CNN

A short explanation of the basic idea behind a CNN will be given. A CNN consists of convolutional and pooling layers. The input data is passed into a stringing of one or more convolutional layers, which are followed by a pooling layer. This convolutional-pooling layer structure can be repeated to create deeper configurations that form a deep CNNs. A convolutional layer applies a filter pattern to the input data. This helps to map the input data to an underlying feature map and to find local connections between them. The filter pattern reduces the number of weights in the network by mapping parts of the input to one feature. Furthermore, the pooling layer shrinks the feature map and merges semantically similar features together [105]. A convolutional layer can consist of more than one filter pattern. Convolutional Neural Networks (CNNs) were originally developed for computer vision, e.g. for image analysis. Since 2011 they have become more and more popular in NLP [38]. In 2014 Kim [92] and Kalchbrenner et al. [86] successfully used CNNs in NLP tasks such as sentence classification and sentence modeling. More complex ANNs, such as Recurrent Neural Networks (RNN), which are also important in NLP, are discussed in the next section.

RNN

Language, since we read words in order, can natively be thought of as a sequence. Thus, modelling language through a FNN like above loses some important language information about which words come in which order in the sequence. MLPs handling of sequences of data is poor, as it is equivalent to trying to understand a sentence while forgetting each word while reading. Added to this, text has rarely a norm for the length and cannot be split up easily while keeping the original information within the split.

Hopfield [71] and Rumelhart [164] solved this by feeding the previous hidden state (see figure 2.4) in the network back to the current state. This allows the network to keep a form of internal memory of what has been seen previously in the sequence. This solves two problems at once: Firstly, one can now model dependencies back in time as the model takes into account what it has seen earlier. Secondly, one now have a structured way of handling different lengths of inputs, as one can keep running each new data point into the model for the entire length of the sequence. RNNs solved these and quickly became the de facto standard for most forms of NLP using Neural Networks [109].

An RNN can also be seen as a feed-forward neural network with a dynamic number of hidden layers that are all set to have the same parameters. Rather than being "deep", the model is "wide" as it is unrolled through time. In contrast to a regular feed-forward neural network, however, it accepts a new input at every "layer" or time step. Specifically, the RNN maintains a hidden state h_t, which represents its "memory" of the contents of the sequence at each time step t. At every time step, the RNN performs the following operation:

$$h_t = \sigma_h(Ux_t + W_h h_{t-1} + b_h) \tag{2.81}$$

$$o_t = \sigma_o(Vh_t + b_o) \tag{2.82}$$

$$\tag{2.83}$$

where σ_h and σ_o are activation functions. The RNN applies a transformation W_h to modify the previous hidden state h_{t1} and a transformation U to the current input x_t, which yields the new hidden state h_t. At every time step, the RNN furthermore produces an output o_t. The figure 2.2.4 illustrates how information from the state of

the network is passed to the next state containing information from previous states, called a 'summary'. The unfolded representation shows the states of the network and the flow of information for three consecutive points in time. To know its history the neural network in current state s_t obtains the summary W from the previous state s_{t-1}. Suppose a training of an RNN in an unsupervised way. Hence, the model trains on unlabeled real-world texts. For each stem it is asked to predict the output word o_t based only on all k previous words x_{t-k},...x_{t-2},x_{t-1}. The prediction is compared to the correct word, if these are not equal the loss is backpropagated. The backpropagation is then able to "change history' to improve prediction accuracy.

[h]

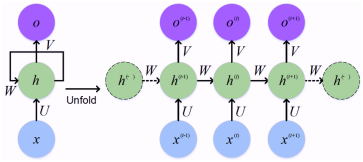


Figure 2.2.4: : Recurrent neural network unfolded in time.

The benefit of this architecture is that information is compressed inside the neural network. Also, these is a certain sence of importance since the weights are not uniformly updated for allprevious states (words). For example, "the car which is green, broke". For this sentence, the word "broke" can more easily be predicted based on the "the car" than on "which is green". It is found that RNNs are able to capture this importance. [173].

In practice, the RNNs are not using the complete history. The cause for this is vanishing gradients. RNNs has trouble learning over a large number of time steps. Training it through Backpropagation has a tendency to become hard, as their gradient either becomes vanishingly small or very large, making convergence unlikely. Vanishing or "exploding" gradients [69] make RNNs difficult to train well."In practice gradients vanish ans simple RNNs become like 7-grams for most words" [173]. Additionally, these models has a tendency to become large, especially in NLP, and therefore takes considerable computational resources to train. There has been some developments in order to improve convergence when training RNNs. Some of the same regularization algorithms used for general MLPs can be carried over, such as L1 [183] or L2 [184] regularization. Others, like Dropout [176] may even be considered very important for some models' convergence [72].

Long Short-Term memory Network LSTMs

Long short term memory networks (LSTMs) were proposed in 1997 and were developed to avoid the drawbacks of plain RNNs [70]. LSTM networks are preferred compared to RNNs for dealing with sequential data as they can retain information for longer time spans, which is necessary for modelling long-term dependencies in common neural language. The LSTM can be seen as a more sophisticated RNN cell that introduces mechanisms to decide what should be ”remembered” and ”forgotten”. The LSTM augments the RNN with a forget gate f_t, an input gate i_t and an output gate o_t, which are all functions of the current input x_t and the previous hidden state h_t. These gates interact with the previous cell stat c_{t-1}, the current input, and the current state c_t and enale the model to selectively retain or overwrite information. So basically, the improvement in a LSTM is the introduction of a forget mechanism. Besides saving information about the last input, they are able to forget earlier processed instances stored in their internal state. Moreover, LSTMs also have another input c_{t1} and output c_t which is called cell state. The LSTM consists of several gates: the input gate, forget gate and output gate. The gate mechanism works similar to a water faucet since the gates regulate the values passed through the LSTM. All gates are based on sigmoidal activation functions.

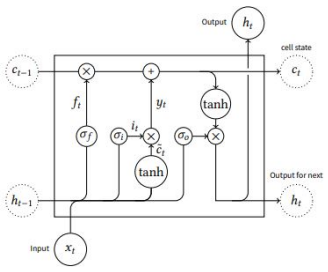


Figure 2.2.5: Architecture of a recurrent cell in a Long Short-Term Memory Network (LSTM) with three different gating mechanisms σ_f, σ_i and σ [20]

The entire mode is defined as follows:

where σ is the sigmoid activation function and $\circ$ is element-wise multiplication, also known as *Hadamard product*.

The Forget Gate: The forget gate enables the LSTM to drop information from cell state c_t . By looking at the input x_t and the previous hidden state h_{t1}, it calculates a value between 0 and 1 for every entry in the cell state c_{t1}. This number indicates whether the

corresponding entry should be kept (1) or deleted (0).

$$f_t = \sigma_f(W_f[h_{t-1}, x_t] + b_f) \tag{2.84}$$

(2.85)

Thereby f_t is the result of the forget gate, W_f and b_f represent the corresponding weights and biases. The output is then multiplied with the previous cell state c_{t1}, as shown in Figure 2.2.5.

The Input Gate: Like the forget gate, the input or update gate σ_i combines the previous internal state h_{t1} and the current input x_t to it . A layer with sigmoid activation decides on the factor they should be included in the new internal state c_t:

$$i_t = \sigma_i(W_i[h_{t-1}, x_t] + b_i) \tag{2.86}$$

(2.87)

The actual new value $\sim$ c_t is generated by a tanh layer with the same two input nodes:

(2.88)

$$simc_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.89}$$

(2.90)

Both results are multiplied and added to the internal state c_{t1} from time step t – 1, which is already filtered by the forget gate:

$$c_t = f_t \circ c_{t-1} + i_t \circ \sim c_t \tag{2.91}$$

(2.92)

Therefore, the input gate works as a regulator over the input sequence. For further calculations c_t is passed on to the next time step and serves as input for it.

The output Gate: The overall output ht of a LSTM-cell is finally modified by the output layer gate σ_o. The cell state from equation (2.92) is taken and passed through another tanh-layer. Similar to the gates explained above, the output gate filters candidates according to h_{t1} and x_t. Moreover, it weights the instances to be removed from the

possible results:

$$o_t = \sigma_o(W_o[h_{t-1}, x_t] + b_o) \tag{2.93}$$

$$\tag{2.94}$$

The calculation of the next state h_t is done by:

$$h_t = o_t \circ tanh(c_t) \tag{2.95}$$

$$\tag{2.96}$$

A LSTM can be applied to any sequence with the length of n. The final result is then represented by the last hidden state of the sequence h_n. In practice it could be necessary to append a linear layer of a FNN to transforms the hidden state into a desired outcome.

Bidirectional long-short Term Memory Network

When modelling language, one must recognize that not all words are predictable by the word before. Consider the sequences “river bank” and “bank account”, “bank” has a contextual dependency in the two sequences, not only on the previous word, but on the word after. This is a problem that regular recurrent networks can struggle with. This problem can be solved however through bidirectionality proposed by Graves and Schmidhuber [60]. This simply means reversing direction of the sequence and feeding this to an independent network and concatenating the resulting hidden states. Bidirectional RNNs were introduced in 1997 by Schuster and Paliwal [168]. Bidirectional LSTMs are based on them and extend the standard LSTM model. The basic idea is to process the sequence x = $x_1, \ldots, x_t$ forwards and backwards. Therefore, a bidirectional LSTM has a forward and backward part.

LSTM cells can be stacked in multiple layers.In most cases, a bidirectional LSTM [59] will be used, which runs separate LSTMs forward and backward over the sequence. The hidden state h_t is the concatenation of the hidden states from the forward and backward LSTMs at time step t:

$$h_t = [h_{fwd}; h_{bwd}] \tag{2.97}$$

In figure 2.2.6, you have extended the simple RNN to work bidirectionally. The states (h_p) in figure 2.2.6 may be from regular recurrent nodes or LSTMS. The Network with the reversed states is simply an independent copy of the original Network. Commonly, the resulting hidden states are concatenated to form:

$$h_p = (h_p - - >, h_1 < --) \tag{2.98}$$

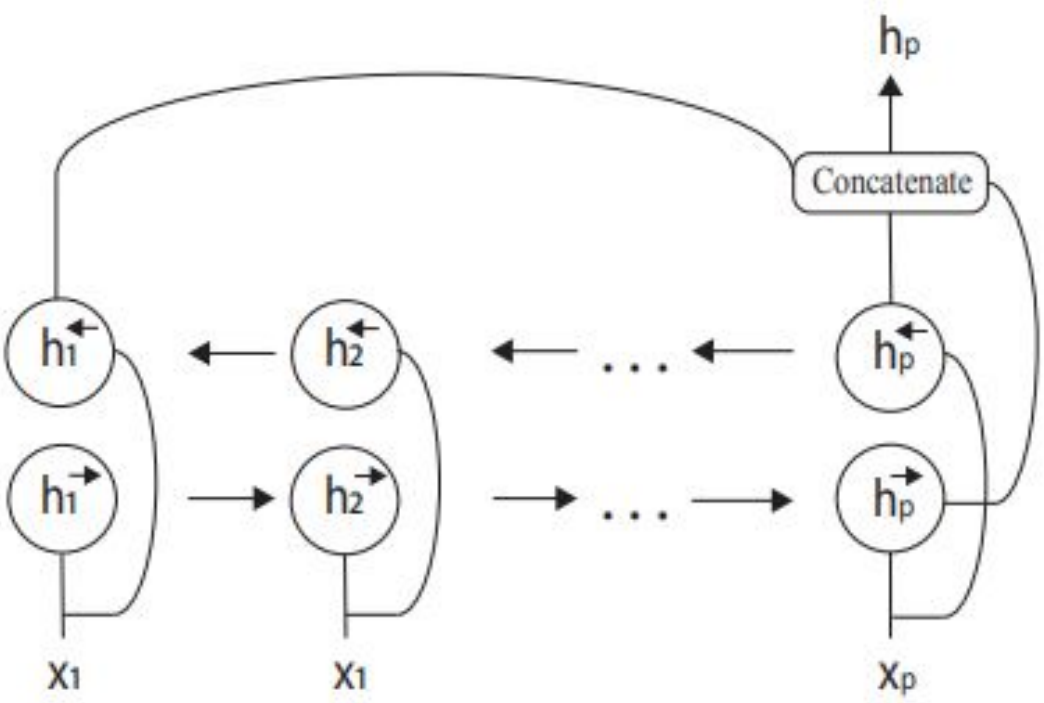


Figure 2.2.6: Bi-directional recurrent neural Network

This final hidden state vector can then be used in modelling in the same way as for simple RNNs.

Their hidden states h_{fwd} and h_{bwd} are connected to the output layer by a FNN. The forward part of the network processes $x_1, \ldots, x_t$ and the backward part $x_t, \ldots, x_1$ [59]. Using the last hidden states the outcome y_t at the output gate is computed:

$$y_t = W_{h_{fwd}y} h_{fwd_t} + W_{h_{bwd}y} h_{bwd_t} + b_y \tag{2.99}$$

By doubling the number of weights, bidirectional dependencies can be modeled. In terms of computational capability, constant scalar increases of complexity is usually not a problem. Given that it adds contextual information to language models, this is often a method employed to improve model representation and predictions.

Autoencoder

An autoencoder is a neural network that is trained to reconstruct its input by compressing it first into a low-dimensional representation. In the simplest case with one hidden layer, it uses an encoder to map the input x to a vector z:

$$z = \sigma(Wx + b) \tag{2.100}$$

where σ is an activation function. a decoder than maps the latent representation back to reconstructed version of the original input x':

$$x' = \sigma'(W'z + b') \tag{2.101}$$

The model is traind to minimize a recontruction loss such as the squared error between the original and the reconstructed input

$$\ell = \left\| x - x' \right\|^2 \tag{2.102}$$

Autoencoders are commonly used to learn representations and have been used in sequential transfer learning 3.2 , domain adaptation.

2.2.3 Training Neural Networks

Reminder: There are two main types of training in machine learning, namely supervised and unsupervised as explained. We utilize both learning techniques, unsupervised learning in the extractive approach and supervised learning in the abstractive approach. In supervised learning, the goal is to train a network on classifying for a specific task, such that it learns to generalize for unseen data. This is done by iteratively updating the weights of a network until finding a set of optimal weights that minimize or maximize some objective function.

Analogous to many machine learning models where closed-form solutions are not available, neural networks are typically trained with stochastic gradient descent. As each model consists of multiple layers, calculating the gradient of the loss function with regard to the parameters $\nabla_\theta J(\theta)$ is non-trivial. To compute the gradient, dynamic programming algorithm is used known as back-propagation [164]

Back-propagation is based on a chain rule of calculus, where given functions y = g(x) and z = f(g(x)) = f(y) defines the derivative $\frac{\partial z}{\partial x}$ of z with respect to x as the derivative of z with respect to y times the derivative of y with respect to x:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{2.103}$$

For factors $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$, and scalar z, one analogously obtain the partial derivative of z with respect to x_i as follows:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{2.104}$$

In vector notation, the gradient of z with respect to x containing all partial derivatives of z with respect to each x_i can be determined using matrix-vector multiplication:

$$\nabla_{x^z} = (\frac{\partial y}{\partial x})^T \nabla_{y^z} \tag{2.105}$$

where $\frac{\partial y}{\partial x} \in \mathbb{R}^{\kappa * >}$ is the Jacobian matrix of g, the matrix containing all partial derivatives. The back-propagation algorithm now performs such a Jacobian-gradient product for each operation in our neural network graph [57]. Let us define a deep feed-forward neural network with L layers, with weight matrices W_l and bias parameters b_l where $lin1, ...L$. The model takes an input x, produces an output $\hat{y}$ and minimizes a cost function $J = L(\hat{y}, y)$:

$$a_l = b_l + W_l h_{l-1} \tag{2.106}$$

$$h_l = \sigma_l(a_l) \tag{2.107}$$

$$\tag{2.108}$$

where $h_0 = x$ and $\hat{y} = h_L$. Forward-propagation proceeds from the first layer,

computing the representation a_l that is then fed through an activation function σ_l, which yields a hidden state a_l, which is provided to the next layer. This is repeated until the output is $\hat{y}$ and the loss J is calculated.

Back-propagation proceeds in backwards order: It starts by computing the gradient $\nabla_{\hat{y}} J$ of the loss function J with respect to the output $\hat{y}$. It then computes the gradient of the representation $h^{(l)}$ and a^l for the last layer, from which it then obtains the gradients on the parameters of the layer. It then continues until it arrives at the gradients of the first layer. The gradient $\nabla_{\hat{y}} J$ of the loss with respect to the output is:

$$\nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y) \tag{2.109}$$

If a squared error loss is used, the gradient is simply:

$$\nabla_{\hat{y}} J = \nabla_{\hat{y}} \frac{1}{2}(\hat{y} - y)^2 = y - \hat{y} \tag{2.110}$$

The gradient of the loss function is obtained with regard to the layer's pre-activation representation $\nabla_{a_l} J$ with the chain rule:

$$\nabla_{a^{(l)}} J = (\frac{\partial h^{(k)}}{\partial a^{(k)}})^T \nabla_{\hat{y}} J = \sigma'(a^{(l)}) \tag{2.111}$$

As our activation function is elementwise:

$$\nabla_{a_l} J = \sigma'(a_l) \circ \nabla_{\hat{y}} J \tag{2.112}$$

This demonstrates why it is desirable for functions to be differentiable. The sigmoid activation function, for instance, has a convenient derivative:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.113}$$

From this, one can now obtain the gradients of the model parameters based on each

parameter's contribution:

$$\nabla_{b_l} J = \nabla_{a_l} J \tag{2.114}$$

$$\nabla_{W_l} J = \nabla_{a_l} J h_{l-1}^T \tag{2.115}$$

then subsequently propagate the gradient to the next lower-level layer and repeat the calculations:

$$\nabla_{h_{l-1}} J = W_l^T \nabla_{W_l} J \tag{2.116}$$

As you can see, in order to compute the gradients on a parameter, the gradients on all computations that are the result of this parameter need to be known. For each layer, the computed gradients are then used to update the corresponding parameters with gradient descent. When optimizing RNNS the gradient needs to be propagated through the past time steps rather than through the depth of the model. Consequently, this is known as back-propagation through time. As the error cannot be back-propagated indefinitely, it is generally propagated until a fixed length that is defined in advance. Longer lengths should enable the model to learn longer-range dependencies.

A common issue encountered when optimizing RNNs in practice is known as exploding of vanishin gradients. During forward propagation, the hidden state is multiplied many times wit hthe weight matrix, once per time step; During back-propagatio, this leads the gradients being multiplied with the same calues over and over. This causes the calues to either explode, i.e. become very large or vanish, i.e. become very small, rendering the model unable to learn. Mitigating the exploding and vanishing gradient problem is of the motivations for the development of the LSTM as explaind in 2.2.2.
Objective Function: When training a neural network, the goal is to maximize or minimize some objective function. A loss function is a type of objective function that is to be minimized [8]. The purpose of a loss function is to measure how well a model predicts the expected outcome for any data point in the training set. Cost function is the term for the performance measure evaluated on the whole training set. A common loss function is the cross-entropy error function as seen in equation presenting the cross entropy, where P is the target distribution and Q is the distribution of the network's

predictions. It is a commonly used objective function in neural networks.

$$https://odr.chalmers.se/bitstream/20.500.12380/300416/1/CSE \quad (2.117)$$

vanishing gradient

2.2.4 Summary

An artificial neural network (ANN) is a computing system loosely inspired by the functions of a human brain. It is constituted of neurons in different layers and connections between them, where each connection holds a certain weight. The essential structure of an ANN includes an input layer, output layer and a number of hidden layers between the input and output layer.

An artificial neuron is a computational unit defined by an activation function and its connecting weights. The purpose of the activation function is to decide if a neuron should be active or not. It does this by taking as input a linear transformation of the outputs from neurons in the previous layer and the weights. The idea is that a neuron activates on certain input patterns that it has learned. Learning is done by altering the weights of the neurons.

The basic ANN is the feedforward neural network (FFNN), or multilayer perceptron (MLP). In an MLP, information flows forward from the input layer through the intermediate layers and to the output layer. For an input vector x, the first hidden layer, h (1), in a network computes h (1) = g(w(1)x + b), where g is some activation function, w(1) is the weight vector in the first hidden layer and b is a bias term. These computations continue in the subsequent layers of a network until the output has been computed. An intermediate layer, such as h (1), can be seen as a vector representation of the input x to an MLP. Neural word embeddings are an example of this. Two common ANNs are convolutional neural networks (CNNs), often used in computer vision, and recurrent neural networks (RNNs), which have been seen more in the field of NLP.

Chapter 3

Natural Language Processing: Theory and related work

Natural language processing (NLP) is the main research area of this thesis. It is related to other fields such as Artificial Intelligence (AI) and Machine Learning (ML).

3.0.1 Definition and distinction

Natural language processing (NLP) aims to read text or listen to speech and extract meaningful information from it. One deviation from this definition is natural language generation, where text is generated. The difficulty of NLP is that humans for every word activate “a cascade of semantically related concepts, relevant episodes, and sensory experiences” [27]. These activations may also differ per context. For example, the activations for “I want more money.” depend on whether the sentence is uttered by a child or employee. The field is associated with artificial intelligence (AI) and Machine Learning (ML). AI is a very boad term and is a way to describe systems that are able to ”think” [76]. In literature, there are many different explanations which make it difficult to define this topic accurately and exactly. For this thesis it will be defined as “a system that has the ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation” by [87]. NLP deal with ”the use of human languages[...] by a computer” [57]. It does have many different applications which all refer to humans’ unstructured natural language. For example, its application areas are :

- Translating texts between (human) languages : **machine translation** - **Question**

answering which is given some context and a question processes question sentences only to point to an answer in the text or to generate an answer sentence. - Classifying words or parts of sentences is done by part-of-speech tagging (for example nouns and verbs) and **Named Entity Recognition** (for example dates, locations, organisations). - **Optical Character recognition** attempts to recognize characters in images and can use NLP knowledge to improve accuracy. - Finding co-references like ”house” and ”it” in the sentence ”The house is white, and it is located on a hill” is done using coreference resolution. - NLP is not limited to text, because if includes speech recognition which transforms speech to text. - Entailement classification contains examples such as ”people formed a line at the end of Pennsylvania Avenue” which is contained in ”logically implied by” ”At the other end of Pennsylvania Avenur, people began to line up for a White House tour” - Determining whether two sentences have the same meaning is called ”semantic text similarity” - **Sentence classification** is the broad tasks of classifying a sentence (for example, the sentiment or intention of user)

Intent classification, machine translation, and named entity recognition are discussed further in section 3.0.3 Thus, the domain of NLP encompasses all interactinos between a computer and a human by the use of written or spoken natural language. It is a research and application field, which is concerned with the manipulation and understanding of natural languages. This processing of human languag is based on understanding the intended meaning of a message; which is difficult even for humans, e.g. when irony or special expressions are used. All components of natural language, such as phonetics, phonology, morphology, syntax, semantics and pragmatics must be taken into account in order to gain complete understanding of a message **Phonetics** is about the acoustic properties of a sound produced by the human vocal tract. It examines how sounds are physically constructed, e.g. with the tongue or the lips. The sound of a particular human language is studied by phonology. For example, the English language has 45 distinguishable sounds called phonemes. Phonetics and Phonology are particularly important aspects in speech recognition when converting sounds into real words that can be processed by a computer. **Morphology** concerns about the meaning and the architecture of words. Stemming and lemmatization, which are described below, are based on this component by transforming words like ”going” back to their word stem ”go”.

syntax: The ordering of words and the building of grammatical correct sentences

is investigated by the syntax. **semantics**: In contrast, semantic examines the meaning of sentences that are constructed by the use of syntax and morphological word forms.

pragmatics: To obtain the intended overall meaning of a message, pragmatics uses the context of the situation. For example, let us assume, somebody asks: ”Could you pass the salt?”. Given the context of the situation, the question is actually a request to pass the salt and not a question if someone is able to do that [24]. Therefore, a computer needs to take all parts of natural language into account to use it

By this definition modern NLP approaches are AI, specifically artificial narrow intelligence [87].

Natural language processing (NLP) aims to teach computers to understand natural language. As the facility for language is abstract, let's take a more concrete view by defining NLP by way of its tasks, which generally map a text to linguistic structures that encode its meaning [172]. The machine learning tools presented in the previous section will be used to learn a mathematical model of this mapping. Specifically, there will be an aim to train a model that can map from an input x consisting of a sequence of words to an output y generally using the principle of maximum likelihood estimation to find a set of parameters θ that maximizes the conditional probability $P_\theta(y|x)$ of our model. We focus on discriminative models that model the conditional probability directly from raw data rather than using a generative model that learns the joint probability distribution P(x, y).

3.0.2 Feature Selection and Preprocessing

Feature selection and preprocessing are significant tasks in Artificial Intelligence . Especially in NLP, this task does have tremendous impact on the success of text analysis [6]. This is mostly caused by the unstructured and arbitrary nature of text data. Furthermore, machines need structure and numerical data. A couple of approaches for this transformation task, e.g. word embeddings or the vector space model, exist. This section's scope lies on the theoretical foundation of different preprocessing and feature selection techniques. This section will be accompanied by the English phrase ”the worst virus is spreading” as an example to illustrate the application of preprocessing. Nevertheless, every routine should be used with care. It is not always the case that a reasonably good preprocessing method leads to better results in every application

[61].

Tokenization

For processing written natural language it is inevitable to split texts into smaller units, which are called tokens. Computers need to distinguish single entities of a text and tokenization is used to create them. Usually tokens represent simple words, which are the smallest independent units of natural language [157]. Furthermore, tokens can consist of idioms or a hyphens, e.g. ”user-generated”. Tokenization breaks running texts into short text entities and is the very first task in any text preprocessing cycle [77] . Besides the partition of small units, whole sentences can also be the output of a tokenizer. A simple word tokenizer can be realized in many languages by splitting the text at the occurrences of space symbols. This simple baseline approach does have a couple downsides, due to the lack of identifying words that semantically belong together [191]. However, a simple tokenizer divides the phrase, which was introduced above, is into the following five tokens: ”The”, ”worst”, ”virus”, ”is”, ”spreading”

Stop Word Removal

A very important approach to reduce the huge raw input space in NLP is stop word removal (swr). Most languages have specific words, which do appear more often than others or do not include much information about the content of the text, e.g. auxiliary verbs or articles [52]. Due to this, it often makes sense to exclude this so-called stop words in further analysis. In English such words could be ”the”, ”a” or ”an” and for French typical stop words are the articles ”le”, ”pour” and ”la”. The elimination could be done by checking the words against a standardized stop word list. These lists are available in literature and are often implemented in different software packages [44]. In our example, ”the” and ”is” are eliminated. Stop word removal should be used with care, especially in sentiment analysis, which attempts to predict a positive or negative intention of a text. The removal would exclude words that are able to change a whole statement, such as ”not” or ”none”.

Stemming

Besides stop word elimination, stemming is a useful technique to map words to their word stems and further reduce the input dimension. This helps to extract the real

meaning of a text and makes the unstructured data better accessible for a machine. The first stemming algorithm based on deleting longest suffixes and spelling exceptions was developed in 1968 [115]. By now, the porter stemming algorithm is a state-of-the-art approach and strips suffixes from words to retain the word stem [148]. While this method performs well in English, there are some drawbacks for the French language, due to the fact that French words are not usually build by adding suffixes. By using the English Porter Stemmer the words ”best”, ”fox” and ”running” are assigned to the following words: worst –> worst, virus –> virus, ”spreading” –> spread

Lemmatization

Lemmatization is the process of mapping every word in a text to their dictionary type or intended originating structure. Verbs are transformed to their infinite form, a noun is reconstructed to it's singular representation and adverbs or adjectives anticipate their positive format [110]. The method is based on morphological analysis and often uses a dictionary, for example WordNet[53], where the lemma of every modified word form could be retrieved. This preprocessing step is similar to stemming and reduces the input space, by mapping different word forms to their common representation. Since lemmatization is supported by dictionary entries, it is able to map ”worst“ to its lemma ”bad“: worst –> bad, virus –> virus, running –> run

Vector Space Model : Bag-of-words

Besides, preprocessing the words themselves, their representations have to be changer into a machine readable format. Meanwhile, a couple of different approaches have been developed to transform text into different kinds of numerical representations. Some of them only represent statistics of a word, such as the one-hot-encoding, and other formats include the word's context. This representation is the main field of research discussed in this thesis.

The Vector Space Model is an approach that transforms a text into one vector. It is based on one-hot-encoding of words. Given a set of textual documents (corpus), it is possible to create a vocabulary with the length of N, the N unique words in the total corpus. The one-hot-encoded word vector represents a word by a vector of length N with 1 at the corresponding vocabulary entry. For example, if the term ”virus” is the i-th uniqye word in the corpus, the vector has the length of N with 1 at the i-th position

and all other entries are 0:

$$\textit{one-hot}(virus) = (0, ..., 1, ..., 0)\textit{at the i-th position} \tag{3.1}$$

The vector space model extends this model to documents. The function ϕ maps any document d to its vector space representation. This distinct words, which are called terms, in the vocabulary are represented by t_1,...,t_n.

$$\phi : d - - > \phi(d) = (tf(t_1, d), (tf(t_2, d), ..., tf(t_n, d)) \in \mathbb{R}^N \tag{3.2}$$

ϕ count the occurrences of each term (tf) in the vocabulary per document. Therefore, the document vector can have more than one entry that is not 0. The function $tf(t_i, d)$ how often the i-th vocabulary word appears in the document d. Besides simply using the term frequency (tf), it is also possible to give every particular word a weighting, according to its relative appearance in the corpus. This could be done by using term frequency divided by Inverse Document Frequency (tf-idf), which gives less meaning to common words in a corpus. To calculate the tf-idf for a term in a document, a normalized term frequency must first be determined. The relative normalized term frequency (ntf_{rel}) is defined as:

$$ntf_{rel}(t_i, d) = \frac{tf(t_i, d)}{\sum_{t_m \in d} tf(t_m, d)} \tag{3.3}$$

It calculates the term frequency of the i-th term t_n in d and divides this by the sum of all other term frequencies in d. The maximal normalized term frequency (ntf_{max}) puts the highest term frequency in d into the denominator:

$$ntf_{rel}(t_i, d) = \frac{tf(t_i, d)}{mac_m tf(t_m, d)} \tag{3.4}$$

Both equations can be used for retrieving an adjusted term importance throughout a particular document d. For incorporating the word relevance with regard to the whole corpus, it is necessary to calculate the Inverse Document Frequency (IDF):

$$idf(t_i) = log(\frac{|D|}{|d : t_i \in D|} \tag{3.5}$$

|D| is the total number of documents in the corpus and the denominator represents the occurrence of the term t_i in all documents. For frequent terms in the corpus, |d : $t_i \in$ D| gets big, which results in fraction closer to 1. By reversing the Document Frequency with the logarithm, the common words are given a lower weighting, because they have less distinctiveness than rarer words. The actual weight w for every word in the document is calculated by the product of normalized term frequency and inverse document frequency:

$$w(t_i, d) = idf(t_i).ntf(t_i, d) \tag{3.6}$$

In some contexts, text will be represented as a bag-of-words (BOW) $x \in \mathbb{R}^{|V|}$ where V is the vocabulary. Each entry x_i corresponds to the number pf occurrences of the i-th word in the vocabulary in the text. This frequency may be weighted with *term frequency-inverse document frequency* (tf-idf), which addictionally reflects how important a term is in a *corpus*, a collection of texts. An *n-gram* is a contiguous sequence of n words in a text. In the standard BOW model, only *unigrams* are considered, sequences of one word. *bigrams* may additionally be considered, sequences of two words. The bag-of-words ignores *grammar* and word order. To capture compositionality and dependencies in the input, throughout this thesis, our preferred way pf representing a sequence of words w_1,...,w_T will be to encode it as a sequence of the corresponding word embeddings x_1, ..., x_T

By using the Vector Space Model, the sparsity of the document vectors could be one major problem, due to the fact that N -length(d) positions are 0. Another issue is that the distance between two different document vectors is very small (curse of dimensionality). Therefore, it is not easy to distinguish between documents, especially when it comes to grouping similar documents, e.g. in clustering.

Evaluation Metrics

NLP systems are typically evaluated with regard to their performance on the test set of the specific task. For binary classification, accuracy is the common evaluation

measure, which is defined as follows:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.7}$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives and false negatives respectively. Intuitively , the number of true predictions is divided by the number of all predictions. For multi-calss classification and even in binary classification there is a need of a more detailed metric to observe the performance per class. The F1 score is used, which is the harmonic mean of precision and recall:

$$F1 = 2.\frac{P.R}{P + R} \tag{3.8}$$

$$Precision : P = \frac{TP}{TP + FP} \tag{3.9}$$

$$Recall : R = \frac{TP}{TP + FN} \tag{3.10}$$

The F_1 metric balances precision, the fraction of correctly predicted instances and recall, the fraction of correctly predicted instances out of all instances of the category. It thus provides a good estimation of the overall quality of a model. Certain tasks may use specialized evaluation metrics, which will be introduced with the corresponding task.

Let's review the main NLP tasks.

3.0.3 NLP Tasks

Part-of-speech (POS) tagging

POS tagging is the task of tagging a word in a text with its corresponding part-pf-speech. A part-of-speech is a category of words with similar grammatical properties. Common English parts of speech are noun, verb, adjective, adverb, pronoun, preposition, conjunction, etc. Part-of-speech tagging is difficult as many words can have multiple parts of speech. Parts of speech can be arbitrarily fine-grained and are typically based on the chosen tag set. The most common tag set used by the Penn Treebank [118] comprises 36 tags. However, POS tags vary greatly between languages due to cross-lingual differences. The creation of a "universal" tagset has been an

ongoing development: Petrov et al. [123] proposed a tag set of 12 coarse-grained categories, while the current tag set of the Universal Dependencies 2.0 [134] contains 17 tags. When applying a POS tagger to a new domain, current models particularly struggle with word-tag combinations that have not been seen before.

Chunking

Chunking, also known as shallow parsing, aims to identify continuous spans of tokens that form syntactic units. Chunks are different from parts of speech as they typically represent higher order structures such as noun phrases or verb phrases. Approaches typically use BIO notation, which differentiates the beginning (B) and the inside (I) of chunks. O is used for tokens that are not part of a chunk. Both POS tagging and chunking act mostly on the grammatical and syntactic level, compared to the following tasks, which capture more of the semantic and meaning-related aspects of the text.

Named entity Recognition NER

NER is the task of detecting and tagging entities in text with their corresponding type, such as PERSON or LOCATION. BIO notation is also used for this task, with O representing non-entity tokens. Entity categories are pre-defined and differ based on the application. Common categories are person names, organizations, locations, time expressions, monetary values, etc. NER is a common component of information extraction systems in many domains. Despite high numbers (≈ 92–93 F1) on the canonical CoNLL-2003 newswire dataset [165], current NER systems do not generalize well to new domains. NER corpora for specialized domains such as medical and biochemical articles [91] have consequently been developed.

Semantic role Labeling (SRL)

SRL aims to model the predicate-argument structure of a sentence. It assigns each word or phrase to its corresponding semantic role such as an agent, goal, or result. The FrameNet project [13] defined the first large lexicon consisting of frames such as Apply heat and associated roles, known as frame elements such as Cook, Food, and Heating instrument. Predicates that evoke this frame, e.g. “fry”, “bake”, and “boil” are known as lexical units. PropBank [94] added a layer of semantic roles to the Penn

Treebank. These semantic roles are specific to each individual verb, but employ the same tags, which—inspired by Dowty [48]—start from Arg0 that roughly indicates a proto-Agent, to Arg1, which designates a proto-Patient, etc., and are usually used in SRL applications.

Dependency parsing

Dependency parsing is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between "head" words and words, which modify those heads. Dependency parsing differs from constituency parsing, which focuses on identifying phrases and the recursive structure of a text. A typed dependency structure labels each relationship between words, while an untyped dependency structure only indicates which words depend on each other. Dependency parsing is used in many downstream applications such as coreference resolution, question answering, and information extraction as the relations between the head and its dependents serve as an approximation to the semantic relationships between a predicate and its arguments.

Topic Classification

Topic classification and sentiment analysis are text classification tasks. They assign a category not to every word but to contiguous sequences of words, such as a sentence or document. Topic classification aims to assign topics that depend on the chosen dataset, typically focusing on the news domain. As certain keywords are often highly predictive of specific topics, word order is less important for this task. Consequently, traditional Bag of words (BOW) models with tf-idf weighted unigram and bigram features are often employed as strong baselines.

Sentiment analysis

Sentiment analysis is the task of classifying the polarity of a given text. Usually this polarity is binary (positive or negative) or ternary (positive, negative, or neutral). Most datasets belong to domains that contain a large number of emotive texts such as movie and product reviews or tweets. In review domains, star ratings are generally used as a proxy for sentiment. Sentiment analysis has become one of the most popular tasks in NLP. Variants of the task employ different notions of sentiment and require

determining the sentiment with respect to a particular target.

Language modelling

Language modelling aims to predict for each word in a text the next word. It is generally known as an unsupervised learning problem, as it only requires access to the raw text. Despite its simplicity, language modelling is fundamental to many advances in NLP and has many concrete practical applications such as intelligent keyboards, email response suggestion, spelling autocorrection, etc.

3.0.4 Language modelling

Language modelling is the task of predicting a sequence of text's probability, often conditioned on being the successor of a given text. People often use language models to predict the next word in a sequence given its predecessor. This can also operate on several levels, e.g., character, word or even sentence. This is generally considered to be an unsupervised task since the data required to train a language model is raw text.

Text Representation

When dealing with natural language, the textual information needs to be represented in some way such that it can be interpreted by a machine. A common way to represent text is by using the previously defined 3.0.2 the Vector SPace Model assuming the Bag-of-words assumption of word embeddings, such as word2vec [125] or GloVe [143], and large vocabularies that will be explained further. However, even with big vocabularies, there is a possibility that out of vocabulary (OOV) words occur. One way to overcome OOV was presented by Sennrich et al. [170]. They proposed to work with subwords instead of words using byte pair encoding (BPE). This method creates embeddings internally for these subword units, and it is explained in section **??**.

Sequence-to-Sequence

The purpose of a sequence-to-sequence (seq2seq) model, or encoder-decoder model, is to map an input of variable length to an output of variable length where the input and output length may differ. Proposed by Sutskever et al. [179], a seq2seq model is comprised of two parts: an encoder and a decoder. Both components contain several

recurrent units that take as input a single element. The encoder encodes the input sequence word by word by computing the hidden state h_i for each timestep i. It then passes the last hidden state h_n to the decoder which uses it as its initial state. The purpose of the final hidden state of the encoder is to give a representation of the whole input sequence in form of a fixed-length vector to the decoder. The decoder then uses the hidden state to generate output y_i and the next state s_i for each timestep. See figure 3.0.1

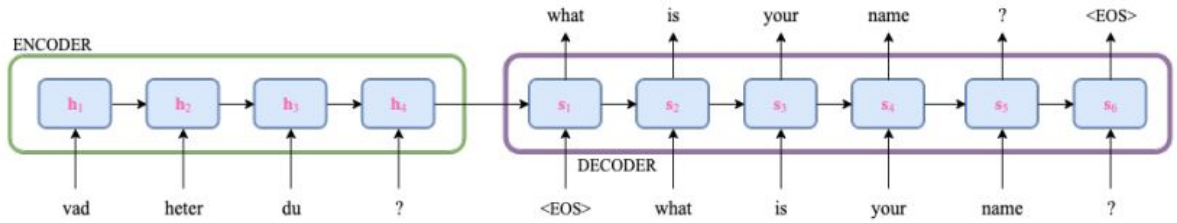


Figure 3.0.1: Sequence-to-Sequence model translating Swedish sentence to an English sentence

for a depiction of an encoder-decoder model. An issue arises when the input sequence is very long and the final hidden state of the encoder needs to encapsulate all of the information in a single fixed-length vector, which leads to loss of information.

Attention

Attention is a technique which was first proposed for RNNs to improve their contextual awareness. Essentially, it works by training a weighted representation of all hidden state vectors at the same time as the recurrence relationships are trained. This allows the network to look backward through the previous hidden states to highlight context information instead of needing to rely solely on the final hidden state representation from the RNN when making predictions. This can allow the model to pick up more subtle and complicated context information.

The attention mechanism was introduced by Bahdanau et al. [12] as a way to circumvent the issue of handling long sequences. Instead of letting the encoder compress the information into a single fixed-length vector the decoder makes use of a context vector c_i together with the previous state s_{i1} to generate the output for the current timestep i. The context vector is a linear combination of the hidden states, h_i, of the encoder and the attention weights α_{ij} for each timestep. The weight α_{ij}, seen

in Equation 3.11 where $e_{ij} = a(s_{i1}, h_j)$ and a is an alignment model parametrized as a FFNN, of each h_i is a probability that reflects how important the hidden state is with respect to the previsous hidden stat s_{i-1} in generating the output y_i

$$\alpha_{ij} = \frac{\exp\{e_{ij}\}}{\sum_k \exp\{e_{ik}\}} \tag{3.11}$$

This attention mechanism enables the decoder to decide which parts of the source sequence to focus on, instead of forcing the encoder to compress all the information into a single vector and passing it to the decoder. See Figure 3.0.2

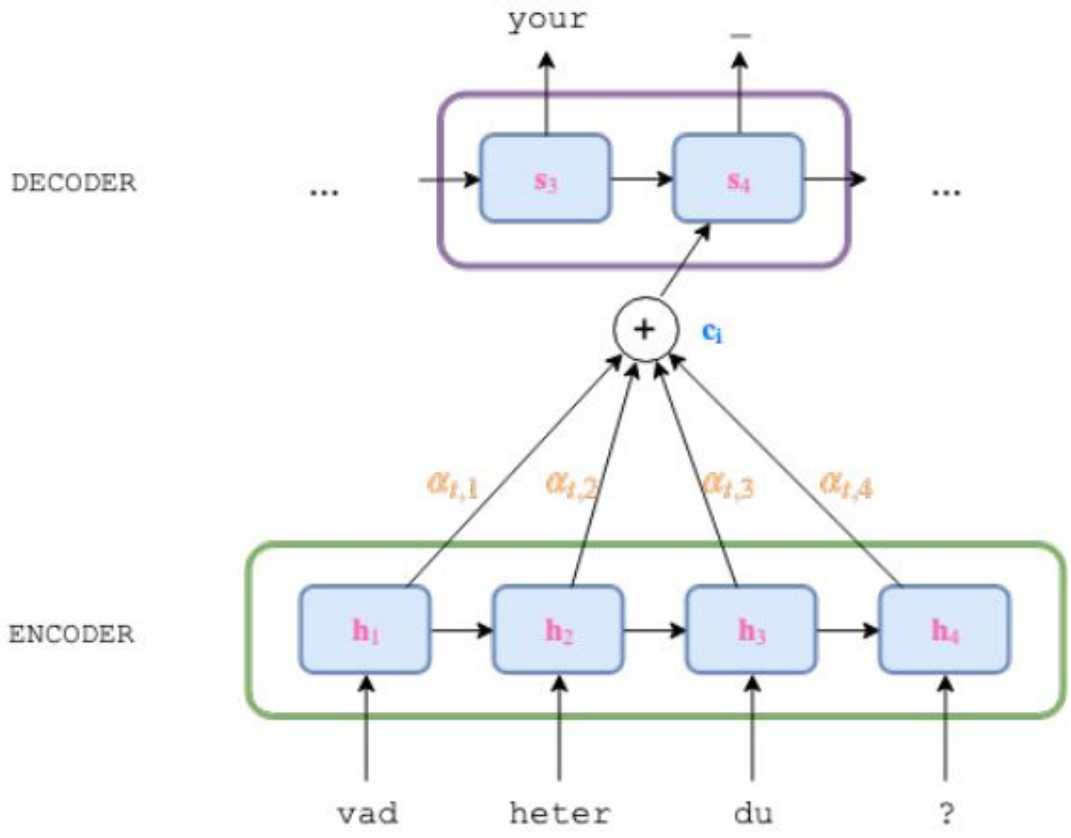


Figure 3.0.2: Attention mechanism illustrated with a translation task. Here, the model tries to predict the next word in the sequence, i.e.n 'name' provided the previous state and the context vector.

The use of Attention is one of the state-of-the-art approaches. It was first applied in machine translation by Bahadanau [12] in 2014. They use a bidirectional sequence to sequence LSTM in order to traslate text from a language to another. The attention

mechanism is close to human's attention. It helps shift the focus on the important part of an information sequence. On which part of how much should we bring attention depends on the task to do and the experience. Besides Bahdanau et al., other attention mechanisms have been introduced by Luong et al. []. The author proposed a local and a global attention system. The global approach is close from Bahdanau's system and includes all instances from a sequence. The local attention focuses only on the local context, aka the nearby words. *

Training Sequence-to-Sequence Models

A common way to train a neural language model is to use a technique called *teacher forcing*.[194]. It is utilized in architectures where the previously generated output is used as input to predict the next output, e.g, encode-decoder networks. Teach forcing is when the model produces an output y(t) that is not desirble, thus, instead of using y(t) as input to generate y(t+1) we feed the model the ground truth at timestamp t+1, i,e, the output we expected from the previous timestamp.

Evaluation: perplexity

A common way to evaluate a neural language model is to calculate its perplexity, which is a measure of how accurately the model predicts unseen data. It is defined as 2^{-l}, where entropy l is $l = \frac{1}{M} \sum_i \in [1, m] log(p(s_i)))$. Here, M denotes the total number of words in the test set and s_i is a sentence in the test set.

3.1 Vector Representation of Language

3.1.1 Traditional word representations

Vector Representation of Language

An essential step in using neural networks of any machine learning model is representing the language as vectors. let's discuss several methods for creating such a vector representatino of language and theory behind them. We can gather all of the words w_i if our vocabulary to form a bag of words as dicussed before.

$$\mathcal{V} = w_i : i \in 1, ..., N \tag{3.12}$$

For example, if the text sequence is "I love using NLP!" then the vovabulary would need to include an index for each of the words and symbols.

$$\mathcal{V} = I, love, using, NLP, ! \tag{3.13}$$

From there, the simplest vector representation x_i to distinguish between words is one-hot vectors which indicate which word of the vocabulary the vector corresponds to:

$$I = [10000] \tag{3.14}$$
$$love = [01000] \tag{3.15}$$
$$using = [00100] \tag{3.16}$$
$$NLP = [00010] \tag{3.17}$$
$$! = [00001] \tag{3.18}$$

However, simple one hot vectors are not a very useful input to most natural language processing tasks, because they are embedded in a vector space that doesn't contain any extra meaning information about the words beign represented nor the relations between them. In fact, the only thing the different vector dimensions mean are: Is this "Love"? Yes or no, but without capturing the expression in this word which is the main idea of the affirmation. Also, as vocabulary grows, one hot vectors become computationally unusable since each word needs its own seperate dimension, so $x_i \in \mathbb{R}^N$

3.1.2 Statistical Language Models and Word Embeddings

A more efficient method is to pass a lower dimensional vector $x_i \in \mathbb{R}^d$ (d < N) which is also embeds language information about the words w_i. This requires less computational power to feed to any NLP task (classification, NER...)as input, and contains more relevant language information than a one-hot vector. This type of lower dimensional representation is called a *word Embedding* since they embed words in a multi-dimensional meaning space where groups of semantically and syntactically similar words near to each other in terms of vector distance. One more important language feature which is also good for creating word embeddings is how words are

used in conjunction with each other, i.e. their co-occurrence. Through looking at the context that words appear in and the probability of observing a word given the observed context on can give a lot of insight to their syntactical and semantic use. This is called Statistical language Modelling.

In the example, ”I love using NLP!” above, there are clearly some words that are less likely to occur in place of ”using”. For syntax reasons, it would be very strange to see and adverb like ”quickly” in that position, and for semantic reasons it would be very strange to see an unrelated word like ”coronavirus”. Important aspects of word meaning, correct syntax and logical relationships between words and phrases can be captured throuh the probability of these words co-occuring [182] Thus a meaningful language feature is the pribability of observing a particular word w_i given the context sequence ${w_j}_{i-n}^{i-1}$ of n words that came before. This probability is called *n-gram* since it is based on an *n*-word wontext.

$$P(w_i|w_{i-1}, ..., w_{i-n}) \tag{3.19}$$

From a aaaaaaatext corpus and with a specific window, one can calculate the co-occurence probability of two words w_i and w_j. This co-occurrence probability, $P_{ij} = P(w_i|w_j incontext)$ is based on counts of occurrences of w_i with and without w_j in its conetxt window. These probabilities create a co-occurrence probability matrix $\check{c} \in \mathbb{R}^{(N*N)}$ for each word pair in the vocabulary. The row vecors of P are the co-occurrence probabilities of a word with every other word in the vocabulary and, theoretically, could also be used as a word-vector representation like the one-hot evectors above. This would encore more clanguage information that the one-hot vectors. However, it would run into the same computational problems with growing vocabulary size. Also, this matrix is relatively sparse, especially for words that don’t occur very often in the text corpus, so not all N dimentions are necessary to effeciently represent this data. Most traditional word embeddings are based on approximating P in lower dimentional space. A classic dimension reduction technique is *principal component alanysis (PCA)* based on the singular value decomposition of the co-occurrence matrix. This approach is appealing because it is unsupervised, meaning that no linguistic rules need to be fed to the system. Also, PCA looks globally at the text corpus for all of the occurrences of a word in all the different context, representing rich language features. However, with large N this technique becomes too computationally expensive.

Latent Semantic Analysis LSA

Latent Semantic Analysis [42] has been one of the most widely used methods for learning dense representations of words. Given a sparse word-word co-occurrence matrix C obtained from a corpus, a common approach is to first replace every entry in C with its pointwise mutual information (PMI) [35] score, thus yielding a PMI matrix P. P is factorized using singular value decomposition (SVD), which decomposes P into the product of three matrices:

$$P = U \sum V^T \tag{3.20}$$

Where U and V are in column orthonormal form and $\sum$ is a diagonal matrix of singular value. One subsequently obtain the word embedding matrix X by reducing the word representation to dimensionality k in the following way:

$$X = U_k \sum_k \tag{3.21}$$

Where $\sum_k$ is the diagonal matrix containing the top k singular values and U_k is obtained by selecting the correspoing columns from U. LSA is a form of dimensionality reduction: while using co-occurrence counts of words as features directly is impractical, reducing them to a lower dimension enables their use in many applications. Closely related to LSA is canonical correlation analysis (CCA), which has also been used to learn representations of words [46].

Brown clustering

One problem with classic language models is that features are extremely sparse as many word pairs are never observed in a corpus. Brown clustering [26] is a seminal unsupervised pretraining technique in NLP that seeks to combat this sparsity by assigning every word to one out of the C classes. Using a class-based bigram model, the conditional probability that a word follows another word can be decomposed as the product of the class transition probabilities $P(c_{i+1}|c_i)$ and the word emission probability $P(w_{i+1}|c_i)$

$$P(w_{i+1}|w_i) = P(c_{i+1}|c_i)P(w_{i+1}|c_i) \tag{3.22}$$

Where c_i is the class of w_i, the i-th word in the data. The authors propose a greedy hierarchical agglomerative clustering algorithm for finding a hard class assignement for each word: The algorithm initially assigns each word to its own class. It the iteratively merges classes so that the average mutual information is maximized until C classes remain. In a post-processing step, words are re-arranger if moving them to a new class increases the average mutual information of the partition. The result of the method is a binary tree that includes the words as its leafs. The path to each word, which consists of the concatenated binary codes is then used as feature in a downstream task. Despite only relying on the bigram counts, Brown clusters reveal similar relations as later neural approaches, such as grouping words together with the same morphological stem, e.g. *performed, perform, performs, performing* and *clustering* related words with different stems such as *attorney, counsel, trial, court, and judge.*

Brown cluster have been applied to a wide range of tasks such as NER [127], [156]] and dependency parsing [[97] [180]]. More recently, they have also been found to be an important signal for part-of-speech tagging on Twitter [136] and cross-lingual NER [121].

Latent dirichlet allocation (LDA)

LDA [21] is a generative probabilistic model of a document that represents each topic as a mixture over latent topics, with each topic being represented as a distribution over words. The parameters of LDA are usually estimated using Gibbs sampling [147]. The most common use case for LDA is to explore the topics in a corpus. However, the posterior Dirichlet parameters, i.e. the distribution over topics can also be used as the representation of a document for text classification [21]. Conversely, the distribution over topics for each word can be used as the word representation. LDA is the only seminal unsupervised pretraining technique that does not clearly belong to either the language models nor to the matrix factorization methods. On closer inspection, LDA can be seen as a form of dimensionality reduction and is thus related to matrix factorization approaches. As a latent variable model, LDA is also similar to HMMs, which can be used as language models.

Pretrained HMM language models

Despite not having been widely adopted, the use of latent variable HMM language models to learn representations for sequence labelling [Huang and Yates, [73], [74], [75]] is the closest non-neural analogue to the current prevailing approach of pretraining language models. [73] Huang and Yates train a latent variable model HMM on an unlabelled corpus to learn the probability of a word appearing before and after another word. They then use these probability distribution as features for the left and right contexts in a sequence labelling model. LSA is applied to these context vectors to obtain a dense representation. [74] Huang and Yates adapt these learned representations to semantic role labelling. Huang and Yates [75] use a Factorial HMM with multiple hidden layers in order to capture more aspects of the word representations. This approach is conceptually strikingly similar to the multi-layered neural network language models used these days.

3.1.3 Word embeddings with neural networks

All previous approaches learn representations of words that are then used as features in downstream models. The following approaches learn dense word represenations with neural networks. In this context, such representations are also known as word embeddings. A common argument for word embeddings is that they are low dimensional, dense, and more expressive than traditional approaches [15]. However classic approaches such as LSA also induc dense representatinos and – provided the settings are the same– are competitive with more recent neural approaches [108].

The dimension problem can be solved in an unsupervied way, using neural techniques as shown by Bengio et al. [19]. His breakthrough was to use a feed forward neural network tp predict the *n*-gram probabilities of the words in the vocabulary given the context that came before.

For each word in order in the text corpus, the *n* previous words were used as context and each word was represented by a trainable, d-dimensional random initialized vector. These vectors were then concatenated and fed through many hidden layers. From there the output layer was a softmax probability over the vocabulary of all the probabilities of seeing each word, given the context words. This objective function ensured that while training to predict *n*-grams, the network was learning linguistically valuable information in the matrix of d-dimensional vectors.

In this seminal paper, Collobort and Weston [37] learn the word embeddings by training a neural network on a corpus $\mathcal{C}$ to output a higher score for a correct word sequence than for an incorrect one. For this purpose, they use a max-margin loss.

$$\mathcal{L} = \sum_{i=C}^{\mathcal{C}-C} \sum_{w' \in V} max(0, 1 - f([w_{i_C}, ..., w_i, ..., w_{i+C}]) + f([w_{i_C}, ..., w', ..., w_{i+C}])) \quad (3.23)$$

This outer sum iterates over all words in the corpus $\mathcal{C}$, while the inner sum iterates over all words in the vocabulary. Each word sequence consists of a center word w_i and a window of C words to its left and right. f(.) is neural network that outputs a score given a word sequence and is trained to output a higher score for a word sequence occurring in the corpus (the left term) than a word sequence where the center word is replaced y an arbitrary word w' from the vocabulary (the right term).

Furthermore, it was shown by Collobert et al. ([38]) that this type of word embedding wasn't just a placeholder for words, but coded important language information. In fact, he showed that word embeddings could be used as the sole input to other machine learning models for applied NLP tasks with state of the art results. Amongst other things, word embeddings have shown to be useful in finding named entities in phrases (Turian et al.[186], can be aligned over languages to help machine translation ([207]), and also in semantic sentence parsing [174].

[!h]

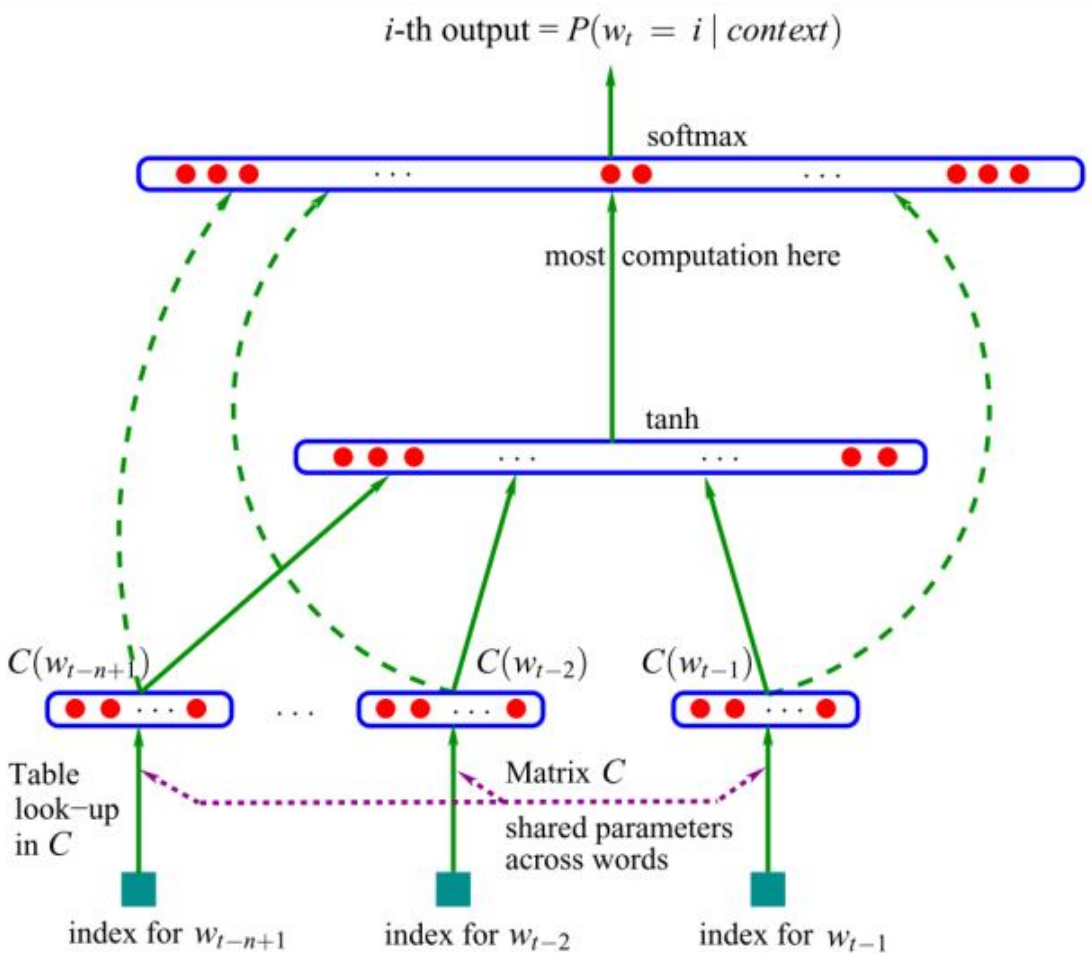


Figure 3.1.1: Feed Forward neural network

Bengio's model had several drawbacks, however. Firstly, it was based on a feed forward neural network so the only way of increasing the context window for the network to learn was by increasing the number of input noes. This meant that these embeddings were expensive to train for recognizing context heavy features in the data.

Word2Vec

To train more efficiently on smaller datasets and with better context information Mikolov et al. ([126], [125]) proposed a RNN approach to embedding creation called the skip-gram model. RNNs, as mentioned, can more efficiently model rich context information since they have a memory of the sequence of words they have seen. The skip-gram model starts with the opposite probability to what is predicted in an n-gram since it's output is then probability of different context words, based on a target word.

This technique proved to embed these word-vectors in a universe of language meaning which was very context rich and semantically interpretable, especially in terms of relations between words. A famous example of these vectors interpretability is that using vector arithmetic the vector for ”King” minus the vector for ”Man” adding the vector ”women” was extremely close in vector space to the vector for ”Queen” showing that the model has learned to represent words with respect to a dimension which one can understand as gender, and another which one can understand as royalty.

Skip-Gram with Negative Sampling SGNS The method used in Word2vec is Skip-gram with negative sampling [126] is arguably the most popular method to learn word embeddings due to its training efficiency and robustness [108]. SGNS approximates a language model but focuses on learning efficient word representations rather than accurately modelling word probabilities. It induces representations that are good at predicting surrounding context words given a target word w_t. The objective is shown in Figure 3.1.2. To this end, it minimizes the negative log-likelihood of the training data under the following skip-gram objective:

$$\mathcal{L}_{SGNS} = -\frac{1}{|\mathcal{C}|}\sum_{t=1}^{|\mathcal{C}|}\sum_{-C<j<Cj\neq 0} logP(w_{t+j}|w_t) \tag{3.24}$$

$P(w_{t+j}|w_t)$ is computed using the softmax function.

$$P(w_{t+j}|w_t) = \frac{\exp\left\{(\hat{x_{t+j}}^T x_t\right\}}{\sum_{i=1}^{|V|}\exp\left\{(\hat{x_i}^T x_t\right\}} \tag{3.25}$$

where x_i and $\hat{x_i}$ are the word and context word embeddings of word w_i respectively. The skip-gram can be often seen as a neural network without a hidden layer. The word embeddings x_i of the input word w_i is then the same as the hidden stated of the model. This word embedding x_i is fed into a softmax layer, where each word has a *seperate* representation $\hat{x_i}$, which represents how it behaves in the context of the input word. Generally, x_i is used as the final word representaion, although combining both x_i and $\hat{x_i}$ can be beneficial [108]

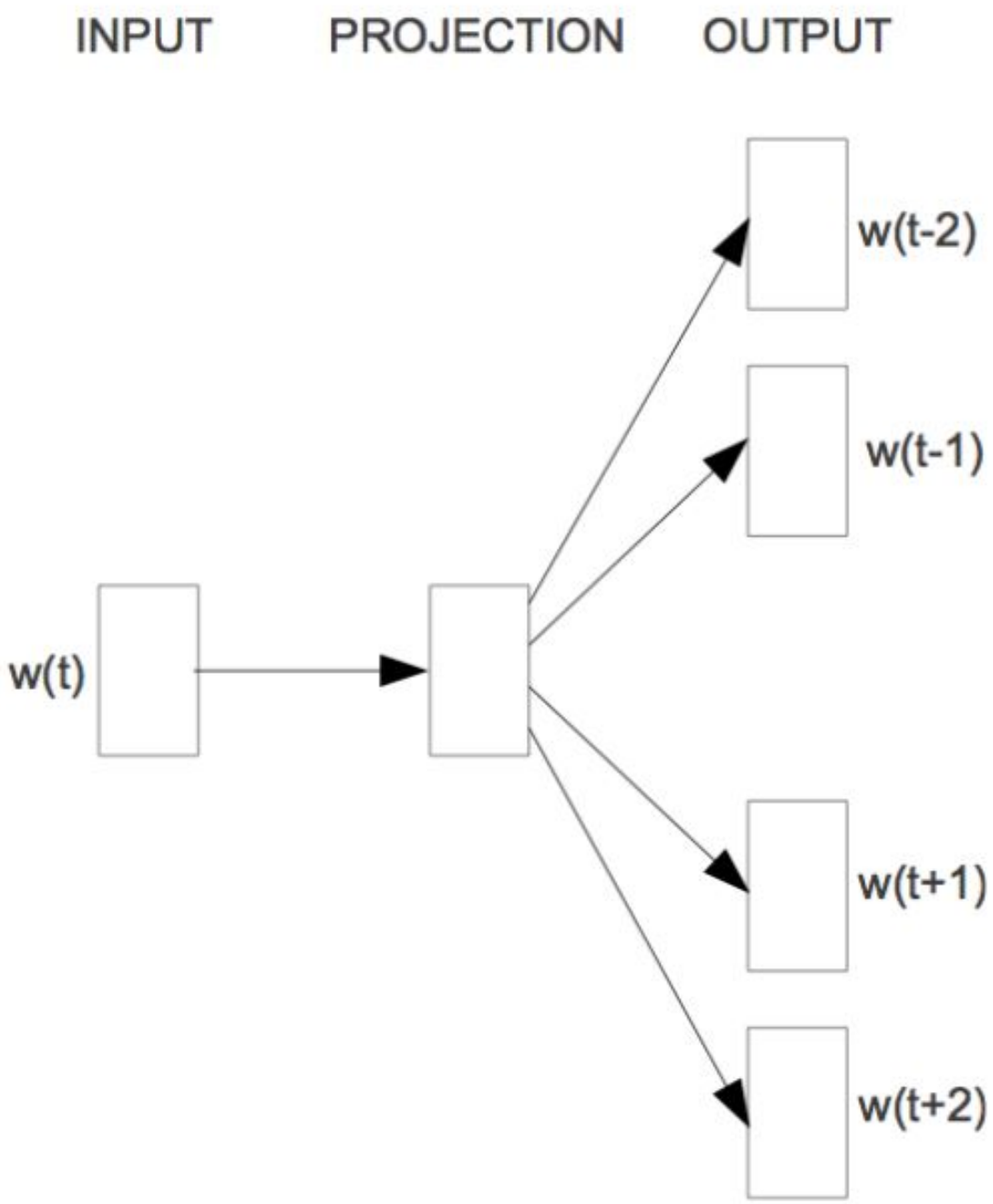


Figure 3.1.2: The skip-gram model

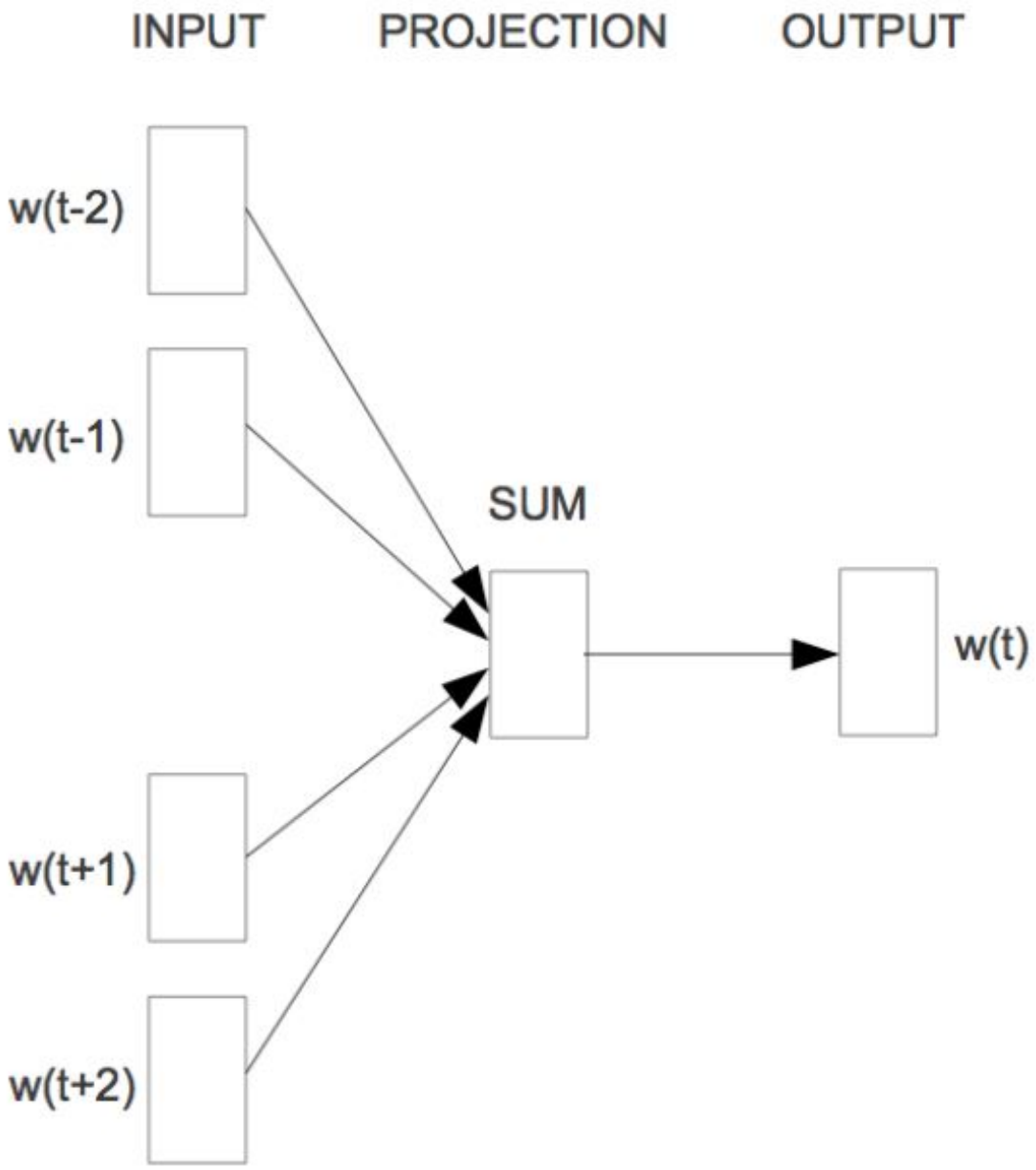


Figure 3.1.3: The continuous bag-of-words model

As the partition function in the denominator of the softmax is expensive to compute, SGNS uses negative sampling, which approximates the softmax to make it computationally more efficient. Negative sampling is a simplification of noise contrastive estimation [Gutmann and Hyv¨arinen, [63]], which was applied to language modeling by Mnih and Teh [131]. Similar to noise contrastive estimation, negative sampling trains the model to distinguish a target word w_t from negative samples drawn from a noise distribution P_n. In this regard, it is similar to the C&W

model as defined above, which ranks true sentences above noisy sentences. Negative sampling is defined as follows:

$$P(w_{t+j}|w_t) = log\sigma(\hat{x_{t+j}^T}x_t) + \sum_{i=1}^{k} \mathbb{E}_{w_i \tilde{P}_n} log\sigma(-\hat{x_i}^T x_t) \tag{3.26}$$

where σ is the sigmoid function and k is the number of negative samples. The distribution P_n is empirically set to the unigram distribution raised to the $3/4^{th}$ power. Levy and Goldberg [107] observe that negative sampling does not in fact minimize the negative log-likelihood of the training data as in Equation 3.24, but rather implicitly factorizes a shifted PMI matrix, very similar to LSA.

Continuous bag-of-words (CBOW) Continuous bag-of-words can be seen as the inverse of the skip-gram architecture: The model receives as input a window of C context words and seeks to predict the target word w_t by minimizing the CBOW objective:

$$\mathcal{L}_{CBOW} = -\frac{1}{|\mathcal{C}|}\sum_{t=1}^{|\mathcal{C}|} logP(w_t|w_{t-C}, ...w_{t-1}, w_{t+1}, ..., w_{t+C}) \tag{3.27}$$

$$P(w_t|w_{t-C}, ...w_{t-1}, w_{t+1}, ..., w_{t+C}) = \frac{\exp\left\{(\hat{x_t}^T x_s\right\}}{\sum_{i=1}^{|V|}\exp\left\{(\hat{x_i}^T x_s\right\}} \tag{3.28}$$

where x_s is the sum of the word embeddings of the words $w_{tC}, ..., w_{t+C}$, i.e. $x_s = \sum_{CjC, j\neq 0} x_{t+j}$. This is depicted in Figure 3.1.3. The CBOW architecture is typically also trained with negative sampling for the same reason as the skip-gram model. Many extensions to the SGNS and CBOW models have been proposed over the years. Le and Mikolov [[104]] extend SGNS and CBOW to model paragraphs and documents by adding a vector that represents the paragraph. Depending on the model, this vector is used to either predict the centre word or the context words. Bojanowski et al. [[22]] extend SGNS by representing each word as a bag of character n-grams, which is particularly useful for modelling words in morphologically rich languages. Pagliardini et al. [137] extend CBOW with compositional n-gram features.

GloVE

The GloVe (Global Vectors) for Word Representation do have another approach for building word vectors. They were initially proposed in 2014 by Pennington et al.[143] and put their focus not solely on the word's local context. GloVe differs from the skip-gram and feed forward word embedding models due to the fact that it is trained on a loss function which takes into account both local co-occurrences from the n-length context windows, but also global count-based co-occurrence probabilities from the text corpus. This is to attempt to encode more of the language features that would be captured in a global method like PCA. This is based on the intuition that including only local context information doesn't give enough information to features about how often words occur in rare contexts. The loss function ends up looking similar to the skip-gram model, but with slightly richer context specific information embedded in the vectors.

Theoretically speeking, Global Vectors learn word representation via matrix factorization. It minimizes the difference between the dot product of the embeddings of a word w_i and its context word c_t and the logarithm of their number of co-occurrences within a certain window size:

$$\mathcal{L}_{GloVe} = \sum_{i,j=1}^{|V|} f(C_{ij})(x_i^T \hat{x}_j + b_i + \hat{b}_j - logC_{ij})^2 \tag{3.29}$$

where b_i and $\hat{b}_j$ are the biases corresponding to word w_i and its context word w_j, C_{ij} captures the number of times word w_i occurs with context word w_j, and f(.) is a weighting function that assigns relatively lower weight to rare and frequent co-occurrences.

Some other popular embeddings models include Latent Semantic Analysis, the FastText embeddings (Joulin et al., 2016) and the ConceptNet embeddings from Speer et al. (2016).

Text Corpora and Pre-Training The choice of text corpora that the model will learn from is nearly more important than the actual word representation model architecture. This has huge importance since different types of languages settings(language, specific vocabulary, style...) have totally different features and vocabularies. Some common text corpora used for training word embeddings are all the articles on Wikipedia, a corpus of newspaper stories, all text data stored on the web, or all posts on Twitter.

The differences in language between Wikipedia, the web at large scale and Twitter are extremely clear and will create totally different language universes that the vocabulary is embedded in. Clearly, training a word-embeddings model is a computationally difficult task. Corpora should be as large and varied as possible to give the richest language features. This means as well that the loss functions will have a big amount of local minima which can make training difficult. For this reason, these types of training problems are notorious for requiring lots of engineering tricks and computing power.

Since it is difficult and computationally expensive for all users to retrain these embeddings, GloVe has released pre-traind word embeddings for each of these three text corpora mentioned above: Wikipedia, the Common Crawl dataset of websites and all data on Twitter. These can be downloaded and used as inputs to other machine learning models for NLP tasks. Every NLP task contains different text data, thus the language universes are slightly different. That's why it is a common practice to tune the word-embeddings weights at the same time as training the model. This allows us to better refine the meaning space of vectors to suit the problem context. Added to this problem, the existence of words which are not in our vocabulary of word embeddings is almost sure. It is most common to initialize these out of vocabulary (oov) words' vectors to random values. Based on their context int the problem corpus, the word embeddings can still be trained to sort these words into the right part of meaning space.

3.1.4 Deep Pretrained representations

All the presented neural network-based methods so far are shallow models that trade expressiveness for efficiency. In order to be scalable, they use a neural network without a hidden layer. This, however, limits the relations they can capture. In analogy to computer vision, using word embeddings is akin to initializing an image model with pretrained representations that only encode edges: they will be helpful for many tasks, but they fail to capture higher-level information that might be even more useful [161]. A model initialized with word embeddings needs to learn from scratch not only to disambiguate words, but also to derive meaning from a sequence of words. This is the core aspect of language understanding, and it requires modelling complex language phenomena such as compositionality, polysemy, anaphora, long-term dependencies,

agreement, negation, and many more. It should thus come as no surprise that NLP models initialized with these shallow representations still require a huge number of examples to achieve good performance.

At the core of recent advances is one key paradigm shift: going from just initializing the first layer of our models to pretraining the entire model with hierarchical representations. If learning word vectors is like only learning edges, these approaches are like learning the full hierarchy of features, from edges to shapes to high-level semantic concepts. The following models consequently use deep neural networks to learn representations. Another advantage of such models is that document representations can be easily obtained as the hidden state of the model.

Autoencoding Dai and Le [40] propose to pretrain an LSTM autoencoder that reconstructs the words in a sentence. Compared to language modelling, which makes a prediction for every word, the autoencoder first processes all words in a sentence and then predicts all words of the input sequence without receiving more information. The autoencoder objective is thus conceptually easier than the language modelling objective as the model does not need to "look into the future" but merely needs to "remember the past". Hill et al. [68] propose to use a denoising autoencoder instead. This model makes the task more challenging by adding noise to the input, which should make the learned representations more robust. The noise consists of randomly deleting words and randomly scrambling non-overlapping bigrams.

Skip-thoughts Instead of reconstructing the words in a sentence, skip-thoughts [205] trains an RNN to reconstruct the words in the preceding and succeeding sentences. Skip-thoughts can thus be seen as an extension of SGNS to the sentence level. This notably requires training on longer documents consisting of ordered sentences. While being conceptually simple, the original skip-thoughts takes weeks to train. Hill et al. [68] propose a cheaper variant that sums the word embeddings of the input sentence, while [80] Jernite et al. present discourse-based objectives that can be used as cheaper alternatives. Their first objective trains a model to predict whether two sentences have been switched. They frame next-sentence prediction as a classification task where given the first three sentences of a paragraph the model must choose the sentence immediately following among five candidates from later in the paragraph. Logeswaran and Lee [114] similarly propose a more efficient formulation of skip-thoughts, which replaces the decoder with a classifier that chooses the target

sentence from a set of candidate sentences. Devlin et al. [45] further reduce next-sentence prediction to a binary classification task of predicting whether a sentence follows another sentence.

Pretrained Language models In this part, there is an overview of the pre-trained language models.

While many of the earlier approaches such as Brown clustering, SGNS, and CBOW are variants of a language model, they were all proposed as approximations to make language modelling more computationally feasible with limited computational resources. With the increase in compute in recent years, it has now become feasible to pretrain deep neural language models. Dai and Le [40] first proposed to pretrain language models, but only considered training on in-domain documents, which limited their approach.[146] Peters et al. pretrain a language model for sequence labelling tasks, while Ramachandran et al.[154] apply a language model to summarization and machine translation. Peters et al. [144] first show that a pretrained language model is useful for a wide variety of NLP tasks. Radford et al. [150] train a deeper model on more data. Devlin et al. [45] introduce a masked language model objective, which randomly masks some words in the input and then predicts only the masked tokens. In contrast to a denoising autoencoder, only the masked words rather than the entire input are reconstructed. Compared to other pretraining tasks such as translation, skip-thoughts, and autoencoding, language modelling is more sample-efficient and performs best on syntactic tasks [203]. The rise of pretrained language models has major implications for transfer learning for NLP. Pretrained language models have achieved improvements between 10-20% on a wide variety of tasks, some of which have been studied for decades. Pretrained language models can help with domain adaptation [83] and enable few-shot and zero-shot learning [Howard and Ruder,[72], Radford et al., [150]]. In addition, representations from pretrained language models capture hierarchical features that are analogous to ones captured by ImageNet models in computer vision [Krizhevsky et al., [99]]: the word embedding layer captures morphological information; lower layers capture local syntax, while the upper layers capture longer range semantics such as coreference [Peters et al., [145]]. Pretrained language model representations have also been shown to implicitly learn logic rules [98].

3.1.5 Multi-task pretraining

Distantly supervised, supervised and unsupervised pretraining tasks complement each other. Using multi-task learning, a model can be trained on different tasks at once. Pretraining on multiple tasks may help mare representations more general, and thus capturing more information, as a hypothesis space that performs well on sufficiently large number of tasks is also expected to perform well on learning novel tasks from the same environment [17]. Subramanian et al. [178] perform multi-task pretrinaing on skip-thoughts, machine translation, constituency parsing, and natural language inference. Similarly, Cer et al. [] combine pretraining on skip-thoughts, response prediction, and naturla language inference while Devlin et al. [45] combine masked language modelling with next sentence prediction.

3.1.6 Architectures

While reliable word representations can be learned with shallow approaches such as SGNS and GloVe, most NLP tasks require representations of sentences and documents. Nevertheless, simple architectures such as an average of pretraind word embeddings are surprisingly strong baselines for sentence and document-level representations. Wieting et al. [193] that training a model based on an average of word vectors outperforms more complex architectures on their paraphrase pretrainign task. Arora et al. [10] propose to additionally subtract the principal component from a weighted average of pretraind word embeddings, while Chen [33] propose to add noise to the average of pretrained embeddings. Another conceptually simple architecture is the deep averaging network DAN[79], which averages word vectors, followed by two hidden layers. [29] use DAN for pretraining and achieve simialr performance compared to a more complex model.

These simple models are fast and easy to train, but recent experience show that deeper model outperform shallow models and scale with the amount of training data and depth of the network. Wieting and Gimpel [192] show that LSTMs outperform an average of word vectors on paraphrase pretraining and propose a hybrid model that combines the two. Conneau et al. [39] experiment with different models and find that BiLSTM that condenses hidden representations with mac-pooling performs best. Kiros and Chan [96] propose a more lightweight version of this model by using convolutions and a dynamic weighting of embeddings instead of recurrent connections. Recently,

the Transformer [188], an architecture -that will be detailed in this thesis- that is based on self-attention has whon convincing reseults [150] [29], [45]

3.2 Transfer learning

This chapter provides an overview of the literature of transfer learning in general and for NLP in particular. It studies how machine learning models can be transferred to data outside of their training distribution and specifically focuses on transfer across different tasks, domains, and languages in NLP 3. As transfer learning has been used to refer to different areas in different contexts, let's first provide a definition of transfer learning. Based on this definition, we will define a taxonomy and review the first 2 prevalent settings of transfer learning in NLP in the corresponding sections:

1. multi-task learning 2. sequential transfer learning 3. domain adaptation 4. cross-lingual learning .

3.2.1 Introduction

In the classic supervised learning scenario of machine learning depicted in Figure 3.2.1,

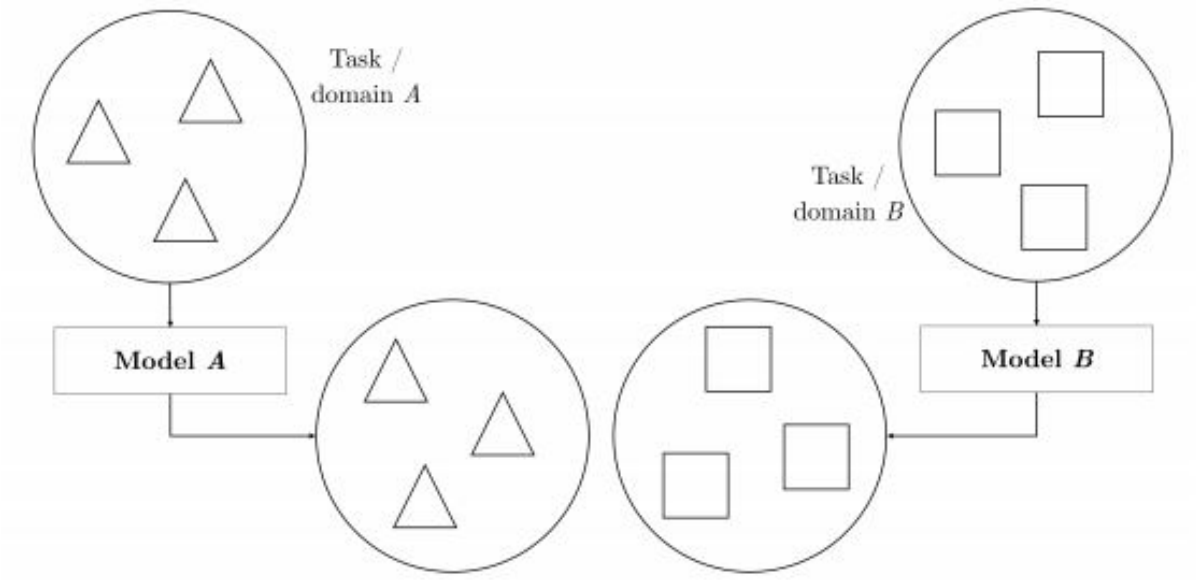


Figure 3.2.1: The traditional supervised learning setup in machine learning

if one intend to train a model for some task and domain A, one assumes that one is

provided with labelled data for the same task and domain. One can now train a model on this dataset and expect it to perform well on unseen data of the same task and domain. In other words, one expect our data to be i.i.d. When given data for some other task or domain B, one require again labelled data of the same task or domain, which one can use to train a new model that is expected to perform well on this type of data.

The traditional supervised learning paradigm breaks down when no sufficient available labelled data for the desired task or domain to train a reliable model. Transfer learning allows us to deal with this scenario by leveraging the data of some related task or domain, known as the *source task* and *source domain*. The knowledge gained in solving the source task in the source domain is stored and applied it to the target task and target domain as can be seen in Figure 3.2.2

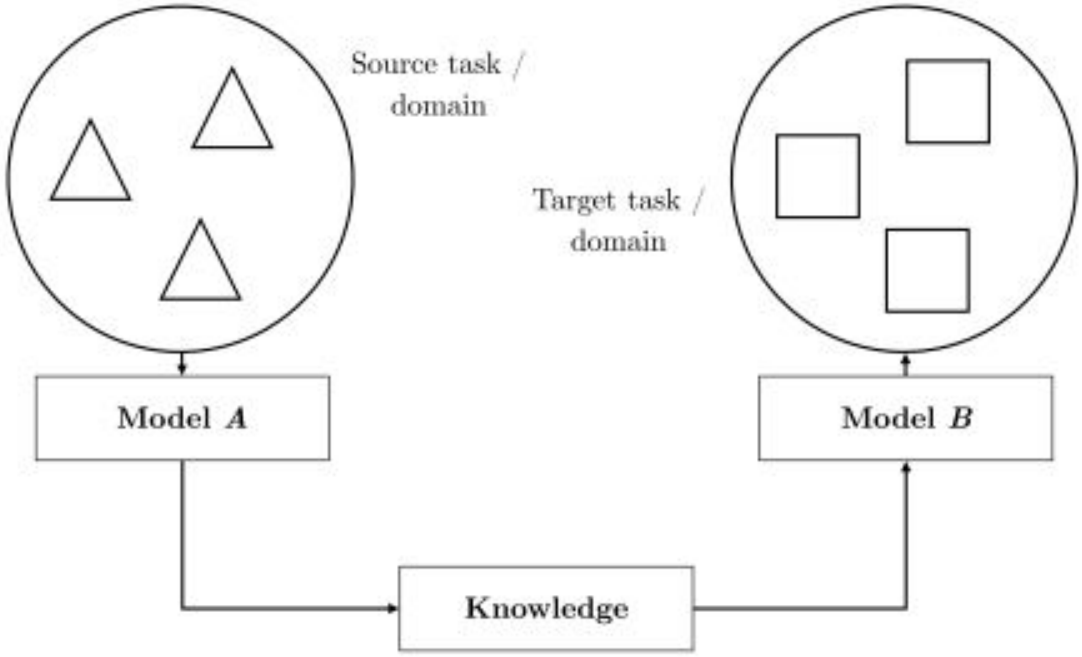


Figure 3.2.2: The transfer learning setup

In practice, we seek to transfer as much knowledge as we can from the source setting to our target task or domain. This knowledge can take on various forms depending on the task and data. In most cases throughout this thesis, it relates to the representations learned by neural network models 2.2.

Definition

We now provide a definition of transfer learning following the notation of Pan and Yang [138] with binary document classification as a running example. Transfer learning involves the concepts of a domain and a task. A domain $\mathcal{D}$ consists of a feature space X and a marginal probability distribution P(X) over the feature space, where X = x_1, $\cdots$, $x_n \in$ X . For document classification with a bag-of-words representation, X is the space of all document representations, x_i is the i-th term vector corresponding to some document and X is the random variable associated with the sample of documents used for training. Given a domain $\mathcal{D} = X, P(X)$, a task $\mathcal{T}$ consists of a label space $\mathcal{Y}$, a prior distribution P(Y), and a conditional probability distribution P(Y | X) that is typically learned from the training data consisting of pairs $x_i \in X$ and $y_i \in Y$. Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to learn the target conditional probability distribution $P_T(Y_T|X_T)$ in $\mathcal{D}_T$ with the information gained from $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. Typically, either a limited number of labeled target examples or a large number of unlabeled target examples are assumed to be available. As both the domain $\mathcal{D}$ and the task $\mathcal{T}$ are defined as tuples, the inequalities between the different members of the tuple give rise to five transfer learning scenarios, which we will discuss below.

Scenarios

Given source and target domains $\mathcal{D}_S$ and $\mathcal{D}_T$ where $\mathcal{D} = X, P(X)$ and source and target tasks $\mathcal{T}_S$ and $\mathcal{T}_T$ where $\mathcal{T} = Y, P(Y), P(Y|X)$ source and target conditions can vary in five ways. The first three scenarios deal with a mismatch between the source and target tasks, i.e. $\mathcal{T}_S \neq \mathcal{T}_T$, while the last two scenarios occur when there is a discrepancy between source and target domains, i.e. $\mathcal{D}_S \neq \mathcal{D}_T$

1. $P_S(Y_S) \neq P_T(Y_T)$. The prior distributions of the source and the target tasks are different, i.e. the documents have different label ditributions. This is important in generative models, which model the prior explicitly [30], [54] but has also been applied to discriminative models. [32]

2. $P_S(Y_S|X_S) \neq P_T(Y_T|X_T)$ The conditional probability distributions of the source and target tasks are different, i.e. source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice and approaches such as

over-sampling, under-sampling, or SMOTE are widely used [31].

3. $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces between the two tasks are different, i.e.documents need to be assigned different labels in the target task. In this case, the most important other distinction is whether the tasks are learned sequentially or simultaneously. Learning multiple tasks at the same time is known as multi-task learning. In practice, this scenario usually occurs with scenario 1 and 2, as it is extremely rare for two different tasks to have different label spaces, but the same prior and conditional probability distributions.

4. $P_S(X_S) \neq P_T(X_T)$. The marginal probability distributions of source and target domain are different, e.g. the documents discuss different topics. This scenario is generally known as domain adaptation. Domain adaptation typically also requires tackling scenarios 1 and 2, as domains may differ in the prior and conditional distributions of their labels.

5. $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the source and target domain are different, i.e. the documents are written in twi different languages. In the context of natural language processing, we will refer to this scenario as *cross-lingual* or *cross-lingual adaptation*

The setting when the source task is different from the rarget task is typically known as *inductive transfer learning*, while in the transductive transfer learning setting source and target tasks are the same.

Taxonomy

Taxonomy is defined for transfer learning for NLP based on these scenarios. Specifically, the taxonomy of [138] is adapted to the transfer learning scenarios that are most commonly encountered in NLP. IN particular, the following changes to their taxonomy are made: - add the NLP-specific setting of *cross-lingual learning* - omit the sample selection bias / covariance shift setting that corresponds to scenario 2 as this typically treated as a special case of domain adaptation. - introduce the term *sequential transfer learning* to highlight the difference to multi-task learning. We treat *self-taught learning* [153]and *unsupervised transfer learning* [41] as instances of this category.

The complete taxonomy for transfer learning for NLP can be seen in 3.2.3.

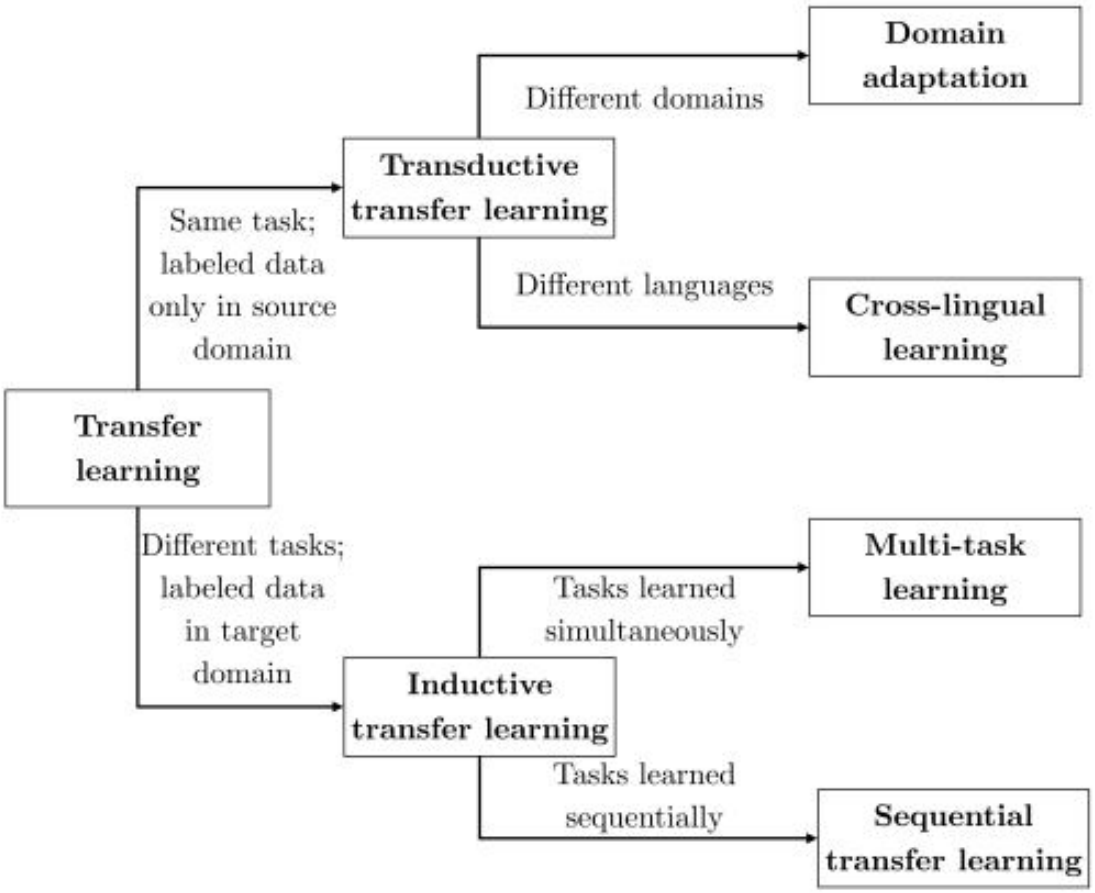


Figure 3.2.3: A taxonomy for transfer learning for NLP

we will review approaches of these four transfer learning settings in the *following*.

3.2.2 Multi-task learning

This section is adapted from: Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. arXiv preprint arXiv:1706.05098.

Multi-task learning (MTL) has led to successes in many applications of machine learning, from natural language processing [37] and speech recognition [43] to computer vision [158] and drug discovery [155]. This section aims to give a general overview of MTL (Multi-Task Learning), particularly in deep neural networks. We will initially motivate MTL from different perspectives. We will then introduce the two most frequently employed methods for MTL in Deep Learning. Subsequently, we will describe mechanisms that illustrate why MTL works in practice. We will provide

context for more recent approaches by discussing the literature in MTL. Finally, we will talk about commonly used types of auxiliary tasks and discuss what makes a good auxiliary task for MTL.

Introduction

Machine learning generally involves training a model to perform a single task. By focusing on one task, however, there is a risk of omitting information that might help us do better on the metric of interest. Specifically, knowledge that comes from the training signals of related tasks is ignored. Alternatively, by sharing representations between related tasks, the model is enabled to generalize better on our original task. This approach is called multi-task learning. Specifically, "MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks" [28].

MTL is also known as joint learning, learning to learn, and learning with auxiliary tasks. Generally, as soon an optimization problem involves more than one loss function, we are effectively doing multi-task learning (in contrast to single-task learning). In such scenarios, it helps to think about what we are trying to do explicitly in terms of MTL and to draw insights from it. Even if we are only optimizing one loss as is the typical case, chances are there is an auxiliary task that could help improve performance on our main task.

Motivation

Multi-task learning can motivated in different ways: Biologically, multi-task learning can be seen as being inspired by human learning. For learning new tasks, the knowledge acquired is often applied by learning related tasks. For instance, a baby first learns to recognize faces and can then apply this knowledge to recognize other objects [189]. From a pedagogical perspective, often tasks that provide us with the necessary skills are learnt first to master more complex techniques. This is true for learning the proper way of falling in martial arts as much as learning a language [25]. Finally, multi-task learning can motivated from a machine learning point of view: Multi-task learning introduces an inductive bias provided by the auxiliary tasks, which cause the model to prefer hypotheses that explain more than one task [28]. This generally leads to solutions that generalize better.

Two methods for MTL in neural networks

So far, the focus was on theoretical motivations for MTL. To make the ideas of MTL more concrete, one should look at the two most commonly used ways to perform multi-task learning in deep neural networks. In the context of deep learning, multi-task learning is typically done with either hard or soft parameter sharing of hidden layers. **Hard parameter sharing** Hard parameter sharing is the most commonly used approach to MTL in neural networks and goes back to Caruana [28]. It is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers as can be seen in Figure 3.2.4

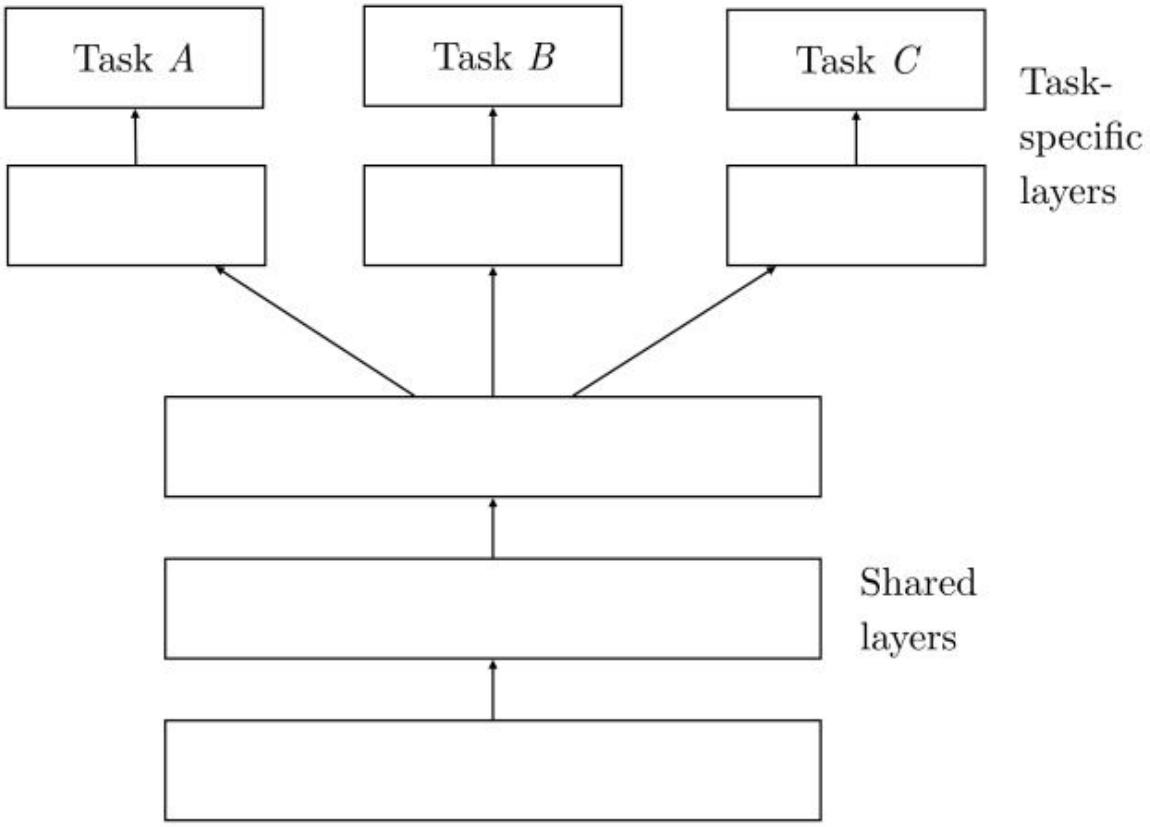


Figure 3.2.4: Hard parameter sharing

Hard parameter sharing greatly reduces the risk of overfitting. In fact, Baxter [16] showed that the risk of overfitting the shared parameters is an order T smaller than overfitting the task-specific output layers where T is the number of tasks. This makes sense intuitively: The more tasks are learnt simultaneously, the more our model has to find a representation that captures all of the tasks and the smaller is our chance of

overfitting on the original task.

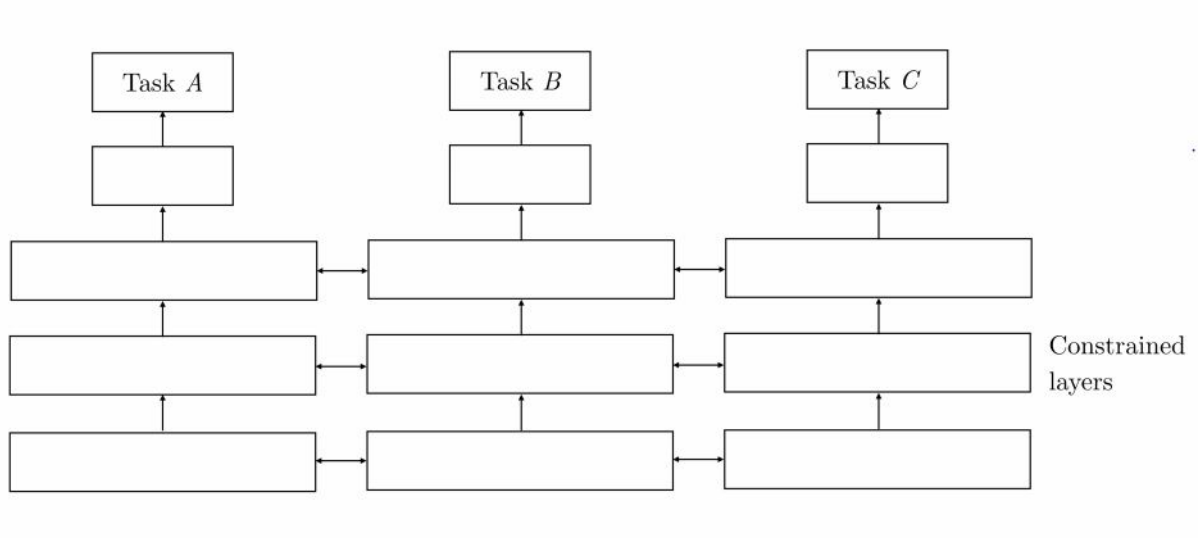


Figure 3.2.5: Soft parameter sharing

Soft parameter sharing In soft parameter sharing on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar, as shown in Figure 3.2.5. Duong et al. [49] for instance use ↕2 distance for regularization, while Yang and Hospedales [200] use the trace norm. The constraints used for soft parameter sharing in deep neural networks have been greatly inspired by regularization techniques for MTL that have been developed for other models, which will be soon discussed.

Why does MTL work?

Even though an inductive bias obtained through multi-task learning seems intuitively plausible, in order to understand MTL better, one need to look at the mechanisms that underlie it. Most of these have first been proposed by Caruana [28]. For all examples, one will assume that one have two related tasks A and B, which rely on a common hidden layer representation F.

Implicit data augmentation MTL effectively increases the sample size that is used for training our model. As all tasks are at least somewhat noisy, when training a model on some task A, our aim is to learn a good representation for task A that ideally ignores the data-dependent noise and generalizes well. As different tasks have different

noise patterns, a model that learns two tasks simultaneously is able to learn a more general representation. Learning just task A bears the risk of overfitting to task A, while learning A and B jointly enables the model to obtain a better representation F through averaging the noise patterns.

Attention focusing If a task is very noisy or data is limited and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. MTL can help the model focus its attention on those features that actually matter as other tasks will provide additional evidence for the relevance or irrelevance of those features

Eavesdropping Some features G are easy to learn for some task B, while being difficult to learn for another task A. This might either be because A interacts with the features in a more complex way or because other features are impeding the model's ability to learn G. Through MTL, the model is allowed to eavesdrop, i.e. learn G through task B. The easiest way to do this is through hints [5], which directly train the model to predict the most important features.

Representation bias MTL biases the model to prefer representations that other tasks also prefer. This will also help the model to generalize to new tasks in the future as a hypothesis space that performs well for a sufficiently large number of training tasks will also perform well for learning novel tasks, as long as they are from the same environment [17].

Regularization Finally, MTL acts as a regularizer by introducing an inductive bias. As such, it reduces the risk of overfitting as well as the Rademacher complexity of the model, which is its ability to fit random noise [175].

Auxiliary tasks

MTL is a natural fit in situations where the interest is in obtaining predictions for multiple tasks at once. Such scenarios are common for instance in finance or economics forecasting where one might want to predict the value of many possibly related indicators; marketing where multiple consumer preferences are modelled at once [7]; or in bioinformatics where one might want to predict symptoms for multiple diseases simultaneously. In scenarios such as drug discovery, where tens or hundreds of active compounds should be predicted, MTL accuracy increases continuously with the number of tasks [155]. In many situations, however, we only care about

performance on one task. In this section, we will thus look at how main and auxiliary tasks should interact to maximize performance and how we can find a suitable auxiliary task in order to reap the benefits of multi-task learning

What layers should tasks share? One of the main considerations in using multitask learning with deep neural networks is to determine which layers should be shared. In NLP, recent work focused on finding better task hierarchies for multi-task learning: [175] show that low-level tasks such as POS tagging and NER should be supervised at lower layers when used as auxiliary tasks. Building on this finding, Hashimoto et al. [64, Hashimoto] pre-define a hierarchical architecture consisting of several NLP tasks as a joint model for multi-task learning. In [166], they propose a hierarchical architecture for semantic tasks. Yang et al. [202] enumerate different ways layers can be shared across two sequence tagging tasks. Rather than constraining a model to a certain sharing structure, Misra et al. [129] use what they refer to as cross-stitch units to learn a linear combination of the output of the previous layers.

How should the tasks interact during training ?

In multi-task learning, minibatches from different tasks are typically sampled uniformly at random. As multi-task learning minimizes a weighted sum of task losses $\frac{1}{T}\sum_{i=1} T\lambda_i\mathcal{L}_i$ where T is the number of tasks, choosing appropriate weights λ for each task is crucial. These weights can be tuned on a validation set just as any other hyper-parameter. The default choice, which works reasonably well in practice is to weigh all tasks equally by setting $\lambda_1 = ... = \lambda_T = c$ where c is an arbitrary constant. More principled approaches are possible such as using uncertainty to weigh the task losses [89].

Alternatively, the sampling of tasks can modified. Given two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$ that are sampled with probabilities p1 and p2 respectively, if p1 = 2p2, we are effectively weighing $\mathcal{T}_1$ with $\lambda_1 = 2\lambda_2$ as examples are seen twice as often. Adjusting the sampling ratio of different tasks thus has the same effect as assigning different weights. Kiperwasser and Ballesteros [95] propose different predefined sampling schedules. In [166], a proportional sampling strategy is proposed, which samples mini-batches proportional to the number of training examples of a task. As multi-task learning becomes a common tool, sampling strategies will become more elaborate. In particular, as schedules are used that adjust the sampling strategy over the course

of training, multi-task learning and sequential transfer learning will move closer together.

Related Tasks used in NLP We now look in more detail in existing NLP tasks, which have been used as auxiliary tasks to improve the performance of a main task.

Speech Recognition: Recent multi-task learning approaches for automatic speech recognition (ASR) typically use additional supervision signals that are available in the speech recognition pipeline as auxiliary tasks to train an ASR model end-to-end. Phonetic recognition and frame-level state classification can be used as auxiliary tasks to induce helpful intermediate representations. Toshniwal et al. [185] find that positioning the auxiliary loss at an intermediate layer improves performance. Similarly, Arik et al. [9] predict the phoneme duration and frequency profile as auxiliary tasks for speech synthesis. **Machine translation**: The main benefit MTL has brought to machine translation (MT) are models that can simultaneously encode and translate to multiple languages, which can not only improve performance, but also enable zero-shot translation: Dong et al. [47] jointly train the decoders; Zoph and Knight [206] jointly train the encoders, while Johnson et al. [82] jointly train both encoders and decoders; finally, Malaviya et al. [117] train one model to translate from 1017 languages into English. Other tasks have also shown to be useful for MT: Luong et al.[116] show gains using parsing and image captioning as auxiliary tasks; Niehues and Cho [133] combine NMT with POS tagging and NER; Wu et al. [197] jointly model the target word sequence and its dependency tree structure; finally, Kiperwasser and Ballesteros [95] use POS tagging and dependency parsing.

Multilingual tasks Similarly to MT, it can often be beneficial to jointly train models for different languages: Gains have been shown for dependency parsing [[49], [8], named entity recognition [56], part-of-speech tagging [51], document classification [139], discourse segmentation [23], sequence tagging [201], and semantic role labelling [132]. Many of these approaches rely on cross-lingual word embeddings for initialization.

Language grounding For grounding language in images or videos, it is often useful to enable the model to learn causal relationships in the data. For video captioning, [140] Pasunuru and Bansal jointly perform next-frame and entailment prediction, while Hermann et al. [66] predict the next frame and the words that represent the visual state for language learning in a simulated environment.

Semantic parsing For a task where multiple label sets or formalisms are available such as for semantic parsing, an interesting MTL strategy is to learn these formalisms together: To this end,[62] Guo et al. jointly train on multi-typed treebanks; [142] Peng et al. learn three semantic dependency graph formalisms simultaneously; [50] Fan et al. jointly learn different Alexa-based semantic parsing formalisms; [204] Zhao and Huang jointly train a syntactic and a discourse parser; and Hershcovich et al. [67] train on four semantic parsing representations. For more shallow semantic parsing such as frame-semantic argument identification, Swayamdipta et al. [181] predict whether an n-gram is syntactically meaningful, i.e. a syntactic constituent.

Representation learning Rather than learning representations based on a single loss, intuitively, representations should become more general as more tasks are used to learn them. As an example of this strategy, Hashimoto et al. [64] jointly train a model on multiple NLP tasks, while Jernite et al. [80] propose several discourse-based artificial auxiliary tasks for sentence representation learning.

Question answering For question answering (QA) and reading comprehension, it is beneficial to learn the different parts of a more complex end-to-end model together: Choi et al. [34] [2017] jointly learn a sentence selection and answer generation model, while Wang et al. [190] jointly train a ranking and reader model for open-domain QA.

Information retrieval For relation extraction, information related to different relations or roles can often be shared. To this end, Jiang [81] jointly learn linear models between different relation types; Liu et al. [111] jointly train domain classification and web search ranking; Yang and Mitchell [199] jointly predict semantic role labels and relations; Katiyar and Cardie [88] jointly extract entities and relations; and in [[166]], we jointly learn coreference resolution, relation extraction, entity mention detection, and NER. **Chunking** Chunking has been shown to benefit from being jointly trained with lowlevel tasks such as POS tagging [Collobert and Weston, [37], Søgaard and Goldberg, [175], Ruder et al., [163]].

Summary

In this section, we have reviewed both the historic literature in multi-task learning as well as more recent work on MTL applied to NLP. We have provided insights on why MTL works. The discussed multi-task learning methods and particularly

sharing of parameters are core to many approaches in other transfer learning scenarios discussed in this chapter. In the next section, we will present the closely related sequential transfer learning paradigm, which trains models sequentially rather than simultaneously.

3.2.3 Sequential transfer Learning

This section gives an overview of sequential transfer learning, arguably the most frequently used transfer learning scenario in natural language processing and machine learning. We will discuss the most common scenario involving a source and target task (§3.3.2). In this context, we will present common distantly supervised (§3.3.2.1), supervised (§3.3.2.2), and unsupervised (§3.3.2.3) source tasks including fundamental methods in natural language processing. We will then discuss common architectures (§3.3.2.5) and transfer methods (§3.3.3). Subsequently, we will talk about the setting using multiple tasks (§3.3.4) and finally review evaluation scenarios (§3.3.5).

3.3 Transformer

The Transformer is a neural network architecture introduces by Vaswani et al. [188] and mainly developed for machine translation. Figure 3.3.1 displays the transformer's architecture that will be detailed in the upcoming parts.

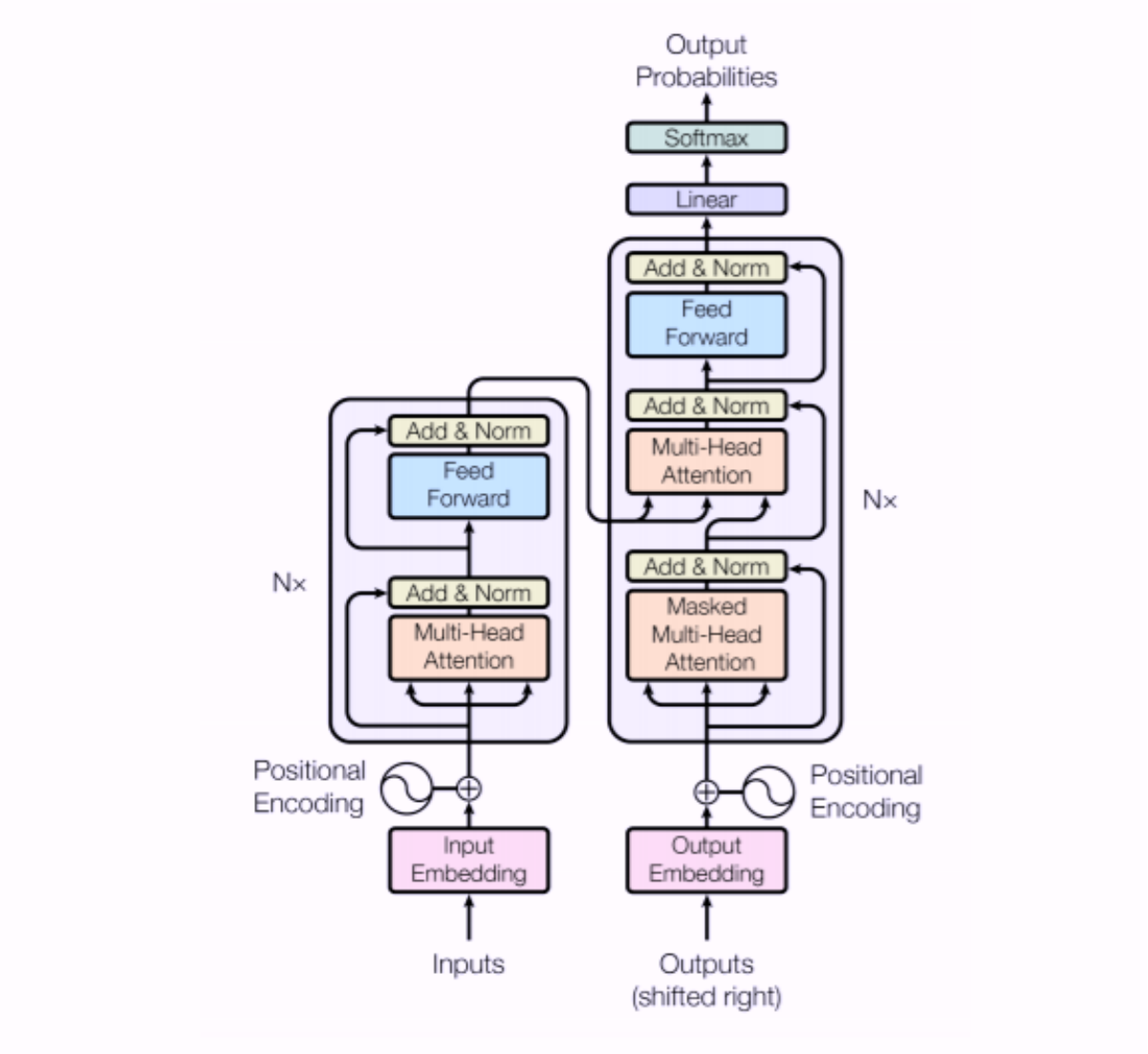


Figure 3.3.1: Transformer architecture [188]

3.3.1 Architecture

The Transformer is composed of two larger building blocks, an encoder and a decoder, see Figure 3.3.2.

Encoder The encoder is composed of two modules. The first is multi-head attention (self-attention) with a residual connection [65], which is followed by layer normalization[11]. Thereafter, the result is fed into a feed forward neural network followed by the same residual with addition to normalization technique.

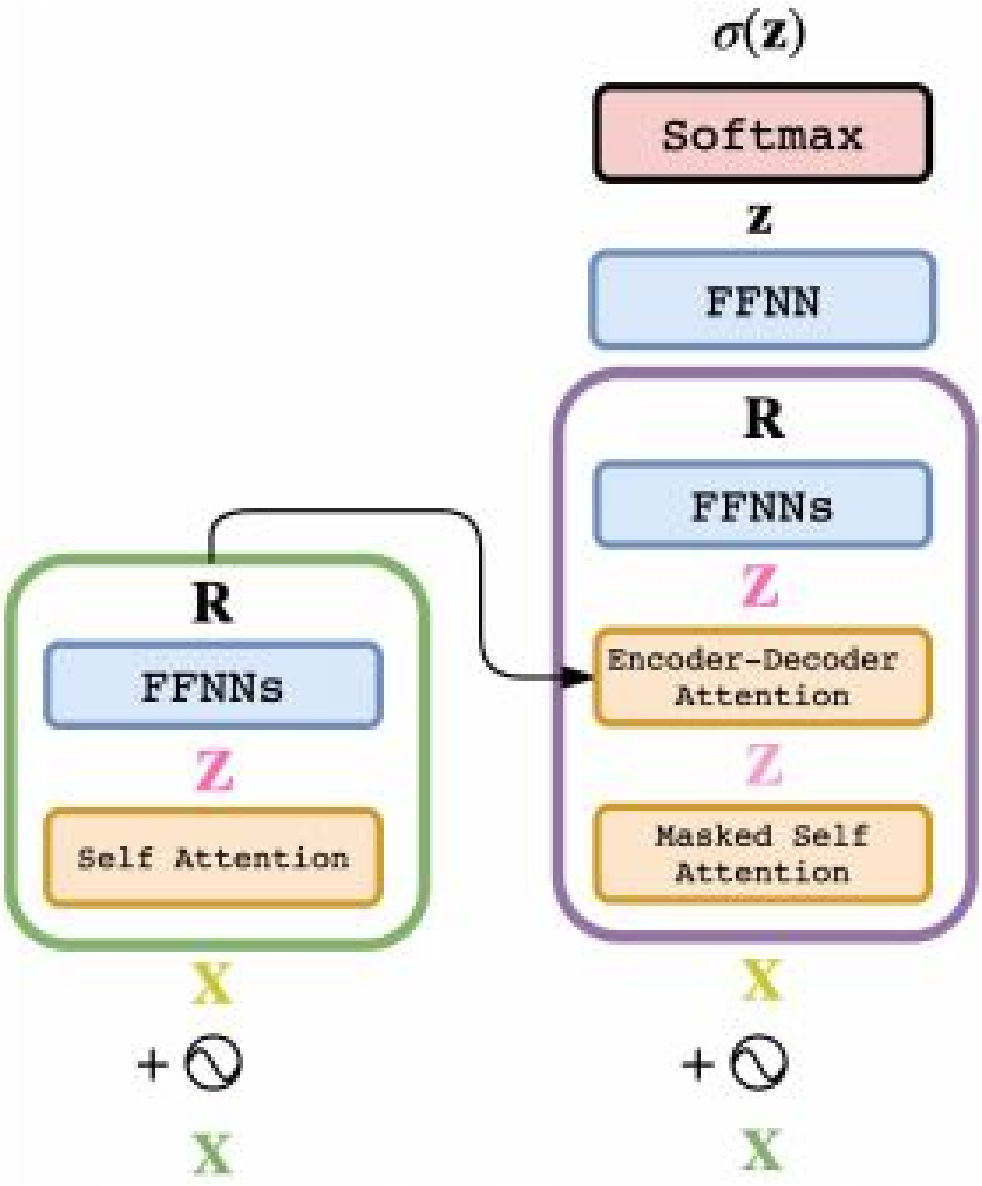


Figure 3.3.2: Transformer architecture [188]. Encode (green) performs self-attention. Decoder (purple) modifies the self-attention to prevent from attending to subsequent positions, and therefore unidirectional. The decoder also attends over the entire encoder using encoder decoder attention. The image omits the residual connection around each submodule followed by layer normalization

Decoder In addition to the two modules from the encoder, the decoder inserts a third module just befor the self-attention. This submodule modifies the existing self-attention in a way to ignore subsequent tokens. Thus the decoder is considered undirectional. The subsequent self-attention is also modified to be an encoder-decoder attention mechanism such that it attends over tle encoder. The previously described

residual connection and layer normalization are applied after each layer.

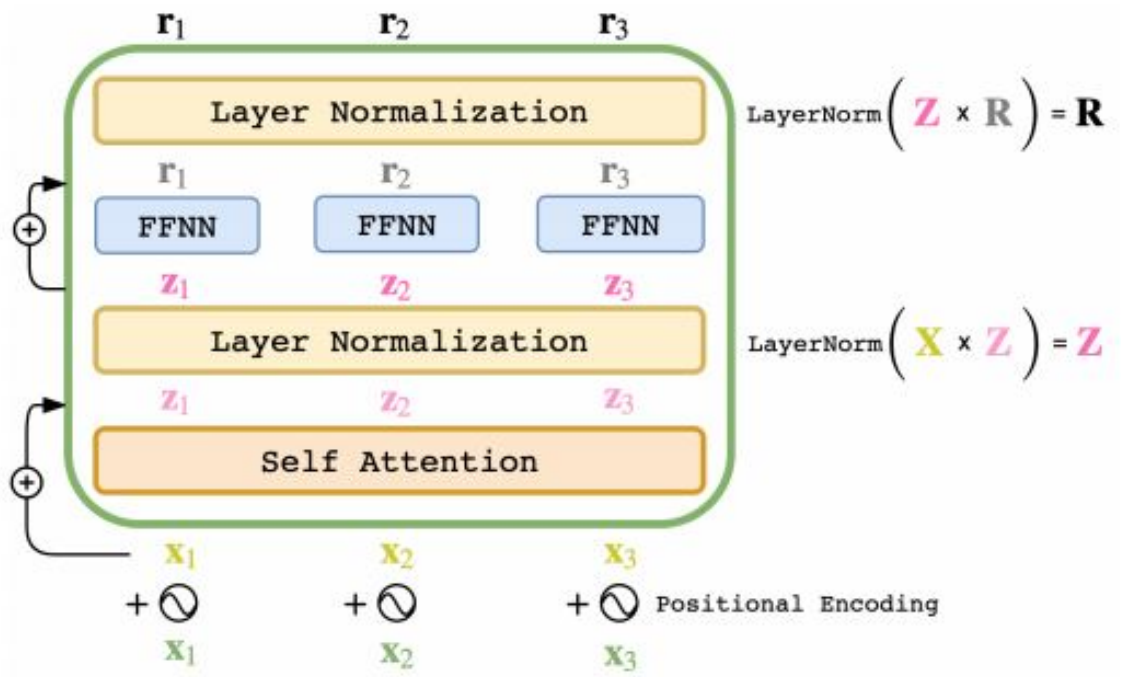


Figure 3.3.3: Inputs, Intermediate calculations and outputs of the Transformer

Positional Encoding The positional encoding appends necessary information about the position of the tokens in the sequence. This is required because the Transformer model doesn't rely on recurrence nor convolutions. The authors propose using sinusoidal functions sinus and cosine to encode the position of the token and the dimensions of the vector. Figure 3.3.3 depicts the larger building blocks of an encoder. In this example Z is the resulting matrix from self-attention which is a stack of z_1, z_2, z_3. R is the resulting matrix which is served as the input to upcoming encoder layer. Left, arrows are groing from x to z and from z to r which depicts the residual connections. These are followed by layer normalization. Both input x_i and output r_i have the same dimension such that the encoder block can be stacked. The intermediate vectors z_i are the vectors calculated by the self-attention mechanism.

3.3.2 Self Attention

The self-attention mechanism can be seen as a three staged process. The stages are vizualized in figure 3.3.4 and described step by stem below..

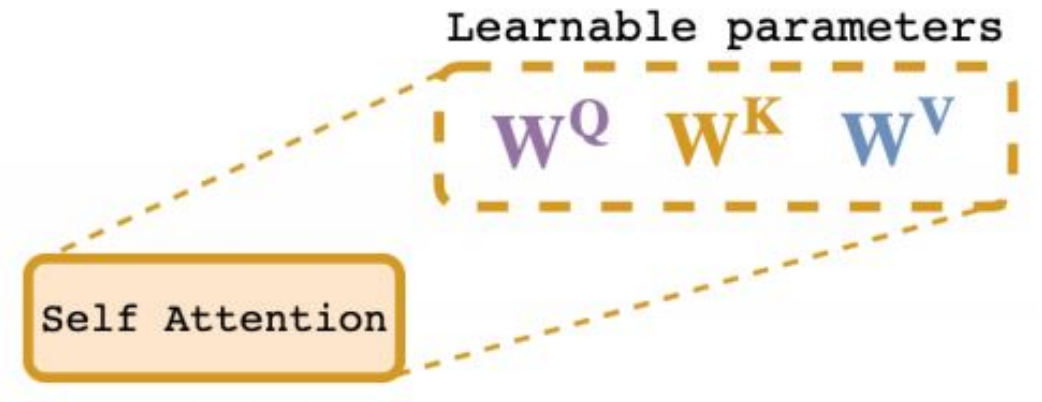


Figure 3.3.4: The learnable parameters in self-attention, masked self-attention and encoder-decoder attention.

we need to explain the different view adopted by the author. Their approach suggests to consider the attention function as a mapping of a query (Q) and a key-value pair (K, V), where the output is the weighted sum of the values and the weight assigned to each value is computed by a alignment function of the query and the corresponding key. This alignment or compatibility function as they name it is described as follow:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where Q, K and V are respectively the query, key and value. d_k refers to the dimension of the queries and keys. The softmax function is a normalization function that accepts a vector z of K real numbers and transforms the components of the vector into a probability distribution, that means after applying softmax, each component z_i of the vector will be bound by (0,1) and the sum of all of them will yield 1. The formula is given as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for \quad i = 1, ..., K$$

Calculating Q, K, V Each embedded representatino of the input is used, alongside the learnable parameters W^Q, W^K and W^V, to compute a query, key and value vector. These are computed by taking the dot product between corresponding matrix and the embedding vector.

Scoring This stage calculates the attention scoring. When scoring the attention for the input "Je" its corresponding query q_1 is used to take the dot product with every key k_i in the sequence. The scores are then normalized by dividing each score by $\sqrt{d_k}$ where d_k is the dimentionality of the key vector. Lastly, a softmax is applied to all of the numbers to scire their importance.

Weighted summation To produce the final output z_1 the weighted sum of the softmax output with corresponding vale is taken.

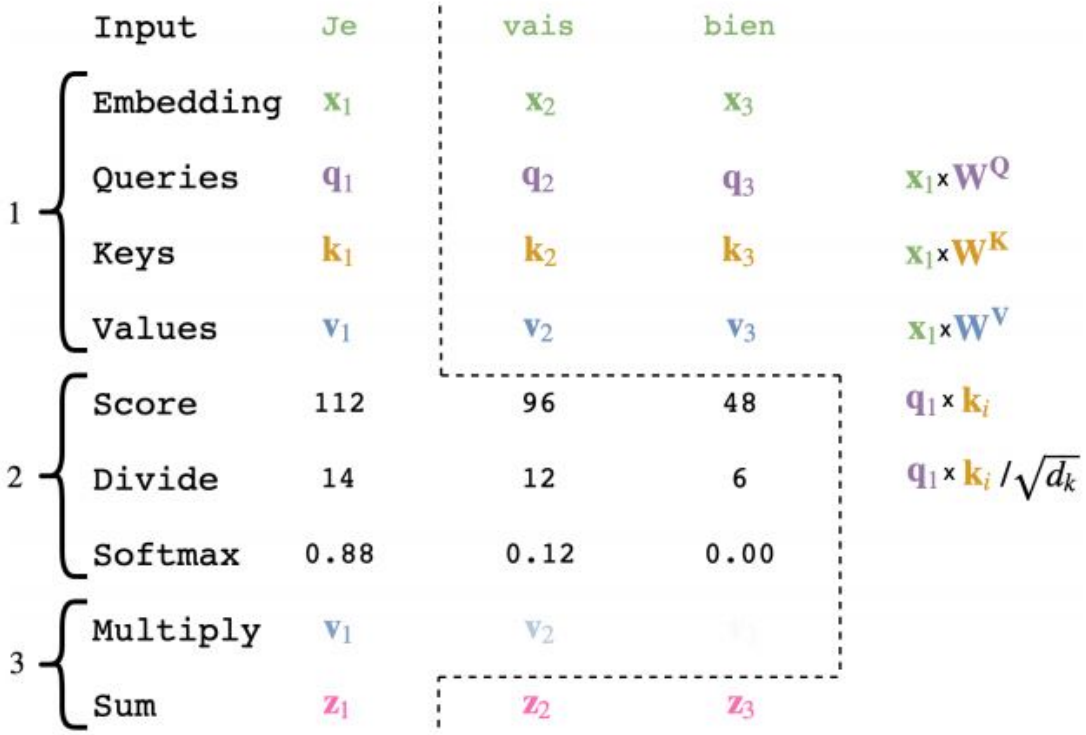


Figure 3.3.5: A three staged process of obtaining the attention output Z given the embeddings of an input sequence X: We see the step number 1: getting z_1, embedding of "je"

However, in the actual implementation vectors are disregarded in favor of matrices which results in faster computations. Figure **??** describes the computations needed to acquire the self-attention output Z from input X.

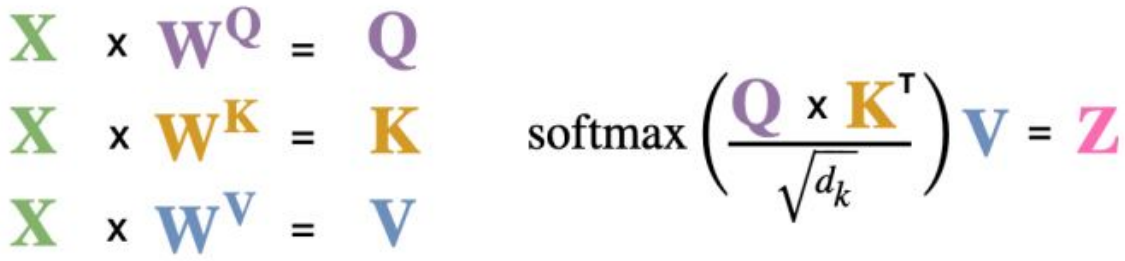


Figure 3.3.6: Query, key and value matrices all calculated using corresponding weight matrix W^Q, W^K, W^V in conjunction with input matrix X. The scaling factor $\sqrt{d_k}$ is the square foot of the dimension of the key vectors.

3.3.3 Multi-head Attention

The transformer does not only use one attention head per layer but eight of them. The first sub-layer is a multi-head self attention layer. Doing this improves the model's capabilities of attending other parts of an input sequence. This gives each layer 24 learnable matrices W_0^Q, W_0^K, W_0^V, ...W_7^Q, W_7^K, W_7^V. This in turn yields Z_0...Z_7 which is multiplied by another weight matrix to give the final output Z of the multi-head attention layer. For the first encoder layer, the input is the input text embedded but for all the other $l = 2, ..., L$ encoder layers, the output of one encoder i is the input of the encoder i+1.

The multi-head mechanism runs through the scaled dot-product attention multiple times in parallel. The resulting scaled dot-product attentions are then concatenated and linearly transformed into the expected dimensions. The authors claim that this approach permits the attention take into account "different subspace representations at different positions" improving the performance of single head attention mechanisms. Figure 3.3.7 illustrates the processes described above.

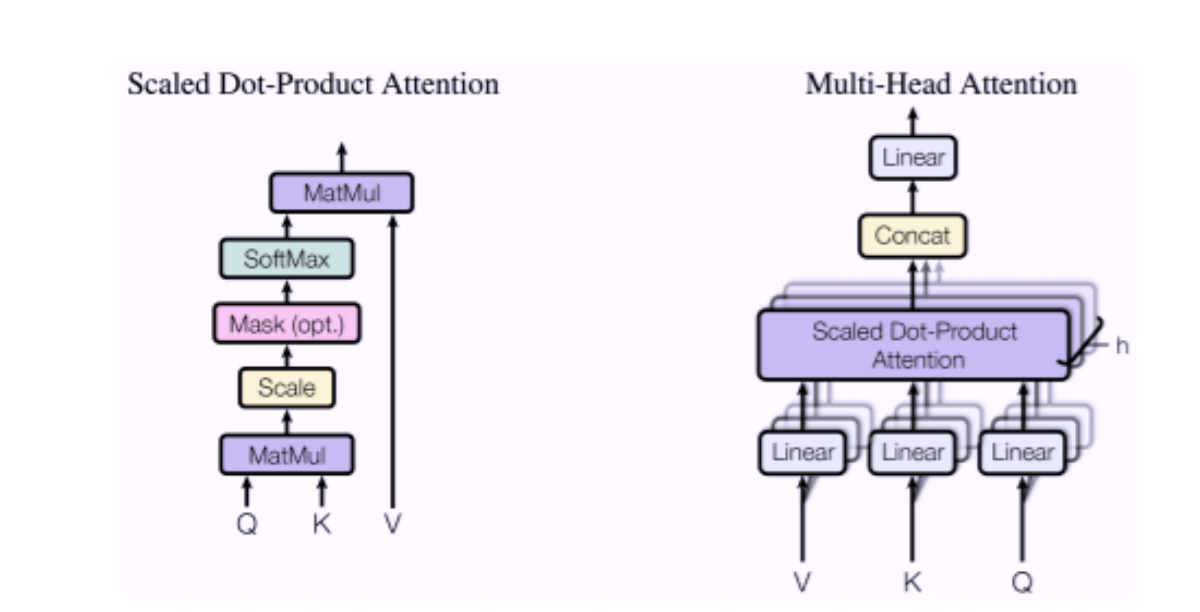


Figure 3.3.7: Multi-headed scaled dot-product self attention

The output of the attention is normalized and passed to the feed forward network. Consequently, the output of the encoder block is fed to the decoder, directly to the block which as an identical structure as the encoder, except this block is preceded by a Masked MultiHead Attention that processes the output embeddings. This masking and the shifting of output embeddings to the right are measures to make sure that the predictions are based on known outputs for previous positions. **Residual Connection and Normalization** We can observe in figure Figure 3.3.1 residual connections (also known as skip connections) short-cutting every sub-layer to the normalization layers. These skip connections serve for both forward and backward passes and are mechanisms to avoid the problem of vanishing and exploding gradients. We've seen similar workarounds in LSTM by preventing the repeated flow of gradients through non-linear activation functions such as sigmoid and tanh. The normalization is applied to mitigate the problem known as covariate shift, which refers to the distributions of the training and test sets being different. This disparity is reduced by fixing the mean and the variance of the summed inputs of the layer.

Feed Forward Networks The sub-layers that transform the output of the encoder and decoder are fully connected feedforward networks. According to the authors of the Transformer paper [188], they implement two linear transformations with a ReLU activation in between.

Now, that we understood the general idea of multi-head Self Attention, let's go through

the mathematical aspect:

The first sub-layer is a multihead self attention layer. for the first encoder layer, this input is X but for all the other $l = 2, ...L$ encoder layers, the input would be the output from the previous encoder H_{l-1}. We start by extracting m differernt representations of this input based on the **scaled dot product attention**. Each of these $i = 1, ..., m$ representations is created un the smae way but with differerntly initialized extraction matrices. This is ti extract different language features and the technique is called m-head attention. The language features that the attention heads find are defined based on three matrices, a so-called Query, Key and Value Matrix $(Q_l^{(i)}, K_l^{(i)}, V_l^{(i)})$. These are calculated seperately for each of the heads and for each of the encoder layers. They are defined as the one head attention described previously, as simple matrix multiplication of the inputs and trainable weight matrices $(W_{Q_l}^{(i)}, W_{K_l}^{(i)}, W_{V_l}^{(i)})$:

$$Q_l^{(i)} = H_{l-1} * (W_{Q_l}^{(i)} \tag{3.30}$$

$$K_l^{(i)} = H_{l-1} * (W_{K_l}^{(i)} \tag{3.31}$$

$$V_l^{(i)} = H_{l-1} * (W_{V_l}^{(i)} \tag{3.32}$$

$$\tag{3.33}$$

These queries, keys and values matrices are combined for each head in a matrix manipulation called scaled dot product attention. This is the manipulation that highlights which input sequences the head pays attention to, and to what extent. This is shown below where ϕ is the softmax funtion and d_k is the dimention of the key matrix $K_l^{(i)}$, which in the case of BERT is always 64.

$$Z_l^{(i)} = \phi(\frac{Q_l^{(i)} * K_l^{(i)}}{\sqrt{d_k}}) * V_l^{(i)} \tag{3.34}$$

To gain information from all the m attention heads, they need to be combined efficiently. These m attention head representations are therefore concatenated and compressed with a matrix multiplication with a layer specific trainable weighting matrix as below:

$$Z_l = Concat(Z_l^{(1)}, ..., Z_l^{(m)}) * W_l \tag{3.35}$$

These internal encoder representation Z_l now have the same size as the input sequences so they can be residually connected to the input and layer normalized to improve training and prevent vanishing gradients. This is shown below for the first vector of the Z_l matrix.

$$z_{l,0}^{'} = LayerNorm(z_{l,0} + h_{(l-1),0} \tag{3.36}$$

Layer Normalization is the technique that will normalize the input values to the later layers to solve the problem of covariate shift where the data changes distribution over the layers of the network. Residual connections are added to prevent vanishing gradients and because it makes training more stable by reducing the number of weight matrix multiplications in deep neural networks. Bot methods are commonly used to stabilize trainign of large networks like BERT however the theory behind them is out of the score pf this project. This internal representations is then passed through a feed forward neural network parameterized by the weight matrices $U_l, \hat{U}_l$. This is to efficiently summarize the most important components of the self attention for passage to the next encoder layer.

$$H_l^{'} = \sigma(Z_l^{'} * U_l) * \hat{U}_l \tag{3.37}$$

Finally, this final representation is normalized and residually connected again, row by row

$$h_{l,0} = LayerNorm(z_{l,0}^{'} + h_{l,0}{}' \tag{3.38}$$

This matrix of row vectors is then the final output of the encoder layer l. This is a summary of relevant language and context information, ready to be passed to the next encoder layer or to be used for classification.

3.4 State-of-the-art

3.4.1 Deep Contextualized Word Representations : ELMo

Word

embeddings based on GloVE and word2vec were the state-of-the-art representation

of words in a numeric format. However they are fixed representations. These shallow structures are able to include context, but only the nearby local context. For example CBOW or skip-gram incorporate 5 to 10 context words in practice that influence a word embedding. Therefore, they lack to project all global connections. The same words are often used acriss many documents in different linguistic contexts(Polysemy). In 2018, Peter et al. [144] developed a "deep contextualized word representation", which overcomes this drawback [144]. These word representations are created with a deep bidirectional Language Model (biLM). Furthermore, the model is pretrained on a large text corpus.

First, let's recall the idea of Language Models (LMs). For a set of N words (w_1, .., w_N) a LM calculates the probability of the whole sequence. It used the conditional probability of the next word w_i given the history of previous words (w_1,...,w_{i-1})

$$p(w_1, .., w_N) = \prod_{i=1}^{N} p(w_i|w_1, w_2, .., w_{i-1}) \tag{3.39}$$

Peters et al. use this underlying structure to build word representations. At first a word embedding v_i is passed into a LSTM with K layers, that processes the sequence in forward direction. Every Layer k = 1, . . . ,K outputs a context-dependent word representation $\vec{h_{i,k}}$ for the word w_i. The top layer in the LSTM is trained to predict the next word w_{i+1} and uses a FNN with a softmax layer on top to compute the output. Another LSTM processes the sequence in reverse direction, as in a bidirectional LSTM. It predicts the previous word w_{i1}. To combine both directions into a biLM, they maximize their joint log-likelihood as optimization criterion:

$$\sum_{i=1}^{N} logp(w_i|w_1, w_2, ..., w_{i-1}, \theta_x, \theta_{\vec{LSTM}}, \theta_s) + logp(w_i|w_{i+1}, w_{i+2}, ..., w_N, \theta_x, \theta_{LSTM}, \theta_s \tag{3.40}$$

θ_x represents the parameters for the input word embeddings E and θ_s mark the parameters for the softmax-FNN. The LSTMs have separate parameters for forward and backward direction. The softmax layer takes the LSTM states to predict the word w_i given the input sequence. Peters et al. call their approach Embeddings from Language Models (ELMo). It combines the input word representation v_i and the hidden states

$\vec{h_{i,k}}$, $h_{i,k}$ from the bidirectional LSTM.$\vec{h_{i,k}}$, $h_{i,k}$ are concatenated to $h_{i,k} = [\vec{h_{i,k}}, h_{i,k}]$. Thus, their K-layered biLM calculates a set V_i of 2K + 1 representations for every word w_i:

$$V_i = v_i, \vec{h_{i,k}} h_{i,k} | k = 1, ...K \tag{3.41}$$

$$= h_{i,k} | k = 0, ...K \tag{3.42}$$

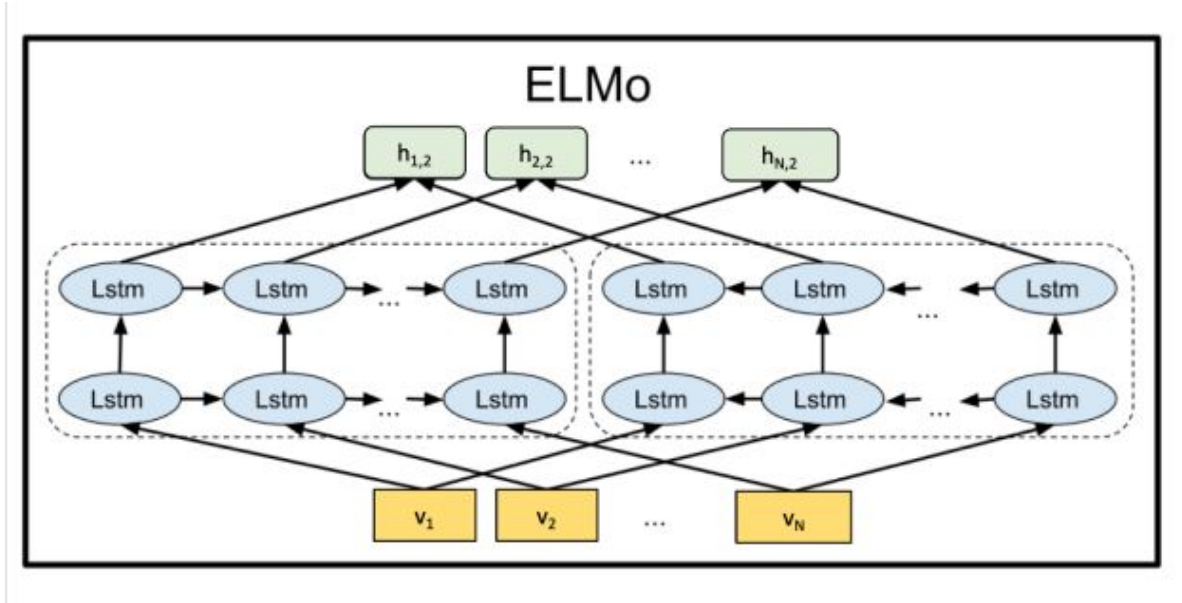


Figure 3.4.1: ELMo uses a 2-layered Bidirectional Language Model as architecture [45]. A sequence of embeddgins $v_1, v_2, ..., v_N$ is passed into the bidirectional LSTM to retrieve their concatenated word representations $h_{1,2}, h_{2,2}, ...h_{N,2}$ that are computed by the second LSTM-layer.

The bidirectional structure of ELMo is shown in Figure 3.4.1. Here, the underlying word representations in V_i are denoted with $h_{1,2}, h_{2,2}, .., h_{N,2}$ for the second bidirectional LSTMlayer. ELMo uses these representations and optimizes them with respect to a given task, e.g. document classification. The word vector

$$v_i^{task} = \gamma^{task} \sum_{k=0}^{K} s_k task h_{i,k} \tag{3.43}$$

is a linear combination of all word representations for word w_i that is optimized with parameters s^{task} and γ^{task} . The different representations in V_i are weighted by softmax normalized weights task . Besides, all ELMo vectors are scaled by γ^{task}. Peters et al.

pretrain the representations in V_i on a large corpus. Afterwards, they learn s^{task} and γ^{task} for every particular task. When classifying documents with a RNN, for example, the previously trained representations v_i^{task} are passed on as input to the RNN. During the RNNs learning phase, the task-specific weights (γ^{task}, s^{task}) are adjusted. The simplest representation of a task-specific word vector, which ELMo can create, is the input embedding itself v_i or the representation $h_{i,K}$ from the last bidirectional LSTM-layer. Their approach outperforms current state-of-the-art methods in different NLP tasks, such as question answering, named entity recognition and sentiment analysis. Using ELMo the task-specific error reduces by 0.7% – 4.7% [144].

Although ELMo representations are an improvement over traditional word embeddings, they are treated as fixed parameters. Thus, a model for a specific task still needs to be trained from scratch.

3.4.2 ULMFit

Closely following the results of ELMo, another development was made with ULMFiT by Howard et al. [72]. ULMFiT is a deep LSTM based model that also does general Language Modelling to improve feature extraction. ULMFiT uses several techniques to ensure that models actually converge well during fine-tuning, such as using AWD-LSTMs proposed by Merity et al. [124], triangular learning rates and learning rate scheduler to avoid catastrophic forgetting. AWD-LSTMs does not just apply Dropout to the recurrent state h, but each gate in the layer as well. These tactics, combined with the model's size again pushed state of the art forward and enabled NLP Transfer Learning [72] properly for the first time. Previous attempts at fine-tuning pre-trained models have proven tricky, and catastrophic forgetting or general poor convergence have long been a problem. These problems have usually become worse with deeper and larger models. While ULMFiT does solve many of these problems, it is hard to train and sensitive to hyperparameter settings for convergence. Thus, Howard and Ruder introduced Universal Language Model Fine-tuning (ULM-FiT), a transfer learning method that can be applied on any NLP task, providing the opportunity of only fine-tuning a model for a downstream task instead of training it from scratch. This was similar to transfer learning in computer vision.

3.4.3 GPT-2

GPT-2 with 117M trainable parameters is a Transformer based language model comprised of 12 slightly modified Transformer Decoder blocks stacked upon each other, see Figure 3.4.2. Each bloc consists of layer normalization and lasily a residual connection around a feed forward neural network. After the 12 blocks, layer normalization is followed.

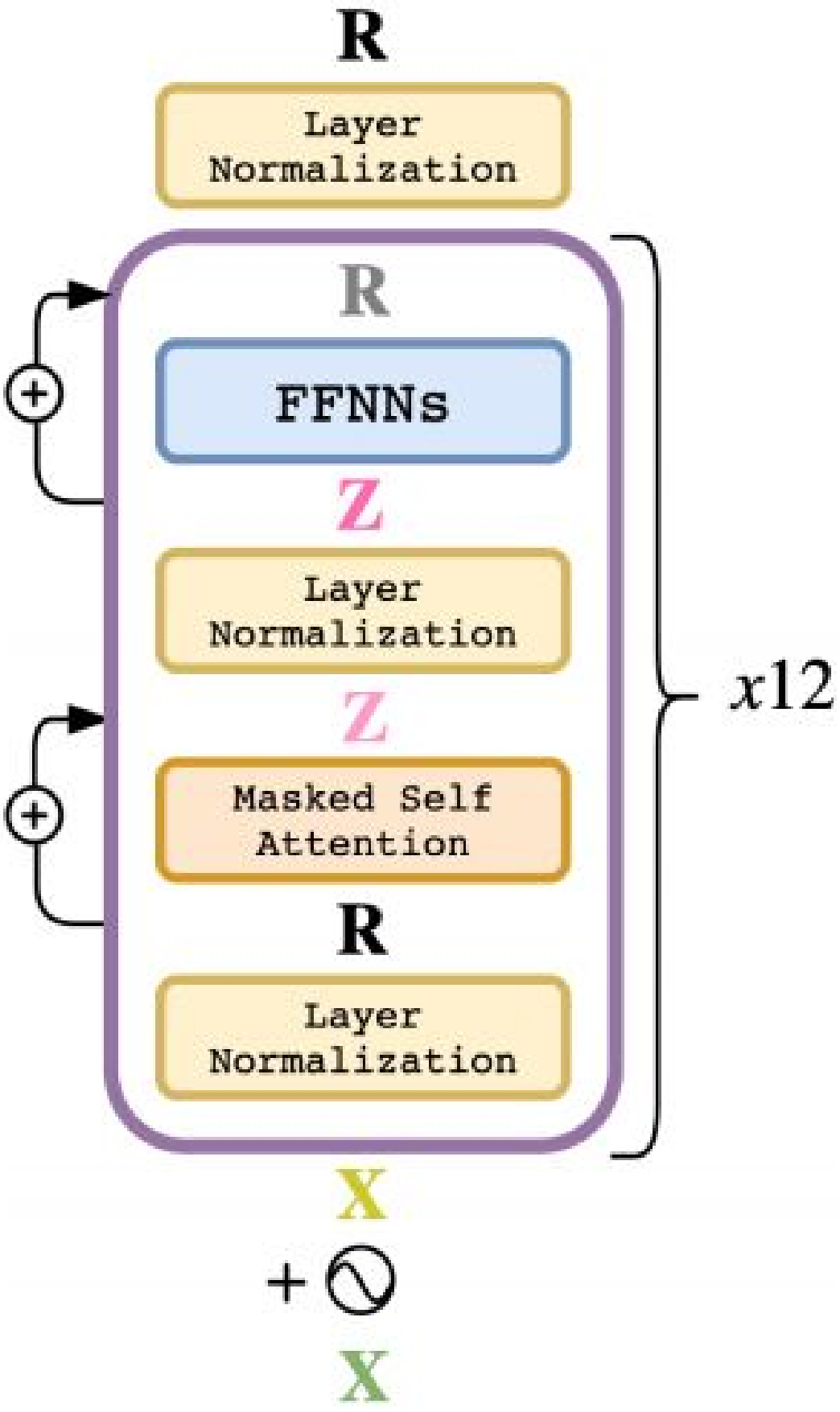


Figure 3.4.2: Architecture of the OpenAI GPT which is solely based on the Transformer decoder.

3.4.4 Bi-Directional Encoder Representations from Transformers (BERT)

We will now discuss the theoretical groundings of BERT, a state of the art language representation and classification from Google. First, we will highlight the background of BERT's creation and the problems with language representation it is supposed to solve. From there we will discuss its neural structure and walk through how the model constructs representations of text data. Finally, we will discuss how BERT's specific input pre-processing and pre-training procedure have made it a widely praised NLP for a range of tasks.

Beyond Word Embeddings

Despite the ease and flexibility of using pre-trained word embeddings in language modelling they have one main drawback: each word is represented by just one vector. This means that a word like "set" which has multiple meanings in different contexts is only embedded in one part of meaning space, thus losing language information. This problem is theoretically taken care of by using RNNs or LSTMs which can carry information continuously over sequences to create a context for the current word. However, practically these models can be insufficient to provide rich context data to predict context-heavy language features such as humor or sarcasm (Zhang et al., [203]). This is partially since long term dependencies are formed recurrently so it can be difficult for a model to know which features to pass forward during training.

This problem of insufficient language context from RNNs has been tackled with bidirectional LSTMs (Graves and Schmidhuber, [60]) and attention mechanisms (Bahdanau et al., [12]) for LSTMs with some success. However, one new method for providing context rich information to the network has been the BERT architecture from Google (Devlin et al., [45]) based on the transformer from Vaswani et al. ([188]).

BERT (Bi-directional Encoder Representations from Transformers) and its underlying machinery, transformers, are based on totally different principles from recurrent neural language models: relies on attention for its representations instead of recurrence.

BERT is a Transformer-based model for language encoding, introduced in [[45]] by Devlin et al. The authors identify the fact that models could only be trained

unidirectionally as one of the big limitations of previous approaches to language modeling. This meant that a token could only be encoded using the information of either the tokens to its left or the tokens to its right, but never using information from both combined at the same time. The objective in creating BERT (Bidirectional Encoder Representations from Transformers) was to create a model that could take the full context of a token into account, left and right.

In order to use BERT for some down-stream task (like machine translation or text summarisation, classification), two steps are necessary: 1. The model needs to be pre-trained. This means the model is not yet trained in any task-specific way, but instead is taught to encode language itself in a sensible way. This is done so the same model can be used for several different down-stream tasks without needing to be trained from scratch. Task-specific training (fine-tuning) is done in the next step. Pre-training results in a general purpose Transformer that can encode input tokens. This pre-training is done on unlabeled training data over two different training tasks, described later in this section.

2. The model can then, once it is initialized with the parameters obtained through pre-training, be fine-tuned to be used for a particular down-stream task. In order to do this, importantly, the architecture of the model itself does not need to be changed. Instead, the same pre-trained model can be applied to several different tasks, by layering task-specific layers on top and training the model on labeled training data pertaining to the desired downstream task.

Since the authors made their pre-trained BERT models (a larger and a smaller one) available for download and free to use, this means that with relatively little effort, these already pre-trained models can be applied to a wide variety of text-based tasks.

The transformer architecture builds on this idea of training weighted representations of the previous hidden states. However, instead of applying attention in conjunction with recurrence, the authors show that one can extract relevant language features simply through applying attention to the underlying word vectors from the original sequence. This process is called encoding the text, and while the original transformer paper used this encoded representation for machine translation, it turns out that training contextually encoded language features is useful in many NLP tasks. Furthermore, since the transformer does not use any form of recurrence relations this allows for significantly more parallelization in training.

Transformer based models such as BERT have been able to achieve state of the art results on a range of NLP tasks. Also, the pre-trained weights of the BERT model are freely distributed online to be downloaded used for transfer learning. We will now explain the mathematical theory behind our comparison model, BERT. Starting with its structural components, the encoder. Proceeding with attention and its representation of data.

Structure of BERT

BERT, as we have said is a state of the art language representation model that can be finetuned to many different Natural Language Processing tasks. At its most basic, BERT is a sequence to sequence language representation model. It takes as an input a sequence of language information $X = (I_0, ..., I_n)$ and outputs a contextualized vector representation $H = (h_0, ..., h_n)$ of the elements of the input sequence.

Note that there are some slight changes in notation from previous sections. Firstly, we don't pass word vectors directly to the BERT model. Instead we pass enhanced word-parts based on the pre-processing defined in the next section. For that reason, we talk about input vectors I rather than word vectors x.

Also, in the RNN setting, we talked about a sequence of words $(x_1, ..., x_N)$ however now we talk about a sequence of input vectors which run over the index $0, ..., p : p > N$. This is because the BERT pre-processing both splits words into word-parts and inserts placeholder tags before the sequence and after sentences. This will be discussed in more detail in the preprocessing section below, however it means that the input vectors do not directly correspond to the underlying words in the sequence.

Due to the pre-sequence placeholder input I_0 and the design of the BERT model, the output representation h_0 becomes a distributed representation of the underlying sequence in the same way that the final hidden state of a RNN does. This representation can also be used directly for classification in the same way that the hidden states in an RNN can be. By adding an extra hidden layer to the original BERT model and activating it with a softmax function, we obtain a classification model.

$$\hat{y} = \phi(V h_O) \tag{3.44}$$

Encoders BERT accomplishes the task of being a highly generalizable language

representation framework through stacking encoders. Encoders, as one could infer from the name, are a neural network architecture taken from the transformer and used to create encoded representations of text.

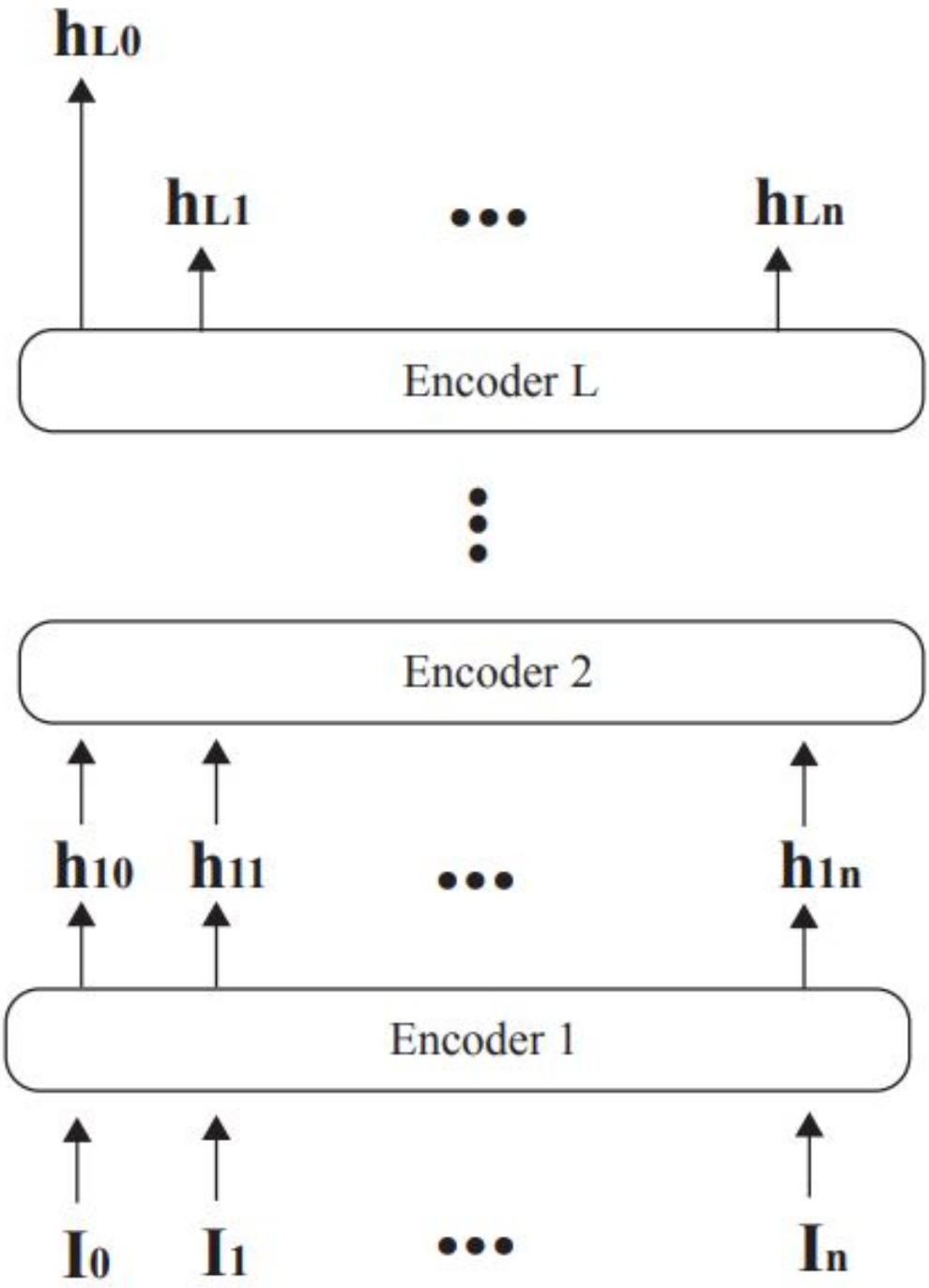


Figure 2.9: Bert layers

Figure 3.4.3: Bert layers

Each encoder layer is designed to abstract language patterns from an input sequence,

forming more nuanced patterns as the information flows up the layers. The input to the first encoder layer is the language inputs and the last outputs are the final encoded language information after being passed through L encoder layers.

$$H_1 = Encoder(X) \tag{3.45}$$

$$. \tag{3.46}$$

$$. \tag{3.47}$$

$$. \tag{3.48}$$

$$H_L = Encoder(H_{L-1}) \tag{3.49}$$

The classification task is then based on the first vector h_{L0} of the final H_L representation.

Each encoder layer is a further combination of two sub-processes. The inputs to the encoder are first passed through a multi-head self attention layer, which uses a series of matrix manipulations to extract language features from the inputs. These manipulations are referred to as multi-head self attention. From there, there is a residual connection and normalization process on the outputs before they are passed to the second sub-layer, a feed forward neural network. This second sub-layer extracts the most important features from the attention layer and after normalization and residual connections sends the outputs onward to the next encoder layer.

This process can seem convoluted, but it is just the same process repeated over and over. First a series of matrix multiplications extract language features from input sequences which are then weighted together by a FNN to highlight the most important features and combine them.

to sum up, Devlin et al. [45] utilized the Transformer encoder for the architecture of their model BERT, which is comprised of 12 Transformer encoder stacked upon each other as depicted in Figure **??**. This means that BERT is ”nondirectional”, i.e., context aware about previous and upcoming tokens. Consider a language model that is unidirectional, such as GPT-2, which is efficiently trained by predicting the next word. ELMo is bidirectional trained by predictiong the next word forward and backward. BERT will be trained to predict the word based on all surrounding words without

sequence consideration, but based on attention mechanism.

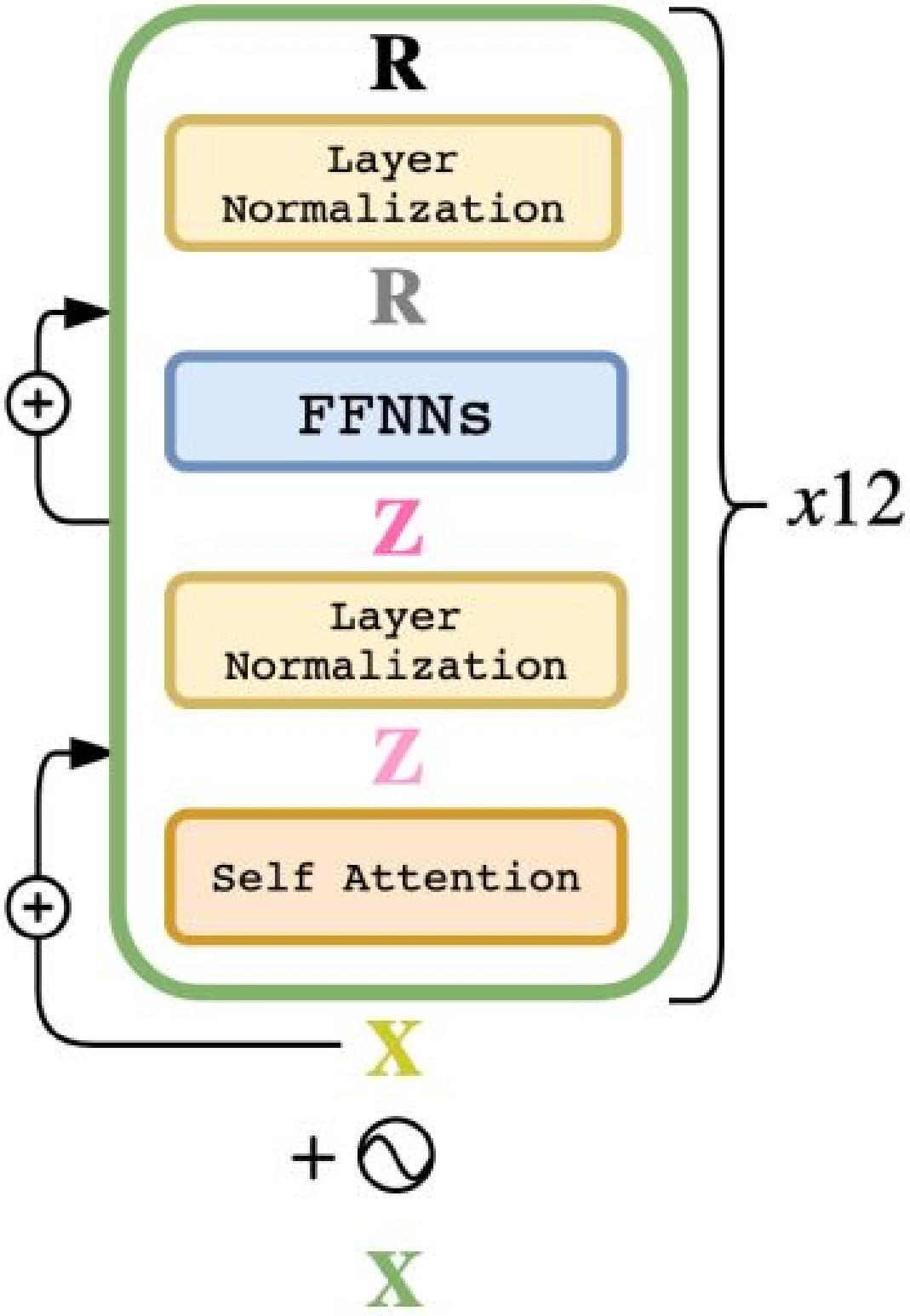


Figure 3.4.4: Architecture of BERT solely based on the Transformer encoder

Training and Fine-Tuning Bert

Much of the matrix manipulation explained in the section **??** Transformer is taken directly from the original transformer implementation. However, BERT proposes some novel ideas for sequence pre-processing and pre-training which also have helped it set state of the art results.

The architecture of the model itself is almost identical to the Transformer architecture described in section **??**. Perhaps more interesting is how textual input is processed and how the model is (pre-)trained: As input, BERT accepts textual sequences that may each be composed of either a single sentence or a pair of sentences, where a ”sentence” means any arbitrary span of contiguous text, not necessarily sentences in the grammatical sense. Each such sequence is preceded by a classification token ([CLS]). The final hidden state for this token can be trained to obtain an aggregated representation of the entire sequence. This is useful for some classification tasks, like summary/non-summary sentence classification for extractive summarisation. If the sequence consists of two sentences, then they are separated by a [SEP] token. Additionally, BERT adds a so-called segment embedding to each token, which indicates whether it belongs to Sentence A or Sentence B. The input representation of each token is obtained by adding together the token’s WordPiece embedding (see [[198]] for details), segment embedding and positional embedding. The latter encodes where in the sequence the token is located. This is necessary, as BERT, being a Transformer model, is not going through the tokens sequentially, and therefore does not ”know” the order of the input tokens. Figure 3.4.5 illustrates the BERT input representation.

Input Sequences As was mentioned before, BERT proposes two novel ideas for coding input data. Firstly, it uses word-piece tokenization and embeddings from Wu et al. ([198]) which splits parts of words to get richer word information and to decrease vocabulary size. They also introduced several BERT specific tokens including the classification token [CLS] and the sentence end token [SEP].

The classification token is important since it always comes at position zero and thus is encoded to the representation vector $h_{L,0}$.

This works because this vector has access to all language information in each self attention layer but doesn’t encode any information itself, thus it becomes a distributed summary of the language information in the rest of the sentence. The sentence

end tokens are also important since they allow the model treat different sentences differently.

Secondly, the BERT combines word-piece embeddings with segment embeddings and positional embeddings.

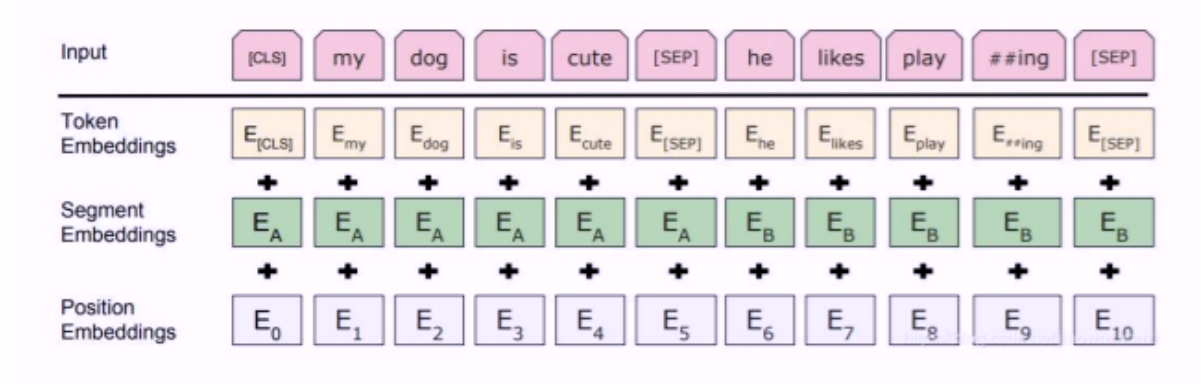


Figure 3.4.5: Embeddings for BERT

This gives the model extra information about which sentence the words belong to and where in the sequence the words come. These positional embeddings are especially important since without them, it is not possible for the scaled dot product attention to differentiate between sentences with the same words in a different order, for example "The cat sat on the dog" compared to "The dog sat on the cat."

The positional embeddings are based on sinusoidal curves. These embeddings are of the same dimension as the word vectors and thus are defined for position i in the text sequence and embedding dimensions 2j and 2j+1 in the embedding vector. We define the values of the position embeddings as below where $d_{embeddings}$ is the dimension of the word embeddings used in word representation, for BERT this is 512.

$$PE_{i,2j} = sin(i/1000^{2j/d_{embeddings}} \tag{3.50}$$

$$PE_{i,2j+1} = cos(i/1000^{2j/d_{embeddings}} \tag{3.51}$$

$$\tag{3.52}$$

These values of positional embeddings described in Equation 3.52 are desirable because they have the property that for any positional difference k in the word sequence, we can represent PE_{i+k} as a linear function of PEi (Gehring et al., [55]).

This allows the model to gain information about the position of words in relation to each other.

Finally, the segment embeddings are a learned embedding vector based on the sentence end tokens to allow the model to represent different sentences in different embedding spaces. This can be helpful to allow the sentences to be divergent in meaning or emotion, which is very relevant to our dataset.

Pre-Training BERT The other key to BERT's general success in representing language is in its training procedure. The pre-trained BERT weights that we use have been trained on two different language representation tasks to make the attention weighting more generalizable. These tasks are next sentence prediction which is a binary classification problem, and word prediction from context which is a multi-class classification problem. These tasks work well together due to a novel masking approach that the authors use.

Once the input representation is obtained, BERT is pre-trained on it by trying to solve two tasks, as mentioned above. These two tasks are the following: 1. Masked Language Model (MLM) Some percentage of the input tokens (in the paper the authors chose 15%) is masked and BERT is tasked with predicting them by using the entire context - left and right. Notably, only these masked tokens are predicted by the model, and no attempt is made to reconstruct the entire input. 2. Next Sentence Prediction (NSP) This task is meant to help the model learn the relationships between sentences: To create the pre-training dataset, for each training instance two sentences A and B are picked from the training corpus. With 50% probability, sentence B will be sentence A's successor, with 50% probability it will be a random sentence from anywhere else in the corpus. BERT is tasked with predicting (binary) if sentence B is indeed sentence A's successor.

For pre-training, the authors used BooksCorpus ([34]) and English Wikipedia texts. The pre-trained models BERTLARGE and BERTBASE are publically available at: https://github.com/google-research/bert

The training works through the following training procedure. Two sentences are sampled from the corpus either following each other or from different parts of the corpus. At the same time, several of the words in sentences A and B are masked with a [MSK] token so the model can't see the words, or changed to a random word in the vocabulary to confuse the algorithm. The model then learns based both on the task of

predicting whether the sentences are consecutive or unrelated and the word prediction task based on context. The two loss functions are added together and minimized for a large sample set. The theory behind this is that learning contextual information is helped by the next sentence prediction task, while the masking task forces the model to keep a higher amount of word-level information in its representations. This also helps the multi-head attention from converging to finding too similar representation features in the encoding since the model is trained also on word prediction, thus the encoding needs to still be grounded in word meaning features.

BERT has advanced the state of the art on several natural language processing benchmarks including beating human performance for the first time on the highly contextual Stanford Question Answering Dataset (Rajpurkar et al., 2018). This should help make BERT perform well on emotion classification tasks and thus makes it a good candidate to train our problem on

In the next two subsections, we will look at variants of BERT that aim to improve on the original model.

3.4.5 RoBerTa

In [112], the authors claim that BERT, as introduced in the original paper [[45]] is actually undertrained and show that with some modifications, significantly better results can be achieved, which are competitive with the performance of every model published after BERT. They name their modified BERT-version RoBERTa. (a robustly optimized BERT approach).

Apart from changing some of BERT's hyperparameters, the main differences from BERT pertain to how the model is pretrained. The main changes from the training suggested in [[45]] are the following:

Dynamic Masking In the original BERT, the masking of the input sequences is done in a static way: Only once in pre-processing. To ensure that BERT will encounter the same sequences with different masking patterns, the training data instances are multiplied by 10 before masking.

RoBERTa, however, applies dynamic masking instead. A new masking pattern is generated every time a sequence is fed to the model. This means that the model will encounter many more different masking patterns of the same instance. This in turn

removes the need to drastically increase the number of training instances.

Full sentences As opposed to BERT, RoBERTa uses exclusively full sentences as input to the model. Such sentences are sampled contiguously from the documents, such that the total length does not exceed 512 tokens. If document boundaries are crossed while sampling, a special inter-document separator token is inserted.

Training in large mini-batches BERT was trained in 1 million training steps with a batch size of 256 sequences. RoBERTa, on the other hand, was trained in only 125.000 steps, using a much larger batch size of 2.000 sequences. The authors express uncertainty over whether they have already found the ideal batch size with this, but it produces better results than the original BERT while taking less time to train.

Larger byte-level BPE

BPE (byte-pair encoding) is a hybrid between character- and word-level text encoding. It bases its encodings on subword units, for example: Instead of encoding the word ”playing” or each of its letters separately, BPE might encode ”play” and ”ing”, building blocks which can be re-used to form other words as well. This allows for a much larger vocabulary as would otherwise be possible. However, a lot of the time in BPE, a large portion of the encodings are encodings of single uni-code characters, which limits the total number of words that can be captured.

It is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. We adapt this algorithm for word segmentation. Instead of merging frequent pairs of bytes, we merge characters or character sequences.

RoBERTa instead makes use of a variation of BPE introduced in [[151]], which is based on bytes instead of characters. This means that less subword units are needed to encode a larger vocabulary. While BERT used character-level BPE with a size of 30.000 subword units and requires the input to be tokenized in preprocessing, RoBERTa requires no such preprocessing and uses byte-level BPE to encode 50.000 subwords.

Longer pretraining on larger data sets RoBERTA was trained over 500.000 steps, while BERT over 100.000, and trained on 160GB of textual training data, resulting in much better end-task performance.

Thus, RoBERTa is a BERT model with a different training approach. RoBERTa removes next-sentence prediction (NSP) tasks and adds dynamic masking, large mini-batches and larger Byte-pair encoding. In BERT the input is masked only once such that it has the same masked words for all epochs while with RoBERTa, masked words changes from one epoch to another. We should also note that because there is no NSP in ROBERTa/Camembert there is no need to add the special tokens [CLS] and [SEP] to the input.

The pretrained RoBERTa model is publically available at: https://github.com/pytorch/fairseq

3.4.6 Camembert

Camembert is based on RoBERTa [113], which replicates and improves the initial BERT by identifying key hyper-parameters for more robust performance.

In this section, the architecture, training objective, optimisation setup and pretraining data that was used for CamemBERT will be described.

CamemBERT differs from RoBERTa mainly with the addition of whole-word masking and the usage of SentencePiece tokenisation 3.0.2.

Architecture Similar to RoBERTa and BERT, CamemBERT is a multi-layer bidirectional Transformer. CamemBERT uses the original $BERT_{BASE}$ configuration: 12 layers, 768 hidden dimensions, 12 attention heads, which amounts to 110M parameters.

Pretraining objective The model is trained on the Masked Language Modeling (MLM) task. Given an input text sequence composed of N tokens x_1,...,x_N, 15% of tokens are selected for possible replacement. Among those selected tokens, 80% are replaced with the special <mask> token, 10% are left unchanged and 10% are replaced by a random token. The model is then trained to predict the initial masked tokens using cross-entropy loss.

Following RoBERTa tokens are masked dynamically instead of fixing them statically for the whole dataset during preprocessing. This improves variability and makes the model more robust when training for multiple epochs.

Since the input sentence is segmented into subwords using SentencePiece, the input

tokens to the models can be subwords. An upgraded version of BERT 666https://github.com/google-research/bert/blob/master/README.md have shown that masking whole words instead of individual subwords leads to improved performance. Whole-word masking (WWM) makes the training task more difficult because the model has to predict a whole word instead of predicting only part of the word given the rest. As a result, WWM is used for CamemBERT by first randomly sampling 15% of the words in the sequence and then considering all subword tokens in each of these 15% words for candidate replacement. This amounts to a proportion of selected tokens that is close to the original 15%. These tokens are then either replaced by <mask> tokens (80%), left unchanged (10%) or replaced by a random token.

Subsequent work has shown that the next sentence prediction task (NSP) originally used in BERT does not improve downstream task performance [[103], [112]], NSP is not used as a consequence.

Optimisation Following [[112]], the model is optimised using Adam [[93]] $(_1 = 0.9, _2 = 0.98)$ for 100k steps. They use large batch sizes of 8192 sequences. Each sequence contains at most 512 tokens. they enforce each sequence to only contain complete sentences. Additionally, they used the DOC-SENTENCES scenario from [[112]], consisting of not mixing multiple documents in the same sequence, which showed slightly better results.

Segmentation into subword units They segment the input text into subword units using SentencePiece [[102]]. SentencePiece is an extension of Byte-Pair encoding (BPE) [[170]] and WordPiece [[101]] that does not require pre-tokenisation (at the word or token level), thus removing the need for language-specific tokenizers. A vocabulary size of 32k subword tokens is used. These are learned on 10^7 sentences sampled from the pretraining dataset. Subword regularisation are not used (i.e. sampling from multiple possible segmentations) in our implementation for simplicity.

Pretraining data Pretrained language models can be significantly improved by using more data [[112], [152]]. Therefore the used data was French text extracted from Common Crawl777https://commoncrawl.org/about/, in particular, OSCAR [[177]] a pre-classified and pre-filtered version of the November 2018 Common Craw snapshot.

OSCAR is a set of monolingual corpora extracted from Common Crawl, specifically from the plain text WET format distributed by Common Crawl, which removes all

HTML tags and converts all text encodings to UTF-8. OSCAR follows the same approach as [[58]] by using a language classification model based on the fastText linear classifier [[85], [84]] pretrained on Wikipedia, Tatoeba and SETimes, which supports 176 different languages.

OSCAR performs a deduplication step after language classification and without introducing a specialised filtering scheme, other than only keeping paragraphs containing 100 or more UTF-8 encoded characters, making OSCAR quite close to the original Crawled data. The unshuffeld version of the french OSCAR corpus is used, which amounts to 138GB of uncompressed text and 32.7B SentencePiece Tokens

3.5 Tokenization

The main tokenizer used in BERT and Camembert are described in this section.

3.5.1 Word-Piece

Wordpiece Model implementation was initially developed to solve a Japense/Korean segmentation problem for the Google speech recognition system. The main advantages is the handling of rare words, words are divided into limited set of common sub-word units for both input and output. (Balance between between flexibility of ”character”-delimited models and the efficiency of ”word”-delimited models.

The approach is data-driven and guaranteed to generate a deterministic segmentation for any possible sequence of characters.

Technique: The main technique is to learn word units from large amounts of text automatically and incrementally by running a greedy algo:

1. Initialize the word unit inventory with the basic Unicode Characters and including all ASCII
2. Build a Language Model on the training data using the inventory
3. Generate a new word unit by combining two units out of the current word inventory to increment the word unit inventory by one. Choose the new word unit out of all possible ones that increases the likelihood on the training data the most when added to the model

4. Goto 2 until a predefined limit of word units is reached or the likelihood increase falls below a certain threshold

Training the segmenter is a computationally expensive procedure if done brute-force as for each iteration(addition of a new word piece unit by combining two existing ones) as all possible pair combinations need to be tested and a new language model needs to be built. It can be done by :

1. Testing only the pairs that actually exist in the training data
2. Testing only pairs with a significant chance of being the best(for example high priors)
3. Combining several clustering steps into a single iteration

A 200k word piece inventory can be built out a frequency-weighted query list in a few hours on a single machine. Different values can be tried: 50k, 100k or 200k and it is not possible to compare perplexity between them. To compare the cost/sentence (log-likelihood used during decoding) can be used.

Example: • sentence: ”Jet makers feud over seat width with big orders at stake” • wordpieces: ”_J” ”et” ”_makers” ”_fe” ”ud” ”_over” ”_seat” ”_width” ”_with” ”_big” ”_orders” ”_at” ”_stake” In the above example, the word “Jet” is broken into two wordpieces “_J” and “et”, and the word “feud” is broken into two wordpieces “_fe” and “ud”. The other words remain as single wordpieces. “_” is a special character added to mark the beginning of a word.

How is the wordpiece model generated ? Data-driven approach to maximize the language model likelihood of the training data, given an evolving word definition. Given training corpus, a number of desired tokens D, the optimization is to select D wordpieces such that the resulting corpus is minimal in the number of wordpieces when segmented according to the chosen wordpiece model.

Compared to the original implementation: Use of a special symbol only at the beginning of the words and not at the bot ends. Cutting the number of basic characters to a manageable number depending on the data (roughly 500 for western languages) Map the rest to a special unknown character to avoid polluting the given wordpiece vocabulary with very rare characters. Total vocabulary of 8k ==> 32k wordpieces achieves good efficiency and fast decoding speed across all pairs of language pairs

tried.

Achievements of wordpiece :

- Balance between the flexibility of characters and efficiency of words.
- Efficient dealing with an essentially infinite vocabulary without resorting to characters only.

3.5.2 SentencePiece

A simple and language independent subword tokenizer and detokenizer is a re-implementation of sub-word units, an effective way to alleviate the open vocabulary problems in neural machine translation. SentencePiece supports two segmentation algorithms, byte-pair-encoding (BPE)[171] and unigram language model [101]. Here are the high level differences from other implementations. Training : SentencePiece trains the segmentation model such that the final vocabulary size is fixed, e.g., 8k, 16k, or 32k. Note that SentencePiece specifies the final vocabulary size for training, which is different from subword-nmt Subword Neural Machine Translation that uses the number of merge operations. The number of merge operations is a BPE-specific parameter and not applicable to other segmentation algorithms, including unigram, word and character.

Previous sub-word implementations assume input sentences are pre-tokenized. This constraint was required for efficient training, but makes the preprocessing complicated because of the obligation of running language dependent tokenizers in advance. The implementation of SentencePiece is fast enough to train the model from raw sentences. This is useful for training the tokenizer and detokenizer for Chinese and Japanese where no explicit spaces exist between words.

Whitespace is treated as a basic symbol

Sentence: Hello_World. Then, this text is segmented into small pieces, for example: tokens: [Hello] [_Wor] [ld] [.]

Lossless tokenization: all the informationis preserved and there is no reliance on manually crafted rules.

3.6 Related work: Domain Specific models

Soon after the publication of BERT, several researchers tested the domain adaptation capabilities of BERT by fine-tuning the language model to specific fields. It has been applied to scientific, biological and clinical data yielding respectively the BioBERT, SciBERT and ClinicalBERT pre-trained BERT language models. This thesis will focus on the first two as their approaches are significantly different from each other but relevant for the current study.

3.6.1 BioBERT

The first domain-specific pre-trained BERT is BioBERT [106], which is optimized for the biomedical field. The authors Lee and Yoon use 4.5B words from the PubMed corpus and 13.5B words from PMC, both archives of biomedical and life science literature. The training of several models was reported to last between 10 days and 23 days depending on the amount of data used and the dataset combinations. The hardware that served the training was composed of 8 NVIDIA V100 GPUs with 32GB memory. They initialize BioBERT with BERTBase, that means, the language model is built on top of the learned weights from English Wikipedia and BookCorpus. Using different combinations of datasets for language model adaptation, the researchers found that the best results were found when combining both PubMed and PMC corpus through 470K pre-training steps, they labeled this BioBERT v1.0. Later, they release a v1.1 using only PubMed but pre-trained with 1M steps and this yielded general performance improvements over the v1.0.

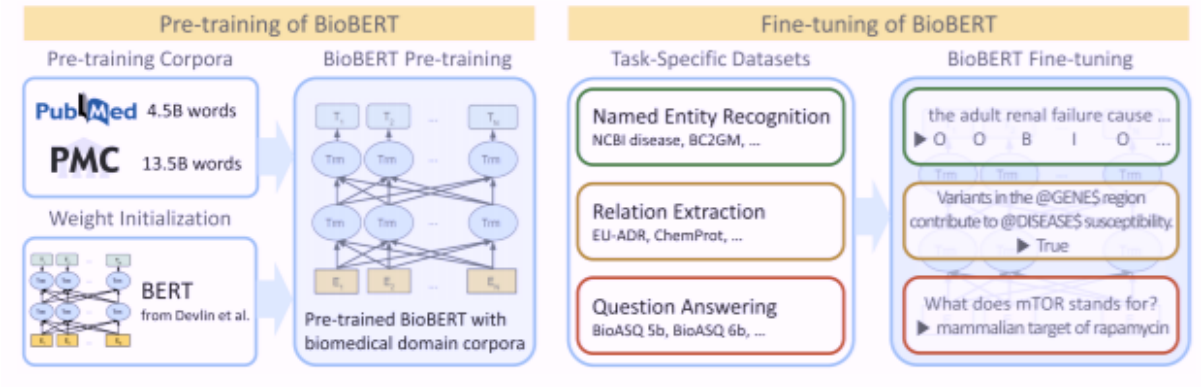


Figure 3.6.1: Overview of the pre-training and fine-tuning of BioBERT (Taken from [34])

The biomedical language model was assessed with an array of 14 task-specific datasets compatible with NER, relation extraction (RE) and QA. To sum up, BioBERT improved the stateof-the-art in 4 out of 8 datasets for NER and 1 out of 3 for RE, always considering F1 score as a metric. For QA, measuring the Mean Reciprocal Rank (MRR), BioBERT exceeded the state-of-the-art in all 3 datasets. BioBERT notably attains higher F1 scores in biomedical NER (0.62) and biomedical RE (2.80), and a higher MRR score (12.24) in biomedical QA.

Lee and Yoon opted for not creating a customized vocabulary, but instead relied on the WordPiece tokenizer to split OOV words into known sub-words or tokens. The main rationale behind this decision is the compatibility and interchangeability between BERTBase and BioBERT models.

3.6.2 SciBERT

Opposed to BioBERT's domain adaptation atop of BERTBase, Beltagy, Lo and Cohan [18] pre-trained from scratch a BERT model using exclusively data from the Semantic Scholar corpus. This corpus includes 1.14M biomedical and computer science papers equivalent to 3.1B tokens and the training process lasted for about a week using a single TPU v33 with 8 cores. The SciBERT model was then thoroughly tested against a collection of 12 task and their corresponding domain specific datasets. These tasks, tackled using embeddings from frozen layers and fine-tuning for posterior comparison, spawned from classification, different sequence labeling and dependency parsing.
SciBERT obtained positive results and achieved state-of-the-art performance for 8 out of 12 tasks. In table 3.1, you can see how SciBERT substantially outperforms BioBERT on two datasets, one NER specific and another one RE specific. The rest of the results are comparable but Beltagy underlines that the training has been done with much less data. It is important to note that all highest results are obtained through the fine-tuning approach. Surprisingly, BERTBase without domain adaptation also managed to achieve for a couple of tasks a performance between the previous state-of-the-art and the new one set by SciBERT.

Task	Dataset	BioBERT	SciBERT
NER	BC5CDR	88.85	90.01
	JNLPBA	77.59	77.28
	NCBI-disease	89.36	88.57
REL	ChemProt	76.68	83.64

Figure 3.6.2: : Comparing SciBERT with the reported BioBERT results on biomedical datasets

The authors of SciBERT complemented the domain adaptation with the insertion of custom vocabulary they named SciVocab that matches the scientific domain. The author measured the difference in performance with and without custom vocabulary and reported an average increase of 0.60 percent on F1 score.

Chapter 4

Engineering-related content: Tools and Environment

In this short section the engineering-related contents are described as well as the research methodology and methods that are used in this degree project.

4.1 Engineering-related and scientific content:

This section will describe applying engineering related and scientific skills; modelling, analysing, developing, and evaluating engineering-related and scientific content. The choice of methods will be described based on problem formulation and the different experiments.

As mentioned earlier, given a theoretical description of methodologies and methods, this part will detail how these are applied in the degree project.

4.2 Libraries

In this short section there is the description of the the different libraries, the machines used for this thesis.

4.2.1 PyTorch

PyTorch is a platform that supports deep learning and provides libraries to easily create ANNs. It is used to build a linear Multilayer Perceptron and a Long-Short-

Term-Memory network. One of the main benefits of PyTorch are implemented models, functions and learning algorithms. PyTorch is based on so-called tensors that represent multidimensional matrices. All calculations are done by tensor operations. Besides, it has implemented support for Graphics Processing Units (GPUs), which makes it easier to train and analyze neural network models on GPUs. Due to many tensor operations that will be done during training it makes sense to use them, as they are designed to do matrix calculations.PyTorch is a deep learning framework that provides maximum flexibility and speed during implementing and building deep neural network architectures. A model can be defined in PyTorch in two steps by sub-classing the torch.nn.Module class specifying the parameters of the model initially and describing then how they are applied to the inputs. The biggest advantages of PyTorch are the dynamic computation graphing and its imperative programming style that performs computations as it goes through each line of the written code.

4.2.2 Additional Python Libraries

Besides PyTorch, other libraries are used to assist the programming part of this thesis. They are listed and briefly described in this sub-section.

SciPy

The SciPy ecosystem is needed when scientific computing is done in Python. It refers to many different packages such as NumPy, pandas and matplotlib. The SciPy library itself offers many different routines for numerical calculation. matplotlib will be applied in this thesis to create diagrams.

NumPy

NumPy's most used functionality is the creation of N-dimensional arrays. Plain Python does not support this fundamental data structure. It also provides implementations for linear algebra and random number generation. In addition, it is used by many other Python packages such as PyTorch to realize the tensor data structure.

pandas

pandas is an open source library that supports data analysis for Python by providing custom data structures such as DataFrames and Series. The structures support the

handling when working with a large amount of data.

scikit-learn

scikit-learn is a collection of data mining and data analysis tools. For example, Random Forest, SVM and various regression models can be found in the catalogue. scikit-learn provides in the programming part different implementations for evaluation metrics, Random Forest and tf-idf.

4.2.3 Github repositories: Transformers library

Transformers [2] provides state-of-the-art general-purpose architectures (BERT, RoBERTa, Camembert, ...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between PyTorch.

4.3 Machines

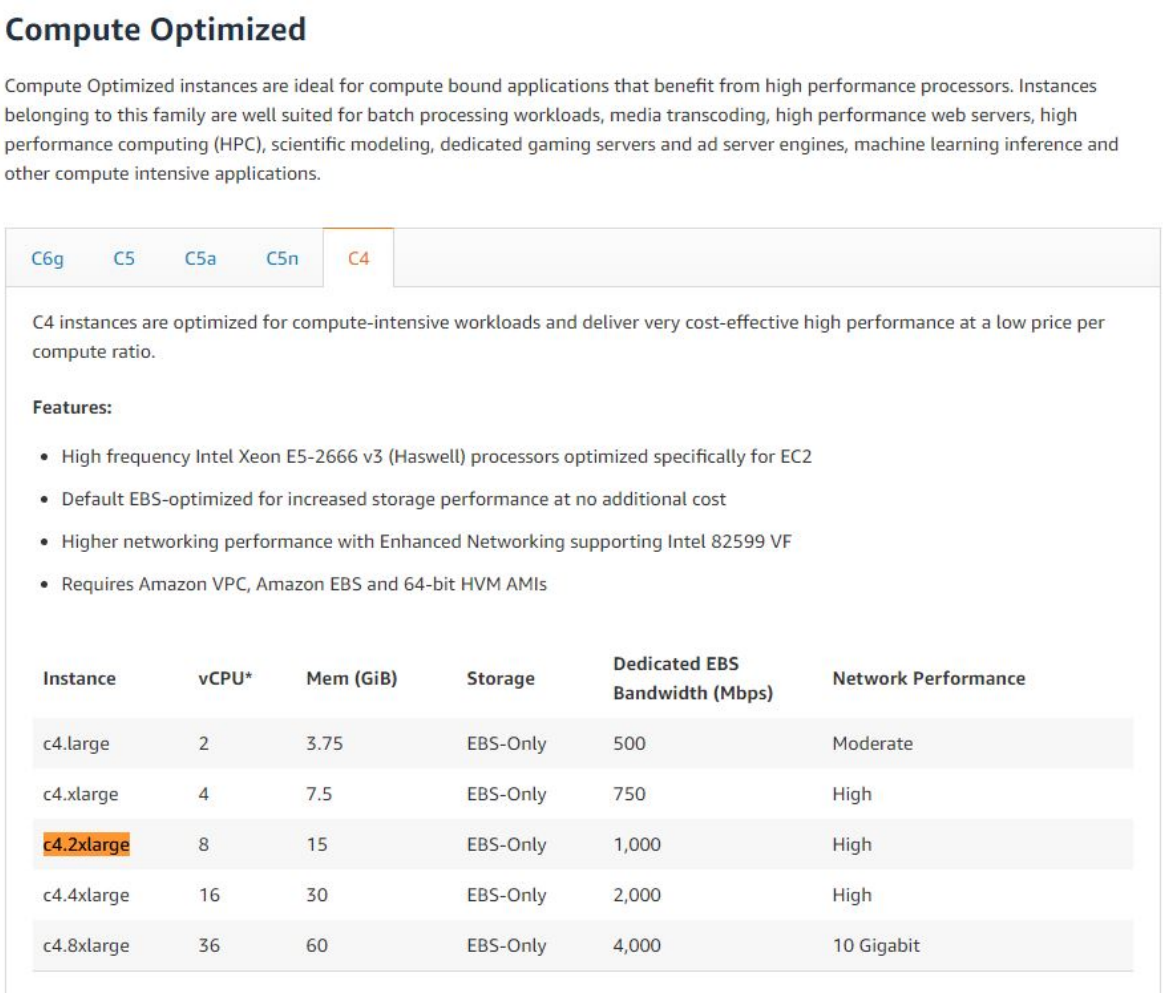
4.3.1 Software

Deep learning in both research and production has been mainly conducted using python as a programming language. This is due to its learning curve and eco-system of data science oriented tools. Between the two major open source machine learning framework, PyTorch is chosen (implemented by Facebook) over Tensorflow (By Google) because of its simpler approach and reduced learning overhead. It is worth saying that at the moment of this thesis, the environment to use GPU with the proposed version of Pytorch was not well documented. Going through all the possibilities had to be done to have a stable environment where code could run normally.

4.3.2 Hardware

All the deep learning computing that train faster on GPU was Performed on AWS machine with a GPU as described in 4.3.2. More experiments were conducted on an instance with other seetings on AWS 4.3.1. The remote machines ran Ubuntu 16.04.6 as operating system.

AWS machines EC2: The first testing were conducted on small datasets on an EC2 instance without GPU. This was the experimental phase of the BERT, Camembert implementations, the strategy used for the adaptation the models on this specific data as well as the experiments conducted to choose the right hyperparameters while optimising the cost of training in terms of time and computation [3]



Compute Optimized

Compute Optimized instances are ideal for compute bound applications that benefit from high performance processors. Instances belonging to this family are well suited for batch processing workloads, media transcoding, high performance web servers, high performance computing (HPC), scientific modeling, dedicated gaming servers and ad server engines, machine learning inference and other compute intensive applications.

C6g | C5 | C5a | C5n | C4

C4 instances are optimized for compute-intensive workloads and deliver very cost-effective high performance at a low price per compute ratio.

Features:

- High frequency Intel Xeon E5-2666 v3 (Haswell) processors optimized specifically for EC2
- Default EBS-optimized for increased storage performance at no additional cost
- Higher networking performance with Enhanced Networking supporting Intel 82599 VF
- Requires Amazon VPC, Amazon EBS and 64-bit HVM AMIs

Instance	vCPU*	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)	Network Performance
c4.large	2	3.75	EBS-Only	500	Moderate
c4.xlarge	4	7.5	EBS-Only	750	High
c4.2xlarge	8	15	EBS-Only	1,000	High
c4.4xlarge	16	30	EBS-Only	2,000	High
c4.8xlarge	36	60	EBS-Only	4,000	10 Gigabit

Figure 4.3.1: Details of the compute optimised machines in AWS, with highlight on the used one c4.2xlarge

model: g4dn.2xlarge Memory : 32 GB GPU : 1 Storage: 225 GB Number of virtual CPUs: 8 [**https://aws.amazon.com/ec2/instance-types/g4/**]

Product Details

	Instance Size	vCPUs	Memory (GB)	GPU	Storage (GB)	Network Bandwidth (Gbps)	EBS Bandwidth (GBps)	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
Single GPU VMs	g4dn.xlarge	4	16	1	125	Up to 25	Up to 3.5	$0.526	$0.316	$0.210
	g4dn.2xlarge	8	32	1	225	Up to 25	Up to 3.5	$0.752	$0.452	$0.300
	g4dn.4xlarge	16	64	1	225	Up to 25	4.75	$1.204	$0.722	$0.482
	g4dn.8xlarge	32	128	1	1x900	50	9.5	$2.176	$1.306	$0.870
	g4dn.16xlarge	64	256	1	1x900	50	9.5	$4.352	$2.612	$1.740
Multi GPU VMs	g4dn.12xlarge	48	192	4	1x900	50	9.5	$3.912	$2.348	$1.564
	g4dn.metal	96	384	8	2x900	100	19	$7.824	$4.694	$3.130

* Prices shown are for US East (Northern Virginia) AWS Region. Prices for 1-year and 3-year reserved instances are for "Partial Upfront" payment options or "No Upfront" for instances without the Partial Upfront option.

Figure 4.3.2: Details of the available machines in AWS with GPU, with highlight on the used one g4dn.2xlarge

AWS EFS: The model needs a big textual corpus to adapt well to this specific context. To store this data on Amazon EFS: Amazon Elastic File System.It provides a simple, scalable, fully managed elastic NFS file system for use with AWS Cloud services and on-premises resources. It is built to scale on demand to petabytes without disrupting applications, growing and shrinking automatically as one add and remove files, eliminating the need to provision and manage capacity to accommodate growth. source: [$https://aws.amazon.com/efs/?nc1 = h_ls$]

Chapter 5

Methodology

In this chapter the methods that were used in order to reach the aims of this thesis are presented. Firstly you can find an introduction of the mail purpose of the thesis that lead to this work. Secondly you can go through the description of the business understanding and the added value of the research. Then, the description of the used datasets and preprocessing is detailed as well as the two approaches taken of the task of language modeling: BERT and CamemBERT explaining the reason behind choosing these models among others. Finally, one can go through the implementation of the approaches, the optimization and the used metrics.

5.1 Introduction

The original aim of the research was to improve an existant system of NER tagging. Previous work of the team constructed a pipeline of Recognition of the Named Entities contained in a text. For a given text sent in Email or in the different social media (Twitter, Facebook..), Axa France would save much time by detecting the most relevant information in the textual data is has (Name, Surname, contract Number, email addresses) automatically. First, this detection enables a better indexation of the textual data: For a given client one can enrich automatically the database by the additional input he states in his conversations with the company. Second, these entities can be used as features that would represent the input utterance for a classification problem: like classifying the main type of message, or for sentiment analysis or even classifying the message's main topic and transferring this message to the specified entity that can treat it. Third, the named entities would have a huge help building an ontology of the

different clients of the company. Finally, a NER system with a good performance will be a good tool to use while anonyomizing the new input data. For a given text, the NER system will detect the most personal data that should be masked (Name, number of contract, phone number..) and hide them for a privacy issue if needed.

The system used was the multilingual BERT as a feature encoder of the input text. The research conducted was on the relevance of the system used to decode the different entities (by comparing Fully-connected and BiLSTM+CRF: Conditional Random Fields). Afterwards, the main topic of research, which leads to this work, is the reliability of the features extracted by the multilingual BERT. In other words, do we have the best representation of the text before applying any NLP task based on this representation.

5.2 Business Understanding and added value

The main work is focused on the best language modeling system that can provide the best representation of the specific French text available in the insurance field in Axa France. The obtention of a good encoder of textual data into numerical representation will minimize the loss of information, and thus minimize the bias related to the encoding phase. In any future NLP Task, one can use this enhanced specific language modeling system as the first brick of the NLP pipeline. We would focus the study of the performance of the quality of dataset, the different hyperparameters and the other bricks and we don't add anymore the bias related to the encoding phase which used to make the systme lose a lot of information. The added value would be this adapted language model which will ensure a better understanding of the textual data and thus a good starting point to any NLP task (NER, classification..). It is an efficient strategy for the upcoming NLP projects with a good work on the background of every NLP task.

5.3 Human-based Annotation of text

The best way to confirm which language model has the best representation of the text would be by applying this newly trained language model on data to provide the word embeddings. These embeddings are input to some NLP task (NER, classification..) and an observation statesa better performance when the new embeddings were used.

However this task has a huge cost in terms of resources: time, human resources. Added to this cost, the annotation of french Insurance data requires a certain familiarity with the insurance field, a good understanding of the french language. This is the first reason why the annotation cannot be outsourced in an annotation tool like AWS SageMaker mechanical Turk: Amazon Mechanical Turk (MTurk) is a crowdsourcing marketplace that makes it easier for individuals and businesses to outsource their processes and jobs to a distributed workforce who can perform these tasks virtually. This could include anything from conducting simple data validation and research to more subjective tasks like survey participation, content moderation, and more. MTurk enables companies to harness the collective intelligence, skills, and insights from a global workforce to streamline business processes, augment data collection and analysis, and accelerate machine learning development.

5.4 Dataset

5.4.1 Data collection

French data available is scarse. Insurance data in French is even more scarce. Specific AXA's Insurance textual data in French is challenging. There are a number of internal resources, mainly emails between the company's contributors, between the company contributors and the clients, listing the different conversations that can be expressed in a formal or unformal way. The other main resource of 'natural data' is the comments, posts and messages on the social networks (Facebook, Twitter, Messenger). Added to this, the available transcriptions of the the client's complaints, vocal messages and vocal conversations with the customer service are available. Mainly the internal data that will be called "Axa-related-data' can be divided into 2 types : structured part consisting of emails and formal conversations, and natural language data consisting of the transcriptions and social media. To overcome the scarcity of data, and to reduce the risk of overfitting, adding internet resources was one of the considerations. There are a number of thesis, archives and articles about insurance that are available in Archive ouverte HAL: an open source repository that holds research output and provides free and immediate access to research results. https://hal.archives-ouvertes.fr/ It offers an API to convenienty fetch documents and store them. There are some open articles and "Deontology Code" in non digital pdf type that was responsible for the input of the short formal sentences.

5.4.2 Data Preprocessing

Before the thesis, the team of the company worked on extracting the Twitter messages using Twitter API. However, some analysis demonstrated that some posts in Twitter had the almost same content changing only some characters, they may be the marketing automatic posts, others were related to a raid against the company with the same message eveytime. A preprocessing of deduplicating the sentence was done using tfidf features to compare the sentences and eliminate those who appeared in a very similar formulation. Some articles and files were in a non digital pdf form, thus extracting the text using special libraries was one of the tasks to insure. There are a lot of libraries enabling this task but they are not consistent, and they present some anomalies in the final extracted text."pdfminer" is used with a postprocessing of the extracted text, like correcting the encodings, removing the images and urls and non relevant text.

5.4.3 Data Volume

The total amount of data collected and preprocessed is 60 million works which is relatively small compared to the data on which is pre-trained BERT from scratch for example.

AXA vs. non-AXA Insurance french text was included mostly, where the majority is AXA's textual resources. The non-AXA insurance text was added to allow the model to generalize more and avoid overfitting on some type of expressions.

Structured French vs. non Structured French: The French AXA insurance Language model should face different NLP use cases, from sentiment analysis to email classification, up to question answering to help AXA contributors. This makes the necessity of the diversity of the data used. The aim was to have a balanced data of spontanuous French and Structured French.

Catégorie	nb_tokens
Non Axa	9769745
AXA	50248445

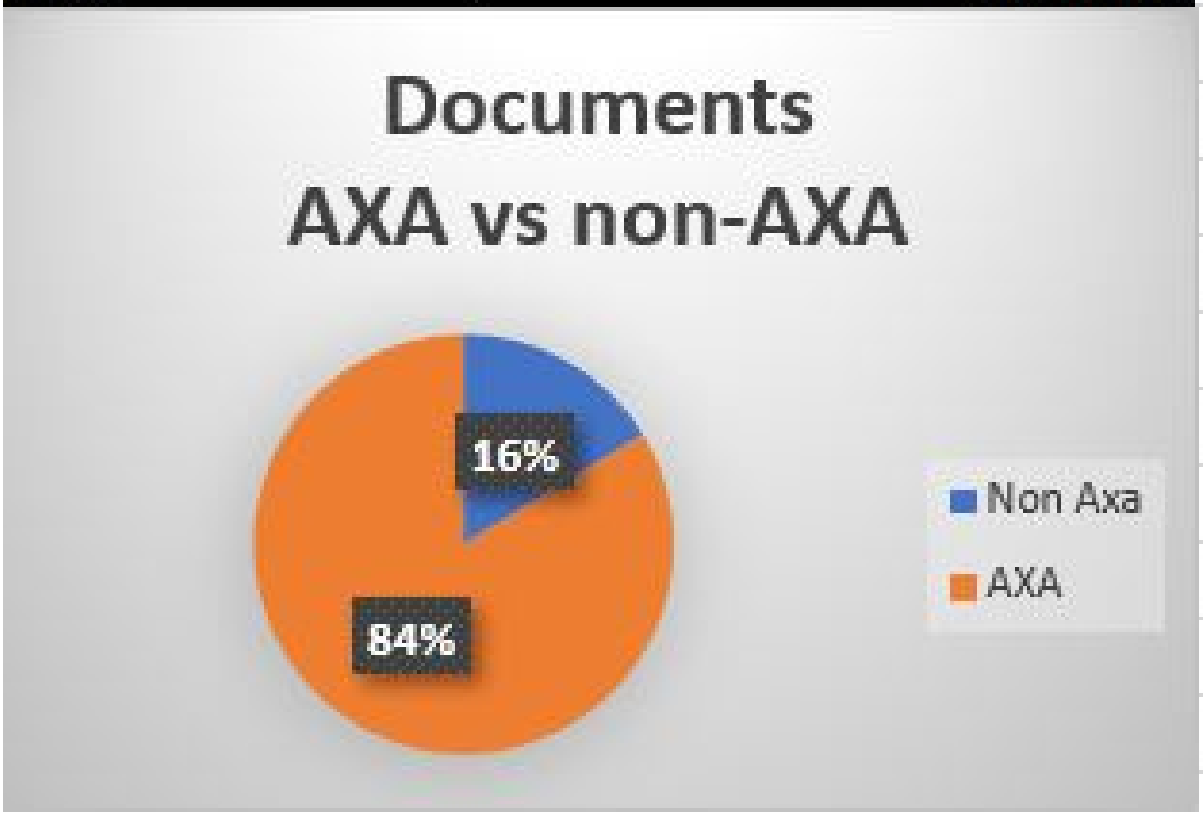


Figure 5.4.1: Distribution of data w.r.t AXA

5.5 Methodology and different choices

5.5.1 Language Model

Language model pre-training has drastically facilitated NLP tasks and made huge improvement in terms of performance of many natural language processing tasks. However, pre-training language models is a computationally expensive task and memory intensive. Having some restrictions in resources makes executing a pre-training from scratch difficult.

5.5.2 BERT

Why BERT? A lot of algorithms explained in the previous sections can be used for language modeling. WHY BERT? Other than being the state-of-the-art in NLP,

Category	Nb_sentences	Nb_tokens
Natural	383000	20862876
Structured	500504	23948577

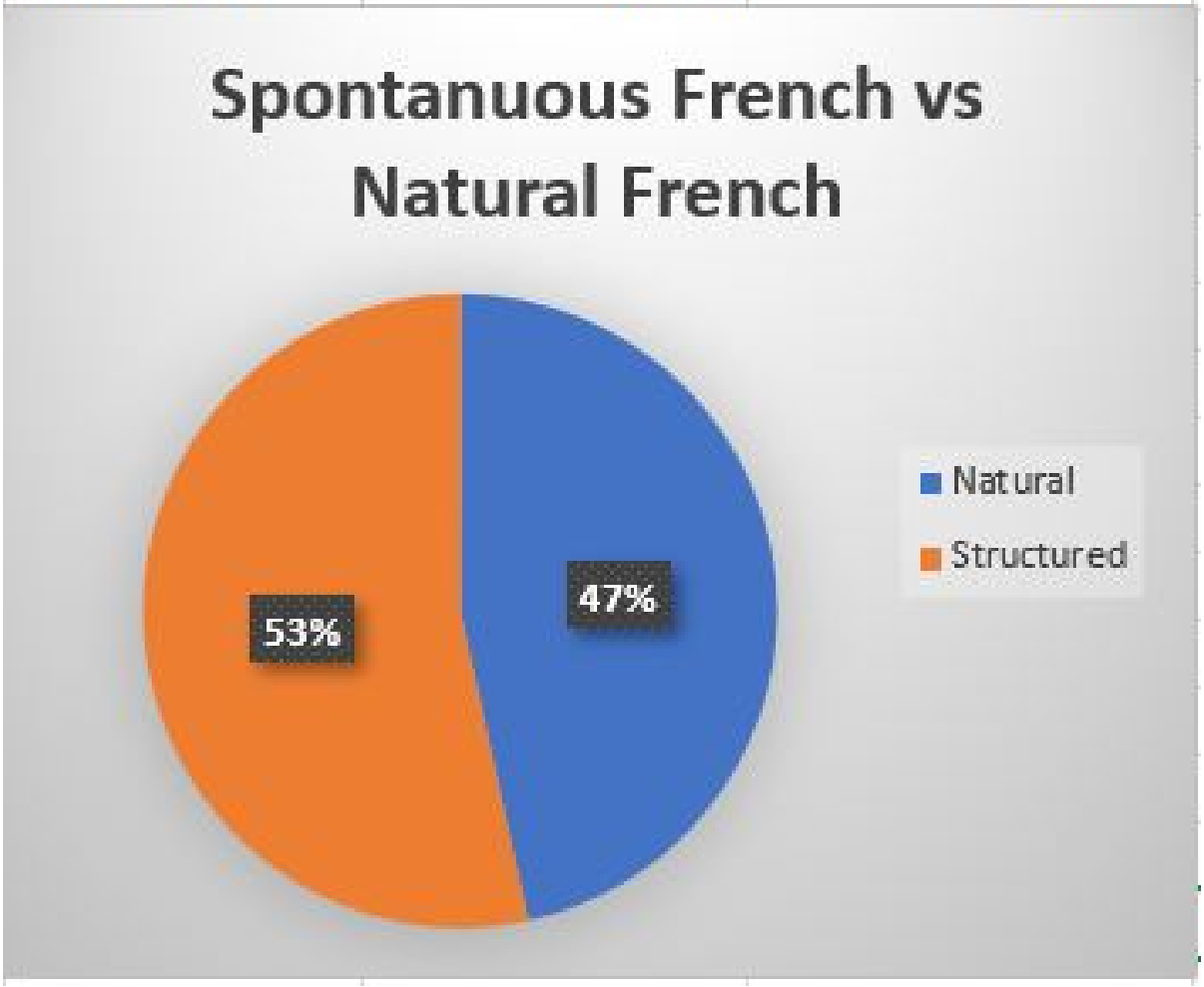


Figure 5.4.2: distribution of data w.r.t natural and formal expression

choosing BERT is based on different reflexions that will explained. First, BERT multiligual uses the Transformers architecture which doesn't suffer from the long sentences as other unidirectional or bidirectional systmes would. The used data can differ in term of length, which means that one can have long emails, and need a language model that can encode the whole context. The inconsistency of the length of input text, makes it necessary to construct a language model even for long sentences and long context. One may consider that such an improvement comes with a computational cost but Transformers need way less computation than the previous proposed solutions.

Second, BERT Transformers are based on the attention mechanism which allows the

better performance on long sequences: with the attention mechanism you can generate long text. Added to that self-attention aligns text with the text which appeared earlier, works well and fast. It is really more human being's intuition. To translate a whole paragraph it would be easier and more efficient to focus on the most important words and context. To understand a whole text, the humain intuition would be to consider the headlines and keep in mind the most important headlines, words and context whitout keeping in mind the whole text (words, expressions).

Third, multilingual BERT is pretrained on 104 languages.The French Language is among them. One would state that the French language challenge in this data can be confronted by the BERT as a baseline since a lot of parameters were trained to encode a lot of information and patterns in the structure, grammar, rules of the French Language. This prepares a good starting point for our specific language modeling. If one would project multilingual BERT to other previous language modeling systems in the human examples: the previous language modeling systems would be like a person who knows some French words but seperately and tries to read and understand a text with whole context. Knowing the Vocabulary isn't sufficient to have a good understanding of an article for example. Multilingual BERT would be the equivalent of a person who has read multiple texts, in different languages including French. The understanding of the article would be easier due to the familiarity with the Language's rules, patterns already seen.

Fourth, BERT has a huge architecture with a lot of parameters. The basic multilingual BERT has 110 million parameters. This would allow the conservation of the fundamental properties of the French language. The transfer learning would allow the model to encode fundamental information about the french language before adapting the model to AXA's data. Having a huge number of parameters allows a confidence that this model wuold be able to encode more complex patterns in the textual data. However, this advantage of having a big model, is a double edged point: we can model the language in a better way but since the model is too complex with a big number of parameters, in case of a scarcity of data, the model can overfit easily.

Another point that makes BERT the first choice to perform language modeling for this insurance French context is the Tokenisation. BERT uses its specific tokenizer. This step is fundamental in the preprocessing phase, since it is responsible for the first treatment of text. Let's take an example: "AXA France est une entreprise d'assurance"

which means ”AXA France is an insurance company”. the tokenization will be the first step in the preprocessing, and the first phase where we can loose information. The tokenisation will decide which parts of the sentence should be chunked or kept as whole token. There are two ways to tokenize the beginning of the sentence 1.”AXA France” is a token, and it would be the best because it is the name of the company and it should be considered as a whole token like ”New York” and not ”New”, ”York”. 2. ”AXA”, ”France” is the second tokenization of ”AXA France”, which looses the fact that it is the name of a wholesome entity. It is equivalent to consider ”New York” as the ”New” the adjective and ”York” the described token. In our example, it is actually considering that ”Axa” is a named entity and ”France” is the country. Embedding this tokenization can lead to a missunderstanding which is that the sentence tries to state that ”Axa” is based in ”France”, while we would like to consider ”AXA France” as a whole entity.

The tokenization is very important in the language modeling and word embedding. It can be done with simple rules (seperating by spaces, punctuation..) but it will be a suffering method in such exceptions. These exceptions are not predictable.Deterministic rule based tokenization will not be able to be generalisable, and it would need an exhaustive list of the exceptions that are not predictable. In order to have a generalisable tokenzation that can perform well on unseen data and unseen exceptions, it is necessary to use more generic method with a possibility of dealing with the unseen situations. This is exactly what BERT provides: It has its proper tokenizer Wordpiece. It chuncks the input sentence into tokens based on a statistical process(WordPiece Tokenization). It has the ability to cope with the rare words and unseen ones. For example the word ”playing” is not present in the vocabulary of multilingual BERT, the tokenizer will search the longest sequence in its vocabulary. It would be ”play”, the tokenizer will do the same operation on the remaining characters ”ing”. let’s say ”ing” and ”play” are in the vocabulary, thus the word ”playing” will be tokenized as [play, ##ing] instead of replacing ”playing” by [OOV] (Out Of the Box). This solution deals with the problem of the rare or out of vocabulary words. However the multilingual BERT Tokenizes ”AXA” which the most frequent word in the textual data of Axa France as [A, ##X, ##A]. This makes it necessary to adapt the BERT tokenization to this specific type of data. Since this process is statistical, it gives more degrees of freedom to adapt this tokenization to the specific text.

Finally, it is really helpful to profit from the community’s work: BERT is open source (including the parameters of pretraining). The implementation is accessible for both

deep learning frameworks Pytorch and TensorFlow.

5.5.3 CamemBERT

During the work done on this Master Thesis, a group of researchers of inria (National Institute for Research in Digital Science and Technology : "Institut national de recherche en informatique et en automatique") were working on adapting RoBerTa on French Language. They submitted on 10 Nov 2019 the paper Camembert. At the beginning of Decemeber 2019, work was still in progress to integrate the new model Camembert to Transformers' Library by the authors of the paper. This model adds a huge added value, since it is trained on only French corpus bigger than the French part of the corpus on which is pre-trained multilingual BERT. Added to the bigger volume, the specialisation of the model on one language, which is the one we're interested in for this Thesis, makes it definitely obvious to work on this Camembert Language as a starting point to train a French insurance language model cable of embedding accurately input text.

5.6 Language Model Fine-tuning

Following the principle of fine-tuning in transfer learning, we pre-train our own domain specific BERT model using the multiligual BERT language model as initialization in the same way as BioBERT 3.6.1 does. If we would pre-train from scratch like SciBERT, we could expect better results, but resource limitations made the expensive required hardware unattainable. Apart from that, it would have entailed a very long computation time, more than a week according to the authors of SciBERT. We pre-train on the other hand, our own domain specific language model using the French CamemBERT language model as initialization.

The Language model finetuning process further trains the model on the specific data making it faster as it should require less data. The training process is the Masked Language Modeling task 3.4.4, which is one of the two original self-supervised tasks for BERT pre-training. We use only this Masked Language Modeling task to have a fair comparison between BERT and CamemBERT. CamemBERT used only MLM as a self-supervised task for pre-training.

Then we fed a data sample containing AXA-specific Insurance documents with 60

million words. The LM fine-tuning lasted about 100 hours on a cloud machine equipped with 1 GPU.

When fine-tuning a pre-trained model on domain specific data, the resulting model has updated weights that represent better the characteristics and the vocabulary of the target domain. Before starting the process, it is required to specify a custom vocabulary file to incorporate into the base model. Therefore, we need to extract the vocabulary relevant for the Insurance domain. A sample of 300000 doucments has been selected from the dataset and the Byte Pair Encoding Algorirthm (BPE) as well as the BERT's wordPiece tokenization has been used to analyze the words and sub-words units. The tokenizer that we used for the BERT architecture is the one from multilingual BERT (Wordpiece). The tokenizer that we used for CamemBERT is the one from inria's CamemBERT(BPEà. This produced a collection of 19063297 words broken down in an array of tokens.

Once the collection of tokens obtained, the new tokens that did not belong yet to the original multilingual BERT vocabulary were added resulting in an increse from 119547 to 105000 +119547 tokens.

we retrain a vocabulary with the wordpiece method, keeping the same size of vocabulary: 119547. When we comapre these subwords to the original multilingual bert vocabulary, we figure that we should add 105000 subwords to the vocabulary in order to have a domain based vocabulary. There is a 119547- 105000/119547% overlap of the original vocabulary with the AXA insurance one.

When we use BPE to analyse the words and subwords units, we fix the vocabulary at 10000 which was the threshold starting from which we had ”Axa” tokenized as one entity. the overlap between these 10000 and the 119547 was (10000-8112) common subwords.

Here we can see that ”axa” is not a whole entity and chunked into 3 subwords : letters while it is the most common short word in the AXA-related datasets.

We retrained the vocabulary with the wordpiece method, keeping the same size of vocabulary we get this tokenization:

we retrained SentencePiece on the domain-specific dataset with 10000 as vocabulary size.

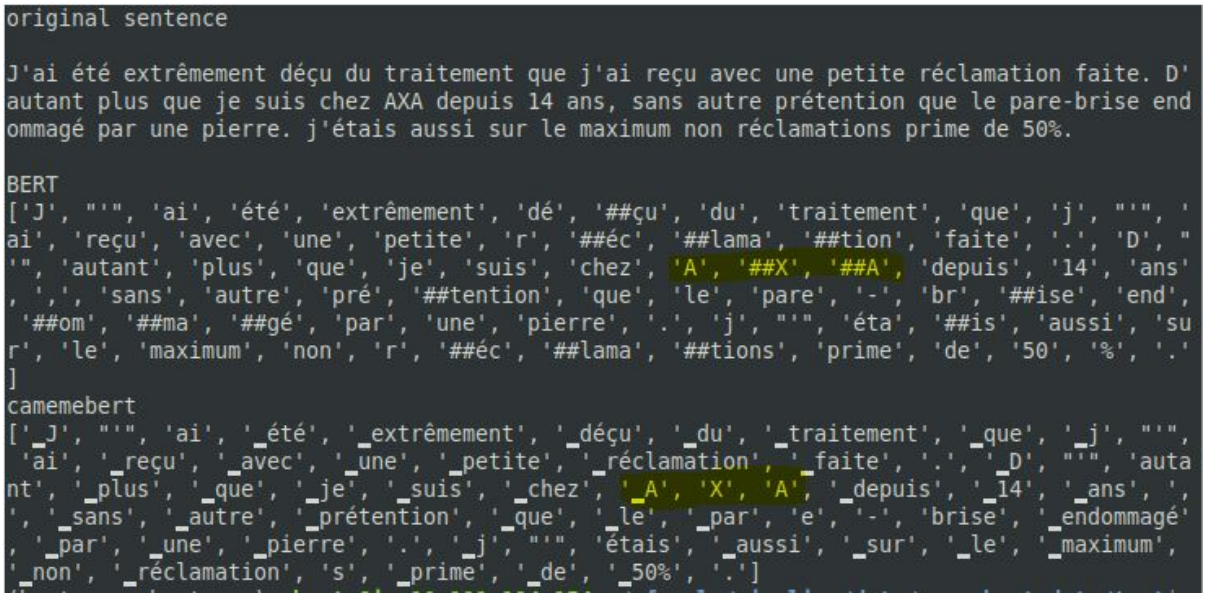


Figure 5.6.1: Tokenization by BERT and Camembert of a sentence of AXA's dataset

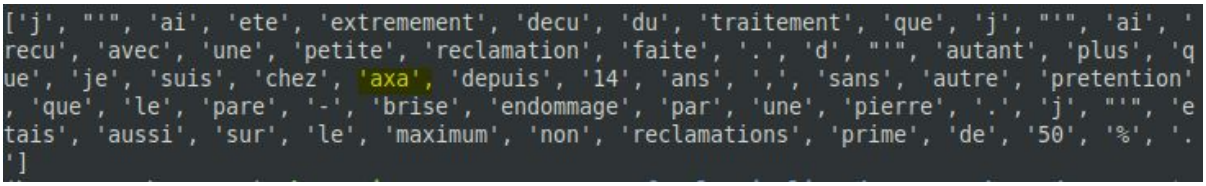


Figure 5.6.2: Tokenization with customised wordpiece

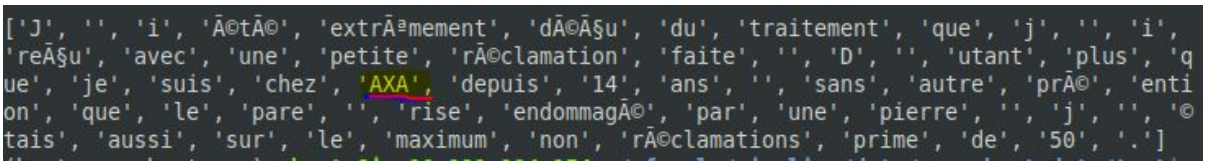


Figure 5.6.3: Camembert tokenization using SentencePiece retrained vocabulary

With the insertion of our custom insurance vocabulary, the tokenization provides different segmentation of words affecting the input sequence for BERT and CamemBERT and we expect to slightly improve the performance in the same way SciBERT did by creating their own SciVoc.

5.7 Implementation

The main research is to run the language modeling on AXA's data to have the better text representation. The two models were completely written within the Pytorch

framework [141]. Devlin et al. [45] released some code [2https://github.com/google-research/bert] for running BERT. This code written in Tensorflow framework [4] allows for tokenizing text and getting BERT model weights among the different proposed pre-trained models. The community provides a whole library "Transformers" [195] that is still implementing the different architectures BERT and Camembert in both Tensorflow and Pytorch. The code examples released at the beginning did not include functions designed for fitting Language Modeling on new big data starting from the pre-trained weights, so specific code had to be written for this thesis.

5.7.1 Hyper-Parameter Optimization

One of the most difficult issues in implementing and training Deep Learning models as big as these two models is choosing the hyperparameters to use in training and in the model architecture. These parameters can be optimized to attain the highest performance. These are the parameters that the model cannot learn by itself and need manual adjustment. Examples of hyperparameters inclued the learning rate, the number of training epochs, the train batch size, the warm-up periode, the dropout rate and all the parameters that could be adjusted in the optimizer like the Adam optimization algorithm. Architectural confirgurations such as the number of hidden layers and nodes could be considered as hyperparameters, but we won't alter these factors since we will be evaluating the BERT out of the Box (oob) against fine-tuned one. We will be evaluating Camembert (oob) against fine-tuned as well. Lack of time and resources we didn't conduct an exhaustive random search over all the hyperparameters and focused on the learning rate, the number of training epochs, the train batch size tuned over a small sample of the data and applied on the whole dataset.

5.8 Metrics

The main training process was Masked Language Model. Thus the metric used were the perplexity 3.0.4. The evaluation loss used was the cross-entropy 2.69.

Chapter 6

Results

In this chapter, the results obtained from our experiments will be presented along with a discussion concerning the results. Lastly we discuss the reflection and model improvements. In this section we call out-of-the-box (oob) every model using the pre-trained weights without any further training.

Word Embeddings of BERT Out of the box: without finetuning

We start with a baseline configuration where we apply the pre-trained multilingual bert model on the test set. We evaluate how does the pre-trained multilingual language model of BERT out of the box (OOB) represent our specific data.

Let's start with a description of this model configuration: There are 12 Hidden attention heads, 12 hidden layers with 3 fully connected layers. Accepting as input 512 tokens at max, using a vocabulary of size 119547. Actually the text used for test is the aubpart of the available corpus. It is tokenized first, then chuncked into parts where each part has less than 512 tokens in order to be accepted by the BERT model. Each token has an index in the BERT's vocabulary. So as input: the BERT model takes the corresponding index of the vocabulary. Added to this, it encodes information about the position of the token in the sentence and goes through the 12 hidden attention heads applying the pre-trained weights, then apply the 3 fully connected layers to have the last-hidden-states holing the outputs of BERT. It is a tuple with the shape (number of examples, max number of tokens in the sequence, number of hidden units in the BERT model). In our case, this will be 2000 (since we only limited ourselves to 2000 examples), 512 (which is the number of tokens in the longest sequence from the

examples), 768 (the number of hidden units in the BERT model).

To summarize the journey of the data:The first step is to use the BERT tokenizer to first split the word into tokens. Then, the special tokens needed for sentence classifications (these are [CLS] at the first position, and [SEP] at the end of the sentence) are added.The third step the tokenizer does is to replace each token with its id from the embedding table which is a component we get with the trained model. The input sentence is now the proper shape to be passed to BERT. Passing the input vector through BERT: The output would be a vector for each input token, each vector is made up of 768 numbers (floats).

The aim is to evaluate the word embeddings by evaluating the perplexity of the language model. If the perplexity is high, then the language model finds it hard to predict a masked token based on the context, which means that the word embeddings have a poor encoding of the textual context.

We first tested this out-of-the box BERT on insurance-related data expressed in a formal way, using well-formed expressions like in thesis and articles for example: First the features are created from the dataset file, by tokenizing the text, by chunking the text to blocks of 512 tokens. This insurance-related test set is subtext of the concatenation of all the insurance related data described above. It contains 4432 examples.

Num examples = 4432
***** Eval results *****
eval loss = 2.3914029434699873
perplexity = tensor(10.9288)

We secondly tested this out of the box BERT on AXA-related data expressed in a natural way:

Num examples = 3560
***** Eval results *****
eval loss = 2.41278353768788
perplexity = tensor(11.1650)

We then tested this out of the box BERT on AXA-related data expressed in a structured

way (emails...):

Num examples = 19491
***** Eval results *****
eval loss = 1.4423957277858068
perplexity = tensor(4.2308)

BERT trained

This subsection will presented the observed results of BERT trained on the whole train dataset and evaluated during training. The learning rate used was 5e-05 where the masked language model probability is 0.15. The model used is the bert base multilingual cased using 1 gpu and fixing the number of train epochs at 100. we fixed the train and the evaluation batch size per gpu at 4.

The hyerparameter block-size has an effect of the number of examples, we tried two block sizes [510, 128]. We tried different Instantaneous batch size per GPU [4, 32, 128]

the number of epochs starting from 2 and 6, 10, 50 to insure that the training can be done, to 100.

Finally we choose the best configuration based on the evaluation: block-size =128, num train epochs =50, batch-size = 4 and maximum number of steps = 3135000 saving steps 31350.

the training Set had 250800 examples which will be trained over 50 epochs with batch size fixed at 4 which leads to 3135000 optimisation steps.But we fixed the maximum number of steps at 470250 because these steps took already 9 days. Actually 31350 steps took 3h30 = 210 min

We conducted while training the evaluation of the trained model on the test set at each checkpoint (31350). The test set includes 110108 examples with the same blocks size of 128 and the same batch size. It gives the following results.

	***** Eval results *****
checkpoint-0	eval_loss = 2.477547967978635
	perplexity = tensor(11.9120)
checkpoint-31350	eval_loss = 2.4354170831515187
	perplexity = tensor(11.4206)
checkpoint-62700	eval_loss = 2.4293250945589673
	perplexity = tensor(11.3512)
checkpoint-94050	eval_loss = 2.3930535357060587
	perplexity = tensor(10.9469)
checkpoint-125400	eval_loss = 2.3913666899654853
	perplexity = tensor(10.9284)
checkpoint-156750	eval_loss = 2.4012745038597734
	perplexity = tensor(11.0372)
checkpoint-188100	eval_loss = 2.391603746980688
	perplexity = tensor(10.9310)
checkpoint-219450	eval_loss = 2.373356234228393
	perplexity = tensor(10.7334)
checkpoint-250800	eval_loss = 2.3729832200366694
	perplexity = tensor(10.7294)
checkpoint-282150	eval_loss = 2.379584653498969
	perplexity = tensor(10.8004)
checkpoint-313500	eval_loss = 2.362384893222222
	perplexity = tensor(10.6162)
checkpoint-344850	eval_loss = 2.3539823830686024
	perplexity = tensor(10.5274)
checkpoint-376200	eval_loss = 2.3697162062309833
	perplexity = tensor(10.6944)
checkpoint-407550	eval_loss = 2.3482893298396283
	perplexity = tensor(10.4676)
checkpoint-438900	eval_loss = 2.346428904313562
	perplexity = tensor(10.4482)
checkpoint-470250	eval_loss = 2.3213006844884467
	perplexity = tensor(10.1889)

Figure 6.0.1: The Evaluation during training of the bert model trained on the whole train dataset and evaluated on the test set

The training starts by creating the features from the dataset file, in our case it took 5 minutes to create the features which includes the step of tokenization for each configuration.

CamemBERT OOB

We apply CamemBERT with the same hyperparameter to have a fair comparison with bert (learning rate=5 e-5, masked language probability =0.15, batch size=4). We start by applying Camembert out of the box(without training) on the whole test set which included 23486 examples. However we had a surprising result, Camembert out of the box applied on our data has an eval loss =5.72790 and perplexity = tensor(307.3245). It models the Axa french insurance text worst than the multilingual bert out of the box. This was quite surprising since Camemebrt had already encoded the french language specifics. To understand this behavior, we evaluated the Camembert out of the box on the different subsets (natural language, more formal language, insurance related data, Axa-related data).

We first tested this out of the box Camembert on insurance-related data expressed in a formal way:

***** Eval results *****
eval loss = 6.742776085299937
perplexity = tensor(847.9112)
We secondly tested this out of the box Camembert on AXA-related data expressed in a natural way ***** Eval results *****
eval loss = 4.582938284159981
perplexity = tensor(97.8013)
We then tested this out of the box Camembert on AXA-related data expressed in a structured way (emails...): ***** Eval results *****
eval loss = 5.7173749403468825
perplexity = tensor(304.1055)
The more structured and formal is the data, the bigger is the evaluation loss and the bigger is the perplexity. We can explain that Camembert were trained on french data available online, which can be less formal text and expressed in a more natural way.

CamemBERT trained

We trained Camembert on the whole dataset using block size of 510. We proceeded the evaluation during training, to insure the convergence and to observe how is the Language modeling fine-tuning on this new specific data in terms of language model. Based on the different results found with camembert oob depending on the type of data: We did evaluation during training on each each type of data we have (AXA related formal data, AXA related natural text, insurance related formal text).

AXA related natural language: 2990 Examples

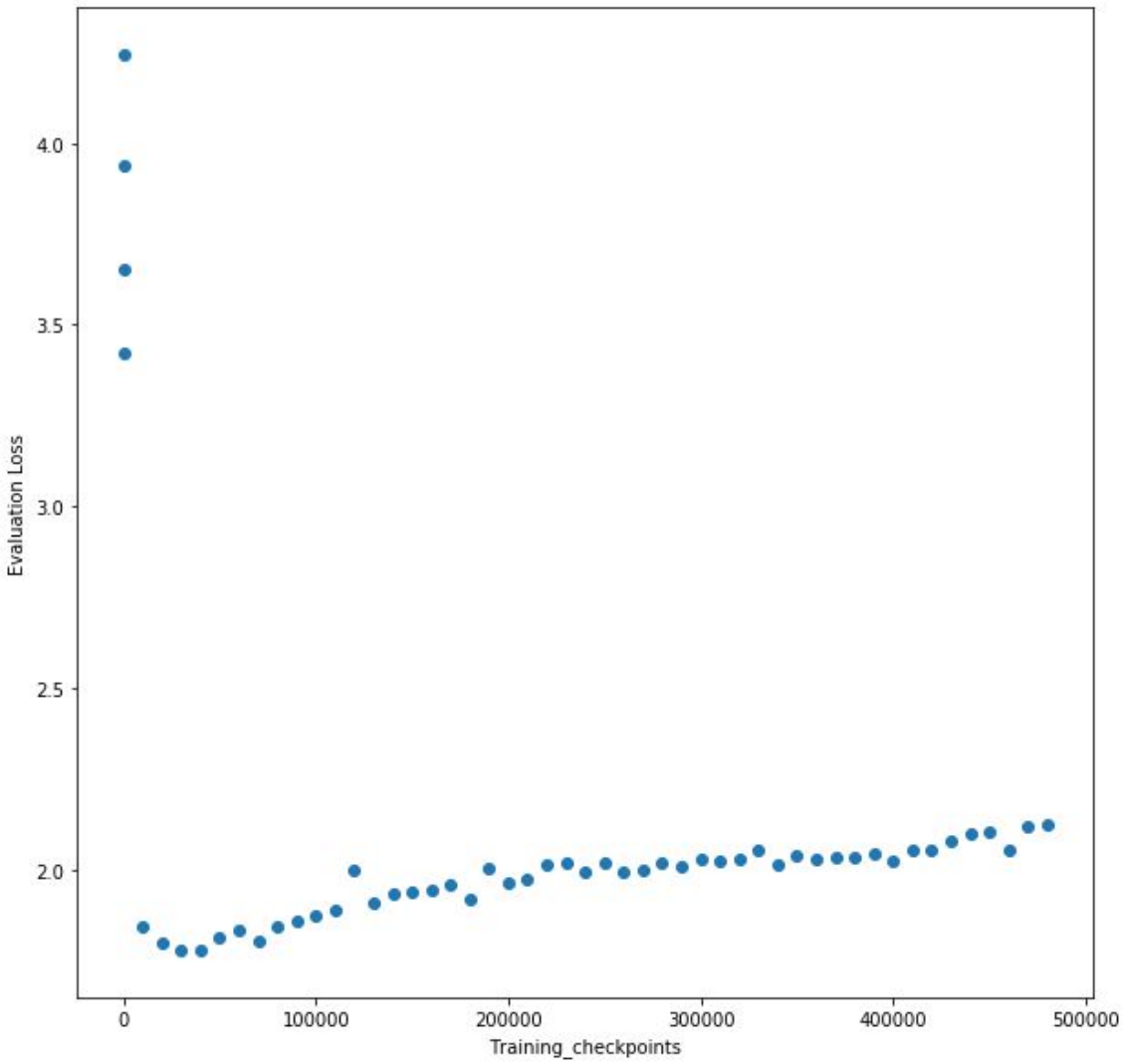


Figure 6.0.2: The Evaluation loss through while training epochs of Camembert on the whole dataset and tested on the Axa related natural language text

As we can see that there was a quick drop of the loss at an early stage and before the 100000th checkpoint the loss hit the minimal value. The evaluation on the test set is

applied while training. During further training, the loss starting from the 100000th checkpoint started going up. This would be overfitting, thus the best phase is the first, in the following figure we zoom on the first checkpoints.

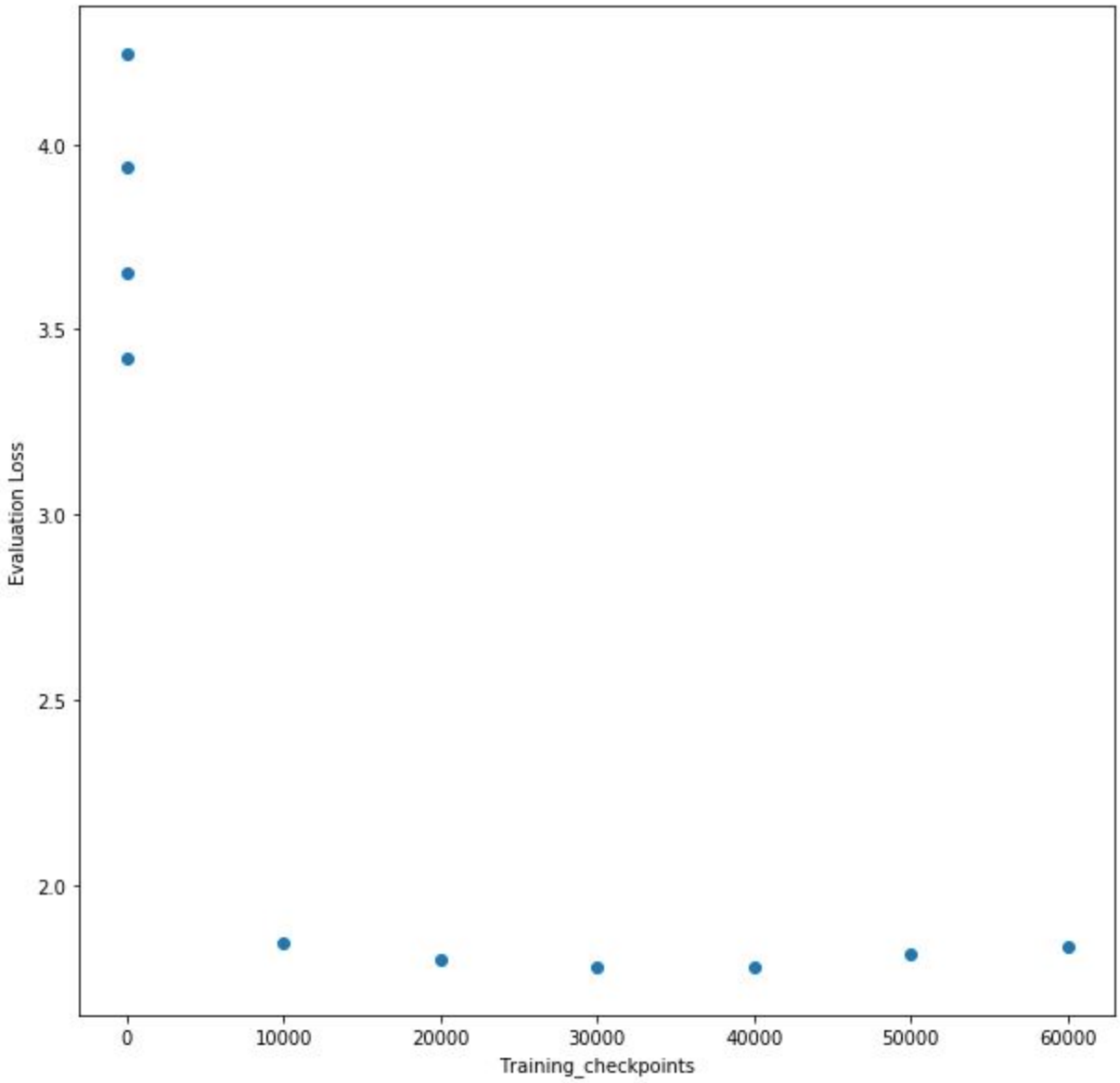


Figure 6.0.3: Zoom on the loss chart docusing on the minimal loss step

While training we only logged every 10000 step. We trained for 480000 steps to have a close behavior to the 470250 steps of Bert. However we can see in this chart, that there was a huge drop of loss at 1000 steps and it started going up again which corresponds to overfitting. We may conclude that Camembert, starting from a big complexity out of the box on this Axa-related natural expressed textual data (tensor(69.6110)) and dropped to tensor(6.32) within just 10000 steps. And the evaluation loss dropped from 4.242923 to 1.845198 within just 10000 steps while BERT had to go through

9 days training to drop from 2.477 to 2.32.

AXA related structured language: 16918 Examples We applied the same trained Camembert on the Axa related structured data (emails, laws related to AXA deontology...). We can find in this figure the loss through the training

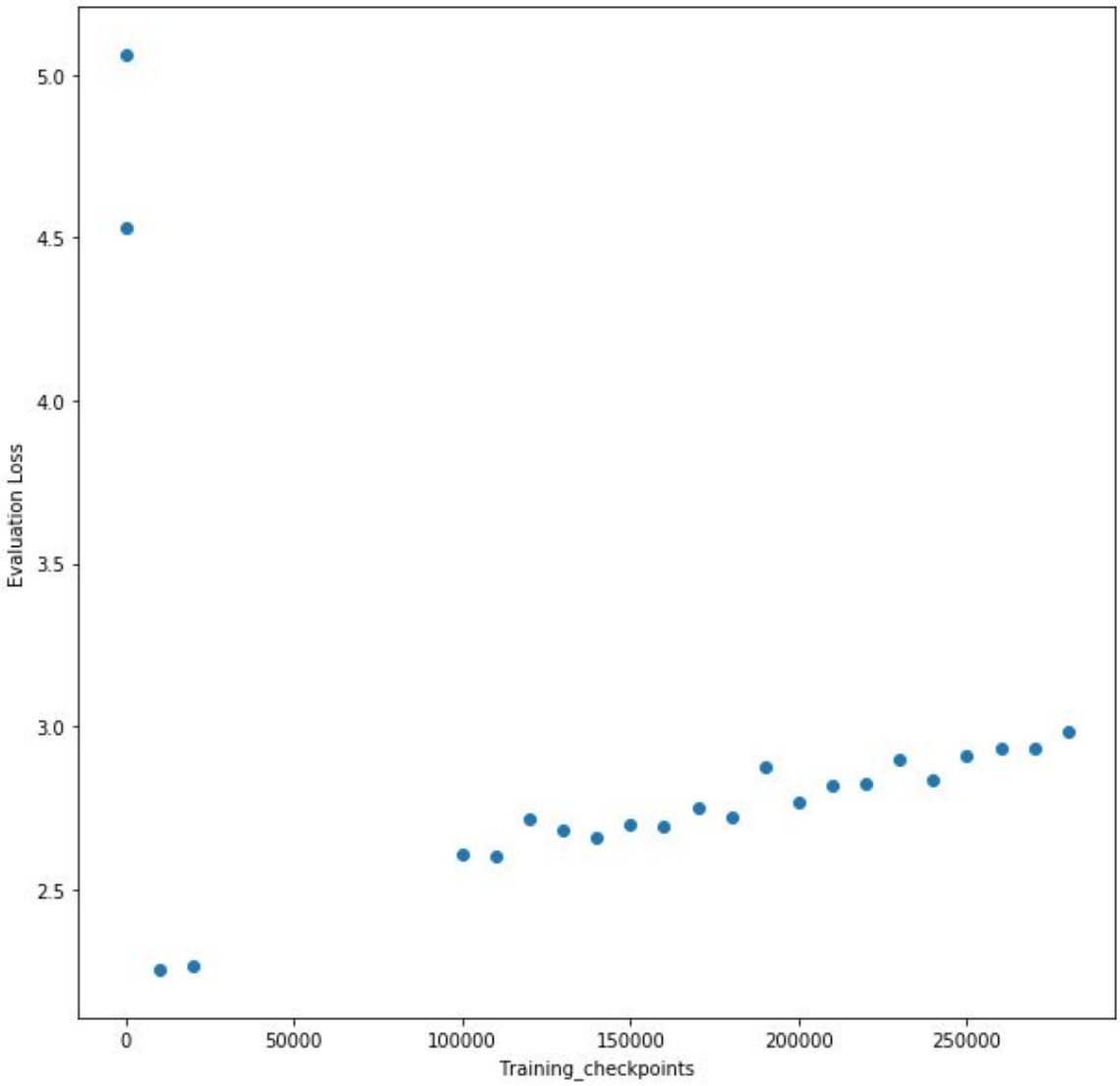


Figure 6.0.4: The Evaluation loss through while training epochs of Camembert on the whole dataset and tested on the Axa related structured language text

Same as the previous example, the loss dropped drastically at the beginning of the training. This makes early stopping necessary to have the lowest loss. We then zoom in in the next figure 6.0.5

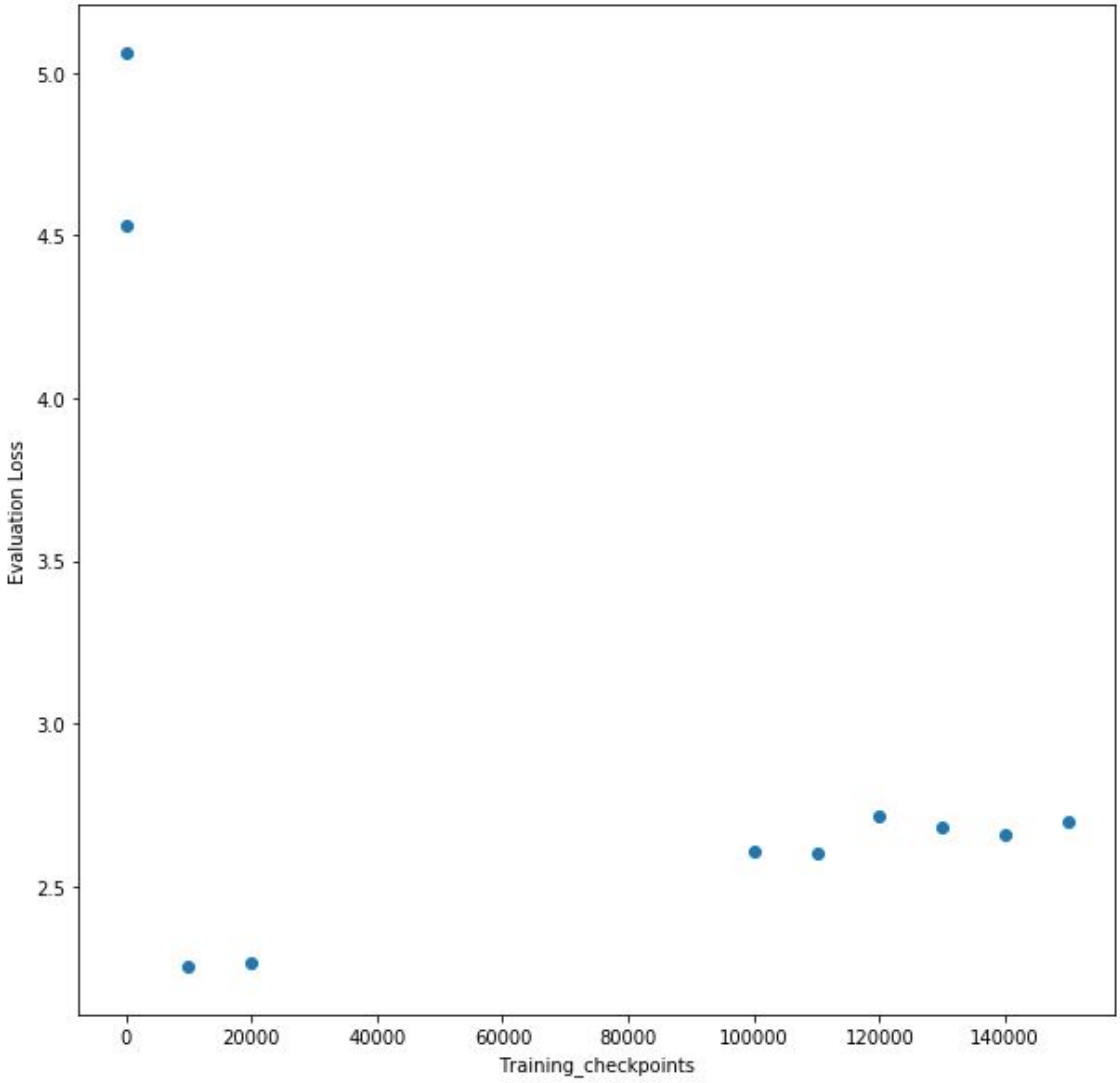


Figure 6.0.5: Zoom on the loss chart docusing on the minimal loss step

These results confirm that camembert converges quickly (within 10000steps). When we continue training it overfits and the evaluation loss grows again even on structured data. To confirm this result we evaluate the trained camembert language model on the insurance structured data which is basically articles and thesis.

Insurance related Data: 23486 Examples This figure 6.0.6, the training was on the whole insurance related data and tested on the whole insurance related data. This would be a merge of the merge of all the datasets tested before.

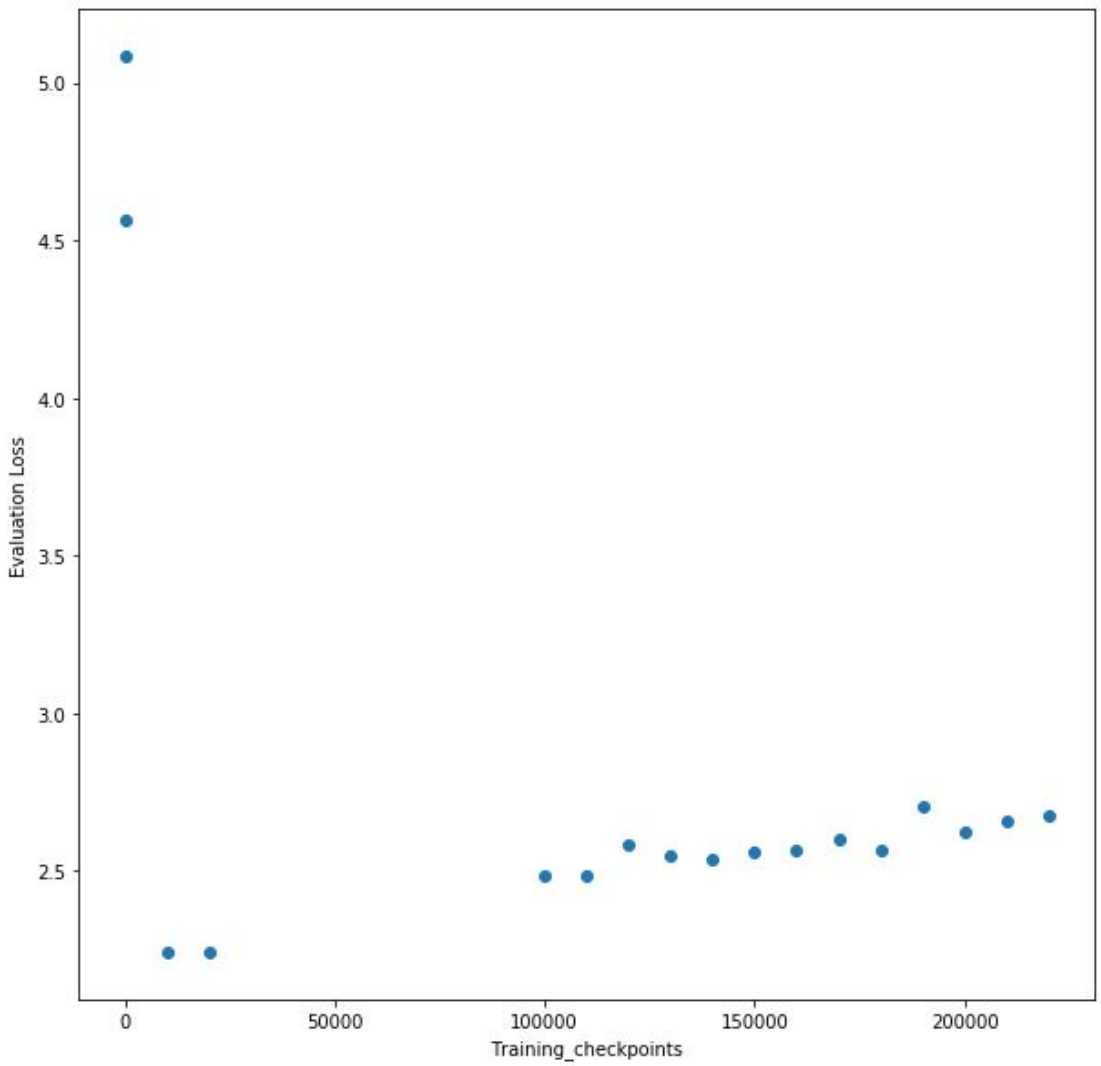


Figure 6.0.6: The Evaluation loss through while training epochs of Camembert on the whole dataset and tested on the whole test set

camemBERT trained with new tokens I manually replaced one of the unused tokens in the vocab file with the new common tokens. It works, but I realized that adding new tokens without the ability to do further pretraining isn't all that useful, especially given small dataset size. I decided not to do it.

6.1 Model Comparison

In this part, we will compare the performance of the two models, their adaptation to the Axa insurance specific data.

Model		Formal text	Natural Text
Bert Out of the Box (no training)	Cross entropy	2.3914	2.4128
	Perplexity	10.9288	11.1650
Bert trained on AXA	Cross entropy	2.3020	1.9806
	Perplexity	9.7328	7.2473
CamemBert OOB	Cross entropy	6.7428	4.58293823
	Perplexity	847.9112	97.8013
CamemBert trained	Cross entropy	2.252016	1.8451984
	Perplexity	9.5069	6.3294

Figure 6.1.1: Comparison of BERT and Camembert Langue Model Adaptation to the Axa insurance textual data

We can see in 6.1.1 that BERT out of the box performs better than Camembert Out of the Box. BERT has almost the same perlplexity for formal and natural language, While Camembert has an important perplexity, and the more formal is the data the bigger is the perplexity. We can interpret these results and understand this behavior by the fact that the multilingual BERT is more generalisable than Camembert. However we observe that after tuning the pre-trained models on the Axa insurance data, BERT language model's perplexity goes slightly down but slowly while Camembert adapts well and quickly to the new French Data needing less time and computation to obtain a better language model. The other difference between the two models, other than the language on which it is pre-trained, there is the tokenization. Being the first transformation applied on the text, the tokenization can have an important impact on the language model and thus on the word Embedding. The final difference between BERT and Camembert is the pre-training tasks, BERT being trained over Masked Language Model and Next Sentence Prediction while Camembert is only trained on Masked Language Modeling task. The paper [120] describes that a pre-training based on both tasks was conducted, and results proved that the Next Sentence Prediction had not a relevant impact. The lack of the NSP may be the responsible for the poorer performance of Camembert Out of the box.

6.2 Reflection and Model Improvements

In order to have a more detailed comparison and understand the impact of the differences between the models (BERT and Camembert), we should extend this study and compare specific tasks like classification, NER tagging or Information Retrieval tasks using the two different language models as a generator of the word embeddings. This would indirectly qulaify the better word embedding according to the one giving a better performance in terms of tasks. However, to conduct such a comparison there is a need of an important volume of annotated data. Annotation is a very costly task, we should ask people to annotate for a long period in order to have the groundtruth. To annotate the Named entities in an average email, detecting names, surnames, phone numbers, contract numbers, emails, organisation, localisation ... would take in average 1 minute. To annotate 100 000 emails (training and testing set), we would need 1666 hours. To insure a good quality of annotation, an annotator can not work more than 5 hours on this task to survive this robotic task. This would take 333 days of annotation. Some other tasks like classification's annotation require a certain knowledge in insurance field: The annotation cannot be outsourced and would have to be done by an insurance specialist which would allocate 1 hour at most of his busy day to annotate.

For all the mentioned reasons, testing the models on NLP tasks and comparing the results of the downstream task was not possible in this thesis, but would be really interesting approach to compare the word embeddings proficiency. Evaluating the word embedding through the results of the downstream task is not suffucient however, since it can be biased by the model applied after the word embedding step.

Not surprisingly, there are many ways of improving our work to get a bette language model which is actually adding special tokens to the vocabulary which wasn't a possible feature in the BERT models in the time of the Thesis but by the time of writing this part, it is an added feature in transformers' repository. This underligns how much evolution and research this model is going through by all the community.

Another comparison that merits to be conducted is using the same tokenization for both models, which were not possible because of the non open source wordpiece, but today (august 2020), HuggingFace provided a Tokenizer library that enables using any tokenizer and made models more flexible to use other tokenizers. The relevant comparison whould be to compare the langue models of a system wordpiece

tokenizer+ BERT; wordpiece tokenizer + Camembert; sentencePiece tokenizer + BERT and SentencePiece tokenizer + CamemBERT. This would be a very interesting future work that will investigate the real effect of tokenization which on the performance, which was hidden in the previous comparison.

Chapter 7

Discussion and Conclusions

This section will present a review of what is presented in this thesis. It will discuss the positive effects and the drawbacks of the research conducted. It will present the conclusions and valid future work.

7.1 Review

We presented an example of transfer learning of the recent BERT model to the insurance domain. Based on related literature, we proceeded with an approach that doesn't require the pretraining of the whole model from scratch, but instead initializes the weights of the model with an existing one, multiligual BERT or Camembert in our case. It is a convenient approach for several reasons:

First, when the volume of available text corpora is not large enough to match the specifications for pre-training an entire new model.

Second, the computational time, still considerably long and dependent on the amount of additional training data, is much shorter than the traditional pre-training that would expand over a week.

As suggested by Sebastian Ruder [[160]], the adaptation requires the source domain and the target domain to share a common base. In our case, it is the understanding of the French language. Following this principle, the adaptation was performed using the French BERT :Camembert and compare with the Multilingual one, as the evaluations during the pre-training of the French model obtained results that were superior to the Multilingual.

The creation of a custom vocabulary should help French Insurance BERT model better the language by weighting the additional embeddings more accurately. Nonetheless, the original vocabulary already includes 10% of the insurance vocabulary. This is one of the characteristics that renders the Insurance domain especially hard to grasp for both humans and machines, many of the day-to-day words are imbued with a different meaning when used in the Insurance context. SciBERT, the pre-trained BERT model for scientific texts [[18]], improves around 0.60% on F1 across all the datasets but their original-scientific vocabulary overlap is just 47%. In our case the effects of vocabulary use are rather diluted, specially when we don't have a battery of evaluations to firmly discern the improvement.

We can conclude that in case of constraints in terms of time or/and resources, Camembert would be relevant to train on the available french domain-specific data and provide the better representation of words (word embeddings).

In case of a small dataset, training this type of large models on a small dataset will not change much the behavior and won't modify deeply the weights. Added to this, we risk overfitting, which why in this case multilingual BERT Out of the Box would be the best choice to have quite good word representations, according to our thesis.

The main guidelines would that BERT would takes much more time to adapt to a domain than Camembert.

7.2 Discussion

The Camembert model have been trained using a corpus scrapped from the web. We believe that the results of fine-tuning the language model would have been much more significant if the model would have been pre-trained on more general information just like $BERT_{Base}$ did with Wikipedia and Book Corpus. This would reduce the bias of the unformal textual data available online. We believe as well that the AXA's French language model would have been much more performant if we had more data, enough data, enough time and resources to pre-train a BERT Architecture or a RoberTa architecture from scratch on this specific data as SciBERT.

The fine tuning process saves considerable amounts of time and yields acceptable results. In the comparative BioBERT(Language model fine-tuning) and SciBERT(language pre-

training), the latter obtains better results but also at the cost of requiring more trainign data and training for over a week on very expensive machines. If models keep growing, this would lead to a not democratic situation for research, where only more favorable position institutions will be able to perform experiments. Other downside that the research community showed concerns about is the increasing interest in deep learning and the practice of it on other social aspects such sustainability. The carbon footprint of these cmoputations is surprisingly high and some researcher like Schwartz [169] already suggest new metrics to tie together energy efficiency and performance of the model. We believe that if this measure would be applied to the comparison between BioBERT and SciBERT, BioBERT would emerge victorious. In our case, the moderate improvements that the AXABERT or AXACamembert provided would probably not justify the pre-training of the model when taking the efficiency considerations that Schwart proposes. However, We will probably have better results when adapting to a target domain that is further from the base domain. Transfer learning in NLP is still at its early stagd and is becoming a key driver in the industry and in research.

One of the biggest weaknesses of BERT is the limitation in the accepted sequence length. 512 tokens is sufficient for many tasks but is very far from enough in order to approach problems like understanding the context of the whole textual file, problmes from the Information Retrieval field. For this kind of tasks, the model isn't an out-of-the-box situation solution but rather a building component capable of tackling auxiliary tasks for later assembling more complex systems. Solving properly the length constraint with other methods is crucial in order to fully exploit the potential of the model BERT or Camembert.

The low resourcing of French Insurance data ans task-specific dataset makes the research in this area challenging.

7.3 Future Work

As discussed above, data scarcity is a major challenge. If there would be more available data and resources to provide expensive machines, we could pre-train from scratch French Insurance BERT model to compare with the current one. The insurance language utilizes some terms and expressions from current language to refer to notions proper to the insurance domain (”santé”, ”auto”,..) being able to remove the ambiguity of these nuances would eventually lead to an performance enhancement of the model.

Extending the contextual word representations with specific world knowledge about insurance and Axa specific language would definitely be an interesting follow-up research.

Studying ways to solve the issue of the maximum sequence length so the BERT model can produce document embeddings is a great challenge but also could produce interesting findings. The document embedding problem has been approached by Mikolov with an extension of Word2Vec called Doc2Vec [104], but there is still much room for improvement. The contextualized word embeddings from BERT would be an interesting approach to produce better document embeddings.

The author of this work believes that in some domains, the only way to attract the professionals to participate in an experiment is in case the experiment itself provides them direct value. Therefore, it is believed that the similarity task, would have a better participation rate from AXA's contributors if it would be a useful tool offered to them. This means, a full system with a search engine where they can search for documents for the current cases they are dealing with, would collect much more information about the performance of the model powering the system. This would be the ideal experiment, but comes also with slight ethical issues.

Once the evaluation has been done, the most common thing to do is to stop the system to avoid maintenance and infrastructure costs. If the tool results useful to the contributors and they integrate it to their daily workflow, it would be unethical to deprive them from this tool. Alternatively, research on the effectiveness of NLP and IR methods could be viable on open knowledge platforms. Many indicators of document relevancy could be anonymously measured through user behaviors and the tool would remain available to the community as long as the project exists.

Other type of projects in NLP, that can have a direct impact of the contributors' daily work, is a system capable of classifying the emails and messages or a Knowledge management tool that helps the contributors find quick answers to their questions or to explore already asked questions about Axa insurance knowledge. However, such system should never return wrong answers. Any error may lead to the contributor's error and thus costly functional error.

7.3.1 Ethical and societal considerations

Several papers had problems of reproductibility [14], [78], [135]. This thesis had problems reproducing the results presented by [45]. It is not a problem about reproducing the results of BERT paper, but a more general problem of lack of the specifics of the code and evaluation. In this case some results were shown specifically for NER and there are speculation that there are some missing details about the model. Generally in scientific papers enough information should be provided to be able to reproduce all the results properly. When either the full algorithm is not revealed (word piece not open source and some data set are not public anymore) or exact hyperparameters are not fully mentioned, it may get harder to have deeper considerations and conclusions about the paper.

Neural Networks are generally considered as ”black boxes”, where a human specialist cannot always answer to the different questions of why, and How an algorithm behaved in a certain way. Looking at neural networks from a control theory perspective there is no general theory about how to sufficiently test and understand a Neural Network in its whole domain. For example in Computer Vision, some studies show that what seems as pure noise for human sight can be considered as a specific object by a Neural Network. For some Neural Networks, some simulation may be helpful but this is computationally unfeasable for NLP Problems since having a large dictionary implies very big combinatorics of word inputs. Identifying weaknesses in the model's classification can thus be complicated.

There are other issue about using Transfer Learning models built by a specific company. BERT is an open collection of pre-trained weights and could in theory run on any hardware, but is built to work on Google's own cloud infrastructure. This create dependecies which makes it difficult to run, or even create BERT outside of thos hardware capapbilities. Historically, with computer Vision for instance, the only hardware acceleration required were GPUs which are relatively easy to configure and work with. BERT's reliance on TPUs or other very powerful computer chips greatly limits its independence.

7.4 Conclusion

This master thesis presented current state-of-the-art techniques in NLP and showed an approach to apply these NLP methods to be adapted on the french language, AXA France's insurance data language more specifically.

Current research in NLP-based text analysis focuses on advanced Artificial Neural Networks that uses meaningful word representations as input. This made the NLP community focus on this aspect of text representation(word embedding). This thesis covered FNNs, RNNs, attention based architectures that are used nowadays to analyse texts with respect to a certain task. The focus of this thesis is how to have the best textual representation (word embeddings). I presented the theory of the deep contextualized word representations like ELMo, ULMFIt to understand the main starting points of change in word embeddings and language modeling. It was very important to understand this emerging field of research of transforming words into meaningful and machine-readable formats using complex deep learning architectures. By providing the theoritical foundations and the literature review in Chapters 2 and 3, the thesis answered the first research question: Which satet-of-the-art NLP-approaches exist to represent accurately the French Axa Insurance textual data ? The focus was mainly set on the language modeling capacity.

The practical part of the this thesis verified that the Axa insurance French Data is different from the common french data. It verified as well that the adaptation of large pre-trained model even on a relatively small dataset has an impact on the language model. Chapters 4, 5 and 6 focus on the methodology of testing, the different results of this adaptation.

I showed that state-of-the-art approaches provide pre-trained models that can have a good textual representation of any type of language in case of no fine-tuning like BERT. The results revealed as well that the language BERTs adpat faster to new specific domains.

we conclude that the generalization of neural networks is a slippery slope. As far as generalization goes, the sequence-to-sequence framework makes it possible to transfer the models to any kind of dataset. However there are limits. Deep learning is shorthand for creating w very complex differentiable, non-linear function and approximate its optimum by minimizing the error with respect to the steepest gradient

change in high-dimensional space. In language each dimension will contain some of the information provided by the dataset with respect to the statistical nature of text: Polysemy, hyponymy, synonymy, named entities. Models like ELMo partially encode the hierarchical nature a word by contextualizing the sentence, paragraph or document it co-occurs with. As a result, all machine learning models that do not explicitly incorporate structural knowledge practically encode complex correlations on unstructured text. Simply expressed, it cannot tell causation from correlation. It does not handle prior knowledge. It is not transparent and explainable, since the solution space has potentioally infinitely many parameter settings. Furthermore, our sequece-to-sequence models with attention suffer from very large parameter requirements. Our biggest model had 280 million parameters, with a vocabulary size of 50.000 or a BPE segmentation of 40.000. It was questionable whether 50.000 tokens are enough to deal with highly specialized domains. It is a pain point to add special tokens and pre-train again on this vocabulary as well.

Further work to improve the results of my approach could be done by using the trained tokenizer to have the new vocabulary to use in the different BERT-type models. This would help chunk better the input sentences and thus have a better tokenisation. Further work to try is to try different tokenizations with the differents architectures and find the best combination. Finally, if annotated datasets available, studying the effect of every word embedding system on the downstream tasks' results would improve the reliability of the conclusions.

To answer the main research questions of this thesis: The latest state-of-the-art pre-trained BERT, GPT models provide a wide spectral of possibilities that can allow a good domain adaptation and thus a better representation of specific French Insurance Axa-related data with less resources. Actually, these pre-trained models prove to have a good encoding of any textual data out of the box already. We can have better word embeddings of French Insurance Axa-related text by fine-tuning these models on the available data. The main conclusions were that multilingual BERT has a good word embedding out of the box but requires a large corpus and long period to fine-tune and to adapt to this specific domain. The French BERT model Camembert had a relatively poor embedding of the insurance data out of the box but was very quick to fine-tune requiring less data. Actually BERT models deal with the out of vocabulary words in the tokenization by chunking them to the largest possible subwords. This makes the training of the tokenization and adapting the model to the tokenization a very

interesting future work. Regarding the requirements of domain adaptation of BERT model, a corpus of 60 million words enabled a better language model, but the more data we have the better is the fine-tuning of this large model.

Bibliography

[1] URL: https://w3techs.com/technologies/overview/content_language (visited on 09/25/2020).

[2] URL: https://github.com/huggingface/transformers (visited on 09/25/2020).

[3] URL: https://aws.amazon.com/ec2/instance-types/ (visited on 09/25/2020).

[4] Abadi, Martín et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[5] Abu-Mostafa, Yaser S. **?**Learning from hints in neural networks**?** In: *Journal of complexity* 6.2 (1990), pp. 192–198.

[6] Allahyari, Mehdi et al. **?**A brief survey of text mining: Classification, clustering and extraction techniques**?** In: *arXiv preprint arXiv:1707.02919* (2017).

[7] Allenby, Greg M and Rossi, Peter E. **?**Marketing models of consumer heterogeneity**?** In: *Journal of econometrics* 89.1-2 (1998), pp. 57–78.

[8] Ammar, Waleed et al. **?**Many languages, one parser**?** In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 431–444.

[9] Arik, Sercan O et al. **?**Deep voice: Real-time neural text-to-speech**?** In: *arXiv preprint arXiv:1702.07825* (2017).

[10] Arora, Sanjeev, Liang, Yingyu, and Ma, Tengyu. **?**A simple but tough-to-beat baseline for sentence embeddings**?** In: (2016).

[11] Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. **?**Layer normalization**?** In: *arXiv preprint arXiv:1607.06450* (2016).

[12] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. **?**Neural machine translation by jointly learning to align and translate**?** In: *arXiv preprint arXiv:1409.0473* (2014).

[13] Baker, Collin F, Fillmore, Charles J, and Lowe, John B. **?**The berkeley framenet project**?** In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*. 1998, pp. 86–90.

[14] Baker, Monya. **?**1,500 scientists lift the lid on reproducibility**?** In: (2016).

[15] Baroni, Marco, Dinu, Georgiana, and Kruszewski, Germán. **?**Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors**?** In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2014, pp. 238–247.

[16] Baxter, Jonathan. **?**A Bayesian/information theoretic model of learning to learn via multiple task sampling**?** In: *Machine learning* 28.1 (1997), pp. 7–39.

[17] Baxter, Jonathan. **?**A model of inductive bias learning**?** In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198.

[18] Beltagy, Iz, Lo, Kyle, and Cohan, Arman. **?**SciBERT: A pretrained language model for scientific text**?** In: *arXiv preprint arXiv:1903.10676* (2019).

[19] Bengio, Yoshua et al. **?**A neural probabilistic language model**?** In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[20] Bianchi, Filippo Maria et al. **?**An overview and comparative analysis of recurrent neural networks for short term load forecasting**?** In: *arXiv preprint arXiv:1705.04378* (2017).

[21] Blei, David M, Ng, Andrew Y, and Jordan, Michael I. **?**Latent dirichlet allocation**?** In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[22] Bojanowski, Piotr et al. **?**Enriching word vectors with subword information**?** In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[23] Braud, Chloé, Lacroix, Ophélie, and Søgaard, Anders. **?**Cross-lingual and cross-domain discourse segmentation of entire documents**?** In: *arXiv preprint arXiv:1704.04100* (2017).

[24] Briscoe, Ted. **?**Introduction to Linguistics for Natural Language Processing**?** In: (2013).

[25] Brown, Lee K et al. **?**Teaching by principles**?** In: (1994).

[26] Brown, Peter F et al. **?**Class-based n-gram models of natural language**?** In: *Computational linguistics* 18.4 (1992), pp. 467–480.

[27] Cambria, Erik and White, Bebo. **?**Jumping NLP curves: A review of natural language processing research**?** In: *IEEE Computational intelligence magazine* 9.2 (2014), pp. 48–57.

[28] Caruana, R. **?**Multitask Learning. Autonomous Agents and Multi-Agent Systems**?** In: (1998).

[29] Cer, Daniel et al. **?**Universal sentence encoder**?** In: *arXiv preprint arXiv:1803.11175* (2018).

[30] Chan, Yee Seng and Ng, Hwee Tou. **?**Estimating class priors in domain adaptation for word sense disambiguation**?** In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. 2006, pp. 89–96.

[31] Chawla, Nitesh V et al. **?**SMOTE: synthetic minority over-sampling technique**?** In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[32] Chelba, Ciprian and Acero, Alex. **?**Adaptation of maximum entropy capitalizer: Little data can help a lot**?** In: *Computer Speech & Language* 20.4 (2006), pp. 382–399.

[33] Chen, Minmin. **?**Efficient vector representation for documents through corruption**?** In: *arXiv preprint arXiv:1707.02377* (2017).

[34] Choi, Eunsol et al. **?**Coarse-to-fine question answering for long documents**?** In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 209–220.

[35] Church, Kenneth and Hanks, Patrick. **?**Word association norms, mutual information, and lexicography**?** In: *Computational linguistics* 16.1 (1990), pp. 22–29.

[36] Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. **?**Fast and accurate deep network learning by exponential linear units (elus)**?** In: *arXiv preprint arXiv:1511.07289* (2015).

[37] Collobert, Ronan and Weston, Jason. **?**A unified architecture for natural language processing: Deep neural networks with multitask learning**?** In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[38] Collobert, Ronan et al. **?**Natural language processing (almost) from scratch**?** In: *Journal of machine learning research* 12.ARTICLE (2011), pp. 2493–2537.

[39] Conneau, Alexis et al. **?**Supervised learning of universal sentence representations from natural language inference data**?** In: *arXiv preprint arXiv:1705.02364* (2017).

[40] Dai, Andrew M and Le, Quoc V. **?**Semi-supervised sequence learning**?** In: *Advances in neural information processing systems*. 2015, pp. 3079–3087.

[41] Dai, Wenyuan et al. **?**Self-Taught Clustering**?** In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 200–207. ISBN: 9781605582054. DOI: 10.1145/1390156.1390182. URL: https://doi.org/10.1145/1390156.1390182.

[42] Deerwester, Scott et al. **?**Indexing by latent semantic analysis**?** In: *Journal of the American society for information science* 41.6 (1990), pp. 391–407.

[43] Deng, Li, Hinton, Geoffrey, and Kingsbury, Brian. **?**New types of deep neural network learning for speech recognition and related applications: An overview**?** In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8599–8603.

[44] Denny, Matthew and Spirling, Arthur. **?**Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it**?** In: *When It Misleads, and What to Do about It (September 27, 2017)* (2017).

[45] Devlin, Jacob et al. **?**Bert: Pre-training of deep bidirectional transformers for language understanding**?** In: *arXiv preprint arXiv:1810.04805* (2018).

[46] Dhillon, Paramveer, Foster, Dean P, and Ungar, Lyle H. **?**Multi-view learning of word embeddings via cca**?** In: *Advances in neural information processing systems*. 2011, pp. 199–207.

[47] Dong, Daxiang et al. **?**Multi-task learning for multiple language translation**?** In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1723–1732.

[48] Dowty, David. **?**Thematic proto-roles and argument selection**?** In: *language* 67.3 (1991), pp. 547–619.

[49] Duong, Long et al. **?**Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser**?** In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 2015, pp. 845–850.

[50] Fan, Xing et al. **?**Transfer learning for neural semantic parsing**?** In: *arXiv preprint arXiv:1706.04326* (2017).

[51] Fang, Meng and Cohn, Trevor. **?**Model transfer for tagging low-resource languages using a bilingual dictionary**?** In: *arXiv preprint arXiv:1705.00424* (2017).

[52] Feldman, Ronen, Sanger, James, et al. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

[53] Fellbaum, WordNet. *An Electronic Lexical Database (Language, Speech, and Communication)*. 1998.

[54] Finkel, Jenny Rose and Manning, Christopher D. **?**Hierarchical bayesian domain adaptation**?** In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. 2009, pp. 602–610.

[55] Gehring, Jonas et al. **?**Convolutional sequence to sequence learning**?** In: *arXiv preprint arXiv:1705.03122* (2017).

[56] Gillick, Dan et al. **?**Multilingual language processing from bytes**?** In: *arXiv preprint arXiv:1512.00103* (2015).

[57] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep learning*. MIT press, 2016.

[58] Grave, Edouard et al. **?**Learning word vectors for 157 languages**?** In: *arXiv preprint arXiv:1802.06893* (2018).

[59] Graves, Alex, Jaitly, Navdeep, and Mohamed, Abdel-rahman. **?**Hybrid speech recognition with deep bidirectional LSTM**?** In: *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE. 2013, pp. 273–278.

[60] Graves, Alex and Schmidhuber, Jürgen. **?**Framewise phoneme classification with bidirectional LSTM and other neural network architectures**?** In: *Neural networks* 18.5-6 (2005), pp. 602–610.

[61] Greene, Zac et al. **?**The nuts and bolts of automated text analysis. Comparing different document pre-processing techniques in four countries**?** In: (2016).

[62] Guo, Jiang et al. **?**Exploiting multi-typed treebanks for parsing with deep multi-task learning**?** In: *arXiv preprint arXiv:1606.01161* (2016).

[63] Gutmann, Michael U and Hyvärinen, Aapo. **?**Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics**?** In: *The journal of machine learning research* 13.1 (2012), pp. 307–361.

[64] Hashimoto, Kazuma et al. **?**A joint many-task model: Growing a neural network for multiple nlp tasks**?** In: *arXiv preprint arXiv:1611.01587* (2016).

[65] He, Kaiming et al. **?**Deep residual learning for image recognition**?** In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[66] Hermann, Karl Moritz et al. **?**Grounded language learning in a simulated 3d world**?** In: *arXiv preprint arXiv:1706.06551* (2017).

[67] Hershcovich, Daniel, Abend, Omri, and Rappoport, Ari. **?**Multitask parsing across semantic representations**?** In: *arXiv preprint arXiv:1805.00287* (2018).

[68] Hill, Felix, Cho, Kyunghyun, and Korhonen, Anna. **?**Learning distributed representations of sentences from unlabelled data**?** In: *arXiv preprint arXiv:1602.03483* (2016).

[69] Hochreiter, Sepp. **?**The vanishing gradient problem during learning recurrent neural nets and problem solutions**?** In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[70] Hochreiter, Sepp and Schmidhuber, Jürgen. **?**Long short-term memory**?** In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[71] Hopfield, John J. **?**Neural networks and physical systems with emergent collective computational abilities**?** In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[72] Howard, Jeremy and Ruder, Sebastian. **?**Universal language model fine-tuning for text classification**?** In: *arXiv preprint arXiv:1801.06146* (2018).

[73] Huang, Fei and Yates, Alexander. **?**Distributional representations for handling sparsity in supervised sequence-labeling**?** In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 495–503.

[74] Huang, Fei and Yates, Alexander. **?**Exploring representation-learning approaches to domain adaptation**?** In: *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*. 2010, pp. 23–30.

[75] Huang, Fei and Yates, Alexander. **?**Open-domain semantic role labeling by modeling word spans**?** In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. 2010, pp. 968–978.

[76] Hurwitz, Judith and Kirsch, Daniel. **?**Machine learning for dummies**?** In: *IBM Limited Edition* 75 (2018).

[77] Indurkhya, Nitin and Damerau, Fred J. *Handbook of natural language processing*. Vol. 2. CRC Press, 2010.

[78] Ioannidis, John PA. **?**Why most published research findings are false**?** In: *PLoS medicine* 2.8 (2005), e124.

[79] Iyyer, Mohit et al. **?**Deep unordered composition rivals syntactic methods for text classification**?** In: *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint*

conference on natural language processing (volume 1: Long papers). 2015, pp. 1681–1691.

[80] Jernite, Yacine, Bowman, Samuel R, and Sontag, David. **?**Discourse-based objectives for fast unsupervised sentence representation learning**?** In: *arXiv preprint arXiv:1705.00557* (2017).

[81] Jiang, Jing. **?**Multi-task transfer learning for weakly-supervised relation extraction**?** In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 1012–1020.

[82] Johnson, Melvin et al. **?**Google's multilingual neural machine translation system: Enabling zero-shot translation**?** In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 339–351.

[83] Joshi, Vidur, Peters, Matthew, and Hopkins, Mark. **?**Extending a parser to distant domains using a few dozen partially annotated examples**?** In: *arXiv preprint arXiv:1805.06556* (2018).

[84] Joulin, Armand et al. **?**Bag of tricks for efficient text classification**?** In: *arXiv preprint arXiv:1607.01759* (2016).

[85] Joulin, Armand et al. **?**Fasttext. zip: Compressing text classification models**?** In: *arXiv preprint arXiv:1612.03651* (2016).

[86] Kalchbrenner, Nal, Grefenstette, Edward, and Blunsom, Phil. **?**A convolutional neural network for modelling sentences**?** In: *arXiv preprint arXiv:1404.2188* (2014).

[87] Kaplan, Andreas and Haenlein, Michael. **?**Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence**?** In: *Business Horizons* 62.1 (2019), pp. 15–25.

[88] Katiyar, Arzoo and Cardie, Claire. **?**Going out on a limb: Joint extraction of entity mentions and relations without dependency trees**?** In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 917–928.

[89] Kendall, Alex, Gal, Yarin, and Cipolla, Roberto. **?**Multi-task learning using uncertainty to weigh losses for scene geometry and semantics**?** In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491.

[90] Keskar, Nitish Shirish et al. **?**On large-batch training for deep learning: Generalization gap and sharp minima**?** In: *arXiv preprint arXiv:1609.04836* (2016).

[91] Kim, Jin-Dong et al. **?**Introduction to the bio-entity recognition task at JNLPBA**?** In: *Proceedings of the international joint workshop on natural language processing in biomedicine and its applications*. Citeseer. 2004, pp. 70–75.

[92] Kim, Yoon. **?**Convolutional neural networks for sentence classification**?** In: *arXiv preprint arXiv:1408.5882* (2014).

[93] Kingma, Diederik P and Ba, Jimmy. **?**Adam: A method for stochastic optimization**?** In: *arXiv preprint arXiv:1412.6980* (2014).

[94] Kingsbury, Paul R and Palmer, Martha. **?**From TreeBank to PropBank.**?** In: *LREC*. Citeseer. 2002, pp. 1989–1993.

[95] Kiperwasser, Eliyahu and Ballesteros, Miguel. **?**Scheduled multi-task learning: From syntax to translation**?** In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 225–240.

[96] Kiros, Jamie and Chan, William. **?**Inferlite: Simple universal sentence representations from natural language inference data**?** In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 4868–4874.

[97] Koo, Terry, Carreras, Xavier, and Collins, Michael. **?**Simple semi-supervised dependency parsing**?** In: *Proceedings of ACL-08: HLT*. 2008, pp. 595–603.

[98] Krishna, Kalpesh, Jyothi, Preethi, and Iyyer, Mohit. **?**Revisiting the importance of encoding logic rules in sentiment classification**?** In: *arXiv preprint arXiv:1808.07733* (2018).

[99] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. **?**Imagenet classification with deep convolutional neural networks**?** In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[100] Krogh, Anders and Hertz, John A. **?**A simple weight decay can improve generalization**?** In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[101] Kudo, Taku. **?**Subword regularization: Improving neural network translation models with multiple subword candidates**?** In: *arXiv preprint arXiv:1804.10959* (2018).

[102] Kudo, Taku and Richardson, John. **?**Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing**?** In: *arXiv preprint arXiv:1808.06226* (2018).

[103] Lample, Guillaume and Conneau, Alexis. **?**Cross-lingual language model pretraining**?** In: *arXiv preprint arXiv:1901.07291* (2019).

[104] Le, Quoc and Mikolov, Tomas. **?**Distributed representations of sentences and documents**?** In: *International conference on machine learning*. 2014, pp. 1188–1196.

[105] LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. **?**Deep learning**?** In: *nature* 521.7553 (2015), pp. 436–444.

[106] Lee, Jinhyuk et al. **?**BioBERT: a pre-trained biomedical language representation model for biomedical text mining**?** In: *Bioinformatics* 36.4 (2020), pp. 1234–1240.

[107] Levy, Omer and Goldberg, Yoav. **?**Neural word embedding as implicit matrix factorization**?** In: *Advances in neural information processing systems*. 2014, pp. 2177–2185.

[108] Levy, Omer, Goldberg, Yoav, and Dagan, Ido. **?**Improving distributional similarity with lessons learned from word embeddings**?** In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.

[109] Li, Jing et al. **?**A Survey on Deep Learning for Named Entity Recognition**?** In: *arXiv preprint arXiv:1812.09449* (2018).

[110] Liu, Haibin et al. **?**BioLemmatizer: a lemmatization tool for morphological processing of biomedical text**?** In: *Journal of biomedical semantics* 3.1 (2012), p. 3.

[111] Liu, Xiaodong et al. **?**Representation learning using multi-task deep neural networks for semantic classification and information retrieval**?** In: (2015).

[112] Liu, Yinhan et al. **?**Roberta: A robustly optimized bert pretraining approach**?** In: *arXiv preprint arXiv:1907.11692* (2019).

[113] Liu, Yinhan et al. **?**RoBERTa: A Robustly Optimized BERT Pretraining Approach**?** In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[114] Logeswaran, Lajanugen and Lee, Honglak. **?**An efficient framework for learning sentence representations**?** In: *arXiv preprint arXiv:1803.02893* (2018).

[115] Lovins, Julie Beth. **?**Development of a stemming algorithm**?** In: *Mech. Transl. Comput. Linguistics* 11.1-2 (1968), pp. 22–31.

[116] Luong, Thang et al. **?**Multi-task Sequence to Sequence Learning**?** In: (2016).

[117] Malaviya, Chaitanya, Neubig, Graham, and Littell, Patrick. **?**Learning language representations for typology prediction**?** In: *arXiv preprint arXiv:1707.09569* (2017).

[118] Marcus, Mitchell, Santorini, Beatrice, and Marcinkiewicz, Mary Ann. **?**Building a large annotated corpus of English: The Penn Treebank**?** In: (1993).

[119] Marsland, Stephen. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[120] Martin, Louis et al. **?**CamemBERT: a Tasty French Language Model**?** In: (2019).

[121] Mayhew, Stephen, Tsai, Chen-Tse, and Roth, Dan. **?**Cheap translation for cross-lingual named entity recognition**?** In: *Proceedings of the 2017 conference on empirical methods in natural language processing*. 2017, pp. 2536–2545.

[122] McCulloch, Warren S and Pitts, Walter. **?**A logical calculus of the ideas immanent in nervous activity**?** In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[123] McDonald, R. **?**A universal part-of-speech tagset**?** In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*. 2012.

[124] Merity, Stephen, Keskar, Nitish Shirish, and Socher, Richard. **?**Regularizing and optimizing LSTM language models**?** In: *arXiv preprint arXiv:1708.02182* (2017).

[125] Mikolov, Tomas et al. **?**Distributed representations of words and phrases and their compositionality**?** In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[126] Mikolov, Tomas et al. **?**Efficient estimation of word representations in vector space**?** In: *arXiv preprint arXiv:1301.3781* (2013).

[127] Miller, Scott, Guinness, Jethran, and Zamanian, Alex. **?**Name tagging with word clusters and discriminative training**?** In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*. 2004, pp. 337–342.

[128] Minsky, Marvin L and Papert, Seymour A. **?**Perceptrons: expanded edition**?** In: (1988).

[129] Misra, Ishan et al. **?**Cross-stitch networks for multi-task learning**?** In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3994–4003.

[130] Mitchell, Thomas M. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[131] Mnih, Andriy and Teh, Yee Whye. **?**A fast and simple algorithm for training neural probabilistic language models**?** In: *arXiv preprint arXiv:1206.6426* (2012).

[132] Mulcaire, Phoebe, Swayamdipta, Swabha, and Smith, Noah. **?**Polyglot semantic role labeling**?** In: *arXiv preprint arXiv:1805.11598* (2018).

[133] Niehues, Jan and Cho, Eunah. **?**Exploiting linguistic resources for neural machine translation using multi-task learning**?** In: *arXiv preprint arXiv:1708.00993* (2017).

[134] Nivre, Joakim et al. **?**Universal dependencies v1: A multilingual treebank collection**?** In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. 2016, pp. 1659–1666.

[135] Olorisade, Babatunde K, Brereton, Pearl, and Andras, Peter. **?**Reproducibility in machine Learning-Based studies: An example of text mining**?** In: (2017).

[136] Owoputi, Olutobi et al. **?**Improved part-of-speech tagging for online conversational text with word clusters**?** In: *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2013, pp. 380–390.

[137] Pagliardini, Matteo, Gupta, Prakhar, and Jaggi, Martin. **?**Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features**?** In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 528–540.

[138] Pan, Sinno Jialin and Yang, Qiang. **?**A survey on transfer learning. IEEE Transactions on knowledge and data engineering**?** In: *22 (10): 1345* 1359 (2010).

[139] Pappas, Nikolaos and Popescu-Belis, Andrei. **?**Multilingual hierarchical attention networks for document classification**?** In: *arXiv preprint arXiv:1707.00896* (2017).

[140] Pasunuru, Ramakanth and Bansal, Mohit. **?**Multi-task video captioning with video and entailment generation**?** In: *arXiv preprint arXiv:1704.07489* (2017).

[141] Paszke, Adam et al. **?**Automatic differentiation in PyTorch**?** In: (2017).

[142] Peng, Hao, Thomson, Sam, and Smith, Noah A. **?**Deep multitask learning for semantic dependency parsing**?** In: *arXiv preprint arXiv:1704.06855* (2017).

[143] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. **?**Glove: Global vectors for word representation**?** In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[144] Peters, Matthew E et al. **?**Deep contextualized word representations**?** In: *arXiv preprint arXiv:1802.05365* (2018).

[145] Peters,
Matthew E et al. **?**Dissecting contextual word embeddings: Architecture and representation**?** In: *arXiv preprint arXiv:1808.08949* (2018).

[146] Peters, Matthew E et al. **?**Semi-supervised sequence tagging with bidirectional language models**?** In: *arXiv preprint arXiv:1705.00108* (2017).

[147] Porteous, Ian et al. **?**Fast collapsed gibbs sampling for latent dirichlet allocation**?** In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 569–577.

[148] Porter, MF. *An algorithm for su x stripping., 14 (3): 130 137*. 1980.

[149] Prechelt, Lutz. **?**Early stopping-but when?**?** In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[150] Radford, Alec et al. *Improving language understanding by generative pre-training*. 2018.

[151] Radford, Alec et al. **?**Language models are unsupervised multitask learners**?** In: *OpenAI Blog* 1.8 (2019), p. 9.

[152] Raffel, Colin et al. **?**Exploring the limits of transfer learning with a unified text-to-text transformer**?** In: *arXiv preprint arXiv:1910.10683* (2019).

[153] Raina, Rajat et al. **?**Self-taught learning: transfer learning from unlabeled data**?** In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 759–766.

[154] Ramachandran, Prajit, Zoph, Barret, and Le, Quoc V. **?**Searching for activation functions**?** In: *arXiv preprint arXiv:1710.05941* (2017).

[155] Ramsundar, Bharath et al. **?**Massively multitask networks for drug discovery**?** In: *arXiv preprint arXiv:1502.02072* (2015).

[156] Ratinov, Lev and Roth, Dan. **?**Design challenges and misconceptions in named entity recognition**?** In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*. 2009, pp. 147–155.

[157] Reenen, P van and Mulken, MJP van. **?**Studies in Stemmatology**?** In: (1996).

[158] Ross, Girshick. **?**Fast r-cnn**?** In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[159] Ruder, Sebastian. **?**An overview of gradient descent optimization algorithms**?** In: *arXiv preprint arXiv:1609.04747* (2016).

[160] Ruder, Sebastian. **?**Neural transfer learning for natural language processing**?** PhD thesis. NUI Galway, 2019.

[161] Ruder, Sebastian. **?**Nlp's imagenet moment has arrived**?** In: *Gradient, July* 8 (2018).

[162] Ruder, Sebastian, Ghaffari, Parsa, and Breslin, John G. **?**Towards a continuous modeling of natural language domains**?** In: *arXiv preprint arXiv:1610.09158* (2016).

[163] Ruder, Sebastian et al. **?**Latent multi-task architecture learning**?** In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4822–4829.

[164] Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. **?**Learning representations by back-propagating errors**?** In: *nature* 323.6088 (1986), pp. 533–536.

[165] Sang, Erik F and De Meulder, Fien. **?**Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition**?** In: *arXiv preprint cs/0306050* (2003).

[166] Sanh, Victor, Wolf, Thomas, and Ruder, Sebastian. **?**A hierarchical multi-task approach for learning embeddings from semantic tasks**?** In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 6949–6956.

[167] Schmidhuber, Jürgen. **?**Deep learning in neural networks: An overview**?** In: *Neural networks* 61 (2015), pp. 85–117.

[168] Schuster, Mike and Paliwal, Kuldip K. **?**Bidirectional recurrent neural networks**?** In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[169] Schwartz, Roy et al. **?**Green ai**?** In: *arXiv preprint arXiv:1907.10597* (2019).

[170] Sennrich,
Rico, Haddow, Barry, and Birch, Alexandra. **?**Neural machine translation of rare words with subword units**?** In: *arXiv preprint arXiv:1508.07909* (2015).

[171] Sennrich, Rico, Haddow, Barry, and Birch, Alexandra. **?**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://www.aclweb.org/anthology/P16-1162.

[172] Smith, Noah A. **?**Linguistic structure prediction**?** In: *Synthesis lectures on human language technologies* 4.2 (2011), pp. 1–274.

[173] Socher, Richard, Bengio, Yoshua, and Manning, Christopher D. **?**Deep learning for NLP (without magic)**?** In: *Tutorial Abstracts of ACL 2012*. 2012, pp. 5–5.

[174] Socher, Richard et al. **?**Recursive deep models for semantic compositionality over a sentiment treebank**?** In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[175] Søgaard, Anders and Goldberg, Yoav. **?**Deep multi-task learning with low level tasks supervised at lower layers**?** In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2016, pp. 231–235.

[176] Srivastava, Nitish et al. **?**Dropout: a simple way to prevent neural networks from overfitting**?** In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[177] Suárez, Pedro Javier Ortiz, Sagot, Benoît, and Romary, Laurent. **?**Asynchronous pipeline for processing huge corpora on medium to low resource infrastructures**?** In: *Challenges in the Management of Large Corpora (CMLC-7) 2019* (2019), p. 9.

[178] Subramanian, Sandeep et al. **?**Learning general purpose distributed sentence representations via large scale multi-task learning**?** In: *arXiv preprint arXiv:1804.00079* (2018).

[179] Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. **?**Sequence to sequence learning with neural networks**?** In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[180] Suzuki, Jun et al. **?**An empirical study of semi-supervised structured conditional models for dependency parsing**?** In: *Conference on Empirical Methods in Natural Language Processing 2009*. 2009, pp. 551–560.

[181] Swayamdipta, Swabha et al. **?**Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold**?** In: *arXiv preprint arXiv:1706.09528* (2017).

[182] Teller, Virginia. **?**Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition**?** In: *Computational Linguistics* 26.4 (2000), pp. 638–641.

[183] Tibshirani, Robert. **?**Regression shrinkage and selection via the lasso**?** In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[184] Tikhonov, Andrey Nikolayevich. **?**On the stability of inverse problems**?** In: *Dokl. Akad. Nauk SSSR*. Vol. 39. 1943, pp. 195–198.

[185] Tsuboi, Yuta et al. **?**Direct density ratio estimation for large-scale covariate shift adaptation**?** In: *Journal of Information Processing* 17 (2009), pp. 138–155.

[186] Turian, Joseph, Ratinov, Lev, and Bengio, Yoshua. **?**Word representations: a simple and general method for semi-supervised learning**?** In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. 2010, pp. 384–394.

[187] Vapnik, Vladimir N and Chervonenkis, A Ya. **?**On the uniform convergence of relative frequencies of events to their probabilities**?** In: *Measures of complexity*. Springer, 2015, pp. 11–30.

[188] Vaswani, Ashish et al. **?**Attention is all you need**?** In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[189] Wallis, Guy and Bülthoff, Heinrich. **?**Learning to recognize objects**?** In: *Trends in cognitive sciences* 3.1 (1999), pp. 22–31.

[190] Wang, Shuohang et al. **?**Reinforced reader-ranker for open-domain question answering**?** In: *arXiv preprint arXiv:1709.00023* (2017).

[191] Webster, Jonathan J and Kit, Chunyu. **?**Tokenization as the initial phase in NLP**?** In: *COLING 1992 Volume 4: The 15th International Conference on Computational Linguistics*. 1992.

[192] Wieting, John and Gimpel, Kevin. **?**Revisiting recurrent networks for paraphrastic sentence embeddings**?** In: *arXiv preprint arXiv:1705.00364* (2017).

[193] Wieting, John et al. **?**Towards universal paraphrastic sentence embeddings**?** In: *arXiv preprint arXiv:1511.08198* (2015).

[194] Williams, Ronald J and Zipser, David. **?**A learning algorithm for continually running fully recurrent neural networks**?** In: *Neural computation* 1.2 (1989), pp. 270–280.

[195] Wolf, Thomas et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2019. arXiv: `1910.03771 [cs.CL]`.

[196] Wolpert, David H and Macready, William G. **?**No free lunch theorems for optimization**?** In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[197] Wu, Shuangzhi et al. **?**Sequence-to-dependency neural machine translation**?** In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 698–707.

[198] Wu, Yonghui et al. **?**Google's neural machine translation system: Bridging the gap between human and machine translation**?** In: *arXiv preprint arXiv:1609.08144* (2016).

[199] Yang, Bishan and Mitchell, Tom. **?**A joint sequential and relational model for frame-semantic parsing**?** In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 1247–1256.

[200] Yang, Yongxin and Hospedales, Timothy M. **?**Trace Norm Regularised Deep Multi-Task Learning**?** In: (2017).

[201] Yang, Zhilin, Salakhutdinov, Ruslan, and Cohen, William. **?**Multi-task cross-lingual sequence tagging from scratch**?** In: *arXiv preprint arXiv:1603.06270* (2016).

[202] Yang, Zhilin, Salakhutdinov, Ruslan, and Cohen, William W. **?**Transfer learning for sequence tagging with hierarchical recurrent networks**?** In: *arXiv preprint arXiv:1703.06345* (2017).

[203] Zhang, Kelly W and Bowman, Samuel R. **?**Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis**?** In: *arXiv preprint arXiv:1809.10040* (2018).

[204] Zhao, Kai and Huang, Liang. **?**Joint syntacto-discourse parsing and the syntacto-discourse treebank**?** In: *arXiv preprint arXiv:1708.08484* (2017).

[205] Zhu, Yukun et al. **?**Aligning books and movies: Towards story-like visual explanations by watching movies and reading books**?** In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27.

[206] Zoph, Barret and Knight, Kevin. **?**Multi-source neural translation**?** In: *arXiv preprint arXiv:1601.00710* (2016).

[207] Zou, Will Y et al. **?**Bilingual word embeddings for phrase-based machine translation**?** In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 1393–1398.

TRITA -EECS-EX-2020:733