Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Towards Explainable Neural Language Models

Master Thesis

2022
Bc. Tomáš Vician

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Towards Explainable Neural Language Models

Master Thesis

Study program:	Computer Science
Field of study:	2508 Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Endre Hamerlik

Bratislava, 2022
Bc. Tomáš Vician

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Tomáš Vician
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Towards Explainable Neural Language Models

Annotation: Recent Language Models - mainly based on Attention mechanism - have shown striking capabilities in many domains and subtasks: Summarization, Question Answering etc.
On the other hand, the scale of their parameters make it hard to explain the behaviour these LMs showcase.

The aim of the thesis is to unravel:

1) Predictions of such LMs: Why does the network behave in a way we registered?
2) Whether their embeddings encode morpho-syntactic features
3) If such features are present, which parts of the model are responsible for them?

The thesis will try to tackle these questions using Probing technique. [1,2]
Expected results include explanations of internal representations of neural LM regarding the morpho-syntactic features of their textual input.

Literature: [1] Alain, G., & Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. arXiv preprint arXiv:1610.01644.

[2] Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. (2018). What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics.

Keywords: Explainable AI, xAI, Natural Language Processing, NLP

Supervisor:	Mgr. Endre Hamerlik
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	30.10.2020
Approved:	18.11.2020 prof. RNDr. Rastislav Kráľovič, PhD.

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Guarantor of Study Programme

.....

Student

.....

Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Vician
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Towards Explainable Neural Language Models
Smerom k vysvetliteľným neurálnym jazykovým modelom

Anotácia: Aktuálne jazykové modely – hlavne založené na mechanizme pozornosti – ukázali pozoruhodné schopnosti v mnohých doménach a podúlohách: sumarizácia, zodpovedanie otázok atď.
Na druhej strane kvôli enormne vysokému počtu parametrov týchto modelov je ťažké vysvetliť ich správanie.

Cieľom diplomovej práce je objasniť:

1) Predikcie týchto neurňových sietí: Prečo sa model správa tak, ako sme zaregistrovali?
2) Či sa v Embeddingoch týchto jazykových modelov vytvárajú reprezentácie morfo-syntaktických konceptov?
3) V prípade ak sú zachytiteľné, ktoré časti daného modelu disponujú týmito reprezentáciami?

Diplomová práca sa pokúsi riešit tieto otázky použitím tzv. Probing techniky [1,2].

Očakávaným výsledkom sú vysvetlenia interných reprezentácií v neurálnom jazykovom modeli z hladiska morfo-syntaktických vlastností vstupneho textu.

Literatúra:

[1] Alain, G., & Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. arXiv preprint arXiv:1610.01644.

[2] Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics.

Kľúčové slová:

Vedúci: Mgr. Endre Hamerlik

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 30.10.2020

Dátum schválenia: 18.11.2020 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: V prvom rade by som rád vyjadril moju vďačnosť voči môjmu školiteľovi, Mgr. Endremu Hamerlikovi, za jeho ochotu viesť ma počas tejto práce zavedúc ma do zaujímavej tematiky spracovania prirodzených jazykov a taktiež za jeho cenne rady a konzultácie. Tiež by som sa rád poďakoval mojej rodine a priateľom za všetkú ich podporu počas mojich štúdií.

Acknowledgement: First of all, I would like to express my gratitude towards my supervisor, Mgr. Endre Hamerlik for accepting me as his supervisee while introducing me to the interesting area of natural language processing, for his advises and consultations during during the writing of this thesis. I would also like to thank to my family and friends for all their support during my studies.

Abstract

Natural language processing has been rapidly advancing in recent years. The used neural models in this domain are more and more complex. Different techniques have been developed to understand their inner representations. Inspired by these techniques, we tried to extract some linguistic features from the inner representations of a chosen model, more precisely the BERT model. We conducted an ad hoc syntactic dependency analysis and used the so-called probing technique to predict parts of speech. We also present our results and discuss possible further work.

Keywords: Explainable AI, xAI, Natural Language Processing, NLP, Transformer, probing

Abstrakt

Spracovanie prirodzeného jazyka sa v posledných rokoch rýchlo vyvíjalo. Používané neurálne modely v tejto doméne sú čoraz komplexnejšie, preto bolo vyvynutých niekolko rôznych techník na pochopenie ich vnútorných reprezentácií. Inšpirovaní týmito technikami sme sa pokúsili extrahovať niektoré gramatické vlastnosti z vnútorných reprezentácií zvoleného jazykového modelu, presnejšie modelu BERT. Vykonali sme ad hoc analýzu syntaktických stromov a použili sme takzvanú probing techniku na predpovedanie slovných druhov. Taktiež prezentujeme naše výsledky a diskutujeme o ďalšej možnej práci.

Kľúčové slová: vysvetliteľná UI, xAI, spracovanie prirodzedných jazykov, NLP, Transformer, probing technika

Contents

List of Figures

Introduction

In this work, we try to investigate the inner workings of given neural language models based on the attention mechanism, namely the Transformer model-based BERT [11]. This model has produced remarkable results in the field of natural language processing, although its size and structure cause it to be black-box-like. For this reason researchers have started to come up with methods providing more insight into the inner workings and representations of these models. One such method is probing. This technique in essence tries to predict some linguistic feature based upon a chosen inner representation of the model. We made two sets of experiments to determine the degree to which some morpho-syntactic features are represented inside the model. The first experiment aims to find the correlation between the weight dynamics of the so-called attention-heads and grammatical dependencies. The second experiment uses a probe, in our case a simple multi-layer perceptron, to determine whether part-of-speech is encoded in the embeddings produced by the model at the given layers.

The structure of our work is simple. At the beginning, we present the broader context, and then we introduce some definitions regarding the topic. The second part describes the data and techniques we used. In the third part, we discuss the results we obtained. In the last part we hand over the conclusions we made while mentioning the possible continuation and future work.

Chapter 1

Related work

Since its emergence, the field of AI has been concerned with natural language processing, which is today more relevant than ever, while going through remarkable progress in recent years. Many long-established systems are being replaced by neural networks. Different neural models are being introduced at a rapid pace. These models are also often more complex than their older counterparts. With the increasing complexity, the black-box nature of models has increased too. These developments led the NLP community of researchers towards the development of novel interpretation and evaluation techniques for these inherently uninterpretable models.

A plethora of methods [1] were used to monitor multiple features of layers in different convolutional neural networks used for image classification. Among others the authors utilized linear classifiers, already referring to them as probes. As they mentioned, the term "probing" was already in use in the context of their investigation. Linear classifier probes were initially related to reflect the nature of information flow between the layers regarding information entropy. The authors however used linear-only probes to investigate the inner representations of given models.

Different natural language processing analysis methods also aim to extract information from the inner representations of neural networks. A survey of Yonatan Belinkov and James Glass [5] collects a notable set of examples of such works, various probing methods (also called diagnostic classifiers) including the core method of the present thesis: non-linear probing of contextualized language models [9].

Probing sentence embeddings

One of the cornerstone studies in this field was published by Alexis Conneau et al. [9], where 10 different probing tasks are designed to capture relatively simple linguistic features, such as the number of words in a sentence, word content of the sentences, tree depth of the hierarchical structure of sentences, verb tense of the sentence and others. The authors use these tasks to examine embeddings that were generated by three selected encoders trained in multiple ways. They report their findings on the

properties of the investigated encoders and training methods alike. Since the BERT [11] model has been introduced with its enormous amount of parameters and remarkable performance with NLP tasks, the interest in understanding its inner workings was immediately substantial.

Among such efforts can be listed this work of Papadimitriou et al. [31], where the authors investigate how Multilingual BERT (mBERT) encodes grammar. They inspect how the high-order grammatical feature of morphosyntactic alignment manifests itself across the embedding spaces of different languages (how "subject" is defined in different languages).

Different probing tasks were also designed for the BERT model by Tenney, Xia, Chen et al. [39] comparing it with other NLP models in terms of explainability. Building on token-level probing the authors introduced the so called edge probing task design and constructed a suite of sub-sentence tasks. They probed word-level contextual representations from four selected models and studied how those models encode sentence structure across a range of syntactic, semantic; local, and long-range phenomena. Representin the sentences as a list of tokens and chose labelled edges (edge from one span of embeddings of tokens to another) they tried to predict the labels on those edges. A few examples of these tasks are part-of-speech tagging of a selected token, named entity labeling and others. Inspired by these trends we decided to conduct our experiments aiming to investigate:

- the object- and subjecthood representation in a finetuned BERT model.
- the Part-of-speech representation in the BERT model using non-linear probes in line with the related work we cited above.

1.1 Artificial neural networks

In this section we describe relevant models and definitions, using the learning material for Deep learning [44] created by Zhang et al. Artificial neural networks (ANN) (also called multilayer perceptrons if having multiple hidden layers, see 1.1) are computing systems designed to recognize hidden relationships in a set of data. They aim to approximate some function on a domain of an input. They are inspired by the biological neural networks (animal or human brains). Further on we will refer to them only as Neural networks. In the practical sense, neural networks are a collection of nodes - artificial neurons - that are connected. We can represent those connections as edges of graphs. These connections, like the synapses of a biological neuron, can transmit signals to other neurons. The strength of those connections is represented by a real value, the weight, assigned to the given nodes, which decreases or increases the strength of the signal. The output of the nodes is computed by a given non-linear function applied on

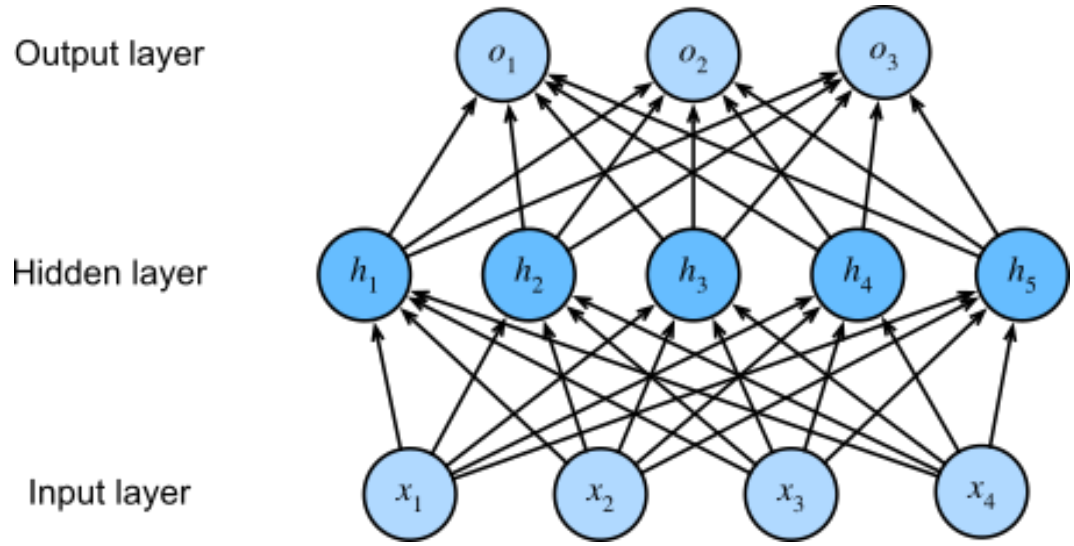


Figure 1.1: An MLP with a hidden layer of 5 hidden units. [44].

the sum of the activity routing the neuron in question. The weighted output may be part of the input of the next node. Neurons typically have a set of learnable parameters, weights that adjusts as learning proceeds. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers that form the network.

An essential property of these networks is their adaptability. As a result of the learning method, they are capable of generating increasingly good results without changing the output criteria. Even though they can be used for unsupervised learning, e.g. KSOM (Kohonen Self-Organizing Maps [26]) being a good example, they are mostly used in the context of supervised learning. Unsupervised learning does not need labeled data, it can detect patterns in the data or create groupings in the data or represent the data in a compressed format. This method often maps the input to the output while being forced to learn about the structure of the date to be able to do this. Supervised learning on the other hand uses labeled data and tries to map an input to a defined output. The training data for supervised models are mostly made out of pairs of the input data and the desired output value or supervisory signal. Let us consider this example: the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ denotes a batch of n input examples each having d input features (d being 4 in the case of the example figure 1.1).

For the MLP with one hidden layer (as is our example 1.1 with h hidden units, we denote by $\mathbf{H} \in \mathbb{R}^{n \times h}$ the outputs of the hidden layer.

The hidden and output layers being fully connected (a neuron is connected with all neurons from the previous layer), we have hidden-layer weights $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ and biases $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$ and output-layer weights $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and biases $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$. We calculate the outputs $\mathbf{O} \in \mathbb{R}^{n \times q}$ of the MLP with one hidden layer:

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)},$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

We can prove, that only adding a hidden-layer does not change the fact that with linear operations the model can only approximate affine functions (for proof and explanation see [44]).

To use the potential of the multilayer architecture a nonlinear activation function σ is needed to be applied to each hidden unit following the affine transformation.

The outputs of these activation functions are called activations. Applying these activation functions, it is no longer possible to substitute the MLP with a linear model (the output often can not be modeled using a linear approximator):

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

As the rows in $\mathbf{X}$ correspond to the examples of batch, let us define σ to apply to its inputs per row (one input example at a time as defined in [44]).

Activations

An activation function is a function that is used in an artificial neural network for improving the networks ability to learn complex patterns in the data. They decide whether a neuron activates or not by calculating the weighted sum and adding bias to it. Activations are differentiable operators transforming input signals to outputs, while adding non-linearity. We will again rely on the descriptions and definitions from [44]:

Softmax

A nice feature of the softmax function [16] is that it transforms any input vector to an other one, which will have all its components in the interval (0, 1) with the sum of 1. It takes a vector $\mathbf{x}$ of n real numbers, normalizing them to a probability distribution of n probabilities divided by sum of exponentials of the input numbers. The resulting components can be thought about as a probability distribution, because their sum equals one. Softmax is typically used in the output layer. It is defined with this equation:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1.1}$$

where x_i are the components of vector $\mathbf{x}$ from x_1 to x_n and j going through all elements.

Sigmoid

The sigmoid function, used by Rumelhart et al. [35] (also called squashing function [44] transforms its inputs, for which the values are from the domain $\mathbb{R}$, to outputs in the interval of (0, 1). We can think of the sigmoid function as a special case of the softmax.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

ReLU

The sigmoid function has mostly been replaced by the Rectified Linear Unit or ReLU. It is the most popular choice, due to its simple implementation and good performance. It was introduced in 2010 by Nair et. al [30] and has good convergence on stochastic gradient descent compared to the sigmoidal. We used this activation function in our experiments too as we achieved faster learning rates with it. For an element x, the ReLU function returns x if it is greater than zero and 0 otherwise:

$$\text{ReLU}(x) = \max(x, 0).$$

Its disadvantage is that it can "die" when in training, due to large gradients, preventing it from reactivation. There is however modified versions of the ReLU function that try to overcome its drawbacks, for example the **pReLU** (parameterized ReLU [17]). This version sums a ReLU with a linear term, so the information gets through always, even when the argument is negative:

$$\text{pReLU}(x) = \max(0, x) + \alpha \min(0, x).$$

Learning algorithms

In the previous subsections, we have shown the structure of a neural network. The key ability of neural networks is their ability to learn. We are going to describe this learning process in this subsection, again citing from [44]. Let us look at the forward pass in the network for the sake of simplicity without the bias term. The input example is $\mathbf{x} \in \mathbb{R}^d$ The temporary inner result is:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x},$$

$\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ being the weight parameter from the hidden layer. After running the inner result $\mathbf{z} \in \mathbb{R}^h$ through the activation function ϕ we get the hidden activation vector with length h,

$$\mathbf{h} = \phi(\mathbf{z}).$$

The hidden variable $\mathbf{h}$ is also an intermediate variable. Assuming the example from 1.1 the parameters of the output layer form a weight of $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$, we get the variable at the output layer with a vector of length q:

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}.$$

Adam

Now we mention the optimization technique we used for the training of our model. As it is out of the scope of this work to list all of the techniques, we will mention only the so-called **Adam** (adaptive movements) optimization algorithm introduced by Kingma et al. [25]. It is one of the widely preferred optimization techniques used in deep learning. We cite the description of this algorithm from [44].

Adam uses exponential weighted moving averages (or leaky averaging) to estimate the momentum and also the second moment of the gradient, that are expressed by these variables (g_t being the gradient):

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$
$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

The β_1 and β_2 are non-negative weighting parameters. Often chosen as $\beta_1 = 0.9$ and $\beta_2 = 0.999$. That is, the variance estimate moves slower than the momentum term. If they are initialized as $\mathbf{v}_0 = \mathbf{s}_0 = 0$ there is a significant amount of bias in the initial phase towards smaller values. This is addressed by using $\sum_{i=0}^{t} \beta^i = \frac{1-\beta^t}{1-\beta}$ to re-normalize terms. Consequently the normalized state variables are given by

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

Having the estimates we the equations can be updated. First, we rescale the gradient

$$\mathbf{g}_t' = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

Instead of the gradient itself, the update uses the momentum $\hat{\mathbf{v}}_t$. Finally, the parameters are updated according to this equation:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}_t'.$$

1.2 Deep learning models

Deep neural networks or more broadly, deep-learning architectures, have been applied to a variety of fields with tremendous success: computer vision, audio recognition, or

more specifically machine vision and speech recognition, bioinformatics and medicine. It is already applied in the industry e.g. in drug design, medical image analysis, social media such as social network filtering but also board game programs, natural language processing (NLP) and many others, where they have produced impressive results. In many of these domains, they offer state-of-the-art solutions for given problems comparable to or sometimes even outperforming top human performance.

Deep learning refers mostly refers to models that have more than 3 layers. It was soon discovered that linear perceptrons have their limitations. [29] As a demonstrative example, it was shown, that a single neuron was unable to implement an XOR function. A network with a non-polynomial activation function with one hidden layer of unbounded width proved to be more "capable", as it may be used as a universal classifier given the necessary conditions are fulfilled (depth, width) [20].

1.3 Attention

Neural networks use attention in a manner that resembles the attention in cognition. For a given task, when a model is dealing with its input, some parts of the input data are often more relevant than others. When the attention mechanism was introduced [3], it was inspired by the supposition that encoder-decoder models had a specific bottleneck, which should be addressed. At the time when phrase-based translators were introduced, the first techniques used multiple components tuned more independently from each other. A new approach was introduced later, as in [24], where one larger neural network was trained, being responsible for processing the input sentence and producing a correct translation. These machine translation models from the family of encoder-decoders encode the input sentence into a fixed-length vector. This approach is the source of the above-mentioned bottleneck. All the information has to be encapsulated in a vector of a given constant size, which is more and more problematic as the length of the input sentence increases.

When addressing this issue, Bahdanau et al. [3] proposed an extension to the encoder-decoder type. The newly introduced model, when generating a word of a translation, looks at the source sentence for clues. A set of positions is associated with the input sentence, in which the model soft-searches for the best relevant information. The next prediction of the model relies on the previously generated words and the aforementioned positions.

RNN Encoder–Decoder

The model proposed by [3] is based on the RNN Enncoder-Decoder framework by [37] and [7]. In this framework the encoder reads the input sentence comprised of vectors

$v = (x_1, \cdots, x_{T_x})$ into an other vector c generated from the sequence of the hidden states.

The decoder is may be designed to predict the next word from the context vector and the previously predicted words. In the design by [3] however, an innovation was introduced. A bidirectional RNN is used as the encoder and the decoder emulates searching in the input sentence when decoding.

The above-mentioned architecture consists of a bidirectional recurrent neural network serving as an encoder. The decoder emulates searching through a source sentence during decoding the translation.

In the proposed architecture the conditional probabilities are defined as:

$$p(y_i|y_1, \ldots, y_{i-1},) = g(y_{i-1}, s_i, c_i), \tag{1.2}$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

The context vector c_i depends on a sequence of annotations $(h_1, \cdots, h_{T_x})$ to which an encoder maps the input sentence. For every annotation, h_i information about the whole input sequence is contained with a focus on the parts around the i-th word of the input sequence.

The weight α_{ij} of each annotation h_j is calculated as follows:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}. \tag{1.3}$$

$$e_{ij} = a(s_{i-1}, h_j)$$

being an *alignment model* which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} and the j-th annotation h_j of the input sentence.

The probability that the target word y_i is aligned to, or translated from, a given source word x_j can be expressed with α_{ij}. In this case, c_i, the i-th context vector, is the annotation that is expected over all the annotations with probabilities α_{ij}.

In the proposed scheme, the authors intended the annotation of each word to summarize the preceding words, as well as the following words. Therefore they proposed to use a bidirectional RNN in a way described below. The given model obtains the annotation for each word x_j by concatenating the forward hidden state and the backward one for h_j. In this way, the annotation h_j contains the summaries of both the preceding words and the next words. Since RNNs are prone to represent more recent inputs better, the annotation h_j is focused on the words near x_j. The produced sequence of annotations is processed later, by the decoder and the alignment model, when computing the context vector.

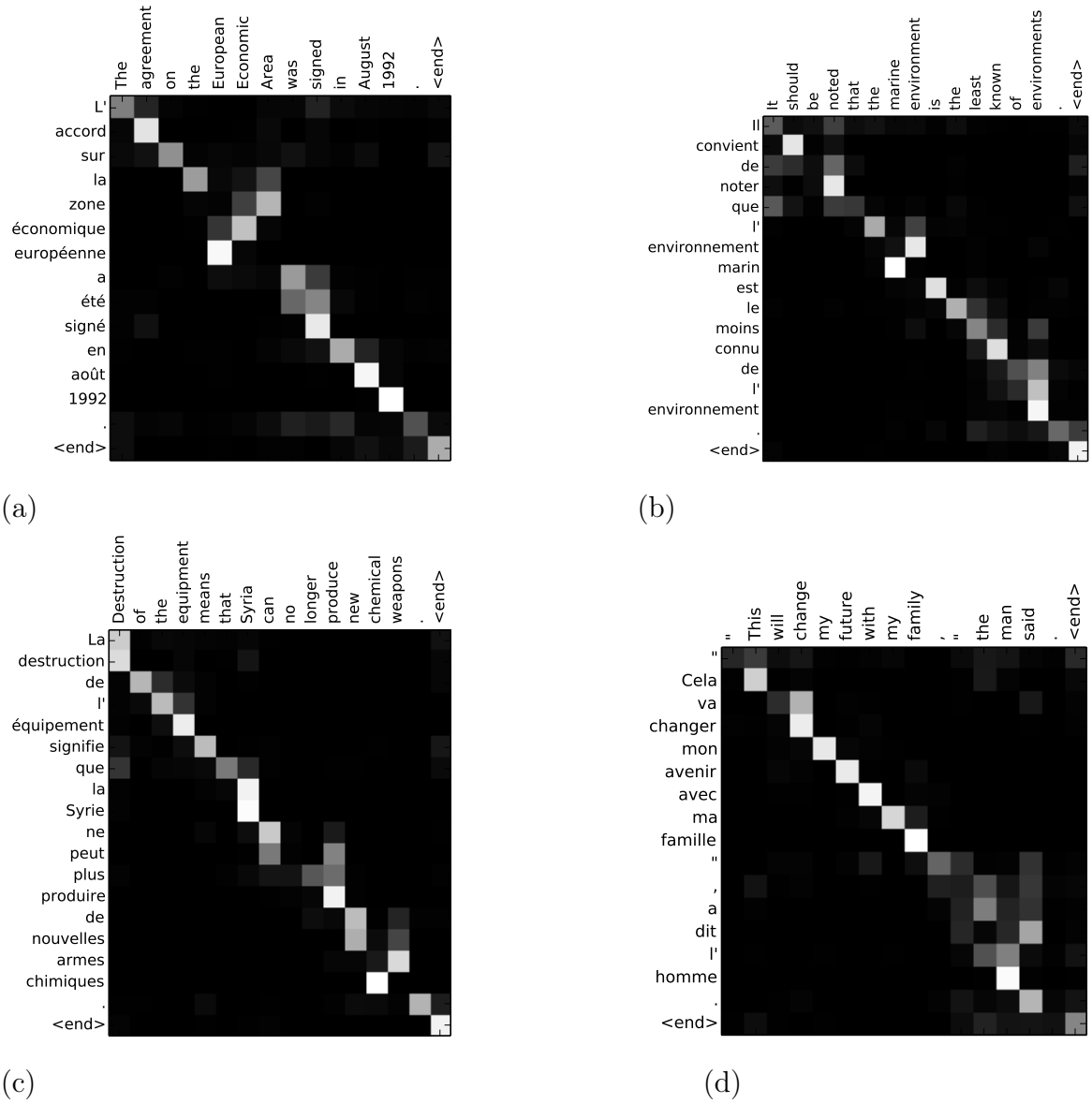


Figure 1.2: Sample alignments found by [3] using RNNsearch-50 (a model trained with sentences of length up to 50 words).

Alignment

The approach put forward by the authors provides a nice intuitive way for inspecting the (soft-)alignment between the words of the generated translation and those of the source sentence. By soft is meant that an alignment is not given in a binary form (aligned and not aligned), but rather weights are generated to indicate the "degree" of the alignment of two given words.

This is done by visualizing the annotation weights α_{ij}, shown in 1.2. The x-axis and y-axis of each plot correspond to the words in the English source sentence and the generated French translation, respectively. The pixels show the weight matrices composed of α_{ij}, the annotation of the j-th source word for the i-th target word using

a grayscale visualization. For every plot, each row of a matrix indicates the weights associated with the annotations. From the annotation matrix, it is visible which positions in the source sentence are considered more important during the generation of the target word.

It is demonstrated in the figures 1.2, that the alignment of words between English and French is largely monotonic. This produces strong weights along the diagonals of the alignment matrices. However, as there are irregularities observable, a number of non-trivial, non-monotonic alignments can be spotted too. Adjectives and nouns often having a different order in the two languages, French and English, stronger wights were produced outside of the diagonal of the figure 1.2 plot (a).

The figure demonstrates the alignment of words of an English sentence and its correctly translated French counterpart. In the three-word sentence part [European Economic Area], the correct translation has the second and third word switched: [zone économique européen]. The RNNsearch was able to correctly align [zone] with [Area], skipping two words ([European] and [Economic]), after which it "looked" back and completed the whole phrase.

While neural networks working in such a way, that the encoder compresses the information to the input of vector of a fixed length, turned out to be an impede performance improvement. Therefore the "networks focus" should be primarily on those more important parts. The relevance of a given part of the data is largely determined by the context.

1.4 Transformer

Until recent years, neural networks and LSTM (long short-term memory) were used by the models providing the best results in the domain of natural language processing including sequence modeling, language modeling, machine translation, and many others. In 2017 a new approach was introduced. Instead of using sequence-aligned recurrent neural networks or convolution the proposed model, [43] relies on self-attention, sometimes also called intra-attention, for the computation of its input and output not needing recurrence. The experiments done by the authors demonstrated not only superior results on many benchmark tasks but also significantly improved the time requirements for their training compared to the state-of-the-art up to that point. Transformer-based models, since their introduction, got very significant attention and were applied to a substantial amount of tasks, mostly in the field of natural language processing, but also in computer vision.

Self-attention is an attention mechanism that relates to different positions of a single sequence for computing a representation of that sequence. The application of

self-attention produced good results in many tasks including summarization, reading comprehension, and many others.

Neural sequence transduction models usually have a structure consisting of an encoder and a decoder. The encoder is mapping the input sequence of symbol representations to a sequence of continuous representations. The decoder is responsible for the generation of an output sequence of symbols, each element at a time, one after another.

The introduction of the Transformer was the first transduction model (reasoning from specific cases to also specific cases, from input to output), which relies solely on self-attention when computing the representations of input and output and not using convolution or a sequence aligned RNN.

The creation of the Transformer model was significantly motivated by the possibility of parallelization. It made possible a shorter training time while increasing the results on different tasks such as translation and later many others. In models such as the ConvS2S [14] which use convolutional neural networks as their building blocks the hidden representations are computed in a parallel manner for the input and output positions. However, unlike in the case of Transformers, relating signals between two input or output positions requires an increasing number of operations when increasing their distance from each other, thus making it more difficult to learn dependencies between points further from each other. In the case of ConvS2S this growth is linear. The Transformer requires only a constant number of operations. However, this improvement is not for "free", due to the reduction in effective resolution because the attention-weighted positions are averaged. The authors of the models handled this drawback with a multi-head attention.

Architecture of the transformer

The two main components of a transformer are an encoder and a decoder part 1.6. Both these components are organized as a stack of six layers of decoders or encoders respectively 1.3.

- **Encoder:** As mentioned earlier, the encoder has a stack of six identical layers. Every one of these layers has two main parts: a multi-head self-attention mechanism, and a position-wise fully connected feed-forward network. The output of the sub-layers is described by this equation: LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function realized by the sub-layer.

- **Decoder:** The decoder is composed of a stack of six identical layers too. An important feature that differentiates the decoder from the encoder component is that apart from the sub-layers of the encoder layers, the decoder layer contains a

[1]source: https://jalammar.github.io/illustrated-transformer/

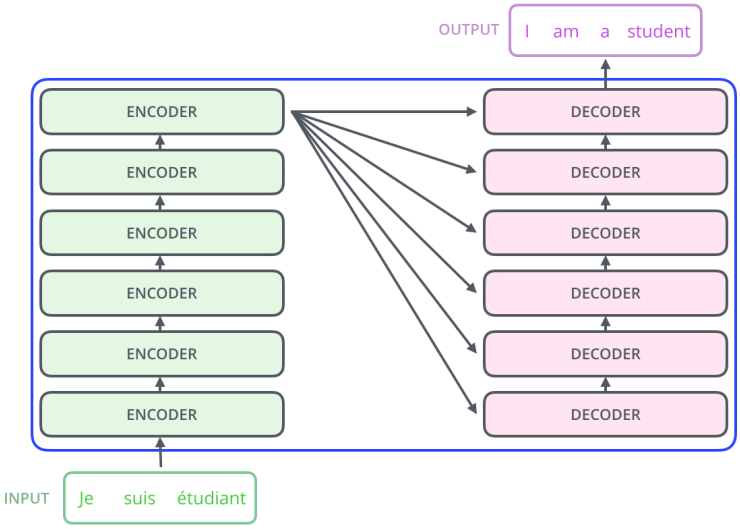


Figure 1.3: The encoder-decoder architecture high level view. [1]

third sub-layer in the middle of the two original ones. The role of this additional sub-layer is to apply multi-head attention to the input of the decoder. In the other aspects, the decoder has a similar structure to the encoder, having connections around the sub-layers with normalization at the end. However, the decoder's self-attention is prevented from reaching subsequent positions which together with the output embeddings shifted, results in predictions depending only on the known outputs at lesser positions.

- **Attention** The function which is referred to as attention by the authors is further described as the so-called query and key-value pair set being mapped to a given output. These three structures, the query, key, and value are different vectors. The computation of an output is the following: the given values a summed according to weights assigned to them. These aforementioned weights are assigned to the values by a compatibility function of queries and keys.

  The processing of the words from the input sequence by the model uses self-attention, which enables to get information from the input sequence at different positions which provides a context for better encoding.

  The notion of self-attention is a fundamental part of [43] and therefore we should also look at it somewhat more precisely.

  A relatively in-depth intuitive description can be found at [2], from which we extracted some explanations and a step-by-step description for understanding the attention mechanism, which is implemented by the authors using matrix operations.

  This process of calculating the self-attention can be described in a few steps.

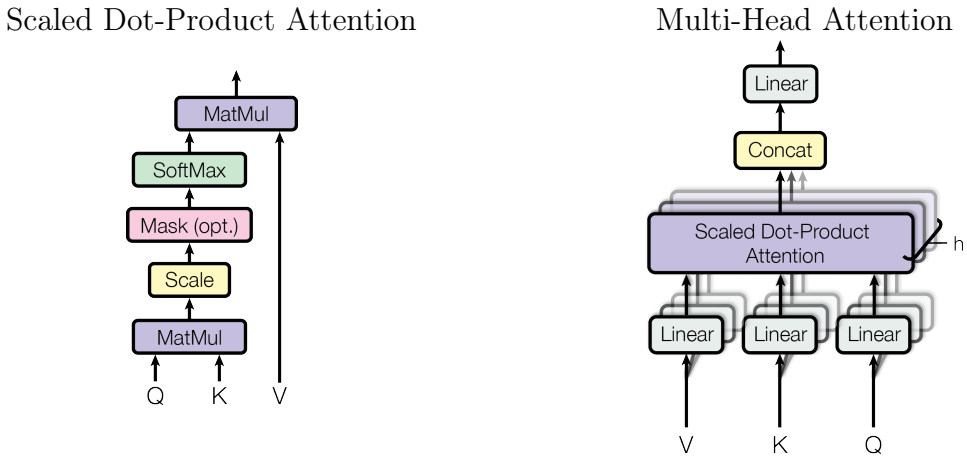


Figure 1.4: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention (consists several attention layers running in parallel). The visualizations are from [43]

At first, for each of the encoder's input vectors, which are usually embeddings of some words, three vectors are created: query, key, and value vectors. These are computed as the result of the multiplication of the embedding by three respective matrices, in which the weights were trained alongside the training process of the whole model.

The next step is the score calculation. For a given word, every word from the input sequence is scored against the current one. This score determines how related are the two words, how much focus should be placed on that other word when encoding the one the model deals with at that time. The score is calculated by taking the dot product of the current query vector with the key vector of the given word being scored.

For example, if the self-attention when being calculated for the word at the first position, the first score is the dot product of $\mathbf{q_2}$ (the query vector for the second world) and $\mathbf{k_1}$ (the key vector for the first word), then the following one is the dot product of $\mathbf{q_2}$ and $\mathbf{k_2}$ and so on. After these operations, the scores are divided by the square root of the key vectors dimensions. This is done for the stabilization of the gradients. In the referenced work this number is 8, as the square root of 64. The softmax operation is then applied to the given results for the normalization of the scores. These softmax scores refer to the magnitude of the impact these original words have at the position being calculated. The highest score is usually given to the word itself, although the other scores are important too, since they may be relevant in a given context.

After the scores are calculated, each value vector is multiplied with the appropriate score. This step results in the more relevant set of words standing out and

the irrelevant ones diminishing (they are multiplied by a very small number). Finally, all these value vectors weighted by the calculated scores are summed. The result of this sum is the output of the self-attention layer for the given position (the given word).

As mentioned earlier, these calculations are done in the form of matrix operations for the sake of better efficiency. The embeddings are joined into a matrix, which is then multiplied with the query, key, and value weight matrices, which have their elements set during the training of the whole model. The results are the query, key, and value matrices corresponding to the input vectors. The above-described steps in this section are expressed by this formula for the calculation of the attention function:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V} \tag{1.4}$$

Scaled Dot-Product Attention

The authors refer to the above function as "Scaled Dot-Product Attention" Figure 1.4. In summary, the inputs are queries, keys (both with dimension d_k) and values (with dimension d_v). The dot products of the queries and keys are calculated than scaled (divided by $\sqrt{d_k}$, the softmax function is applied on the result representing the wights it is multiplied by the values. Since they computed the attention function on a set of queries simultaneously, for efficiency, they are joined into a matrix $\mathbf{Q}$, as indicated on the visualization too. The keys and values are also joined together into the matrices $\mathbf{K}$ and $\mathbf{V}$.

Multi-Head Attention

Another feature of the transformer model is that not a single attention function is used, but multiple ones are used in parallel. This is implemented in the mechanism called by authors as "multi-headed: attention, which is part of the self-attention layer. Its visualization is provided by the referenced article, also shown in Figure 1.4.

The proposed design does not use a single attention function, but rather uses h (the h is the number of parallel attention functions also shown in Figure 1.4) different linear projections (learned during training) of the queries, keys, and values. These linearly projected variants are sent into the attention function in

parallel. The given results are then concatenated and projected again yielding the final result of the Multi-Head Attention.

As analyzed by [2], this design provides an improvement of the performance in two ways. Firstly it enables the model to "focus" on different positions in a sentence, providing a better "grasp" of the concept.

Secondly, the parallel attention heads give the model the ability to get information from multiple subspaces of representations. The model introduced by the authors uses different sets of query, key, and value matrices which were randomly initialized before the training and were trained to result in a nonequal set of matrices. These sets project the embeddings to "a different representation subspace".

The encoders and decoders contain a fully connected feed-forward neural network over the attention sub-layers which is applied to every position (of the vector derived from the input sequence at that position). It contains a ReLU activation between two linear transformations. The model also uses learned embeddings for the conversion of input/output tokens to vectors of the needed size. A linear transformation is used along with the softmax function to retrieve the probabilities of the predicted tokens.

An important aspect of the transformer model is, that it does not contain convolution nor recurrence. However, it has to take into account the sequential nature of its input. For this reason, the authors have decided to include some information about the token positions of the input sequence. They encoded the positions of the input embeddings both for the encoder stack and decoder stack at their bottoms. These encodings can be chosen in many ways, as referred to [15] by the authors. The sine and cosine function was used, with different frequencies, to encode the positions, since this method have resulted in similar success when using learned positional embeddings according to the authors [43]. Here is the functions calculating the positonal encodings:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

In the above formula these are the parameters: i being the dimension and pos being the position. The dimensions of the positional encoding conform to a given sinusoid. The authors chose this function because they hypothesized that it may allow the model to learn to attend by relative positions, because for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

The authors also suggested, that the self-attention mechanism could benefit the interpretability of the model. They analysed attention distributions from their models came forward with an example Figure 1.5. It did not only show how individual attention heads learn a different tasks, but some appeared to exhibit a functionality related to the syntactic and semantic structure of the sentences.

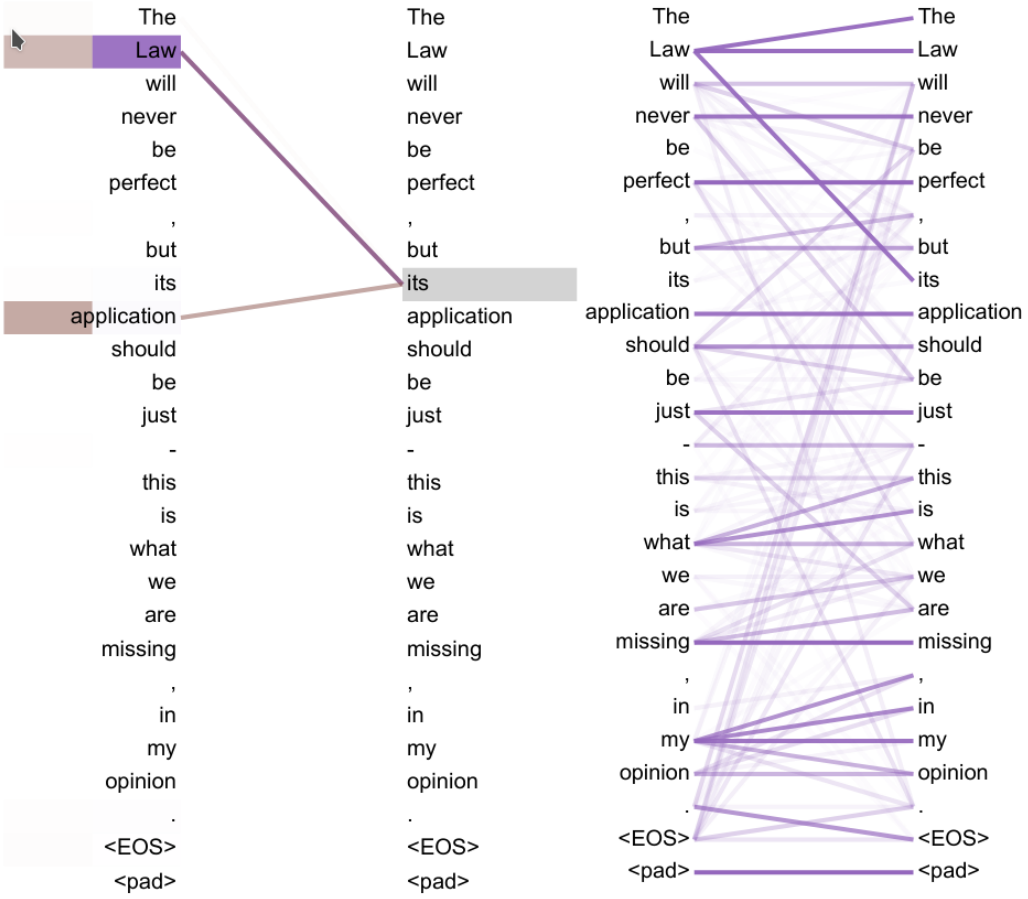


Figure 1.5: Two attention heads from the layer 5 of 6. Indicating anaphora resolution. Right: Full attentions for head 5. Left: Isolated attentions from just the word 'its' for attention heads 5 and 6. The example and its visualization is from [43]

1.5 BERT

The BERT deep learning model (BERT stands for Bidirectional Encoder Representations from Transformers), was introduced in 2019 [11]. The authors designed it to pretrain deep bidirectional representations from an unlabelled text using left context and right context at every layer. It can be fine-tuned with an additional output layer for a large variety of tasks. The Bidirectional Encoder Representations from Transformers (BERT) natural language processing model has achieved state-of-the-art results on

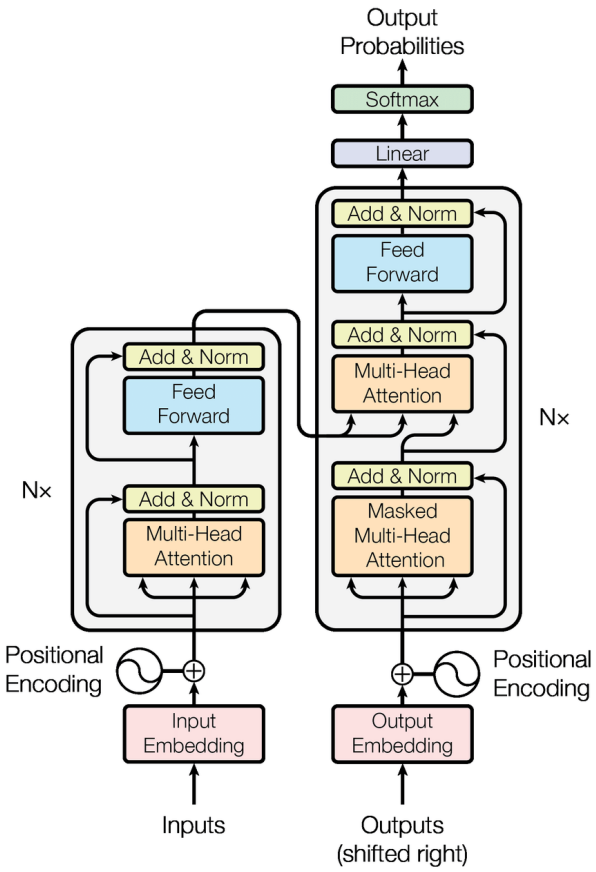


Figure 1.6: The architecture of the Transformer - model from [43]

many tasks becoming one of the leading models in natural language processing in the recent years [33].

The BERT model has two important qualities. One is that it is based on a deep Transformer encoder network, what enables the efficient processing of text inputs using self-attention. The other is its deeply bidirectional feature, meaning that it utilizes both left and right contexts of a text passage in all layers.

The BERT model was pretrained on two tasks. One was language modelling (where some percentage of the tokens were masked and the model was trained to predict them from the context) and the other was sentence prediction (predicting if a given sentence was probable or not following the first sentence). The pre-tained weights were released by the authors of the model, enabling the public to use them with little effort [2].

Those pretrained models come in two sizes. The Base model has 12 layers with a transformer, with the hidden size being 768, 12 sef-attention heads at every layer resulting in 110 million parameters. The second one is the Large model with 24 layers, with a hidden size of 1024, 16 self-attention heads. This variant has around 340 million parameters.

The BERT model was trained on multiple languages since it made its first appear-

[2]https://github.com/google-research/bert

ance, from which the **English BERT** is the most relevant for us in this work. It was initialized with weights pretrained on the so called BookCorpus, which is a dataset prepared from more than 11 thousand books of different genres and more than two million words from texts of the English language Wikipedia.

Currently many pre-trained variant and derivative models are available for many use-cases. One of those is the NER-BERT [27], a model fine-tuned for entity tagging. We used this variant our first experiment. Other variants are RoBERTa, ALBERT or ELEECTRA, each of them having specific advantages. Those are improvement on the masked language modeling (RoBERTa), parameter efficiency (ALBERT) and a comparatively good efficiency even on a small scale (ELEECTRA).

BERT embeddings

Embeddings are numerical vectors that in some sense capture the meaning of words. For example it is often the case, that similar words have numerically similar embeddings. They are needed so that a model can operate on them with mathematical functions. As BERT also works with the embeddings of given words, it has to get them from somewhere before transforming and passing them to the next layer. The BERT models input embedding is created as the sum of three different embeddings as seen on Figure 1.7.

These three initial embeddings are the so called token, segment, and position embeddings. Bert also uses special tokens such as the classification, separator and mask token with each of them having their token embeddings. The token embeddings are created from word tokens which are produced by a method called WordPiece tokenization. This method sometimes separates one word into multiple subwords with their corresponding embeddings.

The word piece tokens are later converted into a vector (with the length of 768 in the case of the base model) by the token embeddings layer.

The segment embeddings denote to which sentence does a subword belong, when the BERT model is dealing with more sentences (it can be fed two sentences, where zeroes are assigned to the first one and ones to the other, as segment embeddings).

Finally, the position embeddings represent the position of a token in the input sequence.

BERT structure

In this paragraph we overview the structure of the Base BERT model from a high level. The vizualization of the the $BERT_{BASE}$ archiecture can be seen at [32]

It is composed of 12 transformer layers stacked on each other, with each of them

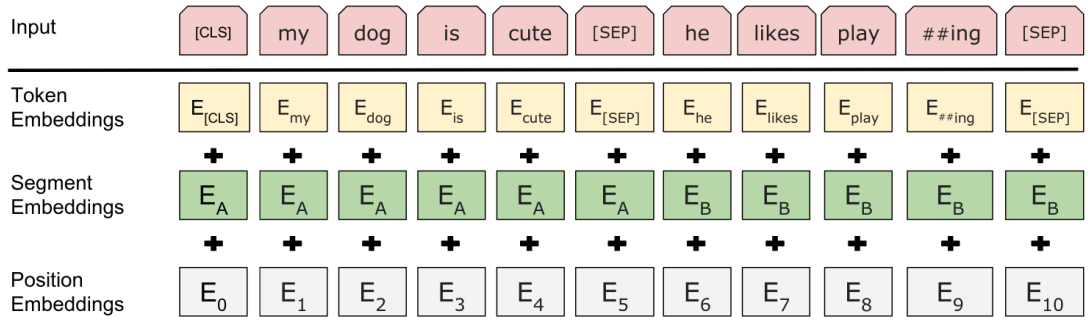


Figure 1.7: The representation of the BERT input. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. The image is from the article introducing BERT [11]

having 12 attention heads. When processing a sequence of tokens, the tokens are embedded into a learned embedding vector of length 768. Those embeddings vectors are then consecutively transformed while passing through the layers of the model.

This transformation process of an embedding in a given layer can be described in the following steps.

At first, the a set of linear projections (which were learned by the BERT model during its training for every layer) is applied on the input embeddings. The results are three vectors of a length 64, that are called key, query and value (often only referred to only as K, Q and V).

After the first step, these three types of vectors are processeed by the self-attention head, resulting in other vectors of length 64 for every triplet of key, query and value. This process is the one we described in more details and shown on a cited figure 1.4. It is important to note, that the self-attention head processes the whole input sequence thus making BERT "aware" of the context.

As the BERT model uses multi-head attention, it utilizes 12 different linear projections creating 12 different key, query, value triplets, each corresponding to a unique self-attention head, each of them relating the tokens to each others in a different way.

The 12 outputs are than concatenated, and the result's linear projection is summed with previous embedding and after being normalized, the sum is sent through a feed-forward layer and summed with a previous state. In these steps the embedding is transformed, and ready to be sent to the next layer, unless being the last one.

1.6 Natural Language Processing

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural

language.

Natural language processing, often shortened as NLP, is a subfield of linguistics, computer science, and artificial intelligence. In this work we are mostly concerned with the interactions between machine learning and human language. We will also refer to human language as natural language.

We can vaguely define Natural Language Processing (NLP) as the automatic software manipulation of natural language, speech and text alike. More precisely, we can look at NLP in the context of given tasks or problems we want to deal with. There is a very large variety of these possible tasks, hence we mention only a few examples while while emphasising those relevant to this work:

Text processing

Word segmentation (Tokenization)

Separation of a text into words or tokens. In case of some artificial languages, for example programming languages such as Python, C++ or others, it may be a more or less straightforward task. The code is tokenized and ready for further processing. In many cases natural languages present a trivial challenge too, when the boundaries of the words align with those of the words in text. However, for some languages it is not the case. For some tasks this approach is not satisfactory, since the words may contain multiple tokens. These issues may arise from the nature of the language (e.g. Chinese, Japanese etc.), the task, or implementation. Some words not being in the set of possible tokens of the tokenizer may result in the splitting of words not known, to multiple subwords. The procedure of tokenization is sometimes also used to create a bag of words (BOW) for example for document classification [3]. The specific tokenization of words used for the BERT model is described in the BERT section 1.5 along with the BERT architecture.

Morphological analysis

Stemming and Lemmatization

Two different word normalization procedures are often used for pre-processing words or whole documents. Both of these techniques are a form of reduction of derived or inflected words to their base forms. (e.g. "observer" to the root form "observe"). The difference between Stemming and Lemmatization is, that while the former uses a set of rules, the latter works with a dictionary to return the so-called lemma (dictionary form).

[3] https://machinelearningmastery.com/deep-learning-bag-of-words-model-sentiment-analysis/

Morphological segmentation

This process has two parts: firstly the word is separated into its morphemes, secondly, the morphemes are identified by their classes. The difficulty of this task varies a lot between the languages. While in the case of the English language the morphology is fairly simple, other languages may present a greater challenge regarding the morphological segmentation. The agglutinative languages are good examples since in their case a simple dictionary method is not possible.

Part-of-speech (PoS) tagging

Part-of-speech (PoS) is a grammatical category of lexical items (words) that have similar grammatical properties. Words being in the same part-of-speech category often show similar behavior regarding their syntax, having similar roles in the grammar. In the English language, we can list words into these main categories: noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection, numeral, article, and determiner. It may be noted, that often more granular categorization is used, creating subcategories for the aforementioned ones.

The PoS tagging, or grammatical tagging, is the task to determine the part of speech for each word in a sentence or more generally in a whole corpus. Since many words may represent multiple different parts of speech, a naive dictionary approach is not sufficient. Many words, especially common ones, can serve as multiple parts of speech. For example, "cool" can be an adjective ("cool lemonade") or verb ("cool down the reactor"); or the word "out" can be of multiple different parts of speech. Initially, PoS tagging was a manual process. However, with the advancement of computer science, the effort of algorithmization of this process started to increase with the growing computing capabilities.

At first rule-based methods were used, like the Brill's tagger [6]. Later a stochastic approach was taken too, using for example Markov models or dynamic programming for determining the PoS tagging. Later on, machine learning models were utilized reaching accuracy above 97% [41]. Despite the high rates of correct tagging, there is still room for improvement and experimentation. Possible new approaches were made possible as evolutionary computation was introduced [36].

Syntactic analysis

There are many sub-problems related to syntactic analysis. Some examples are sentence breaking, parsing, and grammar inference (grammar induction). The first one is the process of finding the boundaries of sentences in a text. It is relevant, since some punctuation characters may show up in a text not only indicating the end of a sentence but also as abbreviations, fractions, ordinal numbers, and so on. Grammar inference

or induction is a category of processes that try to "learn" a formal grammar. The goal, in this case, is to generate a formal grammar that describes the syntax of the given language the best.

Parsing

Parsing is one of the most relevant task types from the category of syntactic analysis regarding this work. Its objective is to find the parse tree of a sentence, to be able to answer questions concerning its grammar. One of the problems when parsing a sentence is the possible ambiguity. A sentence may have a very large number of possible parse trees. Even though most of these parses are meaningless, their algorithmic distinction is not so obvious. For this reason, straightforward deterministic solutions are not viable. We distinguish two main categories of parsing: dependency parsing and constituency parsing.

A constituency parse tree breaks a text into sub-phrases. The leaves or terminals in these kinds of trees are the words of the sentence themselves. On the other hand, the inner nodes, or non-terminals of the tree represent different phrase types. The edges which represent the inclusion of a super-category are not labeled. In our example 1.9 for the sentence "This tree is illustrating the constituency relation." we see a few types of phrases: S - sentence, NP - noun phrase, VP - verb phrase, V - verb, D - determiner, N - noun. These types are of the main categories, but larger granularity is possible too, with more specific types, elements extended by additional information and the possible components of these trees (dependency parsing trees too) may vary from language to language.

In the case of dependency parsing, every node in the tree graph represents a word, with the edges showing the relationships between those words. Words grammatically dependent on a word are represented as child nodes of the nodes corresponding to the latter. The example presented 1.10 is a dependency parse tree of the sentence "This tree demonstrates the dependency relations.".

Lexical semantics

Lexical semantics studies the meaning of words, as described by [13], from where we introduce some relevant definitions. It investigates the correlation between the meaning of the lexical units and the structure of the language or syntax. The areas of lexical semantics include the study of semantic relations among the elements of a vocabulary or the structures of the words themselves. The basic blocks or units, also called syntactic atoms, of the analysis in lexical semantics are not only the words in a language (the lexicon) themselves, but also sub-words such as affixes (prefixes, post-fixes, and many

others), but also extend to compound words and phrases. In other words, lexical units can stand on their own, for example in the case of root words or as parts of compound words, in which case they may be referred to as free morphemes. On the other hand, they may be attached to other units such as prefixes and suffixes. These are called bound morphemes.

Named entity recognition

Named entity recognition (NER) deals with the task of mapping given words or groups of words from a text to defined semantic types, for example, a location, organization, event, or person. In some natural languages, such as English, capitalization may help, but in others (e.g. German) it is of less use, therefore this task can not be easily done with simple mechanical rules. Even in a language, such as English, capital letters do not always indicate named entities. Some other languages do not even use the concept of capital letters (Chinese for example). As this task logically implies the need to automatize, more sophisticated methods were introduced. The context of a word needs to be taken into consideration when determining if it refers to a named entity. Named entity recognition is often the basic block of applications dealing with natural languages, hence increasing its importance. Among others, NER is employed in machine translation, question answering, and summarization.

Sentiment analysis Sentiment analysis is used to retrieve subjective information or opinions from a given text, from a short comment or review to a longer one such as documents. It may be used for public opinion scanning, on social media, or aimed at the personalized advertisement and other forms of marketing.

Among other common NLP tasks are word sense disambiguation, or tasks dealing with relational semantics (which often deals with the semantics of whole sentences in relation to each other). The first one tries to tackle problems arising from the ambiguity of natural languages. It asks the question, which is the most likely meaning associated with a word given a specific context. We can list many problems from the category of relationship semantics. Some examples are relationship extraction or semantic parsing. Relationship extraction means given a specific corpus of text, the identification of relationships among named entities (e.g. which street is in which city). Semantic parsing is a task aimed at producing a formal representation of the semantics of a sentence or a whole text. This is a broad task often requiring the execution of many sub-tasks, such as semantic role labeling (assigning labels to words or phrases in a sentence indicating their role in the semantics of the given sentence), word sense disambiguation.

1.7 Explainabe AI (XAI)

Recent years have brought significant improvements of the state-of-the-art models in different fields of machine learning and more precisely in deep learning too. However, with the growing complexity of models, the interpretability decreases.

The broad term explainable artificial intelligence refers to such a type of AI that produces results in a manner somewhat comprehensible to humans. It does not have "black box" like attributes such as many models in machine learning, or at least it provides some extra means to interpret its "behavior" giving clues why it "made" a given decision.

XAI may be relevant due to legal reasons and in the law itself, as explanations of algorithmic decisions may get increased trust due to their increased explainability, and sometimes their correct interpretation may be legally required [12]. The user experience can also be improved by XAI, when no legal requirements are made regarding the explainability, increasing the trust in a given product. XAI is often used to explain the current or past decisions of a system or may try to provide explanations for possible decisions in the future. The relevant literature often distinguishes between white-box and black-box algorithms, which has a big relevance in machine learning too. In machine learning, we can consider those models to be white-box, that produce their results in a way understandable by the experts in a given field. And those models, which are hard to explain or understand (their inner workings, dynamics, or their decisions), oftentimes even by the relevant experts, we may refer to as black-box models.

Explainable AI for Natural Language Processing

So far we have described various black-box models. From their nature arises the need for explainable AI. In the following survey [10], in 2020, the authors made an overview of Explainable AI (XAI), focusing on Natural Language Processing (NLP). As it was noted in the survey, the natural language processing systems have often been based on techniques that are inherently explainable. Into this category belong the decision trees, logistic regressions, and hidden Markov models. In the recent past, however different black-box techniques are more and more popular end relevant. Such is deep learning using different language embeddings as features. These methods while increasing the quality of models and results too, come at a price of interpretability.

Probe

As we have mentioned, neural network models are often regarded as so called black boxes. For the reason on getting some insight into the inner working of such models the idea of probes was proposed [1], where classifiers were trained for the purpose of

understanding the dynamics and roles of the intermediate layers of some chosen black box models. In the mentioned work linear classifiers were used referring to them as probes. The training of these probes were entirely independent from the examined models themselves. As this work observed some interesting properties on models for computer vision, our ambitions were different. The objectives of our works aimed at natural language models, more specifically the BERT model.

To move in this direction we got inspiration from [9], where the authors introduced 10 different probing tasks capturing different linguistic features of sentences, across different encoders with different training settings. Their approach was to use multi-layer classifiers, in "the case relevant properties are non-linearly encoded in the sentence vectors." This motivated us to proceed in a similar manner, also using multi-layer classifiers.

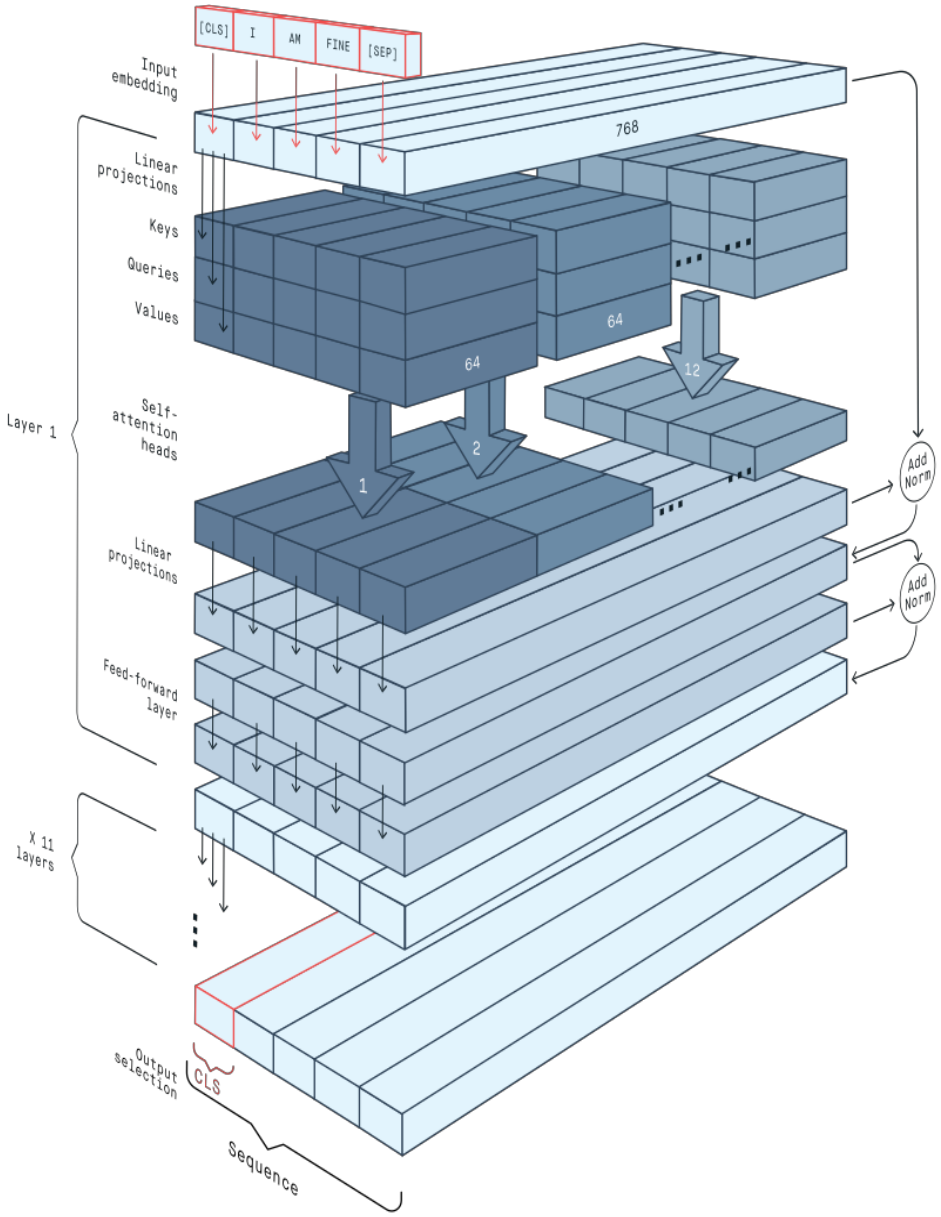


Figure 1.8: The vizualization of BERT model from [32]. The first layer with the transformer is visualized in full size while the other layer are being hidden for brevity.

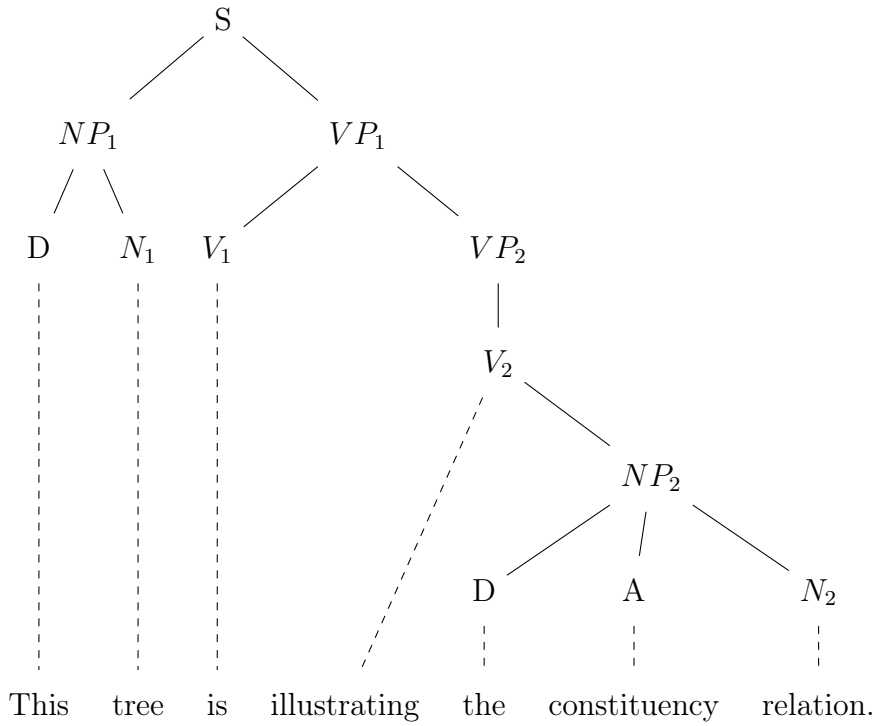


Figure 1.9: Constituency parse tree

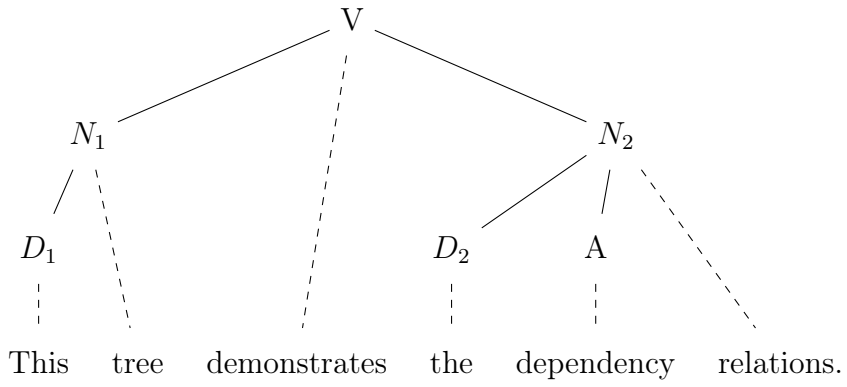


Figure 1.10: Dependency parse tree

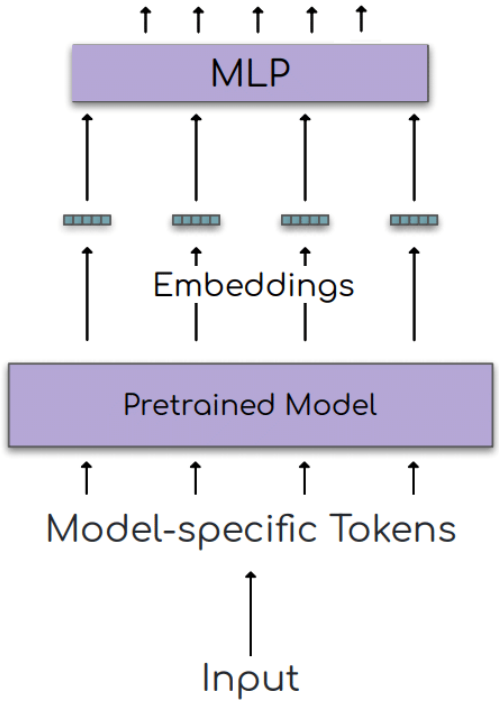


Figure 1.11: The vizualization of the probing technique on a models embeddings from [23] The investigated model is in a static state (not in training mode) while the probe tries to predict the chosen information from the inner representations.

Chapter 2

Experiments

In this chapter, we are going to introduce the data we used for our experiments. For the two of those experiments, we used a different set of data. The first one is a set of sentences with no labels, while the other one is a set of sentences with per token labels.

2.1 Cased sentences

In linguistic typology, the order of the subject, verb, and object in a sentence is an important classificator of languages. The different languages can be categorized according to the order these words observe in an unmarked sentence (a sentence where this order does not differ from the typical one in the language).

Words belonging to these categories make up the basic structure of a sentence. The English language is dominated by the S-V-O order for example (example sentence: Tom ate a banana.). This trio makes up the structure of sentences to which other sentence parts align too. For the Verb-Subject and Verb-Object dependency analysis between tokens, we created a labeled one from the Kaggle dataset from [28],

The original dataset we used for this experiment was an unlabelled one, containing unique cased sentences without punctuation. It consists of 101103 English language sentences with a size of 6MB. The file format is `csv`, which stands for comma-separated values. Each row in the file contains the sentence number (starting from zero) and the sentence separated by a comma. The rows are separated with the end of line character. Since we needed sentences tokenized with grammar tree tokens for our experiment, we had to use an automatic parser to do the job, to compensate for the lack of a large enough dataset labeled according to our needs. To generate sentences with dependency parse tree tokens, we decided to use the SpaCy [19] open-source software library, designed for advanced natural language processing. This library was written in the Python [42] and Cython [4] programming languages. The SpaCy library enables us to conveniently download from their codebase different natural language processing

pipelines. There are multiple choices for different the English, with different size and effectiveness [1]. These are the small, medium, large, and the largest ones respectively. We have chosen the small version for computational and time effectiveness, which is reported by the authors to still have a reasonable accuracy on the needed task. For the labeled dependency parse labeling it is 90% [18]. For the comparison, the largest model has better accuracy by 5 %, but its size is 10 times larger.

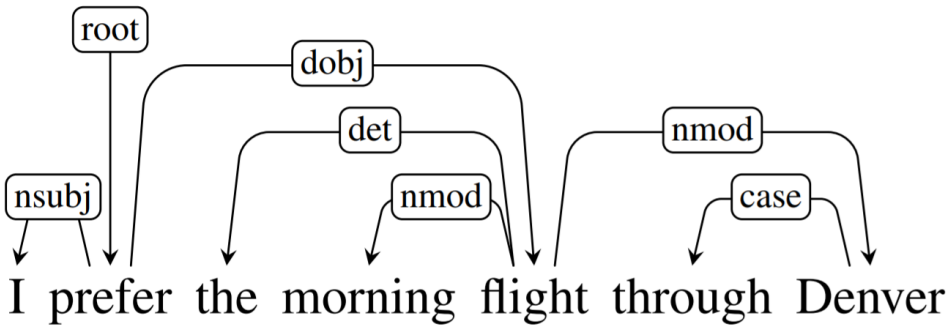


Figure 2.1: A sentence's dependence structure[2]

After reading the raw sentences and having them tokenized, we saved the result in a text file. For a given sentence, the resulting data was saved in the following form: an outer list containing these elements: the sentence itself, the list of its tokens, the list of the positions of the subject related tokens, the list of the subject related tokens, the list of the positions of the verb related tokens, the list of the verb related tokens, the list of the positions of the object-related tokens and the list of the object-related tokens. We also saved generated data-set to a text file in such a format, which was easily readable by another function, when needed. In this case, the elements of the outer list were concatenated on the hash (#) character, and the elements of the inner lists were joined on the underscore (_) character, since these two characters were not in the data-set, and enabled a simple extraction for the function reading the data for later usage. It was convenient to save those data since for the next time we ran the script, we could read the data from the saved file, saving the time needed for the pipeline to process the sentences one by one.

[1] en_core_web_sm, en_core_web_md, en_core_web_lg and en_core_web_trf source: https://spacy.io/models/en

[2] https://www.analyticsvidhya.com/blog/2021/12/dependency-parsing-in-natural-language-processing-with-examples/

2.2 BERT probe dataset

From the subject-verb-object triplets, we could have assumed that the trained BERT model has inner representations for verbs and nouns. However, the analysis of the attention-heads was insufficient to confidently draw such conclusions. The inner representation can not be interpreted in themselves. However, the higher layers of the model are able to "interpret" them with non-linear transformations, therefore we decided to substitute/approximate those non-linear transformations with MLP probes. In this section, we describe the dataset we have chosen for our probing task.

2.2.1 CoNLL-2000 dataset

We used a data corpus [40] created from sentences with separated words each labeled with a part-of-speech tag and chunk tag. The description of the dataset indicates that the dataset was created for chunking tasks, whereas "text chunking consists of dividing a text in syntactically correlated parts of words." An example sentence is given by the authors: **"He reckons the current account deficit will narrow to only # 1.8 billion in September"**, which is chunked in the following way: **[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September] .** The description further states, that text chunking is an intermediate step towards full parsing, which was a shared task for CoNLL-2000. The training and test data for this task is the one we use, but not for parsing, but for the PoS tags. The corpus contains 211727 tokens for the training data and 47377 tokens for the test data. The tokens belonging to different sentences are separated by an empty line.

The PoS tags used can be found in this table 3.5. Those tokens, that did not fit the tags (brackets, apostrophes...) were assigned the value 0.

2.2.2 Token processing

We read the dataset mentioned in the previous subsection with a Python script. We separated the words belonging to different sentences into a class together with their PoS and chunk tags.

In this experiment, we were using the BERT base cased model and tokenizer from the PyTorch-Transformers library [3] to create the embeddings from our dataset containing cased PoS tagged sentences.

We iterated over the sentences while tokenizing the words with the pre-trained BERT tokenizer. The token sequence was then passed into the model, and as passed through, we saved the input embedding and the output embeddings of all the 12 layers

[3] https://pytorch.org/hub/huggingface_pytorch-transformers/

of the given BERT model. When a word was separated into multiple tokens, we saved the embedding for the first token in the word. All these data (sentence number, word number in sentence PoS tag, chunk tag, embedding) were saved. Due to the fact that one run produces all the embeddings and a large number of words, this process had to be done in multiple parts, since all the embeddings could not fit into the memory. Then all the metadata was saved into a file, and the embeddings into separate files, per layers they were extracted from later. One file containing the embeddings and the other data (raw subword, sentence number, token number, and so on) of the training set having a size of more than 2 GB, we used the PyTorch's save function to save the embeddings, utilizing compression, (this time excluding other data,) in the form of dictionaries, which reduced the size of the used files during the experiments. With these methods, we created a training set for each layer of the size of around 812 MiB, and from the test data, the embedding files reached the size of almost 190 MiB. In total, the size of data we worked with exceeded 13 GiB.

2.3 Dependency evaluation

With the first experiment we looked at the individual heads in a given model. The experiment we conducted was structured in a linear manner. At first we iterated over the raw data-set, applied the spaCy small [19] pipeline to get the parse tree labels of the sentences. We have also saved this results in a separate file. There was the possibility to chose the large pipeline, however we have not used it since its accuracy is not significantly better on the task we needed it for and its size is almost 80 larger from the small version we used. After the initial phase we applied the `dslim/bert-base-NER` model on the initial sentences to produec the outputs of the attention-heads. This version is a fine-tuned BERT model [11] for Named Entity Recognition. More precisely, this given model is a bert-base-cased model fine-tuned on the English version of the CoNLL-2003 Named Entity Recognition dataset. [40]. The authors trained it to recognize four entity types: location (LOC), organizations (ORG), person (PER) and Miscellaneous (MISC). We have chosen this model due to its accessibility, whereas it can be easily imported from the transformers Python library and is ready to use.

We also used the corresponding tokenizer with the BERT model of our choice. Since this tokenizer sometimes does not match its output with the used spaCy tokenizer, we used a library which helps to effectively find the alignment between BERT and spaCy produced tokens [38].

When applying the BERT model on the token sequences, we saved the attentions of the model. The attentions are a tuple of torch [8] tensors of the float datatype. The

number of elements in the tuple is the number of layers in the model, which is 12 in our case. These tensors are of this shape:
$batch\_size \times num\_heads \times sequence\_length \times sequence\_length$.
The batch size was 1 in our case since we were inserting one sentence at a time into the model. The number of heads is also 12 in this particular model, and the sequence length is the number of tokens produced from the input sentence (including the two special tokens at the start and at the end of the token sequence marking the sentence margins). For each head we extracted the corresponding tensor slice representing the attention weight matrices between the tokens. Since the spaCy generated tokens were labeled with their parse tree labels, we extracted those, which were the subjects of our interest, connected them to their corresponding BERT tokens, and if in the i-th layer the j-th head had a maximal weight between tokens corresponding to searched sentence units, we incremented a counter, which counted the hits at given positions. In parallel, we used another function for the same dependencies, counting the k maximal weights in a head, in our case setting k to be equal to 3.

Whether a given token corresponds to a sentence unit was computed by the slightly modified version of a function from an SVO (subject-verb-object) extraction script [4]. When we iterated through all the sentences, we saved the frequency for the given heads for the maximal attendance to a dependency relation, and the number of those relations being present in a sentence.

2.4 Non-linear per-token probing

In the second and more significant set of experiments, we used a multi-linear perceptron (MLP) as a probe. We trained it on the embeddings of tokens from PoS tagged sentences measuring the accuracy (in the case of single-target multi-class classification being equal with micro-averaged F1 score). For creating the multi-layer perceptron as the probe we used the `nn` class from the PyTorch library [5]. The MLP class in our is the one implementing the neural network for the probe itself. The class named data_set inherits from the PyTorch Dataset class, containing the data. This structure of the codebase was inspired by [6] borrowing some of its elements. We used the DataLoader class which loaded the data from the data_set class to the network for training and evaluation. The initial data (embeddings and tags) however was read from the disk, where it was saved (saving a lot of time for every run) as a .pt file by the functions in the `data_handler.py` script. This script contained auxiliary functions for data ma-

[4] https://github.com/Dimev/Spacy-SVO-extraction/blob/master/main.py
[5] https://pytorch.org/docs/stable/nn.html
[6] https://colab.research.google.com/github/bentrevett/pytorch-image-classification/blob/master/1_mlp.ipynb

nipulation, storage, and even graph generation from the results of probings. The MLP was used either with two or one hidden layers throughout our experiments, with the non-linear ReLU layer. We experimented with different sizes of the hidden layer, batch size of training inputs utilized in one iteration, and number of epochs. The perceptron predicted a label for an input embedding by applying the Softmax function on the final layer and using Argmax to get the position of the most probable label. During the training phases, we used the cross-entropy loss function imported from the `nn` PyTorch library under the name `CrossEntropyLoss`.

Chapter 3

Results

In this chapter we will discuss the results obtained from our experiments.

3.1 Dependency analysis

The first experiment we conducted to look under the hood of the BERT model was an analysis of its heads in relation to the subjet-verb and verb-object dependency. We created a data-set with from unlabelled sentences using the spaCy library, with the small English pipeline. Then we applied the BERT model on the sentences, and examined the weights at the attention heads. We used two variations of the test. In the first one, we only looked at the position of maximum value in the attention head, and in the other one we looked at the k largest weights, while in our experiment k was set to 3. If the aforementioned maximal value was at the position of the attention between the controlled dependency, we saved the this information in a matrix indicating scores for the heads in layers and on given positions. It can be demonstrated with the following example. When examining the second layer, we look at the third attention head. We search for the position of the maximal weight in this head. Let us denote this position as (r, c). As we described in 1 , the heads can be considered as two dimensional matrices with the size of the input sequences token number on both dimensions. Lets consider we are examining the subjects dependency relation from the verb. If the r-th token is a subject related one and the c-th token is a verb related one, we increment the the r-th row on the c-th position in the score matrix, also incrementing the counter for the current relation being examined. With this counter we divide the score matrix in the end, getting the ratios how those heads are involved in the dependency. From these values we created the heatmaps 3.1. When examining the top 3 weights in the head, we used a similar approach, treating all 3 positions as equal. This may distort the overall picture a bit, but the tendency was the same. The heads at lower layers were better indicators of the dependencies. These results indicates, that indeed, some

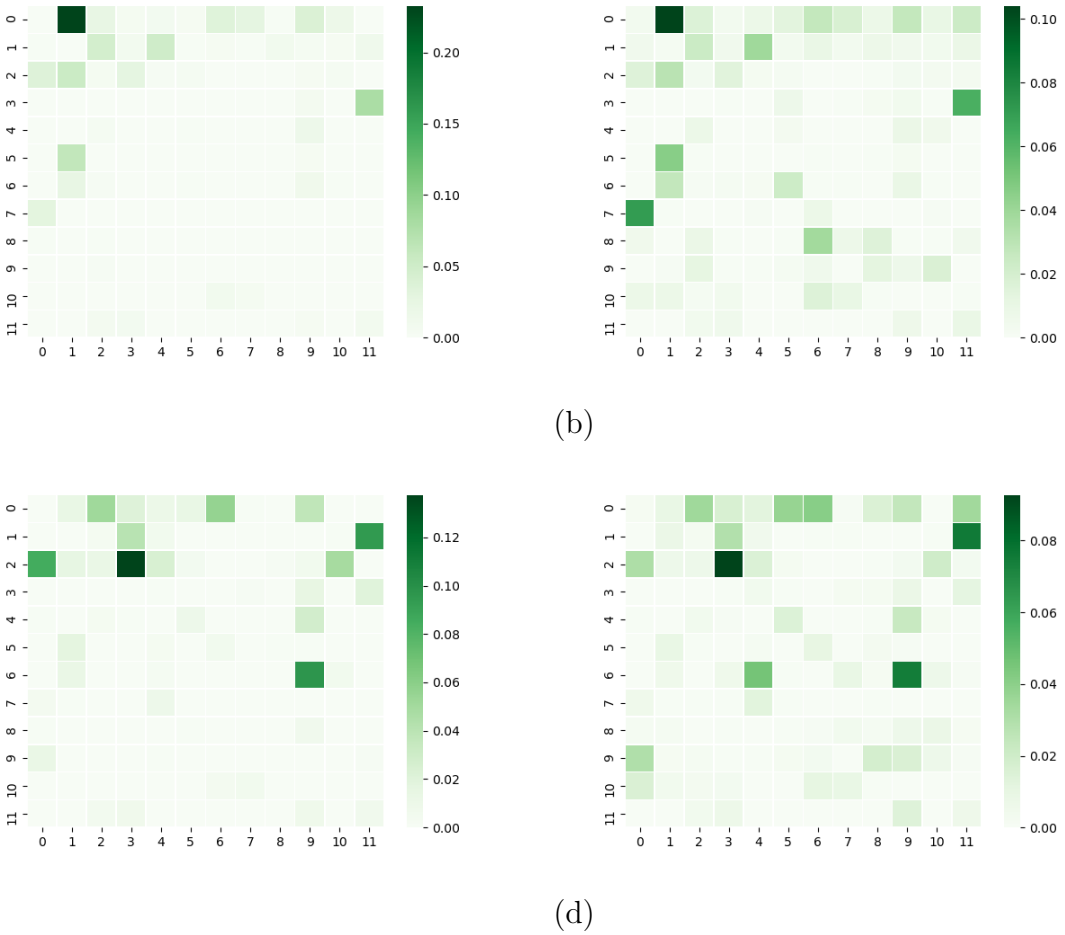


Figure 3.1: Heatmaps of maximal head attentions per given dependencies (a) subject-verb (b) subject-verb 3 largest per head (c) verb-object (b) verb-object 3 largest per head

parts of the BERT model can capture a given syntactic feature better than the others. However, this maximal weight method used in this experiment does not show to be a reliable enough indicator of the given dependencies. As shown in [22], higher complexity syntactic features are usually captured in the higher levels of the BERT model. This information with our findings indicates that attention weight does not fully indicate the tested dependency relations. Our results are also in line with the literature that a BERT model often does represent linguistic of syntactic features in more complex ways, as opposed to some cases, as int the example [34], where a single unit in an LSTM (long short-term memory) was pointed out as an indicator of sentiment. However, we also have to note the huge size difference in the size of the two models (BERT vs. the aforementioned LSTM), what may be a factor to this phenomena. The two compared the results of the two test variants and achieved very similar results.

The table 3.1 shows how the maximal (or top three) attentions in the heads correlated with the given dependency. Here the results are summed per layer, not distin-

layer	S-V	S-V%	S-V(3)	S-V(3)%	VO	VO%	VO(3)	VO(3)%
1	21257	39.69%	67932	26.65%	7088	20.61%	39773	23.04%
2	8056	15.04%	30395	11.93%	5048	14.68%	21190	12.27%
3	7708	14.39%	20516	8.05%	11585	33.69%	31341	18.15%
4	4693	8.76%	20137	7.90%	1296	3.77%	5266	3.05%
5	1257	2.35%	6741	2.64%	1477	4.29%	8122	4.70%
6	3625	6.77%	13121	5.15%	1045	3.04%	4494	2.60%
7	2326	4.34%	17735	6.96%	4126	12.00%	26065	15.10%
8	1742	3.25%	21325	8.37%	531	1.54%	3833	2.22%
9	305	0.57%	21467	8.42%	327	0.95%	4935	2.86%
10	269	0.50%	15708	6.16%	552	1.61%	14222	8.24%
11	888	1.66%	12836	5.04%	359	1.04%	7978	4.62%
12	1433	2.68%	6960	2.73%	957	2.78%	5414	3.14%

Table 3.1: The results shown on the heatmaps on Fig. 3.1 summed per layer

guishing the different heads, showing that the largest dependencies were in the lower layers. An interesting development of the experiment could be to examine the type of subject-verb dependency captured in the heads, with larger granularity maybe better precision could be achieved.

3.1.1 Result enumeration

Here we note some basic statistical data regarding the experiment. From 101103 sentences, the spaCy parser extracted 53559 verb-subject dependencies indicated by the maximal weight in a head. For the three maximal weights in the head this result was 254873 verb-subject dependencies. For the verb-object dependencies, the maximal attention indicated it 34391 times, and for the top three weights, the maximal attention weight indicated this relation 172633 times. This was possible due to the fact, that the BERT tokenizer oftentimes separated a single word into multiple tokens.

3.2 Probing

The probing, sometimes also called diagnostic classifiers, is the usage of encoded representations of some model to train a classifier on a probing task in question. In the context of NLP it is often designed in a way to capture some linguistic features. When the probe performs well on the given task we may assume that the inner representations encoded the linguistic phenomena we investigated.

We have decided to probe for the part-of-speech of words in sentences. As a probing mechanism we have decided to use a multilayer perceptron.

layer	$epoch_1$	$epoch_2$	$epoch_3$	$epoch_4$	$epoch_5$	**test**
0	84.35%	85.47%	85.86%	85.84%	85.83%	85.24%
1	88.62%	89.84%	90.44%	90.73%	91.00%	91.03%
2	91.60%	92.93%	93.41%	93.67%	93.96%	94.11%
3	92.26%	93.40%	93.86%	94.20%	94.26%	94.42%
4	92.88%	94.04%	94.44%	94.64%	94.84%	94.81%
5	92.98%	94.22%	94.70%	94.94%	95.04%	95.02%
6	92.92%	94.45%	94.82%	95.04%	95.24%	**95.29%**
7	92.59%	94.01%	94.50%	94.80%	95.03%	95.04%
8	92.17%	93.92%	94.36%	94.66%	94.81%	94.88%
9	91.65%	93.40%	94.12%	94.37%	94.48%	94.50%
10	90.96%	92.88%	93.73%	94.02%	94.23%	94.26%
11	90.57%	92.53%	93.29%	93.60%	93.88%	93.96%
12	87.73%	90.68%	91.64%	92.47%	92.73%	92.84%

Table 3.2: Validation accuracies of epochs compared to the test accuracy

In the first settings, we used two hidden-layers of size 50, a batch size of 100, and trained the model in 5 epochs. We were probing the embeddings of the BERT model (bert-base-cased). The results can be seen in 3.2. The first columns denote the validation accuracy at a given epoch and the last column contains the accuracies on the test set. As we can see, the best result was achieved on the sixth layer, for which reason we decided to run further probing tests on that specific layer.

In the next batch of experiments we tested different hyperparameters for which we can see the results in 3.3. From these results we can see the accuracy decline when increasing the batch size. We also observe that an increase in the epochs of the model training have a small impact on the final accuracy.

3.2.1 Probing with masking

Here we describe the probing test executed on masked [1] embeddings. In these settings we applied a particular mask on the embedding vectors and probed those masked embeddings. Here we applied the probe on the the embeddings from the sixth layer of the BERT model, due to the fact that at that particular layer the probe has shown a strong accuracy as shown in the table 3.2.

A trivial example is when we fully masked all the embeddings, resulting in zero vectors of size 768. The first experiment of this kind was done with an MLP with one

[1] We multiplied the elements in the embeddings with the mask elements. The i-th embedding element with the i-th mask element. We did this with every embedding of all the tokens. The mask contained ones and zeroes only.

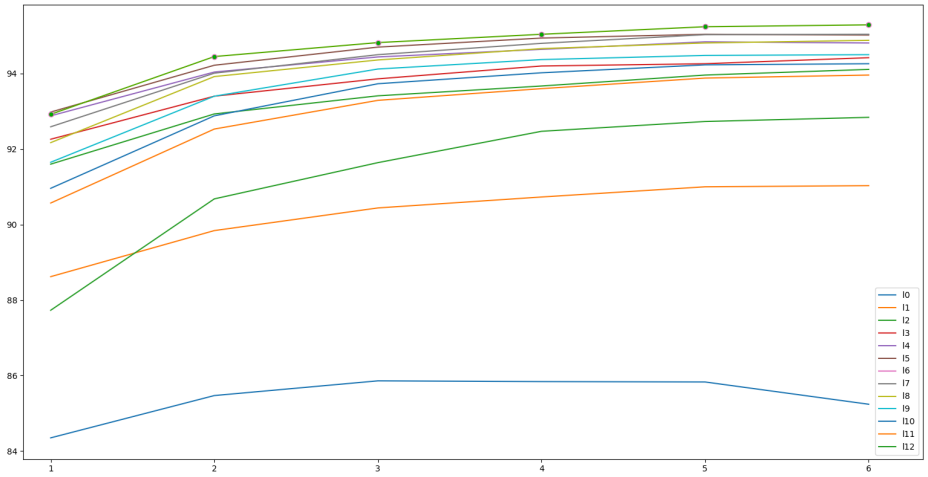


Figure 3.2: Validation accuracies of the probes trained on embeddings on different layers. The last points from the right show the test accuracies of the trained probe. As visible, the best performance was achieved on the sixth layer.

hidden layer of size 10 learning for 1 epoch only. The test accuracy, in this case, was 10.59%, indicating that the model was only guessing, but since the number of possible classes was 37, it meant that the predictions were not uniformly random. Since in that case the probability of a correct prediction should have been around 2.75%. The perceptron has most likely acquired a "sense" of the probability distribution of classes. In the next probing, we increased the size of the hidden layer to 50 from 30. Here the test accuracy has increased to 13.28%.

In the next set of probing tests, we increased the epoch numbers to 5. First, we trained a perceptron with a small hidden layer of size 10. At each epoch, the validation accuracy was near 13.40% and the final test accuracy was around 10.70%. When increasing the hidden layer size to 50, we have achieved a test accuracy of 13.34%. From these results we can assume that the limit of correct blind classification is a little above 13%, stemming from the distribution of chunk types in the dataset. This seems to be also supported by the fact that when increasing the hidden layer size to 200, the test accuracy even decreased a little, possibly caused by overfitting.

The next probing we executed was with a less trivial mask. We masked the first half of the embeddings and let the classifier learn for 3 epochs. This time we used a mini-batch of size 10 and used one hidden layer of size 30. Surprisingly, the validation accuracy at the end of the first epoch was 93.17% and increased throughout the run to 93.94% at the end of the second epoch and finally reaching 94.12%. This slight increase indicates that the model has learned to predict the PoS tags relatively fast, possibly due to the size of the dataset. The test accuracy was even better, reaching 94.28%.

epochs	*batch*	dim_{hid}	*acc*	*epochs*	*batch*	dim_{hid}	*acc*
1	10	50	94.70%	3	100	70	94.93%
1	10	70	94.91%	3	500	50	93.26%
1	100	50	93.14%	3	500	70	93.86%
1	100	70	93.62%	5	10	50	95.72%
1	500	50	84.50%	5	10	70	95.91%
1	500	70	87.80%	5	100	50	95.13%
3	10	50	95.50%	5	100	70	95.31%
3	10	70	95.62%	5	500	50	94.31%
3	100	50	94.85%	5	500	70	94.72%

Table 3.3: Test accuracies for different hyper parameters with two hidden layers of size 50 and 70

Due to the fact that the classification achieved such high accuracy, we theorized that the relevant information is captured more in the second half of the embeddings. For this reason, we repeated this task, but with a mask applied on the second half of the embeddings.

In these settings, the results were similar, from initial valiadation accuracy of 92.9% going up to the final validation accuracy 93.90%. The accuracy on the test set of the data was 93.5%.

With this test we could rule out the previous possibility, that the relevant information determining the PoS tag of a word in a sentence was mainly contained in the second half of the embeddings.

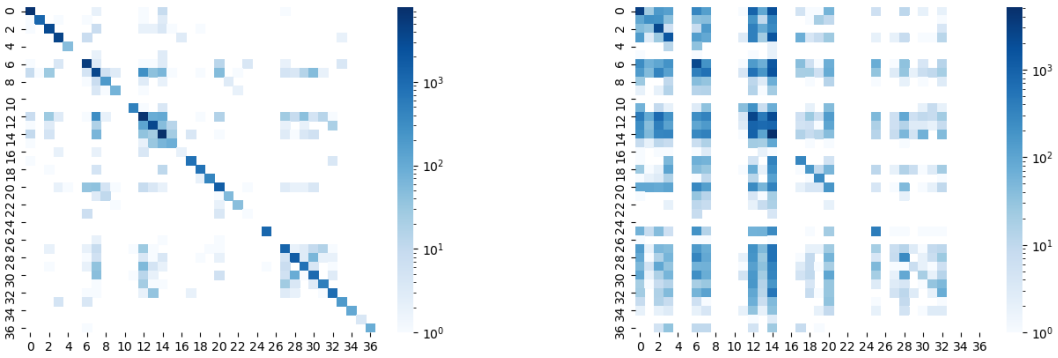
From these previous experiments, we concluded that the PoS correspondence in a given embedding is encoded more robustly and it is not concentrated in a small number of specific elements of the embedding.

Randomized masks After the previous probe tasks we decided to randomize the masks, which we applied to the embeddings. We compared how our probe performs on embeddings with a different number of elements set to zero. The results can be seen in 3.4 The first column shows how much percent of the embeddings are set to zero by the mask. The other columns contain the training and validation accuracies at the given epochs of the model training. In all the cases we randomized the positions of the zeros in the masks, which we fixed and applied on all the embeddings both in the training and test set. With this method, the probe had less and less information available from the embeddings. The last row in the table shows the results when the whole embedding was set to zero. In this case, the probe could rely only on the learned probability distribution of the tags in the dataset, as we did a slightly different probe in the previous experiment. This can be considered as a baseline of the performance of the

masked	tr1	v1	tr2	v2	tr3	v3	test
50%	86.91%	93.25%	93.66%	93.77%	94.16%	94.05%	93.82%
60%	85.18%	92.18%	92.82%	93.00%	93.43%	93.36%	93.12%
70%	82.17%	90.45%	91.34%	91.58%	92.07%	92.01%	91.43%
80%	75.96%	86.46%	87.53%	87.86%	88.53%	88.57%	87.82%
90%	59.49%	70.90%	73.06%	74.27%	75.41%	75.99%	72.08%
92%	55.65%	66.49%	68.75%	70.41%	71.33%	72.02%	67.37%
94%	50.45%	61.34%	63.20%	65.31%	65.78%	67.23%	61.80%
96%	42.55%	50.30%	52.83%	54.74%	55.40%	56.25%	51.22%
98%	33.68%	38.33%	40.25%	41.42%	42.76%	43.05%	38.80%
99%	24.04%	26.78%	27.37%	27.49%	28.26%	28.40%	25.71%
100%	13.41%	13.40%	13.56%	13.40%	13.54%	13.40%	12.74%

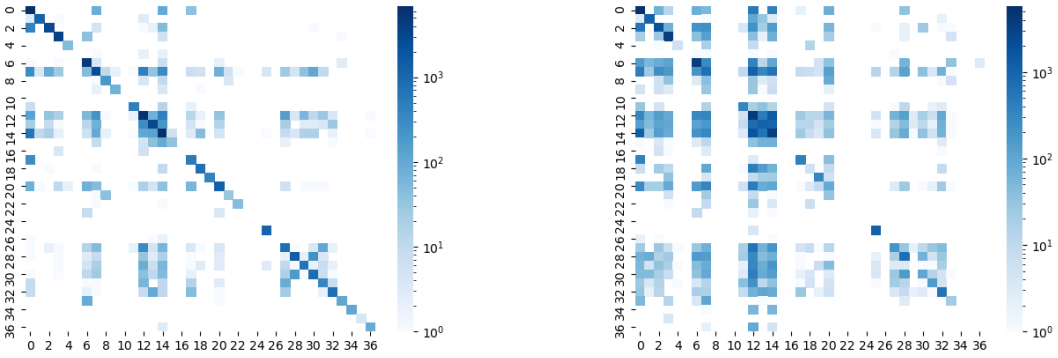
Table 3.4: Probing the sixth layer with different embedding masks

probe with the given hyperparameters (10 epochs, mini-batch of size 10 and 30 neurons in the only hidden layer). The results of these random masked probings show, that considering even only the 20% of the embeddings, the probe had an accuracy by50% higher. Surprisingly, when 98% of the embeddings were masked, the probe performed twice as better as the baseline. From these findings, we can conclude that only a small random portion of an embedding has a high indication towards the PoS corresponding to the word, of which sub-word has produced the embedding. This means that the PoS category of the word is encoded in the embeddings of the BERT model in a redundant way.



(simple) (masked)

Figure 3.3: Confusion matrices of the probing results on the 6-th layer. The left one is the result of a probe trained and tested on embedgings masked on 98%. The labels at rows are the correct labels and the ones under the columns are the predicted ones. For the explanation of the numerical labels see the table in the appendix A 3.5. Darker colours represent more predictions, while a logarithmic color scale is used so that the small errors are visible too.



(simple) (masked)

Figure 3.4: Confusion matrices of the probing results on the initial embeddings. The left one is the result of a probe trained and tested on embeddings masked on 98%. The labels in rows are the correct labels and the ones under the columns are the predicted ones. For the explanation of the numerical labels see the table in the appendix 3.5. Darker colors represent more predictions, while a logarithmic color scale is used so that the small errors are visible too.

Conclusion and future work

When controlling the attention heads of the BERT-NER model for the largest attentions (weights) and correlating them with subject-verb and verb-object dependencies, we have concluded that these correlations are more significant in the lower layers of the model. As we can see in the figures of 3.1, int the case of subject-verb dependencies (the two top heatmaps), the maximal weight was the most correlated with the dependency on the first layer, and with the second attention-head. Similar results were confirmed when correlating the mentioned dependency in the sentences with the three largest weights. When counting the maximum weight attention coincidence with the verb-object dependency, we also found the largest correlations in the lower layers, this time in the attentions produced by the attention-head on the third layer, with the largest correlation at the 4-th attention head. These results were also confirmed by the top-3 wight correlation test. We also looked at the per-head sum of correlations on the specific layers of the pre-trained BERT model layers 3.1. With the subject-verb dependency, the maximal weight correlated in almost 40% of the cases in the first layer, and for the verb-object dependency, this sum was almost 34%. We have to note here that these numbers are the results of sums per layers, and are not from individual heads. The best correlation with the subject-verb dependency for a single attention-head was only above 20% (layer 1, head 2), and for the verb-object dependency, it was around 13% (layer 3, head 4).

We also have to note, that in the case of the top-3 weight correlations we did not differentiate those weights according to their proportions, which could have potentially skewed the overall picture, however despite small differences, this multi-maximal correlation has shown a similar picture (the figures on the right in 3.4 and 3.3). From these findings, we could infer, that the maximal weights in any of the attention-heads are not reliable indicators of the dependencies in question.

However, we have to note, that we assess this undertaking to investigate the attention-heads with probing seems to be more complex regarding the time, computation power, and evaluation requirements. For these reasons we proceeded with the second experiment where we applied a probe on the BERT models token embeddings, probing for part of speech categories. This experiment had the benefit of not needing to relate different tokens with each other as in the previous experiment, although the

part of speech class is not independent of the context of a sentence, since a word form can be of different part of speech (e.g. answer, control, experience, and many others).

As for the probe, we decided to execute the probing technique with a simple multilayer perceptron on the classical BERT model (having 12 transformer layers) pretrained on on English language using a masked language modeling (MLM) objective [11]. We applied the probe on the per-token embeddings of the model, which are the outputs of the given layers of the BERT, when processing a token. We concluded that these layers encode the part of speech (PoS) correspondence in a quite robust way. First, we used a multilayer perceptron with two hidden layers with size 50, training them with batch size 100 for five epochs 3.2. We observed the best part of speech prediction accuracy on the middle layers, namely the sixth layer, where the accuracy was almost 95.3%. The embeddings from the both layers around the 6. one, the 5. and 7. produced an accuracy of over 95%. We can also note, that the worst performance was shown on the input embeddings (layer 0), being around 85%. The fact that the embeddings from the 1. layer have produced an accuracy of 91%, almost increasing the accuracy by 6% compared to the input embeddings (no other increase was observed of such magnitude among the embeddings produced by transformers), we observed that the pass through the transformers greatly benefited the part of speech predictions. We then continued with probings on the embeddings from the 6. layer, experimenting with different hyper-parameters 3.1. We achieved the best accuracy when training for 5 epochs with a batch size of 10 with two hidden layers, getting a precision rate of 95.91%. The next set of probings we have done was on embeddings we masked to a different degree. In hose experiments, we used only one hidden layer in our perceptron, since it did not lead to a dramatic decrease in the accuracies. A baseline result was, when we masked the whole embeddings, giving the machine only a chance of learning the frequency of the data labels. After experimenting with the MLP hyper-parameters, we managed to achieve accuracy of over 13%. If the 37 different labels had an equal chance to appear in the dataset, this result should have been less than one percent. This indicates a drawback of the data having the PoS tags distributed in an uneven way. This, however, may stem from the nature of natural language sentences, where on average some kind of words are much more likely to appear, than others. The next experiments were probings on the embeddings, where we masked the first half of the embeddings in one case and the second half of the embeddings in the other one. In both cases we achieved an accuracy of over 90%, indicating that the part of speech correspondence in the embeddings is not a matter of a few chosen positions, but rather the information is encoded in a more robust and probably redundant way.

The final set of experiments conducted in the category of masked sentences, we applied random masks with a given number of zeroes on the embeddings. 3.4. Here we observed, that even with a large portion of the embeddings being masked, the accuracy

of PoS prediction was high. Only after randomly masking more than 96% of the embeddings has the accuracy decreased below 50%. These results were demonstrating a degree of robustness of the PoS encoding in the model. We can also make some interesting observations looking at the the confusion-matrices 3.4 and 3.3. When no mask was applied on the embeddings, the errors were less wrong classification, and even some of the wrong ones were "near" the correct ones. If we look at the legend of labels in 3.5, we see some tags are more related than others. For example, the tags from 7 to 9 are different adjective types, the tags 12 to 15 are noun types and from 27 to 32 are verb types. If we look at the confusion matrices, we see darker regions at the cross-section of the previously mentioned intervals. These regions indicate that often the wrong labelings are wrong only a "little bit". For example, instead of correctly labeling an embedding as a singular noun (class 12), the MLP labels it as a plural noun (class 13). The confusion matrices for the masked probing also indicate an interesting phenomenon. As the model is unable to learn due to the large mask, it starts guessing, producing the columns on the figures under the more frequent labels in the data.

After looking at these experiments, we have to also mention some drawbacks, based upon which possible directions are adequate for continued work. One such problem is, as we have already mentioned, the dataset being unbalanced regarding the number of PoS occurrences. Comparing our results with similar probings on different datasets could potentially provide some additional information regarding this problem. Those results could be cross-validated based on their given characteristics.

Another issue is determining how much a given model has "actually learned" a task or the given language it was trained on. Further work could include the introduction of different control tasks inspired by the work of [9] For example an untrained BERT model (initialized with random weights) could produce the token embeddings on the same dataset as it was done in our work. Probing this untrained model a baseline could be established for the comparison with the experiments we have conducted. An accuracy contrast could provide more insight into the degree the pre-trained model actually contains the given morphosyntactic information.

For future work, we see multiple possibilities based on our first experiments. As for future work, regarding the first experiment, a possible continuation would be to apply a probe, similarly to what we did in our second experiment, to probe the heads of the BERT model at the layers for the given subject-verb-object connections in the sentences. This could be contrasted with the work [39] of Tenney, Xia, et al., who introduced the so-called edge-probing on contextualized word representations, where among other tasks they conducted probing on dependencies. Another possibility would be to look at the specifics of the subject-verb dependencies looking for clues that correlate with the maximal attention. Including some specific morphological characteristics could increase the correlation with the maximal attention-weight. These experiments

could easily be solved with a post hoc analysis.

Appendix A

In this appendix we included the table with the part-of-speech tags of our dataset [2]. Unmatching tokens (commas, parentheses, etc.) were given the default value of zero.

[2]https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

1	CC	Coordinating conjunction	19	PRP$	Possessive pronoun
2	CD	Cardinal number	20	RB	Adverb
3	DT	Determiner	21	RBR	Adverb, comparative
4	EX	Existential there	22	RBS	Adverb, superlative
5	FW	Foreign word	23	RP	Particle
6	IN	Prepos./subord. conj.	24	SYM	Symbol
7	JJ	Adjective	25	TO	to
8	JJR	Adjective, comparative	26	UH	Interjection
9	JJS	Adjective, superlative	27	VB	Verb, base form
10	LS	List item marker	28	VBD	Verb, past tense
11	MD	Modal	29	VBG	Verb, gerund/present part.
12	NN	Noun, singular or mass	30	VBN	Verb, past participle
13	NNS	Noun, plural	31	VBP	Verb, non-3rd p. sing. pres.
14	NNP	Proper noun, sing.	32	VBZ	Verb, 3rd p. sing. present
15	NNPS	Proper noun, plur.	33	WDT	Wh-determiner
16	PDT	Predeterminer	34	WP	Wh-pronoun
17	POS	Possessive ending	35	WP$	Possessive wh-pronoun
18	PRP	Personal pronoun	36	WRB	Wh-adverb

Table 3.5: Part-of-speech tags enumerated

Appendix B

The software used for the experiments and visualizations can be found in the data storage medium attached to the physical copy of this work. It contains two folders in a zipped folder:

- **svo:** This folder contains the scripts for the dependency experiments. The main script is `apply_transformer.py`. The files `utils.py` and `data_handler.py` contain auxiliary functions and other functions for generating visualizations.
- **probe:** This folder contains the scripts for the probing. The main file is `probe.py` while `data_handler.py` contains the auxiliary functions.

The datasets have to be downloaded from the sources in literature because of their size. Some dependencies may have to be resolved before running the scripts too. For more information see the import sections of the scripts.

Bibliography

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2016.

[2] Jay Alammar. The illustrated transformer, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[4] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[5] Yonatan Belinkov and James R. Glass. Analysis methods in neural language processing: A survey. *CoRR*, abs/1812.08951, 2018.

[6] Eric Brill. A simple rule-based part of speech tagger. In *Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, March 1992. Association for Computational Linguistics.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[9] Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[10] Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. A survey of the state of explainable AI for natural language processing. *CoRR*, abs/2010.00711, 2020.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[12] Lance Eliot. The need for explainable ai (xai) is especially crucial in the law. *SSRN*, 2021.

[13] Dirk Geeraerts. Lexical semantics, 01 2017.

[14] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[15] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[18] Matthew Honnibal and Ines Montani. Spacy models.

[19] Matthew Honnibal and Ines Montani. spaCy 3: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. 2021.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[21] Noman Islam, Zeeshan Islam, and Nazia Noor. A survey on optical character recognition system. *CoRR*, abs/1710.05703, 2017.

[22] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, July 2019. Association for Computational Linguistics.

[23] Devin Johnson, Denise Mak, Drew Barker, and Lexi Loessberg-Zahl. Probing for multilingual numerical understanding in transformer-based language models. 10 2020.

[24] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA, oct 2013. Association for Computational Linguistics.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[26] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, January 1982.

[27] Zihan Liu, Feijun Jiang, Yuxiang Hu, Chen Shi, and Pascale Fung. NER-BERT: A pre-trained model for low-resource entity tagging. *CoRR*, abs/2112.00405, 2021.

[28] Kaggle Kerneler (Kaggle bot) Michael Fekadu, Vishnu Nkumar. Sentences, no punctuation. `https://www.kaggle.com/datasets/mfekadu/sentences`.

[29] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[30] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML 2010*, pages 807–814, 2010.

[31] Isabel Papadimitriou, Ethan A. Chi, Richard Futrell, and Kyle Mahowald. Deep subjecthood: Higher-order grammatical features in multilingual BERT. *CoRR*, abs/2101.11043, 2021.

[32] Peltarion. Visualization of the bert model. `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/bert-encoder`.

[33] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *CoRR*, abs/2003.08271, 2020.

[34] Alec Radford, Rafal Józefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *CoRR*, abs/1704.01444, 2017.

[35] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[36] Ana Paula Silva, Arlindo Silva, and Irene Rodrigues. A new approach to the POS tagging problem using evolutionary computation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 619–625, Hissar, Bulgaria, sep 2013. INCOMA Ltd. Shoumen, BULGARIA.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[38] Yohei Tamura. Robust and fast tokenizations alignment library for rust and python, 2021.

[39] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. *CoRR*, abs/1905.06316, 2019.

[40] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[41] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259, 2003.

[42] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[43] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[44] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.