Deconstructing Ranking Abilities of Language Models

Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Data Science
Fachgebiet Wissensbasierte Systeme
Forschungszentrum L3S

Thesis by
Fabian Beringer

First examiner:	Prof. Dr. Wolfgang Nejdl
Second examiner:	Dr. Sowmya S. Sundaram
Advisor:	Jonas Wallat

Hannover, xx.xx.2022

Abstract

Nowadays, information retrieval plays an important role in our daily lives. Whether we're searching the web, shopping for products online, or trying to find our favorite movies on a streaming platform: An information retrieval system will be responsible for tackling these tasks. As a consequence of recent advances in natural language processing, employing large pre-trained language models as part of a text retrieval pipeline has become a common approach[1]. However, despite their proven effectiveness, these neural network based models are functional black boxes, meaning it is not clear to us as to how they arrive at certain decisions. To get a better understanding of the inner workings of such a model, we apply the recently emerging *probing* paradigm. By employing a diagnostic classifier, this approach enables us to analyze how certain properties are encoded within a model's hidden representations. Unlike previous research that has focused on general linguistic properties, we explicitly study the layer-wise distribution of ranking related knowledge throughout the popular BERT model, a large neural network that has been trained on massive amounts of text data. In this thesis, we provide evidence that BERT not only stores ranking related concepts, but also orders them in a hierarchical manner. Moreover, we leverage our findings to design a multi-task learning setup which infuses task specific information at different layers of BERT, in order to improve the model's ability to rank.

[1] `https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193`

Contents

1 Introduction

1.1 Motivation

Over the last couple of years, contextualized language representations from deep neural network (DNN) models have become the go-to approach for tackling natural language processing (NLP) tasks. In particular, the *transformer* (Vaswani et al., 2017) and its variants, combined with large-scale, unsupervised pre-training, have shown unprecedented performance in a variety of NLP benchmarks.

However, unlike traditional approaches, these models consist of billions of automatically learned parameters, effectively turning them into huge black box functions (Samek et al., 2021). Because of this, understanding why and how such a model arrives at a certain decision becomes a challenge in itself. Yet, as more and more NLP systems rely on these kinds of models, gaining a better understanding of their internal workings becomes crucial, especially when facing problems such as learned social biases (Nadeem et al., 2021; Bender et al., 2021; Kurita et al., 2019), falsely motivated predictions (Ribeiro et al., 2016; Zhang and Zhu, 2018) or simply for determining causes of prediction error. In addition, a better understanding might provide insights on model weaknesses and guide model improvement, e.g. when adapting a model to new domains.

Recent work on understanding neural language models has aimed at measuring the extent to which certain information is encoded in their word representations (Tenney et al., 2019a,b; Hewitt and Liang, 2019a). To achieve this, a *probing* classifier or *probe* is trained to predict certain linguistic properties from these representations. The probe's performance is then expected to reflect how well said property is captured. By employing multiple probing tasks, the presence of different types of information can be estimated. For example, by training a probe for part-of-speech tagging, it can be tested whether it is possible to extract part-of-speech information from the representations.

While the probing paradigm recently emerged for understanding how neural language models encode general linguistic properties, it has yet to be explored, as to what kind of information is stored regarding *downstream tasks* when fine-tuning such models. In our particular case, we're interested in the task of *text retrieval*. Until now, little work has been put into explicitly probing language models for ranking.

However, when considering how information retrieval already affects our daily lives, (e.g. web search, online shopping, searching streaming services) and that for this pur-

pose, large language models are already widely in use[1][2], getting a better understanding of these models is undoubtedly important. For this reason, we want to shed light on the inner workings of the *BERT* (Devlin et al., 2019) transformer model, in the context of ranking. We choose this model, as most recent neural IR approaches are built on BERT or its descendants, i.e. improved or specialized variants that usually share the same core architecture.

1.2 Problem Statement

Given a pre-trained BERT language model, we want to find out as to what extent its hidden representations capture commonly known ranking properties. We further want to know how this kind of information is distributed across different layers of the model and whether these insights help us build a better ranking model. Our research questions are as follows:

- Does BERT encode known ranking properties?
- If this is the case, which layers capture which property?
- Does the distribution change if we fine-tune the model for ranking?
- Can we improve BERT's ranking abilities based on those findings?

1.3 Contribution

This thesis is structured into two parts: Firstly, we apply the probing paradigm in order to understand how a BERT model captures certain ranking properties. We perform extensive analysis and evaluate how ranking related information is distributed throughout the model.

Secondly, we leverage the insights gained from our probing experiments by designing a multi-task learning setup for BERT. We show that infusing additional task information can result in improved ranking performance when compared to a standard fine-tuning approach.

1.4 Thesis Outline

- In the next chapter, we provide the foundations needed for this thesis. We briefly discuss traditional ranking approaches (Section 2.1), then provide the basic ideas

[1] `https://blog.google/products/search/search-language-understanding-bert/`
[2] `https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193`

behind machine learning and deep learning (Section 2.2, Section 2.3) and explain the transformer model (Section 2.4), which is the main subject of our study.

- Chapter 3 presents previous work done in the fields of neural IR, probing and multi-task learning which is related to this thesis.
- In Chapter 4 we present the datasets that were used in this thesis.
- Chapter 5 explains our approach to probing, including task design (Section 5.1), the experimental setup (Section 5.2) as well as the evaluation measures (Section 5.3) that were used.
- In Chapter 6 we present the results from the probing experiments and provide an in-depth analysis.
- Chapter 7 builds upon the findings from Chapter 6. We introduce a novel multi-task learning (MTL) setup, to aid the fine-tuning of BERT for ranking. Again, we present our results and provide analysis (Section 7.3). Furthermore, we perform an ablation study that investigates how this MTL setup behaves in a low resource scenario (Section 7.4).
- In Chapter 8 we conclude this thesis, discuss limitations of our approach (Section 8.1) and give an outlook on potential future work (Section 8.2)

2 Foundations

2.1 Information Retrieval - Ranking Text

Ranking is an integral part of the information retrieval (IR) process. The general IR problem can be formulated as follows: A user with a need for information expresses their *information need* through formulation of a query. Now given the query and a collection of documents, the IR system's task is to provide a list of documents that satisfy the user's information need. Furthermore, the retrieved documents should be sorted by relevance with respect to the user's query in descending order, i.e. the documents considered most relevant should be at the top of the list. (Christopher D. Manning and Schütze, 2008)

While given this formulation the task might appear simple at first, there are several caveats to look out for when it comes to ranking. For instance, there is no restriction on the structure of the query Carpineto and Romano (2012). While we might expect a natural language question like "What color are giraffes?" a user might decide to enter a keyword query like "giraffes color". The same applies to documents: Depending on the corpus we are dealing with, the documents might be raw text, structured text like HTML or even another type of media e.g. image, audio or a combination thereof.

Another possible issue is a mismatch between information need of the user and the corresponding query. Even if we find a perfect ordering of documents with respect to the query, we can not know for certain that the query actually reflects the user's information need. The user might not even know exactly what they're looking for until discovery through an iterative process, i.e. the information need is fuzzy and can not be defined by a single query from the start (Belkin et al., 1993; Christopher D. Manning and Schütze, 2008; Chen et al., 2020a).

Furthermore, a query might require additional contextual information in order for an IR system to find relevant documents (Kekäläinen and Järvelin, 2002; Bai et al., 2007; Tamine and Daoud, 2018). For example, depending on the time at which a query is prompted, the correct answer might change: "Who is the president?" should return a different set of documents as soon as a new president has been elected. Also, since not specified further, it is up to interpretation what country's president the user is interested in. This also shows that some queries might depend on the user's location, i.e. the user might be interested in the president in their respective country. In addition, the corpus might not be static either and change or grow over time. For example, web search has to deal with an ever-growing corpus: the Internet.

While this list of issues is not comprehensive, at this point the complexity of the

ranking problem should have become apparent. Because this work focuses on the ranking of text in the context of web search, we will now give a formal definition with that scenario in mind:

Given a set of $|Q|$ natural language queries $Q = \{q_i\}_{i=1}^{|Q|}$ and a collection of $|D|$ documents $D = \{d_i\}_{i=1}^{|D|}$, we want to find a scoring function $s : Q \times D \to \mathbb{R}$, such that for any query $q \in Q$ and documents $d, d' \in D$, it holds true that $s(q, d) > s(q, d')$ if d is more relevant with respect to q than d'.

To give the reader a more concrete idea and as we are going to build upon it throughout this thesis, we will now discuss two traditional approaches to text retrieval which, unlike neural IR, are based on exact matching, meaning query and document terms are compared directly. Moreover, opposed to neural models, they're "bag of word" models, meaning queries and documents are treated as sets of terms without considering order.

2.1.1 TF-IDF

Term Frequency - Inverted Document Frequency weighting (TF-IDF) is a traditional ranking approach that, given a query, assigns a relevance score to each document based on two assumptions:

1. A document is relevant if terms from the query appear in it *frequently.*
2. A document is relevant if the terms shared with the query are also *rare* in the collection.

From these assumptions, two metrics are derived:

1. Term-Frequency

$$w_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

   where $\mathrm{tf}_{t,d}$ is the count of term t in document d. The logarithmic scaling is motivated by the idea that a document does not linearly become more relevant by the number of terms in it: A document containing a term 10 times more often doesn't necessarily mean it is 10 times more important, e.g. the document might just be very long and contain more words in general. Note that this is just one possible normalization scheme out of many.

2. Inverted Document Frequency

$$\mathrm{idf}(t, d) = \log \frac{|D|}{\mathrm{df}_t} \tag{2.2}$$

   where df_t counts the number of documents that a term occurs in over the full corpus. This way, terms that occur less frequently relative to the corpus size will

receive a higher IDF score and those that are more frequent will receive a lower score.

To compute TF-IDF we can simply sum over the product of TF and IDF for each term in the query to produce a relevance score:

$$\text{score}(q, d) = \sum_{t \in q} w_{t,d} \times \text{idf}_t \tag{2.3}$$

2.1.2 BM25

BM25 builds on the TF-IDF scores by introducing additional hyperparameters and accounting for document length through a normalization term (Robertson et al., 2009). That way, short documents are preferred over long documents, if both have the same term frequency.

$$\text{bm25}_{t,d} = \text{idf}_t \frac{w_{t,d}}{k_1\big((1-b) + b\frac{\text{len}(d)}{\text{avglen}}\big) + w_{t,d}} \tag{2.4}$$

Here, the free parameters k_1 and b determine the how quick the bm25 function saturates with increasing term frequency (typically, $k_1 = 1.2$) and the degree to which document length affects the score, respectively. $\text{len}(d)$ is the length of d and avglen the average document length over the corpus.

2.2 Machine Learning

Machine learning (ML) encompasses a set of statistical methods for automatically recognizing and extracting patterns from data. Typically, we can distinguish between two main types of machine learning: *supervised learning* and *unsupervised learning*. (Bishop and Nasrabadi, 2006)

In the case of supervised learning, we have a set of training instances $X = \{x_i\}_{i=1}^N$ and corresponding labels $Y = \{y_i\}_{i=1}^N$, assigning a certain characteristic to each data point. For example, this characteristic might be a probability distribution over a set of classes or a regression score. If each y_i represents one or more categories from a fixed set of classes $C = \{c_i\}_{i=1}^{|C|}$, it is called a classification problem.

Now given the training data and labels, the goal is to find a hypothesis $h : X \to Y$ that explains this data, such that for unseen data points $x' \notin X$, the labels $y' \notin Y$ can be inferred automatically. One way to estimate the generalization ability of a model or algorithm is to divide the dataset into a training and a test set, and only train on the training set while using the test set for evaluation. If the test set models the full distribution of data adequately, it may act as a proxy for estimating the error on unseen samples.

In contrast, when performing unsupervised learning there is no access to any labels whatsoever. Characteristics of the data need to be learned solely from the data X itself. Examples for this include clustering where X is clustered into groups, representation learning which usually tries to find vector representations for X, as well as dimensionality reduction which, if each X is already a vector, tries to compress them into more compact but still informative representations.

That being said, the lines separating supervised and unsupervised learning have become blurry. Especially with the emergence of semi-supervised approaches and "end2end" representation learning, modern ML methods often integrate parts of both.

2.3 Deep Learning

Deep learning is a subfield of ML that makes use of a class of models called Deep Neural Networks (DNN). Typically, DNNs find application in the supervised learning scenario and are often used for classification tasks. In the following we explain the basic mechanisms of DNNs and common approaches to train them. For additional details, please refer to Goodfellow et al. (2016).

2.3.1 Deep Neural Networks

In essence, a Deep Neural Network (DNN) is a function approximator $f : \mathbb{R}^n \to \mathbb{R}^m$ that applies a series of non-linear transformations to its inputs in order to produce an output. In its simplest form, an input vector $x \in \mathbb{R}^n$ is multiplied by a single weight matrix, a bias vector is added, and the resulting vector is passed through a non-linear activation function σ:

$$f(x) = \sigma(Wx + b) \tag{2.5}$$

where $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are learnable parameters. This model is commonly referred to as single layer feed-forward neural network (FFN) or single layer perceptron.

When used for classification, a single layer FFN is limited to problems that solely require linear separation in order to be solved. To learn more complex, non-linear decision boundaries, multiple layers can be applied in sequence.

An L-layer DNN can be described as follows:

$$\begin{aligned} h^{(1)} &= \sigma^{(1)}(W^{(1)}x + b^{(1)}) \\ h^{(2)} &= \sigma^{(2)}(W^{(2)}h^{(1)} + b^{(2)}) \\ &\vdots \\ f(x) &= \sigma^{(L)}(W^{(L)}h^{(L-1)} + b^{(L)}) \end{aligned} \tag{2.6}$$

Common choices for σ include:

- $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid)
- $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (hyperbolic tangent)
- $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ (softmax)
- $\text{ReLU}(x) = \max(0, x)$ (rectified linear unit (Nair and Hinton, 2010))

2.3.2 Optimization

Arguably, the most common way for optimizing a neural network are the gradient descent algorithm and its variants. For this, an objective function $J(\theta)$ is defined, based on the DNN's outputs and corresponding target labels over the training set.

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; \theta), y_i) \tag{2.7}$$

Here, $\mathcal{L}$ is a differentiable loss function and θ represents the vector of all learnable parameters of the neural network f.

Gradient Descent

For gradient descent, the gradient of $J(\theta)$ with respect to θ is computed and scaled by a hyperparameter called learning rate η. If the objective is to minimize, the scaled gradient is subtracted from the original parameter vector.

$$\theta_{new} = \theta - \eta \nabla_\theta J(\theta) \tag{2.8}$$

By repeating this procedure iteratively, we can gradually minimize $J(\theta)$. Common choices for $\mathcal{L}$ include:

- **Cross Entropy Loss**

  $$\text{CE}(y, \hat{y}) = -\sum_{k=1}^{C} y_k \log \hat{y}_k \tag{2.9}$$

  For classification tasks, where y_k is the ground truth probability of class k and $\hat{y}_k$ the corresponding prediction.

- **Mean Squared Error**

  $$\text{MSE}(y, \hat{y}) = (y - \hat{y})^2 \tag{2.10}$$

  In the case of regression.

Stochastic Gradient Descent

The aforementioned algorithm is also known as the batch gradient descent (BGD) variant. Stochastic Gradient Descent (SGD) differs from BGD in the number of training samples that are used for a gradient update. Where BGD uses the gradient of the full training set for updating θ, SGD only considers a single, randomly picked sample for each update. Not only can this approach be more efficient, since fewer redundant computations are performed, due to its stochastic nature it is also more likely to break out of local minima, allowing additional exploration for better solutions (Ruder, 2016).

Mini-batch Gradient Descent

While SGD's high variance during training makes it more likely to escape local minima, it can also come with the disadvantage of unstable training. In this scenario, convergence might be hindered by overshooting desirable minima.

To mitigate this issue, we can simply use more than one sample in order to achieve a more accurate estimate of the full gradient. Now, at each step a small subset of the dataset is sampled to reduce variance and stabilize training while retaining a level of stochasticity. This variant of gradient descent is called mini-batch gradient descent.

Building on mini-batch gradient decent, many algorithms have been introduced in the context of DNNs, which employ further optimizations in order to improve convergence speed and quality. Notable examples include:

- Adagrad (Duchi et al., 2011)
- RMSProp (Hinton et al., 2012a)
- Adam (Kingma and Ba, 2014)

Algorithm 1: Mini-Batch Gradient Descent with batch size k, learning rate η

Data: $X = \{(x_0, y_0), ..., (x_n, y_n)\}$ training examples and target labels.
Input: function f with trainable parameters θ
initialize θ with random values ;
while *not converged* **do**
  $B \leftarrow$ next k training pairs $\in X$;
  $\theta \leftarrow \theta - \eta \nabla_\theta \big(\frac{1}{k} \sum_{(x_i, y_i) \in B} \mathcal{L}(f(x_i; \theta), y_i) \big)$;
end

Backpropagation

Because a neural network may consist of multiple layers and therefore, is a composition of multiple non-linear functions, computing the gradient w.r.t. each parameter of the network can become a non-trivial and even cumbersome task, if done by hand. One popular way of automatically computing the gradients of a DNN is the backpropagation algorithm (Rumelhart et al., 1988).

Backpropagation is a direct application of the chain rule for calculating the derivative of the composition of two functions. Given two differentiable functions $f(x)$ and $g(x)$, the chain rule states that the derivate of their composition $f(g(x))$ is equal to the partial derivative of f w.r.t. g, times the partial derivate of g w.r.t. x.

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x} \tag{2.11}$$

Let $a^{(k)} = W^{(k)} h^{(k-1)} + b^{(k)}$ be the intermediate output of an L-layer DNN at layer k, before passing it through an activation function σ (See 2.6). With a single application of the chain rule we can compute the gradient of the objective function J w.r.t. $a^{(L)}$ like so:

$$\frac{\partial J}{\partial a^{(L)}} = \frac{\partial J}{\partial \sigma(a^{(L)})} \frac{\partial \sigma(a^{(L)})}{\partial a^{(L)}} \tag{2.12}$$

If we now apply the chain rule a second time, we can produce a term for computing the derivative w.r.t. $W^{(L)}$.

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial J}{\partial \sigma(a^{(L)})} \frac{\partial \sigma(a^{(L)})}{\partial a^{(L)}} \frac{\partial a^{(L)}}{W^{(L)}} \tag{2.13}$$

Note that we now only need to know the derivatives of J, σ and $a^{(L)}$ separately, in order to compute the derivative of their composition. By recursively applying this rule, we can compute partial derivatives of J w.r.t. parameters of the DNN, up to an arbitrary depth, as long as all functions it is composed of are differentiable.

By modeling the chain of operations in a DNN as a computation graph, deep learning frameworks like PyTorch (Paszke et al., 2019) or Tensorflow (Abadi et al., 2015) are capable of automatically performing backpropagation, as long as each operation's derivative is known and pre-defined in the library.

2.3.3 Regularization

Regularization includes a number of techniques to improve the generalization capabilities of an ML model. If an ML model achieves a low error rate on training data but a high error rate on test data, it is said to be *overfitting*. In this scenario, the model has essentially "memorized" the training data and can no longer adapt to unseen examples.

Regularization techniques tackle this problem by limiting the hypothesis space of models through favoring simple solutions over complex ones.

Weight Decay

Weight decay constraints the number of possible hypothesis by adding a penalty based on a model's parameters. For example, L2-regularization encourages small weights that lie on a hypersphere by adding the sum of squares over all parameters to the loss function.

$$J(\theta) = \mathcal{L}(\theta) + \lambda||\theta||_2^2 \tag{2.14}$$

As L2 is only a soft constraint, its effect can be regulated by hyperparameter λ.

Dropout

Dropout is a DNN specific method that, during training time, randomly sets entries in the input vector of a layer to 0 with probability p (Hinton et al., 2012b). The initial idea of this approach is to prevent groups of neurons from co-adapting, i.e. requiring the activation of one another in order to detect a certain feature. If dropout is employed, a neuron can no longer rely on the presence of another neuron. Dropout can also be interpreted as a way of training an ensemble of sub-networks of the original network which share the same parameters.

2.4 Transformer Models

One of the most prominent deep learning architectures of the past years is the transformer (Vaswani et al., 2017). The transformer and its variants have set multiple state-of-the-art records in a variety of NLP tasks (Devlin et al., 2019; Liu et al., 2019b; Clark et al., 2020; Shoeybi et al., 2019; Brown et al., 2020), and have since been adapted to other domains such as computer vision (Dosovitskiy et al., 2020) or audio generation (Dhariwal et al., 2020).

Unlike previous methods for embedding words in vector spaces, which relied on word co-occurrence to learn a single fixed vector for each token (Pennington et al., 2014; Mikolov et al., 2013), transformers are able to produce *contextualized* word embeddings. This means that the representation of each word in a sequence changes dynamically based on its context, i.e. all other words in the sequence.

In this section we will discuss the architecture and ideas behind it and explain one of the most popular training approaches for NLP, named BERT (Devlin et al., 2019).

2.4.1 Architecture

The transformer architecture is based on a single building block which, after an input layer, is repeatedly applied in order to form the full model. Throughout this thesis we will also refer to these building blocks as layers, e.g. a 12-layer model consists of an input layer followed by 12 blocks. A single transformer block consists of:

- a multi-head attention layer
- a point-wise feed-forward layer
- residual connections
- layer normalization

We will first explain the input layer, then go over each of these elements and explain how a transformer block is constructed from them.

2.4.2 Input Layer

The transformer's input layer takes in a sequence of tokens and produces continuous representations by selecting corresponding vectors from a learned embedding matrix $M \in \mathbb{R}^{d \times |V|}$. Here $|V|$ denotes the size of the vocabulary and d the hidden dimension of the model. Because the transformer model itself does not encode the order of its inputs, *positional encodings* are added to the initial embeddings for injecting positional information.

One way for generating positional encodings is to introduce a new set of learned embeddings of dimension d, one for each input position. However, this approach requires a fixed maximum input length, as all embeddings have to be defined before training. Alternatively, Vaswani et al. (2017) propose to use sine and cosine waves as a function of the position:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.15}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.16}$$

where pos and i denote position along sequence and hidden dimension respectively. In terms of machine translation, they've found this approach to perform nearly identical to learned embeddings.

2.4.3 Multi-Head Self-Attention

The Attention Mechanism

Originally, attention mechanisms have been proposed in the context of neural machine translation (NMT) (Bahdanau et al., 2014; Luong et al., 2015). Particularly, they were used for aligning words from a source language with their corresponding translations, i.e. pointing out the source words that are relevant for predicting the next translation target. The alignment is important, as different languages usually do not share the same word order, making a sequential word-to-word translation infeasible.

Generally speaking, attention is a mechanism that allows a model to focus on parts of its inputs, usually while considering a certain context. This could for example be words in a text (inputs) that are regarded important for answering a question (context) (Xiong et al., 2016) or patches of pixels in an image (inputs) that are important for detecting a certain object type (context) (Xu et al., 2015).

Given a sequence of N input vectors $X = (x_1, \ldots, x_N) \in \mathbb{R}^{N \times d}$ and M context vectors $C = (c_1, \ldots, c_M) \in \mathbb{R}^{M \times d}$, we can describe the general attention mechanism as follows:

$$s_{ij} = a(x_i, c_j) \tag{2.17}$$

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^{N} \exp(s_{kj})} \tag{2.18}$$

$$h_j = \sum_{k=1}^{N} \alpha_{kj} x_k \tag{2.19}$$

where $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a scoring function that assigns importance scores to input x_i given context c_j. The attention weights α_{ij} are then used to produce a context-sensitive representation h_j as weighted sum over X.

Common choices for a include:

- $a(u, v) = u \cdot v$ (dot product)
- $a(u, v) = w^\top \tanh(Wu + Uv)$ (additive)
- $a(u, v) = \sigma(w^\top \tanh(Wu + Uv + b) + c)$ (MLP)

Self-Attention

Self-Attention is a special case of attention where input vectors and context vectors stem from the same input sequence. It can be seen as the model attending over a sequence, given the sequence itself as context. In (Vaswani et al., 2017) a third set of *value* vectors is introduced, resulting in three sequences termed *query*, *key* and *value* vectors, in analogy to memory lookups.

To produce these vectors, the initial input sequence is transformed by three different learned weight matrices, namely $W^{(q)}, W^{(k)} \in \mathbb{R}^{d \times d_k}$ and $W^{(v)} \in \mathbb{R}^{d \times d_v}$.

$$Q = XW^{(q)} \tag{2.20}$$

$$K = XW^{(k)} \tag{2.21}$$

$$V = XW^{(v)} \tag{2.22}$$

Then, using the obtained query and key vectors Q and K, a matrix of attention scores is computed and matrix multiplied with V:

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.23}$$

As a consequence, each vector in the resulting sequence becomes an attention weighted sum over the vectors in V. Note the scaling by $\sqrt{d_k}$, which is supposed to prevent oversaturation of the softmax function due to large dot-products.

If we look at this from the memory lookup perspective: Query vectors Q are matched with key vectors K, in order to produce compatibility scores. These scores are then used to retrieve value vectors from V via soft-lookup.

Multi-Head Attention

Self-Attention can further be extended to multi-head attention by running multiple self-attention layers in parallel, then concatenating and projecting their outputs:

$$\text{MultiHeadAttention}(Q, K, V) = \left[\Big\|_{i=1}^{h} \text{SelfAttention}_i(Q, K, V)\right] W^{(o)} \tag{2.24}$$

where h is the number of attention-heads, $||$ denotes concatenation and $W^{(o)} \in \mathbb{R}^{d_v h \times d}$ is a learned matrix for projecting back to the model's original hidden dimension.

Note that in the default case, each attention layer has its own weight matrices. However, we omit layer indices to keep the notation more simple.

2.4.4 Point-wise Feed-forward

The point-wise feed-forward component is a 2-layer MLP that is applied to each position along the sequence dimension, meaning weight parameters are shared across all positions. It follows the following architecture:

$$\text{FFN}(x) = \text{ReLU}(xW^{(0)} + b^{(0)})W^{(1)} + b^{(1)} \tag{2.25}$$

2.4.5 Residual Connection

After applying a layer or block of layers in a DNN, if we add the inputs back to its outputs, we introduce a residual connection or skip connection:

$$\text{Residual}(x) = f^{(k)}(x) + x \tag{2.26}$$

where $f^{(k)}(x)$ is the layer at depth k.

The most prominent motivation for residual connections is that they facilitate the training of deeper neural networks. A common issue with deep neural networks is the vanishing gradient problem. As computing gradients through backpropagation relies on a series of multiplications of potentially small values (Section 2.3.2), gradients tend to become smaller the further we propagate back. This makes training very deep networks harder, as early layers might receive little to no updates.

Since $\text{residual}'(x) = f'(x) + 1$, the gradient of the residual connection will be > 1, even if the gradient is < 1, alleviating the effect of vanishing gradients. It can also be interpreted as preserving more of the initial input information throughout the network, treating the DNN layers as an addition to the identity function instead of a full transformation of the input.

2.4.6 Layer Normalization

Layer Normalization (Ba et al., 2016) is another technique for training deeper neural networks. When training machine learning models on numerical features, it is common practice to normalize inputs, e.g. such that their distribution is centered at 0 and has a standard deviation of 1. This way, there's less variance across features, resulting in more stable training and hence improving convergence.

However, since DNNs pass features through multiple layers, there's no guarantee that hidden representations will maintain a reasonable scale, meaning each layer might have to adapt to a new distribution (Ioffe and Szegedy, 2015). Layer Normalization tackles this problem by computing mean $\mu^{(k)}$ and standard deviation $\sigma^{(k)}$ over the feature dimension of each hidden layer k:

$$\mu^{(k)} = \frac{1}{d}\sum_{i=1}^{d} z_i^{(k)} \tag{2.27}$$

$$\sigma^{(k)} = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(z_i^{(k)} - \mu^{(k)})^2} \tag{2.28}$$

here $z_i^{(k)}$ denotes the i-th output of layer k with hidden dimension d, before applying an activation function.

These layer statistics are then used to normalize the hidden layer representation $z^{(k)}$:

$$\text{LayerNorm}(z^{(k)}) = \gamma^{(k)} \circ \frac{z^{(k)} - \mu^{(k)}}{\sigma^{(k)}} + \beta^{(k)} \tag{2.29}$$

where $\gamma^{(k)}$ and $\beta^{(k)}$ are learned parameter vectors for layer k and $\circ$ denotes the element-wise product. Further, $\gamma^{(k)}$ and $\beta^{(k)}$ are additional learnable parameters for adjusting scale and shift of the normalized distribution if required.

2.4.7 The Full Transformer Block

The full transformer block can be described with the following equations:

$$A = \text{MultiHeadAttention}(X, X, X) \tag{2.30}$$
$$Z = \text{LayerNorm}(A + X) \tag{2.31}$$
$$\text{TBlock}(X) = \text{LayerNorm}(\text{FFN}(Z) + Z) \tag{2.32}$$

It consists of a multi-head attention layer and a point-wise fully connected layer, each followed by residual connection and layer normalization.

2.4.8 BERT Pre-Training

A key part leading to the success of transformer models is their effectiveness in large-scale transfer learning. In the context of NLP, the transfer learning procedure typically consists of two stages: First, a model is pre-trained on large amounts of human generated text data (e.g. from the Internet), using language modeling as an objective. Then, the model is fine-tuned on a downstream task for which only limited amounts of data are available. As the model has already learned basic language concepts during pre-training, it can now leverage this knowledge for the new task instead of learning it from scratch.

One particular approach to transfer learning for transformers, which has pushed the state-of-the-art on several NLP benchmarks, is BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019). Unlike previous pre-training approaches that optimized for traditional *left-to-right* language modeling (Howard and Ruder, 2018; Peters et al., 2018a; Radford et al., 2018), BERT employs a bidirectional objective.

In left-to-right language modeling the goal is to predict the next word in a sequence of words, given the preceding words. BERT on the other hand employs a *masked language model* (MLM) objective. Here, tokens from the input sequence are randomly masked out or replaced with random tokens. The model then has to predict what the actual token should be.

This means that, during pre-training, traditional language models only have access to the context left to their current prediction, while BERT has access to context in both directions.

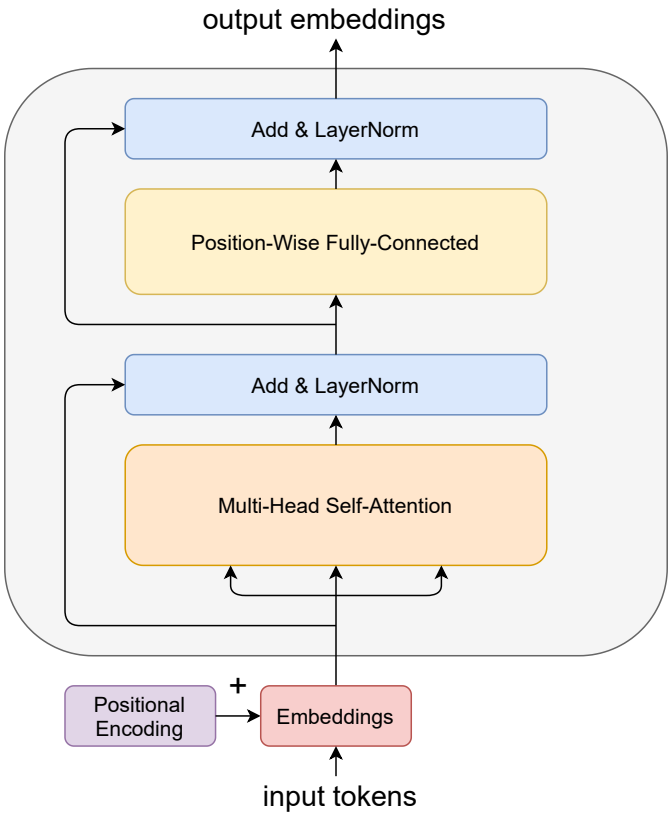

Figure 2.1: Schematic overview of the transformer block and the transformer's input layer at the bottom. Based on Vaswani et al. (2017)

In addition to MLM, BERT introduces a *next-sentence prediction* (NSP) task where, given two sentences, the model has to determine whether the second sentence follows the first, or was selected randomly. This is motivated by language tasks often requiring to model the relation between two sentences.

While many improved training procedures have been proposed since BERT, which often replace NSP or introduce additional objectives (Clark et al., 2020; Liu et al., 2019b; Yang et al., 2019; Lan et al., 2020), MLM still remains as an integral part for most approaches.

2.5 Probing

Probing is a method for estimating the presence of certain properties in dense vector representations of neural network models. The motivation behind probing is to under-

stand what kind of information a DNN stores in its hidden representations. Knowing this can produce insights on what properties a DNN might require for learning a specific task.

To achieve this, a diagnostic classifier (*probe*) P is trained on the fixed representations of a *subject model* $\mathcal{M}$, to predict property Y. In NLP, we typically probe for linguistic properties in word embeddings. For example, we could estimate the presence of part-of-speech (POS) information by predicting POS tags. Based on the probe's performance, e.g. when compared to embeddings from a random baseline, we can conclude how well the property can be decoded from the subject model's representations.

3 Related Work

There are mainly three fields of work this thesis builds upon. In this section we will first introduce related work in neural information retrieval that leverage pre-trained transformer models. We then provide a quick overview of previous work in probing for NLP, as well as Multi-task learning approaches.

3.1 Neural IR with Transformers

In this section we present work on text ranking using transformers. If the reader is interested in a general overview of neural information retrieval, we recommend (Onal et al., 2017; Mitra and Craswell, 2018; Guo et al., 2020).

With the breakthrough of BERT (Devlin et al., 2019) achieving state-of-the-art in various NLP benchmarks, it didn't take long for researchers to also try applying the pre-trained BERT model to the information retrieval problem. In (Nogueira and Cho, 2019), a BERT model is fine-tuned on the MS Marco dataset (Nguyen et al., 2016) to perform re-ranking of text passages. Even though a simple approach was used, which treated relevance estimation as binary classification, it outperformed the existing baselines on the MS Marco leaderboard at that time. Now often termed monoBERT, this approach has become a popular baseline for neural text ranking.

Later, in (Nogueira et al., 2019) a new ranking objective is proposed that computes relevance scores over all possible document pairs in a candidate set. This pair-wise objective is titled duoBERT and achieves better ranking performance with a quadratic increase in computational cost with respect to the size of the candidate set. Hofstätter et al. (2019), on the other hand, do not rely on pre-trained models and instead redesign a neural ranking architecture based on transformer encoders to be more efficient, titled transformer kernel. Further variants building upon this approach are introduced in (Hofstätter et al., 2020; Mitra et al., 2021).

Unlike the previously mentioned methods that leverage a single model to estimate relevance of query-document pairs, *dense retrieval* approaches focus on pre-computing embeddings and directly operating on them during the retrieval phase, e.g. by nearest neighbor search. To achieve this, the common approach is to use two separate encoder models (e.g. two BERT models) for document and query, that produce a fixed size embedding for both. The encoders can then be trained such that a query and a relevant document result in similar embeddings, e.g. by optimizing for cosine similarity between

both vectors (Humeau et al., 2020; Khattab and Zaharia, 2020; Reimers and Gurevych, 2019).

Additional methods for transformer based ranking include knowledge distillation (Chen et al., 2020b; Hofstätter et al., 2020), cascading models for efficiency (Nogueira et al., 2019) and many more. For a more comprehensive survey on transformers in the context of IR, we refer the reader to (Yates et al., 2021).

3.2 Probing Language Models

Recently, a body of research has emerged that focuses on understanding dense vector representations of pre-trained neural language models. Through *probing*, it can be inferred how well certain linguistic properties can be decoded from such representations by predicting these properties with a diagnostic classifier. For instance, Conneau et al. (2018) employ 10 linguistic probing tasks in order to evaluate the representations of different deep sentence encoders. Not only do they probe for simple sentence statistics such as sentence length and word-count, but also for more complex syntactic properties like syntax tree depth. They find that compared to bag of words baselines, these models are generally better at encoding linguistic properties. Similarly, Hewitt and Manning (2019) design a probe which is able to verify that even full syntax trees can be decoded from a BERT model.

Different from Conneau et al. (2018), Belinkov et al. (2017) probe neural machine translation models at *multiple layers*. By employing part-of-speech prediction as probing task, they show evidence that morphological features are better captured at lower layers of these models. Likewise, contextualized embeddings of three different bidirectional language model architectures are probed in (Peters et al., 2018b), with the findings suggesting that lower layers specialize in local syntactic relationships while higher layers model longer range semantic relationships. Supporting this further, Tenney et al. (2019a) deconstruct the language abilities of a BERT model by probing for tasks on different semantic levels. They find that the distribution of linguistic properties across layers suggests hierarchical feature extraction, similar to classical NLP pipelines.

In (Tenney et al., 2019b), a general probing framework termed *edge-probing* is presented, which allows linguistic probing for numerous kinds of tasks. The framework is then used to analyze the impact of contextualization of embeddings. They show that contextualized representations are also generally better at encoding syntactic information when compared to their non-contextualized counterparts.

In another study, Wallat et al. (2020) specifically probe BERT for the distribution of knowledge across different layers. In addition, they do not limit their experiments to the base model, but also investigate fine-tuned models, including ranking as a downstream task. They find that a significant amount of knowledge is stored at intermediate layers and that ranking models suffer less from forgetting than those trained for span

prediction tasks. Merchant et al. (2020), further investigate the effects of fine-tuning on BERT embeddings. Coinciding with Wallat et al. (2020), they find that most information is retained during fine-tuning and that primarily higher level layers are affected by forgetting.

Closely related to this thesis, Choi et al. (2022) probe BERT in the context of ranking. They evaluate whether IDF scores can be decoded from BERT embeddings at different layers of the model and find IDF information to be mostly captured at early layers.

While the general framework of applying a probe classifier to hidden vector representations of a neural language model has been widely adopted, it also comes with drawbacks. On its own, a probe's accuracy on a given task does not convey how well the probed property is encoded, as it is not clear how hard it is to achieve said accuracy. For instance, there is no indication on whether achieving 80% accuracy on a particular task is good or not. One approach to tackle this issue is to introduce a random baseline, i.e. a model with randomly initialized parameters. By probing both, the random baseline and the subject model at hand, a comparison of probe accuracy between both models can be made. Unfortunately, it is a common observation that the difference in accuracy is rather low which makes it hard to draw conclusions from such a comparison (Zhang and Bowman, 2018). Furthermore, it is not clear whether a probe simply learns the task, i.e. memorizes a mapping from embeddings to labels, or actually decodes the property from the word representations. To tackle this issue Hewitt and Liang (2019b) propose the idea of *control tasks*. They construct random tasks by label perturbation on the original tasks. The difference in probe accuracy between both tasks is then termed *selectivity* and used to quantify a probe's ability in decoding task information from the model's embeddings. Unfortunately, analogous to random baselines, this approach often suffers from differences in accuracy that are too small which Hewitt and Liang (2019b) address by limiting dataset size. Moreover, these control tasks require manual design, introducing additional overhead for each new task.

Due to the constraints of random baselines and control tasks, (Voita and Titov, 2020) propose a new metric to quantify probe performance which is the *minimum description length* (MDL) of a probe. The metric is grounded on information theory (Shannon, 1948) and designed to not only reflect the performance of a probe, but also the effort required for achieving that performance. It neither requires the manual design of control tasks nor additional probing on a random baseline. (Voita and Titov, 2020) empirically show the metric to be more robust than accuracy. For more details on MDL, please refer to Section 5.3.1.

3.3 Multi-task Learning for NLP

Multi-task learning (MTL) encompasses methods that learn a single model from multiple tasks or datasets. One motivation for this is to learn a model that is better at generalizing

across different tasks. In this setting, each task is treated equally important.

Another motivation is to improve a model's performance on a particular task by providing additional information from auxiliary tasks. In this case, performance on the main task is most important.

Due to the increased complexity of having to account for multiple tasks, MTL comes with a variety of possible design choices, regarding both architecture and training setup. For instance, Chen et al. (2021) name three general MTL architecture categories:

- Tree-like architectures: a single encoder model is followed by task-specific decoders.
- Parallel feature fusion: each task has a single model, however these models fuse features on one or multiple levels.
- Supervision at different levels: task specific decoders are applied at different levels of a single model.

Another design choice is how to sample data from multiple datasets. For example, a straight forward approach is to sample proportionally to each dataset size (Sanh et al., 2018). However, this can become problematic in cases where some datasets are simply too small compared to the other tasks. Another possibility is to perform task-oriented sampling (Zhang et al., 2017) where samples for each task are drawn with the same probability. As these are just some of the design choices that need to be considered for MTL. For a more comprehensive overview we recommend (Chen et al., 2021; Worsham and Kalita, 2020).

There have been several approaches to address NLP problems with multi-task learning. For instance, (Collobert and Weston, 2008) proposed a convolutional neural network architecture that jointly learns 6 different NLP tasks, including part-of-speech tagging, named entity recognition and language modeling.

More recently, in the context of transformer models, Phang et al. (2018) introduce an additional training stage to BERT, in which BERT is trained on a supplementary task before fine-tuning. Similarly, Liu et al. (2019a) also perform additional training prior to fine-tuning, however using a tree-like architecture, they train on multiple tasks simultaneously.

In (Raffel et al., 2019), a framework is proposed that enables casting of multiple language tasks into a single text-to-text format, allowing training those tasks with the same objective. They were not able to find sampling proportions that are capable of producing results comparable to a pre-train then fine-tune approach. However, when applying MTL as an intermediate step they've observed improved performance similar to Liu et al. (2019a).

In the context of neural ranking, Maillard et al. (2021) present a method for multi-task learning that improves generalization across different domains. Likewise, Fun et al. (2021) present a retrieval-optimized multi-task learning approach, but make use of a sequential training schedule.

4 Datasets

4.1 TREC 2019 - Deep Learning Track

The TREC 2019 deep learning track focuses on studying text retrieval on large-scale data (Craswell et al., 2020). It provides two datasets, one for passage retrieval and one for document retrieval. The datasets are based on MS MARCO (Nguyen et al., 2016) which consists of $\sim$ 1M real world user queries from the Bing search engine and a corpus of $\sim$ 8.8M passages. The passages in the passage dataset are extracted from the document dataset, resulting in multiple passages for each document. Because of this, the document dataset contains less than half the number of all samples.

Since the models we study in this thesis are limited in input length, we will focus on the passage retrieval dataset (TREC2019), unless stated otherwise. Further, we will refer to passages from this dataset as documents, to be consistent with the common information retrieval terminology.

Dataset	**Train**	**Validation**	**Test**
Passage	502,939	55,578	200
Document	367,013	5,193	200

Table 4.1: Number of queries for each dataset split in the two TREC 2019 datasets.

Passage	**Document**
8,841,823	3,213,835

Table 4.2: Corpus size for each TREC dataset.

While with TREC2019, two types of tasks are provided, namely full ranking and re-ranking, we will only perform the re-ranking task. This means, given a pool of 1000 documents for each query, we need to provide an ordering, such that relevant documents are placed at the top. On average, a query has $\sim$ 1.1 relevant documents in its pool which were marked as relevant by human annotators. Note that each annotator only had access to $\sim$ 10 passages during annotation, meaning a pool is likely to contain false negatives, i.e. relevant passages that are not marked as such.

Conversely, documents in the document dataset are automatically marked as relevant, if they contain at least 1 relevant passage.

Task	Query	Document	Target
BM25	how many days to defrost	The best way to safely and quickly defrost chicken, he says, is...	11.44
SEM	how many of grams a sugar should a person have in one day?	According to the Institute of Medicine, Food and Nutrition Board, the average person needs 0.32 grams of protein per pound of body weight...	0.316
NER	what company was skittles made by?	[**Wrigley**] is a company that makes and sells gum, hard candies, and lollipops...	Organization
COREF	at what age do [**kittens**] get their first shots	Kittens this age will begin to eat regular cat food, and will begin to use a litter box. [**They**] are still quite small at this age...	True
FACT	Big Brother 18 is hosted by Emma Willis.	Big Brother 2017, also known as Big Brother 18, is the upcoming eighteenth series of the British reality television series Big Brother, hosted by Emma Willis ...	Supports

Table 4.3: Training examples from each probing dataset and their respective targets. Target spans within the text are highlighted in square brackets.

4.2 Probing Dataset Generation

For all of our probing tasks (Section 5.1), we leverage automatically generated datasets[1] from the TREC2019 passage-level test set. To achieve this, 60k query-document pairs are sampled from the test set of which 40k are used as training set and 10k as validation and test set, respectively. Then, existing tools are used to extract the properties that we are interested in and used to label the data. Details on label generation for each task are described in Section 5.1. Examples for each task are shown in Table 4.3. Throughout this thesis, we will abbreviate each task name as follows: Semantic similarity (SEM), named entity recognition (NER), coreference (COREF), fact checking (FACT).

[1]The code for generation was provided from: `https://github.com/Heyjuke58/tinse_probing_datasets`

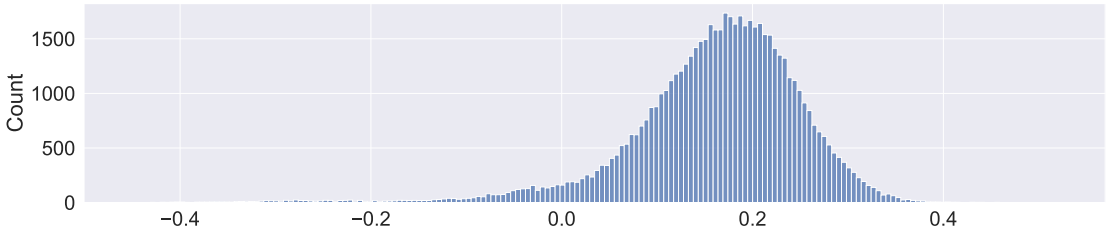

Figure 4.1: Distribution of SEM target scores.

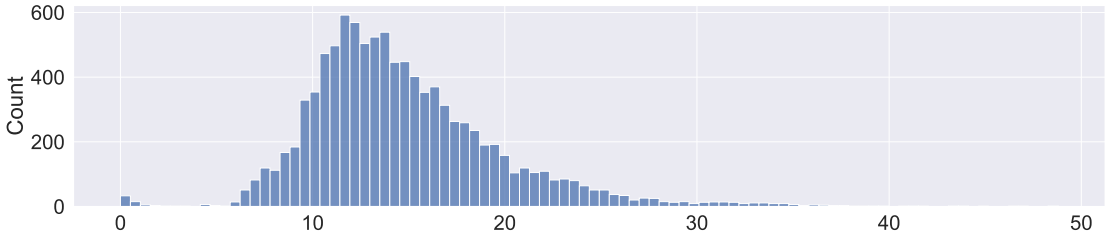

Figure 4.2: Distribution of BM25 target scores.

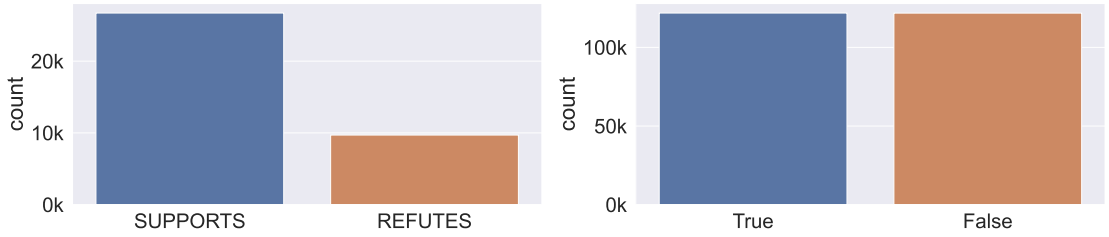

Figure 4.3: Distribution of FEVER and COREF labels. COREF negatives are sampled for each positive.

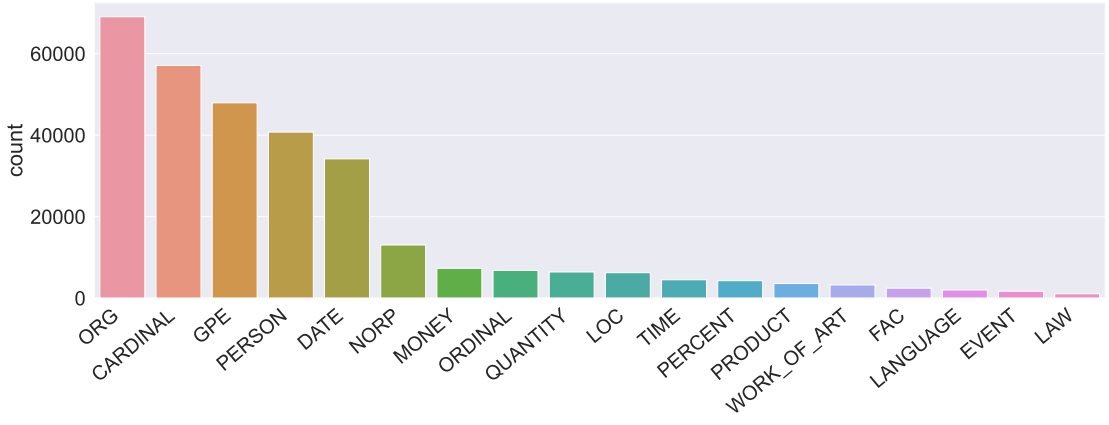

Figure 4.4: Distribution of named entity types in the NER dataset.

5 Approach

Our goal is to understand whether certain properties that we expect to be relevant for ranking can be decoded from the hidden representations of a pre-trained transformer model. Additionally, we want to know to which degree these properties are encoded at different layers within the model.

To test this, we conduct the following experiment: First, we generate a set of datasets $D_{\text{probing}} = \{D_1, D_2, \ldots, D_n\}$, each aimed at predicting a property Y_i $(i \in (1, \ldots, n))$ that, based on traditional ranking methods, can be considered relevant for ranking. Then, for each dataset, we train a probing classifier $P_i : \mathbb{R}^d \to \mathbb{R}^c$ on top of the fixed hidden representations $H^{(k)} = \{h_i\}_{i=1}^N \in \mathbb{R}^{N \times d}$ of subject model $\mathcal{M}$. This procedure is repeated at every layer k. Finally, we compare the classifier's performance (Section 5.3) across layers to get a relative measure of how task-specific information is distributed throughout $\mathcal{M}$. To better put our measurements into perspective, we employ a random baseline model $\mathcal{B}$ which is also probed for each task separately. The probing procedure for BERT is illustrated in Figure 5.1.

Following the probing experiments, we then attempt to develop an improved fine-tuning procedure, based on our findings.

5.1 Task Design

We propose a set of classification tasks with ranking properties as target variable. (Tenney et al., 2019a) findings suggest that BERT stores language concepts in a hierarchical manner, with lower-level semantics being captured in early layers, while higher level concepts can be found closer to the output layer. To test whether this holds true for ranking, we choose tasks that require different levels of semantic abstraction in order to be solved. In this section we provide a list of the tasks we've chosen and explain the reasoning behind our selection, as well as how labels are generated automatically.

BM25 Prediction

BM25 (Section 2.1.2) is a well-known text-retrieval method and still considered a first choice when it comes to computational efficiency. As a heuristic designed around exact term matching, BM25 compares query and document on a symbolic level, without the notion of higher level semantics. Being able to decode BM25 from BERT embeddings

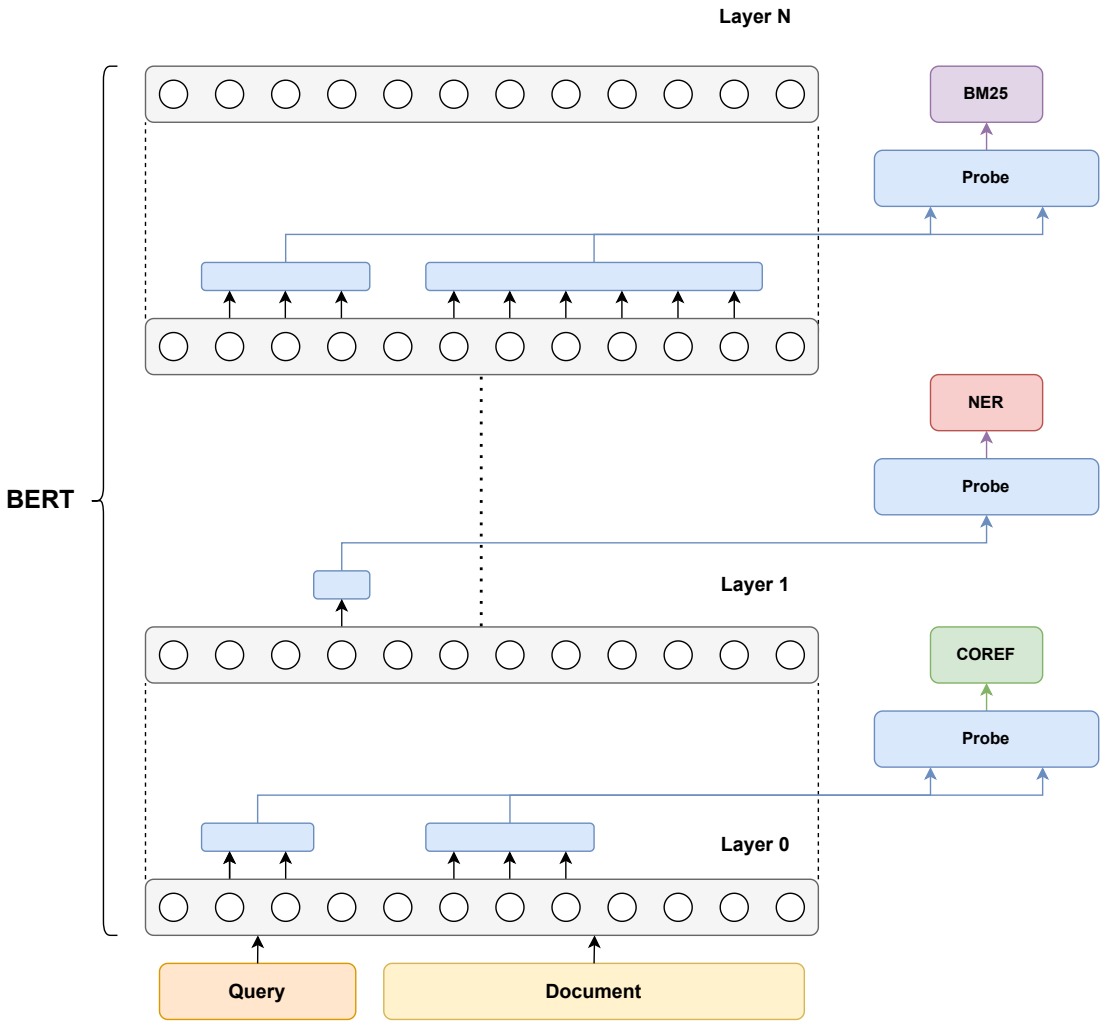

Figure 5.1: Schematic overview of the probing procedure. For the sake of clarity, this example only shows one task per layer. However, in our experiments each property is probed at *every* layer.

might give a hint on whether exact matching properties like term-frequency and corpus level statistics like inverted document-frequency are encoded by the neural ranking models.

To generate BM25 scores for each query-document pair, the Elasticsearch BM25 implementation[1] is used.

[1] `https://www.elastic.co/de/elasticsearch/`

Semantic Similarity

To measure semantic similarity, we compute cosine distance between query and document vectors in the GloVe (Pennington et al., 2014) embedding space:

$$\text{Sim}(q, d) = \frac{q^\top d}{||q|| \times ||d||} \tag{5.33}$$

While these dense word representations encode semantics, unlike BERT they are not contextualized, meaning they do not change based on surrounding words in a sentence. In that sense, using semantic similarity as ranking measure may be interpreted as a kind of soft term-matching: Words between query and document that are similar in meaning increase the estimated relevance.

In order to be able to compute semantic similarity between query and document, first all words are embedded in the GloVe vector space. Then the average embedding for query and document over the sequence dimension is computed and the resulting two vectors are compared.

Coreference Resolution

Coreference resolution is the task of deciding whether two mentions in a text refer to the same entity. To model coreference we use binary classification over two spans of text in query and document, respectively.

We argue that recognizing whether an entity is shared between query and document can be an important indicator on whether the document is relevant. For instance, knowing that "the US president" in a query refers to "Joe Biden" in a document is certainly helpful for determining relevance.

To detect and score entity pairs, the neuralcoref pipeline extension for spaCy[2] is used. It consists of a rule-based mentions-detection module, followed by a feed-forward neural network which produces a binary coreference score for each detected pair.

Named Entity Recognition

Similar to coreference resolution, recognizing entities can be used to match concepts between query and document when performing retrieval. However, in the case of named entity recognition this matching is less restricted, as only entity *types* are considered instead of specific entities. For example, given the query: "How much HP does a jaguar have?", a document that contains car entities should be prioritized over one with animal entities.

For our task, we solely test the general ability of the model to encode entities, meaning we treat document and query as a whole and predict entity types of separate text spans.

[2] `https://github.com/huggingface/neuralcoref`

To identify named entities, spaCy's (Honnibal and Montani, 2017) named entity recognition module is employed. It is capable of detecting and assigning one of 18 different types of entities. Naturally, only pairs that contain at least one entity are being included.

Fact Checking

For fact checking, we leverage the existing FEVER (Thorne et al., 2018) dataset. Here, instead of query and document, a claim and evidence text are provided. The goal is to classify whether the evidence supports or refutes the claim. This task requires high level semantic reasoning, making it relevant for more fine-grained ranking, i.e. providing documents that contain *coherent* answers to a query and not only similar ones.

Since this is the only dataset that we don't sample from TREC2019, we can simply leverage the existing labels of FEVER and sample claim-evidence pairs from its train set.

5.1.1 Probing Models

Subject Models

Subject of our probing experiments is the pre-trained BERT (Devlin et al., 2019) transformer model. In particular, we use the *bert-base-uncased*[3] variant, which consists of 12 layers, with $h = 12$ attention heads each and a hidden dimension of $d = 786$. The model has been trained on BooksCorpus (Zhu et al., 2015) and text passages from English Wikipedia[4], which consist of 800 million and 2.5 billion words, respectively.

In addition, we probe two fine-tuned *bert-base-uncased* models that we term *bert-msm-passage* and *bert-msm-doc*. Both are trained on datasets from the TREC2019 deep learning track (Craswell et al., 2020). While *bert-msm-passage* is trained to predict relevancy of *passages* given a query, *bert-msm-doc* is trained on *document*-level data. Thus, for this purpose we use the TREC2019 passage- and document-level dataset respectively.

Note that all three models share the same architecture and only differ by which datasets they were trained on. Having access to additional versions of *bert-base-uncased* that were fine-tuned specifically for ranking, gives us a way to compare if and how the distribution of information throughout the model changes when being adapted to the new task.

Finally, as a baseline model, we probe *random-embeddings*. For this we take a BERT embedding matrix and initialize its weights from a normal distribution $\sim \mathcal{N}(0, 1)$ and fix it during probing.

[3] `https://huggingface.co/bert-base-uncased`
[4] `https://en.m.wikipedia.org/`

Probe

One problem in probing arises when trying to choose a probe of appropriate complexity (Hewitt and Liang, 2019b). If the classifier is too complex, it might end up modeling new, complex features itself and hence, rely less on the information that is already present in the subject model's representations. On the other hand, if the classifier is too simple, it might not be able to properly decode the information at all.

While there has recently been debate on whether more complex models are actually problematic for probing (Pimentel et al., 2020), following previous literature (Tenney et al., 2019b,a; Hewitt and Liang, 2019b) we've decided on a simple 2-layer MLP which, despite its simplicity, is still capable of modeling non-linear relationships. Specifically, we use the same MLP as (Tenney et al., 2019b):

$$\hat{P}(x) = \text{LayerNorm}(\tanh(xW^{(0)} + b^{(0)}))W^{(1)} + b^{(1)} \tag{5.34}$$

where $W^{(0)} \in \mathbb{R}^{|S|d \times d}$, $W^{(1)} \in \mathbb{R}^{d \times c}$ and $b^{(0)} \in \mathbb{R}^{d}$, $b^{(1)} \in \mathbb{R}^{c}$ are learned parameters with hidden size d and number of target classes c.

Since some tasks require operating over one or multiple spans $S = \{(\text{start}_i, end_i)\}_{i=1}^{|S|}$, we further need to use a pooling mechanism, such that a fixed-length representation can be provided to the probe. Again, like (Tenney et al., 2019b) we employ the simple attention pooling operator also used in (Lee et al., 2017, 2018):

$$\begin{aligned} \alpha_n &= \frac{\exp(w^\top h_n)}{\sum_{k=i}^{j} \exp(w^\top h_k)} \\ \text{pool}(h_i, h_{i+1}, \ldots, h_j) &= \sum_{k=i}^{j} \alpha_k (W h_k + b) \end{aligned} \tag{5.35}$$

where $w \in \mathbb{R}^d$, $W \in \mathbb{R}^{d \times d_{probe}}$, $b \in \mathbb{R}^{d_{probe}}$ are learned parameters which, in the case of multiple spans, are not shared between spans. The pooled span representations are concatenated and passed to the MLP, resulting in the full probe:

$$P(h_1, \ldots, h_N) = \hat{P}\Big(\Big\|_{(i,j) \in S} \text{pool}(h_i, h_{i+1}, \ldots, h_j) \Big) \tag{5.36}$$

5.2 Probing Setup

5.2.1 Fine-tuning Subject Models

We fine-tune a *bert-base-uncased* model on both, TREC2019 passage- and document-level datasets (Section 4.1), to obtain ranking subject models *bert-msm-passage* and *bert-msm-doc* (Section 5.1.1), respectively. Each model is trained with a binary cross-entropy objective, for a maximum of 20 epochs. Early stopping is performed after 3

epochs, if no increase in validation MAP is observed. For hyperparameters, we keep the default settings suggested by (Devlin et al., 2019), using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1e-5, a mini-batch size of 16 and linear increase in learning rate over the first 1000 steps. As BERT uses a fixed set of learned positional embeddings, the input length is limited to 512 tokens. Therefore, we truncate any passages and documents exceeding this maximum length after applying BERT's word-piece tokenization (Wu et al., 2016).

Fine-tuning is performed on a server with 256GB of memory, using a single Nvidia A100 GPU.

5.2.2 Probe Training

All three subject models (Section 5.1.1) are probed at their intermediate output representations, meaning a classifier is learned for each layer separately. This includes the initial sequence of non-contextualized embeddings from the input embedding-matrix, which we refer to as layer 0. We repeat this procedure for all of our proposed tasks (Section 5.1).

As objective function, all probing tasks use cross-entropy loss (Equation (2.9)) which is necessary to compute MDL (Section 5.3.1). In order to achieve this, we cast regression tasks to classification tasks by binning them into $k = 10$ categories. To reduce the influence of outliers, we truncate any values that are further than two standard deviations away from the mean. This means any value $< \mu - 2\sigma$ will be assigned to class 1 and $> \mu + 2\sigma$ to class 10.

We employ the Adam optimization algorithm (Kingma and Ba, 2014) with a learning rate of 1e-4 and a batch size of 32 for updating the probe's parameters, while the subject model's parameters remain fixed. Learning rate is halved each epoch if the validation loss does not improve. The maximum number of epochs is set to 50, and we stop early if validation loss has not improved over the course of 10 epochs. We set the probe classifier's hidden size to $d_{probe} = 256$ and apply dropout with rate 0.3 before the output layer.

All probing experiments are conducted on a server with 128GB of system memory, using a single Nvidia GTX 1080ti GPU.

5.3 Evaluation Measures

In the following, we explain the evaluation measures that we employ for probing and any subsequent experiment.

5.3.1 MDL

As mentioned in Section 5.1.1, selecting a proper size for a probe can be difficult. With a large probing classifier it becomes unclear whether the property probed for is decoded, or the classifier simply learns the task at hand. One way to address this problem is to compare how well the classifier performs on randomly initialized baseline representations (Zhang and Bowman, 2018). Further, (Hewitt and Liang, 2019a) propose the use of *control tasks*, for which task labels are randomly assigned, and a probe is selected based on the difference in accuracy to the original task. Both approaches, however, often do not reflect a large difference in accuracy when compared to the original representations or task.

As a solution, instead of accuracy, (Voita and Titov, 2020) propose an information theoretic approach for measuring probe performance. By recasting learning a probe model to transmitting label data with the least amount of bits, a new measure can be applied: The *minimum description length* (MDL) required for transmitting the task labels, given the probed representations. Not only does MDL measure the probe's predictive performance, it also takes into account the amount of effort that is requierd for achieving said performance. The effort can manifest in model size or the amount of required training data.

Since (Voita and Titov, 2020) find that MDL is a more representative and also reliable measure than accuracy, we also choose it as preferred method for measuring probe performance. To compute MDL, we use the online code definition (Rissanen, 1984). For this, the probing dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ is divided into timesteps $1 = t_0 < t_1 < \ldots < t_S = n$. After encoding block t_0 with a uniform code, for each following timestep a probing model P_{θ_i} is trained on the samples $(1, \ldots, t_i)$ and used to predict over data points $(t_i + 1, \ldots, t_{i+1})$. The full MDL is then computed as sum over the codelengths of each P_{θ_i} and the uniform encoding of the first block:

$$\begin{aligned} \mathrm{MDL}(y_{1:n}|x_{1:n}) &= t_1 \log_2 c \\ &- \sum_{i=1}^{S-1} \log_2 P_{\theta_i}(y_{t_i+1:t_{i+1}}|x_{t_i+1:t_{i+1}}) \end{aligned} \tag{5.37}$$

where c is the number of target classes. Following (Voita and Titov, 2020), we choose timesteps at 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.25, 12.5, 25, 50 and 100 percent of the dataset.

5.3.2 Compression

Because it depends on the number of targets in a probing dataset, it is not reasonable to directly compare MDL between different tasks. A common way to turn MDL into a relative measure is to compute *compression*. Compression divides the codelength that would result from a uniform encoding by the actual MDL:

$$\text{compression} = \frac{n \log_2(c)}{\text{MDL}(y_{1:n}|x_{1:n})} \tag{5.38}$$

This means compression is a measure of how much easier it is for the probe to decode a property from the probed representations, or in other words how well they encode the task labels.

5.3.3 Accuracy

As a secondary measure for probing, we further employ accuracy:

$$\text{Accuracy}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\hat{y}_i = y_i] \tag{5.39}$$

This is simply the percentage of predictions $\hat{y}$ that match the correct target y.

5.3.4 Ranking

Typically, ranking metrics require a set of queries $Q = \{q_i\}_{i=1}^{|Q|}$, with each query being associated with a list of labeled candidate documents $C = \{c_i\}_{i=1}^{|C|}$. The list of candidate documents is expected to be ordered in terms of their predicted relevance and each document has a ground truth label with respect to the query.

MRR

Mean reciprocal rank measures the inverse position at which the first relevant document occurs in the result set, averaged over all queries:

$$\text{MRR}(Q, C) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}(q, C)} \tag{5.40}$$

Here, $\text{rank}(q, C)$ refers to the rank, i.e. the position of the first document in C that is labeled as relevant.

Precision@k

Precision@k (P@k) measures precision at a specific threshhold in C, i.e. only the top-k results are considered when computing precision:

$$\text{Precision@}k = \left(\frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}} \right)_k \tag{5.41}$$

MAP

Unlike MRR, mean average precision (MAP) considers the rank of all relevant documents in the candidate set, not just the first one. Initially, average precision (AP) is computed over the set of candidates for a query:

$$\mathrm{AP}(q, C) = \frac{1}{|C_{rel}|} \sum_{k=1}^{|C|} \text{Precision@k} \times y_k \tag{5.42}$$

where $|C_{rel}|$ is the total number of documents marked as relevant w.r.t. the query and y_k is the ground truth relevance at position k in C. Then taking the average AP over all queries gives us MAP:

$$\mathrm{MAP}(Q, C) = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{AP}(q, C) \tag{5.43}$$

While MRR indicates the ranking quality given the top-ranked document, MAP provides a better picture of the full candidate set ranking.

NDCG

Normalized Discounted Cumulative Gain (NDCG) quantifies a ranked list relative to its ideal ranking. NDCG can also be applied to non-binary labels. At first, a gain is computed, that rewards early placement of relevant documents in C, by logarithmically scaling relevance label y_i as a function of the position and summing over all labels:

$$\mathrm{DCG} = \sum_{i=1}^{|D|} \frac{y_i}{\log_2(i+1)} \tag{5.44}$$

Then, the ideal DCG is computed by sorting C according to the ground truth labels and using it to scale DCG.

$$\mathrm{NDCG} = \frac{\mathrm{DCG}}{\mathrm{IDCG}} \tag{5.45}$$

This way, a perfect ordering (of which multiple may exist) will result in an NDCG of 1 while any suboptimal ordering will lie in $[0, 1)$. Because of the logarithmic scaling, more importance is attributed to relevant documents early in the list.

Again, like Precision@k, by limiting the result set to the first k samples, we can also compute NDCG@k.

6 Probing Results

In this section we will analyze and discuss the results from the probing experiments. Since the overall distribution of properties across layers follows a similar pattern across models, we will first focus on the distribution on *bert-base-uncased*, then discuss the effects of fine-tuning in the following section.

6.1 Distribution of Ranking Properties

Line graphs for our results are visualized in Figure 6.1 and Figure 6.2 respectively. Firstly, a general trend we can observe across tasks is an increase in both compression and accuracy over the first couple of layers up to layer 4-6, suggesting that ranking concepts arise mostly in the mid-range of layers. Then, after a peak at some mid to upper level layer, we observe a constant decrease until the last layer. It is notable that accuracy appears to exhibit a less stable curve, with sudden peaks and drops in between adjacent layers, especially in the case of coreference resolution (COREF) and semantic similarity (SEM) (Figure 6.1). This observation coincides with (Voita and Titov, 2020) findings, that MDL and as a consequence compression, are a more stable measure for probing than accuracy.

For the most part, based on compression, ranking properties can be decoded more efficiently from our trained models than from the random baseline, meaning the probe model is not solely adapting to the task, but instead leveraging information present in the pre-trained representations.

We can further observe that the difference in compression between early layers and the peak value varies depending on the task. This indicates that some properties are more uniformly distributed across layers, while others are more concentrated at a particular layer. For example, decoding named entities results in similar compression scores from layer 1-11, while SEM shows a distinct peak at layer 4.

To better understand this behavior, we can have a look at the row-normalized heatmap in Figure 6.3 of the *bert-base-uncased* model. We can see that COREF and fact checking strongly center around layers 4-7 and 6-9 respectively, with fact checking showing a slightly wider spread. While BM25 exhibits a similar pattern, it is less focused around a particular layer, but instead more evenly distributed from layer 4-6. SEM and named entity recognition (NER) on the other hand look almost identical, with a rather flat distribution that is more prominent in early layers 1-4.

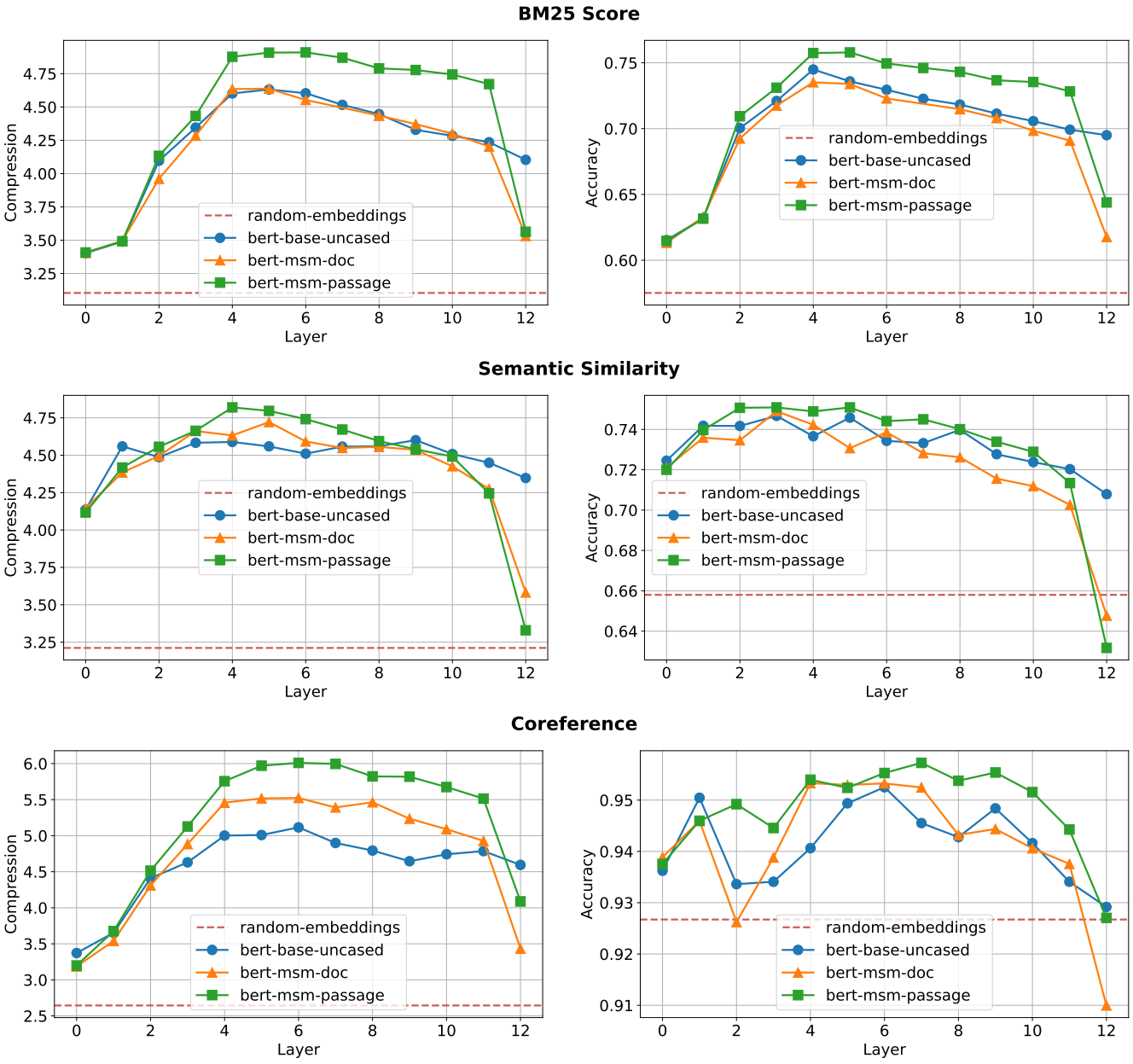

Figure 6.1: Layer to probing score for BM25, semantic similarity and coreference properties. We report accuracy on the test set.

When considering that COREF and fact checking are both tasks that require higher level semantics, based on the observation in (Tenney et al., 2019a), it makes sense that

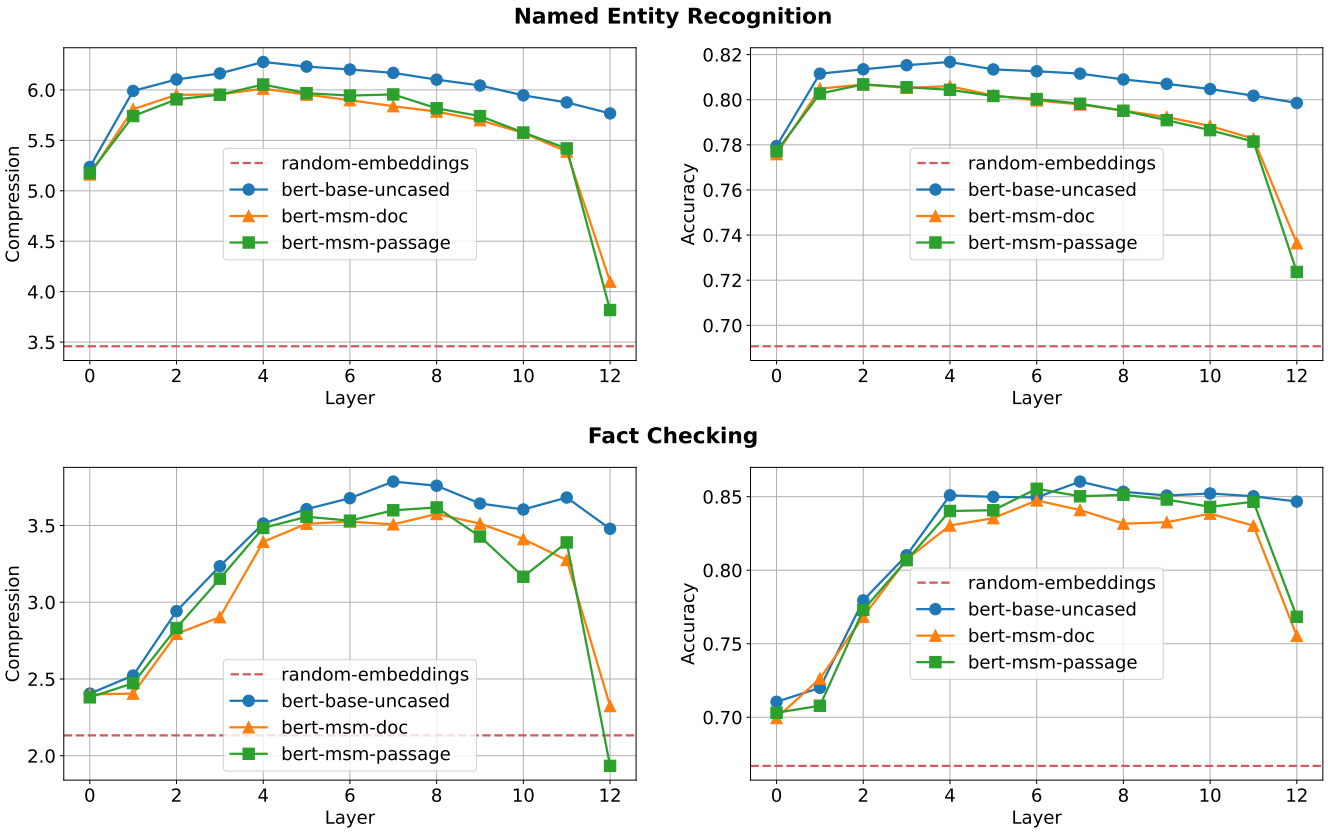

Figure 6.2: Layer to probing score for NER and fact checking properties. We report accuracy on the test set.

both distributions are leaning more towards mid to upper layers. On the other hand, based on this, we would expect a lower level concept such as SEM to be mostly present in early layers, especially since it is a property that can already be captured by non-contextual word embeddings. However, except for a slight peak at layer 1, we observe the property to be more evenly spread across layers. We hypothesize that, as a fundamental concept upon which higher level tasks build on, SEM is a property of the embedding space that needs to be preserved throughout the whole model.

Further, even though NER appears to be a higher level concept than SEM at first, considering that similar categories of entities are likely grouped together in the embedding space (Mikolov et al., 2013; Pennington et al., 2014), a relation to SEM property becomes apparent. Given that NER requires us to predict *categories* of entities, it might explain that NER shares a very similar distribution with SEM.

BM25 depends on both local and corpus-level frequency statistics of words. Finding

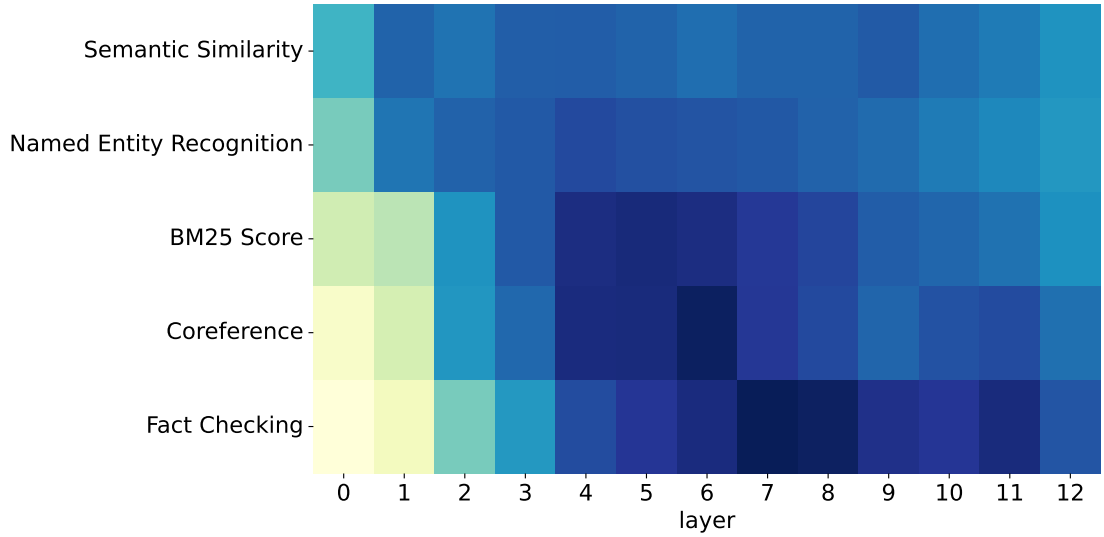

Figure 6.3: *bert-base-uncased*: compression as a function of task and layer, row-normalized. Darker colors represent higher values.

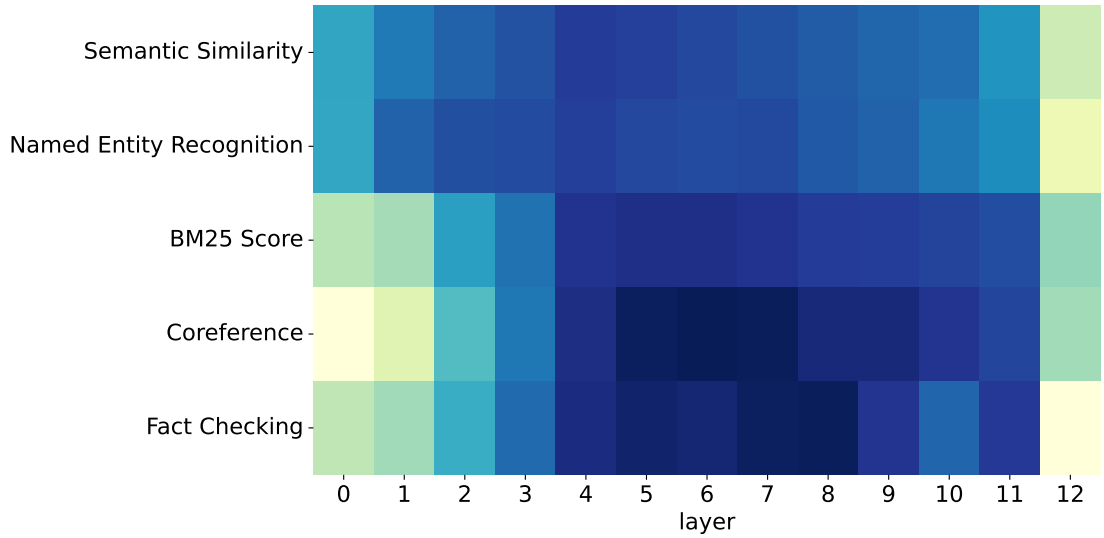

Figure 6.4: *bert-msm-passage* - compression as a function of task and layer, row-normalized. Darker colors represent higher values.

that layers 4-6 are best for inferring BM25 might indicate that some kind of direct term comparison between query and document is modeled within this range of the model. Choi et al. (2022), who probe BERT for IDF, find that the IDF property is present in

early layers and constantly decreases up to the final layer. However, unlike IDF, BM25 also encodes term frequency and considers the length of a document, supporting the idea that local interactions are more prominent in the mid-layers, while corpus level statistics are encoded more evenly across the model.

Finally, since compression is a relative measure that compares the probe's MDL to the MDL of a uniform encoding, it is possible to compare it *between* tasks. For this, we show absolute compression values in Figure 6.5. Firstly, the highest compression can be measured for NER with values > 6. Further, a compression around 4.5 is achieved for BM25, SEM and COREF, with COREF showing some peaks closer to 5. In contrast, for fact checking, we barely reach a compression of > 3.5. These findings indicate that NER is the most easily extractable property from the pre-trained base model, while fact checking appears to be the most difficult.

6.2 Effects of Fine-tuning for Ranking

First of all, we can observe that for BM25, SEM and COREF resolution, the *bert-msm-passage* representations result in higher compression scores for most layers, suggesting that the extractability of these properties indeed increases through fine-tuning for ranking. Whereas this holds true for *bert-msm-doc* in COREF, for the other two tasks we can observe values closer to *bert-base-uncased*. Moreover, the difference starts to first become apparent around layer 2-3 and begins to increase from there on. This coincides with (Merchant et al., 2020), who find that a change in BERT's representations primarily occurs in the upper layers after fine-tuning.

Surprisingly, for both NER and fact checking, *bert-base-uncased* preserves most of the property while the fine-tuned versions appear to lose some of it. However, this difference also grows with increasing depth.

Another pattern resulting from fine-tuning that we can observe, is a sudden drop in compression at the last layer across all tasks. This might be a sign that through fine-tuning, the final layer becomes more specialized and as a result loses some of the general linguistic knowledge that has been acquired during pre-training. This is also supported by (Wallat et al., 2020; Merchant et al., 2020).

We can gain further insights on how fine-tuning affects BERT's representations by comparing Figure 6.3 and Figure 6.4. For SEM, we can see that, after fine-tuning, the property's distribution seems to center more around layer 4 and fade out gradually towards both sides. In contrast, the pre-trained representations exhibit a more even distribution of the SEM property, with no apparent center. Similarly, the concentration of the NER property in early layers becomes more pronounced.

For BM25, a more spread distribution that is shifted towards upper layers can be observed. This behavior is similar to the observations of (Choi et al., 2022), who found the IDF property to be more prominent in higher layers of BERT after fine-tuning for

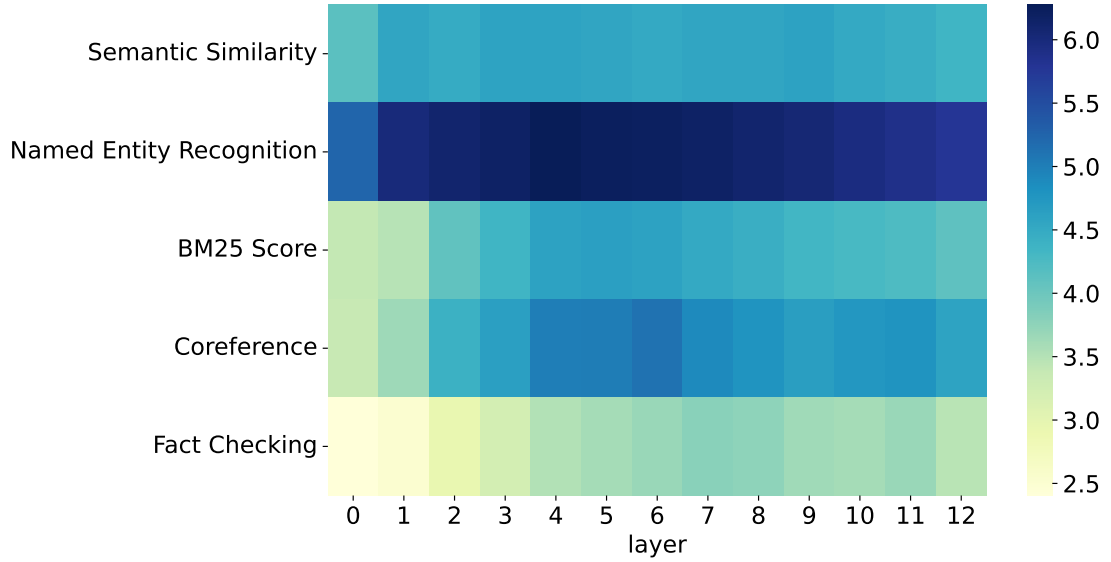

Figure 6.5: *bert-base-uncased* - compression as a function of task and layer, absolute values.

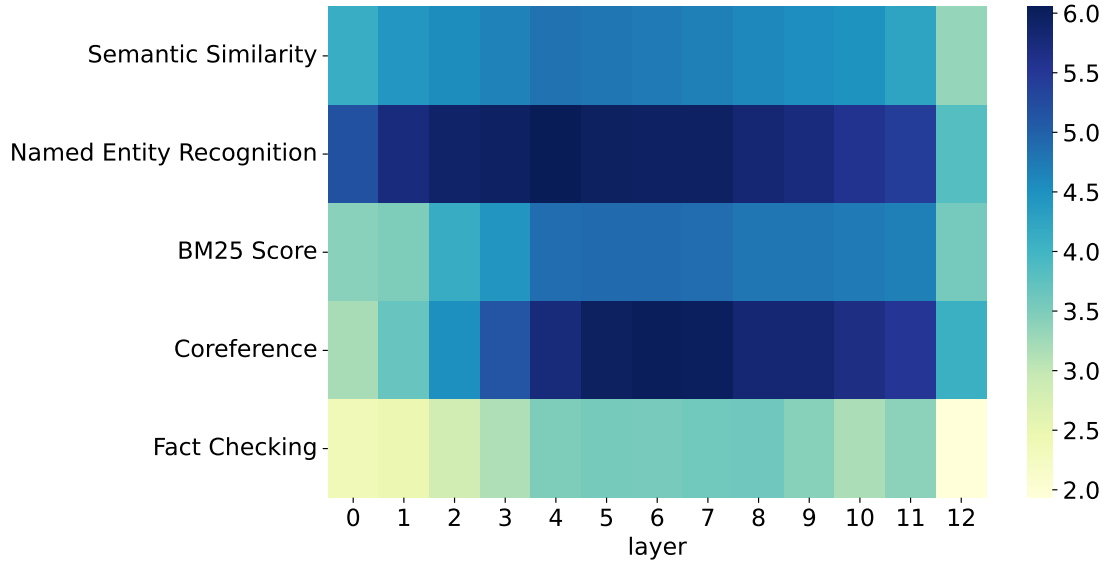

Figure 6.6: *bert-msm-passage* - compression as a function of task and layer, absolute values.

ranking. Likewise, COREF related information appears to shift towards higher layers, though with a distribution that concentrates more around a single layer. On the contrary,

the fact checking property appears to spread further across layers after fine-tuning.

Lastly, we can use a heatmap of the absolute compression values to visualize how well knowledge with respect to different tasks is extractable from *bert-msm-passage*. While it was already recognizable from the line-plots that for the SEM and BM25 properties, *bert-msm-passage* representations achieve higher compression, from Figure 6.6 it becomes much more apparent that the COREF property is easier to decode from the embeddings than prior fine-tuning with almost double the compression across the board, which may be evidence that entity matching is an important part of a neural ranking model.

7 Informed Multi-Task Learning

Based on our findings from the probing experiments, we now want to address the question on whether the knowledge obtained from probing can be leveraged to improve the BERT model for ranking.

Thus, our new objective becomes re-ranking: Given a query q and a set of candidate documents $C = \{c_i\}_{i=1}^{|C|} \subseteq D$, we want to learn a ranker $s : Q \times D \to \mathbb{R}$, such that for any two candidate documents $c, c' \in C$, it holds true that $s(q, c) > s(q, c')$ if c is more relevant than than c', regarding q.
During probing we made these key observations:

- Different properties are best captured around specific layers.
- Fine-tuning for ranking may amplify the presence of a property in specific layers.

Therefore, we want our experiment to satisfy the following criterions:

- Exploit the fact that ranking properties emerge in intermediate layers.
- Use the knowledge on which layers capture what property best.

The main idea of our approach is to aid fine-tuning by infusing task knowledge into specific layers through *multi-task learning*. We want to simultaneously learn ranking properties at the layers that we found to be best at capturing them, especially when being fine-tuned for ranking.

To achieve this, for each task, we first select a layer that we found to efficiently encode the corresponding ranking property during probing. We then fine-tune the *bert-base-uncased* model to perform ranking on the TREC2019 dataset. At the same time, we jointly learn classifiers on top of BERT's intermediate layer representations of the pre-selected layers, to predict the respective ranking properties.

7.1 Model Architecture

Given the intermediate layer representations of the pre-trained *bert-base-uncased* model $\{H^{(i)}\}_{i=0}^{12}$, with $H^{(i)} = (h_1, h_2, \ldots, h_N)$ being the sequence of token embeddings at layer i, for each task (Section 5.1), we select a layer based on the probing results.

We then apply average pooling across the sequence dimension to retrieve a fixed size embedding:

$$\text{pool}(h_i, h_{i+1}, \ldots, h_j) = \frac{1}{j-i} \sum_{k=i}^{j} h_k \tag{7.46}$$

In the case of tasks that require multiple spans, we first average along each span $(i, j) \in S$ and then concatenate the resulting vectors:

$$\text{multi-pool}(h_1, \ldots, h_N) = \Big\|_{(i,j)\in S} \text{pool}(h_i, h_{i+1}, \ldots, h_j) \tag{7.47}$$

Following the pooling we apply a simple MLP classifier of the form:

$$\text{MLP}(x) = \text{ReLU}(xW^{(0)} + b^{(0)})W^{(1)} + b^{(1)} \tag{7.48}$$

7.2 Experimental Setup

7.2.1 Datasets and Sampling

Because our objective is now re-ranking on TREC2019, we can no longer rely on the original probing datasets, as they were sampled from the test set and therefore would cause test set overlap. Instead, we sample new query-document pairs from the TREC2019 train set. Our sampling goes as follows: We uniformly sample 100k train queries for each task. Then, for each query we retrieve 10 documents from the corpus using BM25, resulting in a dataset size of 1M samples which is approximately the size of the TREC2019 ranking dataset. Each task dataset is then constructed by applying the same procedure as in Section 4.2 and Section 5.1, to automatically generate labels.

We exclude the fact-checking task, as we do not have a straightforward way to automatically generate samples and the dataset itself is too small when compared to the other datasets. Furthermore, due to time constraints we do not include COREF when training more than 2 tasks jointly.

7.2.2 Training

During training, we assemble mini-batches of size 32, by sampling from the TREC2019 train set and all generated task datasets with a probability proportional to their size. As loss function we use the objective proposed in (Aghajanyan et al., 2021):

$$\mathcal{L}(y, \hat{y}) = \sum_{t \in \text{tasks}} \frac{\text{CE}(y_t, \hat{y}_t)}{\log c_t} \tag{7.49}$$

where c_t is the number of target classes and y_t, $\hat{y}_t$ are predictions and ground-truth labels with respect to task t, respectively. It scales each task loss, such that all losses would have equivalent values, if the class distribution were uniformly distributed, along with the predictions. Analogous to the probing experiments, regression tasks are cast to classification tasks by binning the targets into $k = 10$ categories, so that c_t is defined properly. We train each model for up to a maximum of 100 epochs and perform early stopping after 3 epochs of no improvement in MAP on the TREC2019 validation set. As optimizer, we use Adam (Kingma and Ba, 2014) with a learning rate of 2e-6 and linearly increase the learning rate over the first $10k$ steps. Each task specific classifier has a hidden size of 128 and dropout with rate 0.2 is applied before each layer.

In addition to binary cross-entropy loss, we further experiment with a pair-wise margin loss for the ranking task:

$$\text{MarginLoss}(y^+, y^-) = \max(\lambda - y^+ + y^-, 0) \tag{7.50}$$

where $\lambda = 0.2$ is the margin and y^+, y^- are the model's predictions for a relevant and non-relevant document, respectively.

Each model is trained on a server with 256GB of system memory, using either a single Nvidia A100 or V100 GPU.

7.3 Results

7.3.1 Single Additional Task

For runs in which a single additional task is introduced (Table 7.1), we can observe a general improvement across the board. It also becomes apparent that the selection of layer which the additional task is trained on actually plays a role: For BM25, SEM and COREF, layer 12 shows worse or equal performance for most metrics, when compared to layers that were selected based on our probing results. For instance, BM25 consistently outperforms the baseline at layers 5 and 6, but performs significantly worse than the baseline at layer 12.

Interestingly, this does not apply to NER which, except from MRR, instead performs best at the final layer. We hypothesize that, due to the NER property having a rather flat distribution across layers (Figure 6.3), infusing additional NER information might work equally well at most layers.

In terms of MAP, we also find NER to yield the highest benefit while the highest increase in MRR, NDCG and precision stems from the additional COREF task. Considering that probing showed the most absolute increase in COREF information when fine-tuning for ranking (Figure 6.5, Figure 6.6), this might be additional evidence for coreference being a useful ability for a neural ranker to learn. Moreover, the performance of NER which seems to be closely related to COREF, suggests that entities in general play an important role for learning to rank.

Tasks	Layer	MAP	MRR	NDCG@10	P@10	avg
TREC	12	0.436	0.926	0.678	0.784	0.706
+BM25	5	0.437	0.947	0.682	**0.791**	0.714
	6	0.439	**0.953**	**0.690**	0.772	0.714
	12	0.420	0.912	0.659	0.749	0.685
+NER	4	**0.447**	**0.950**	0.685	0.788	**0.717**
	5	**0.444**	0.934	0.680	0.772	0.708
	12	**0.447**	0.944	**0.688**	**0.791**	**0.717**
+SEM	1	0.436	0.934	0.682	0.784	0.709
	4	0.440	0.928	0.682	0.779	0.707
	12	0.436	0.928	0.669	0.774	0.702
+COREF	5	0.442	**0.965**	**0.694**	**0.798**	**0.725**
	7	0.425	0.944	0.668	0.770	0.702
	12	0.442	0.948	0.681	0.788	0.715

Table 7.1: Test results for single additional task runs. The first row corresponds to a baseline that was solely trained on TREC2019. +[TASK] indicates multi-task training on TREC2019 and [TASK] simultaneously. The top 3 runs for each metric are highlighted in bold.

7.3.2 Full Multi-task

The results for the full multi-task runs are shown in Table 7.2. Surprisingly, all runs perform equal to or worse than the baseline, with a single exception being the layer combination $5, 1, 12$ on MRR. One hypothesis why this might be the case, is that the ranking task is simply overshadowed by the other tasks, since with this setup only $\frac{1}{4}$ of the dataset specifically addresses ranking. To test this hypothesis, we did perform another run where each additional task was limited to 200k samples each. However, we could not observe any improvement from this approach.

Another hypothesis suggests that too many tasks start to conflict with one another, leading to hidden representations that are no longer coherent with each other and as a consequence, less useful for learning the ranking task. Interestingly, (Aghajanyan et al., 2021) found their multi-task learning setup to perform best when at least 10 tasks were used, meaning it is also possible that 3 additional tasks are simply not sufficient.

Nonetheless, we can still observe that selecting intermediate layers based on probing metrics generally results in higher performance than simply employing the additional task at layer 12.

Tasks	Layers	MAP	MRR	NDCG@10	P@10	avg
TREC	12	**0.436**	0.926	**0.678**	**0.784**	**0.706**
+BM25, +SEM, +NER	5, 1, 12	0.425	**0.950**	0.675	0.772	0.705
	5, 4, 6	0.431	0.930	0.670	0.758	0.697
	6, 4, 12	0.423	0.891	0.656	0.749	0.680
	5, 5, 5	**0.436**	0.911	0.677	0.767	0.698
	12, 12, 12	0.414	0.921	0.656	0.735	0.681

Table 7.2: Test results for joint training using 3 additional tasks. Best scoring metrics are highlighted in bold.

7.3.3 Pair-wise Objective

Again, we report the results for our pair-wise experiments in Table 7.3. This time we report both, runs with one and multiple additional tasks in a single table. Like in the point-wise experiments, with a single additional task we do observe a general improvement over the baseline. This time NER does not provide as much of an improvement as in the point-wise scenario. Solely the MAP score improves while the other metrics even degrade compared to the baseline. On the other hand, similar to the point-wise approach, COREF improves in NDCG and precision. However, it significantly drops in MRR, suggesting that, while the overall ordering of relevant documents improves, the most relevant document is less frequently ranked in top positions.

The decrease in MRR seems to be a general pattern for pair-wise training. When considering that the pair-wise loss focuses on ordering documents with respect to each other, instead of learning whether a single document is relevant or not, this appears makes sense: It becomes less likely for the model to predict strong positives which are then placed in the top positions.

Most surprisingly, in the pair-wise scenario the three task MTL setup still improves over the baseline and even results in the highest average across metrics. This might indicate that having a non-classification ranking loss does create less conflict with the other classification tasks than a binary classification loss.

7.4 Additional Ablation Study

The prior experiment has shown that explicitly infusing additional task information at specific layers, can result in increased ranking performance. Because of this, we further want to investigate whether this could be a valid augmentation technique in a limited data scenario.

For this, we conduct the following ablation study: Given a subset of TREC2019 query-document pairs, we resample multiple times from this subset and automatically create

Tasks	Layers	MAP	MRR	NDCG@10	P@10	avg
TREC	12	0.433	**0.965**	0.681	0.772	0.713
+BM25	5	**0.452**	**0.965**	0.685	0.786	0.722
+NER	5	0.445	0.953	0.667	0.767	0.708
+SEM	1	0.450	0.951	0.687	0.781	0.717
+COREF	5	0.451	0.924	**0.701**	**0.798**	0.719
+FIRST 3	5,1,5	0.449	0.959	**0.701**	0.791	**0.725**

Table 7.3: Test results for multi-task training with the pair-wise ranking objective. FIRST 3 refers to BM25, NER and SEM.

additional training samples for each of the tasks used in the MTL experiment. We then proceed to train using our proposed multi-task method (point-wise) and repeat the experiment for different subset sizes and MTL configurations.

7.4.1 Results

In the following tables we present the results of our ablation study for different sample sizes and MTL setups. The first table (Table 7.4) represents the baseline which did not use any MTL augmentation.

At first, we tried limiting the augmentation to a single task (BM25), since this has been shown to work best in our previous experiments. However, when resampling from the same small set, we can only observe a consistent benefit at sample size $100,000$ (Table 7.5).

We then continued to sample from all tasks which resulted in an overall drop in performance compared to the baseline (Table 7.6). Suspecting that the other tasks overshadowed ranking, we further tried limiting the total sample size of additional tasks to 50% of the ranking sample size (Table 7.7). However, while this approach performed better than using the full sample size, it was still outperformed by the baseline.

#samples	MAP	MRR	NDCG@10	P@10
1000	0.379	0.844	0.545	0.653
10,000	0.401	0.917	0.623	0.730
100,000	0.423	0.917	0.651	0.760
500,000	0.424	0.922	0.673	0.784
1,000,000	0.465	0.915	0.691	0.805

Table 7.4: Test results for model trained on limited sample sizes of TREC2019.

#samples	MAP	MRR	NDCG@10	P@10
1000	0.355	0.864	0.548	0.633
10,000	0.389	0.867	0.595	0.686
100,000	0.431	0.940	0.674	0.765
500,000	0.421	0.947	0.676	0.765
1,000,000	0.430	0.919	0.683	0.770

Table 7.5: Test results for model trained jointly on limited TREC2019 sample and generated BM25 dataset of equal size.

#samples	MAP	MRR	NDCG@10	P@10
1000	0.344	0.850	0.527	0.628
10,000	0.395	0.884	0.600	0.693
100,000	0.425	0.940	0.657	0.758
500,000	0.424	0.912	0.658	0.753
1,000,000	0.454	0.953	0.676	0.770

Table 7.6: Test results for model trained jointly on limited TREC2019 sample and generated BM25, SEM and NER datasets of equal size.

#samples	MAP	MRR	NDCG@10	P@10
1000	0.346	0.828	0.532	0.644
10,000	0.390	0.921	0.626	0.721
100,000	0.408	0.922	0.647	0.747
500,000	0.431	0.922	0.665	0.765
1,000,000	0.432	0.961	0.662	0.751

Table 7.7: Test results for model trained jointly on limited TREC2019 sample and generated BM25, SEM and NER datasets with each dataset being 50% of the sample size.

8 Conclusion

Throughout this thesis we've explored to what extent different ranking abilities are encoded by a BERT model's word representations. Through probing, we found that different layers are better at encoding certain properties. Whereas lower level layers were better at capturing task knowledge that is helpful for simple, match based query-document relations (SEM, BM25, NER), we found mid- to high-level layers to hold more complex semantic information (COREF, FACT CHECK).

Furthermore, we've investigated how fine-tuning affects the distribution of properties by probing a version of the model that has been adapted for ranking. We were able to observe a shift in distribution towards mid- and upper-layers and a consistent loss of information in the final layer, regardless of the task. In addition, by comparing absolute compression values between tasks, we found that the overall presence of property information increased through fine-tuning, suggesting that these properties are indeed relevant for learning a ranking model (excluding FACT CHECK). Interestingly, this effect was especially pronounced with the COREF property.

Then, based on our findings, we've designed a new multi-task learning setup to aid fine-tuning BERT for ranking and found that infusing task information at specific layers can indeed help to learn a better ranking model and that the choice of layer does actually matter.

For this, we've investigated two ranking objectives, point-wise and pair-wise, with different additional task combinations. Through this, we found that for the point-wise objective, providing a single additional task would result in increased ranking performance. However, when further increasing the number of tasks the performance would drop. Surprisingly, we could not observe this decrease in performance when using the pair-wise ranking objective. Instead, the overall performance increased nonetheless.

8.1 Limitations

Firstly, there is a general limitation that arises when using the probing framework. Even though a probe can help us estimate whether a certain property is extractable from a model's representations, we can not conclude that the model itself actually uses that information during inference. While we can gather more evidence on whether a property is important for a downstream task, by comparing pre-trained embeddings to a fine-tuned model, this is still no guarantee.

Another problem is the quality of automatically generated task data. Because some of our tools used for label generation also rely on learned models, this will certainly result in some wrong predictions, meaning our task labels are noisy to a certain extent, ultimately causing less reliable probing results.

Regarding our MTL experiments, while we could observe a general improvement in ranking performance across the board, this improvement was rather marginal. Whether this is due to the model being too simple, lacking quality in additional task data, or simply because the model already has access to most of the infused information, has yet to be explored.

8.2 Future Work

Considering the limitations of this thesis, we suggest the following directions for future work:

- There are still more ranking properties that one might think of. Probing BERT for additional ranking properties can provide further insights on neural ranking.
- By using more sophisticated approaches, the automatic label extraction process might be improved, in order to produce higher quality probing datasets.
- In this thesis we've only used the TREC2019 dataset. Leveraging additional datasets might give us more general insights and also help with MTL learning.
- Since we've used a fairly simple MTL architecture, it might be beneficial to extend our approach with more advanced architectural designs.

Whereas this is just a small selection of potential improvements building on this thesis, there is still much more work to be done in understanding the capabilities of neural ranking models. Interpretability techniques other than probing are undoubtedly equally important to explore, if we want to gain an in-depth understanding of these models.

Plagiarism Statement

I hereby confirm that this thesis is my own work and that I have documented all sources used.

Hannover, xx.xx.2022

(Fabian Beringer)

List of Figures

List of Tables

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Manè, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viègas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

A. Aghajanyan, A. Gupta, A. Shrivastava, X. Chen, L. Zettlemoyer, and S. Gupta. Muppet: Massive multi-task representations with pre-finetuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5799–5811, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.468. URL `https://aclanthology.org/2021.emnlp-main.468`.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

J. Bai, J.-Y. Nie, G. Cao, and H. Bouchard. Using query contexts in information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, page 15–22, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935977. doi: 10.1145/1277741.1277747. URL `https://doi.org/10.1145/1277741.1277747`.

Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, and J. R. Glass. What do neural machine translation models learn about morphology? In *ACL*, 2017.

N. J. Belkin et al. Interaction with texts: Information retrieval as information seeking behavior. *Information retrieval*, 93(55-66), 1993.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL `https://arxiv.org/abs/2005.14165`.

C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. *Acm Computing Surveys (CSUR)*, 44(1):1–50, 2012.

L. Chen, Z. Tang, and G. H. Yang. Balancing reinforcement learning training experiences in interactive information retrieval. *CoRR*, abs/2006.03185, 2020a. URL `https://arxiv.org/abs/2006.03185`.

S. Chen, Y. Zhang, and Q. Yang. Multi-task learning in natural language processing: An overview. *CoRR*, abs/2109.09138, 2021. URL `https://arxiv.org/abs/2109.09138`.

X. Chen, B. He, K. Hui, L. Sun, and Y. Sun. Simplified tinybert: Knowledge distillation for document retrieval. *CoRR*, abs/2009.07531, 2020b. URL `https://arxiv.org/abs/2009.07531`.

J. Choi, E. Jung, S. Lim, and W. Rhee. Finding inverse document frequency information in bert, 2022. URL `https://arxiv.org/abs/2202.12191`.

P. R. Christopher D. Manning and H. Schütze. Inroduction to information retrieval. 2008. URL `https://nlp.stanford.edu/IR-book/`.

K. Clark, M. Luong, Q. V. Le, and C. D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555, 2020. URL `https://arxiv.org/abs/2003.10555`.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390177. URL `https://doi.org/10.1145/1390156.1390177`.

A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. 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. doi: 10.18653/v1/P18-1198. URL https://aclanthology.org/P18-1198.

N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and E. M. Voorhees. Overview of the TREC 2019 deep learning track. *CoRR*, abs/2003.07820, 2020. URL https://arxiv.org/abs/2003.07820.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N19-1423.

P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever. Jukebox: A generative model for music, 2020. URL https://arxiv.org/abs/2005.00341.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

H. Fun, S. Gandhi, and S. Ravi. Efficient retrieval optimized multi-task learning. *ArXiv*, abs/2104.10129, 2021.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng. A deep look into neural ranking models for information retrieval. *ArXiv*, abs/1903.06902, 2020.

J. Hewitt and P. Liang. Designing and interpreting probes with control tasks. *CoRR*, abs/1909.03368, 2019a. URL http://arxiv.org/abs/1909.03368.

J. Hewitt and P. Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China, Nov. 2019b. Association for Computational Linguistics. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275.

J. Hewitt and C. D. Manning. A structural probe for finding syntax in word representations. In *NAACL*, 2019.

G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *University of Toronto, Technical Report*, 2012a.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012b. URL `http://arxiv.org/abs/1207.0580`.

S. Hofstätter, M. Zlabinger, and A. Hanbury. TU wien @ TREC deep learning '19 - simple contextualization for re-ranking. *CoRR*, abs/1912.01385, 2019. URL `http://arxiv.org/abs/1912.01385`.

S. Hofstätter, S. Althammer, M. Schröder, M. Sertkan, and A. Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *CoRR*, abs/2010.02666, 2020. URL `https://arxiv.org/abs/2010.02666`.

S. Hofstätter, H. Zamani, B. Mitra, N. Craswell, and A. Hanbury. Local self-attention over long text for efficient document retrieval. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

M. Honnibal and I. Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

J. Howard and S. Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018. URL `http://arxiv.org/abs/1801.06146`.

S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SkxgnnNFvH`.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

J. Kekäläinen and K. Järvelin. Evaluating information retrieval systems under the challenges of interaction and multidimensional dynamic relevance. 2002.

O. Khattab and M. Zaharia. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*, page 39–48. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450380164. URL `https://doi.org/10.1145/3397271.3401075`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

K. Kurita, N. Vyas, A. Pareek, A. W. Black, and Y. Tsvetkov. Measuring bias in contextualized word representations. *arXiv preprint arXiv:1906.07337*, 2019.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1eA7AEtvS`.

K. Lee, L. He, M. Lewis, and L. Zettlemoyer. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1018. URL `https://aclanthology.org/D17-1018`.

K. Lee, L. He, and L. Zettlemoyer. Higher-order coreference resolution with coarse-to-fine inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 687–692, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2108. URL `https://aclanthology.org/N18-2108`.

X. Liu, P. He, W. Chen, and J. Gao. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504, 2019a. URL `http://arxiv.org/abs/1901.11504`.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019b. URL `http://arxiv.org/abs/1907.11692`.

M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

J. Maillard, V. Karpukhin, F. Petroni, W.-t. Yih, B. Oguz, V. Stoyanov, and G. Ghosh. Multi-task retrieval for knowledge-intensive tasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1098–1111, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.89. URL `https://aclanthology.org/2021.acl-long.89`.

A. Merchant, E. Rahimtoroghi, E. Pavlick, and I. Tenney. What happens to BERT embeddings during fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop on*

Analyzing and Interpreting Neural Networks for NLP, pages 33–44, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.blackboxnlp-1.4. URL https://aclanthology.org/2020.blackboxnlp-1.4.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

B. Mitra and N. Craswell. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, 2018. URL https://www.microsoft.com/en-us/research/publication/introduction-neural-information-retrieval/.

B. Mitra, S. Hofstätter, H. Zamani, and N. Craswell. Improving transformer-kernel ranking model using conformer and query term independence. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 1697–1702, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3463049. URL https://doi.org/10.1145/3404835.3463049.

M. Nadeem, A. Bethke, and S. Reddy. Stereoset: Measuring stereotypical bias in pretrained language models. In *ACL*, 2021.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

R. Nogueira and K. Cho. Passage re-ranking with bert. *ArXiv*, abs/1901.04085, 2019.

R. Nogueira, W. Yang, K. Cho, and J. J. Lin. Multi-stage document ranking with bert. *ArXiv*, abs/1910.14424, 2019.

K. D. Onal, Y. Zhang, I. S. Altingövde, M. M. Rahman, P. Senkul, A. Braylan, B. Dang, H.-L. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. de Rijke, and M. Lease. Neural information retrieval: at the end of the early years. *Information Retrieval Journal*, 21:111–182, 2017.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018a.

M. E. Peters, M. Neumann, L. Zettlemoyer, and W.-t. Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium, Oct.-Nov. 2018b. Association for Computational Linguistics. doi: 10.18653/v1/D18-1179. URL `https://aclanthology.org/D18-1179`.

J. Phang, T. Févry, and S. R. Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *CoRR*, abs/1811.01088, 2018. URL `http://arxiv.org/abs/1811.01088`.

T. Pimentel, J. Valvoda, R. Hall Maudslay, R. Zmigrod, A. Williams, and R. Cotterell. Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4622, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.420. URL `https://aclanthology.org/2020.acl-main.420`.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL `http://arxiv.org/abs/1910.10683`.

N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL `http://arxiv.org/abs/1908.10084`.

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL https://doi.org/10.1145/2939672.2939778.

J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Trans. Inf. Theory*, 30:629–636, 1984.

S. Robertson, H. Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109:247–278, 2021.

V. Sanh, T. Wolf, and S. Ruder. A hierarchical multi-task approach for learning embeddings from semantic tasks. *CoRR*, abs/1811.06031, 2018. URL http://arxiv.org/abs/1811.06031.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL http://arxiv.org/abs/1909.08053.

L. Tamine and M. Daoud. Evaluation in contextual information retrieval: Foundations and recent advances within the challenges of context dynamicity and data privacy. *ACM Comput. Surv.*, 51(4), jul 2018. ISSN 0360-0300. doi: 10.1145/3204940. URL https://doi.org/10.1145/3204940.

I. Tenney, D. Das, and E. Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy, July 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452.

I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. V. Durme, S. R. Bowman, D. Das, and E. Pavlick. What do you learn from context? probing

for sentence structure in contextualized word representations. *ArXiv*, abs/1905.06316, 2019b.

J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1074. URL `https://aclanthology.org/N18-1074`.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

E. Voita and I. Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.14. URL `https://aclanthology.org/2020.emnlp-main.14`.

J. Wallat, J. Singh, and A. Anand. BERTnesia: Investigating the capture and forgetting of knowledge in BERT. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 174–183, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.blackboxnlp-1.17. URL `https://aclanthology.org/2020.blackboxnlp-1.17`.

J. Worsham and J. Kalita. Multi-task learning for natural language processing in the 2020s: where are we going? *CoRR*, abs/2007.16008, 2020. URL `https://arxiv.org/abs/2007.16008`.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. R. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. S. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019. URL `http://arxiv.org/abs/1906.08237`.

A. Yates, R. Nogueira, and J. Lin. Pretrained transformers for text ranking: Bert and beyond. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 1154–1156, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382977. doi: 10.1145/3437963.3441667. URL `https://doi.org/10.1145/3437963.3441667`.

H. Zhang, L. Xiao, Y. Wang, and Y. Jin. A generalized recurrent neural architecture for text classification with multi-task learning. *CoRR*, abs/1707.02892, 2017. URL `http://arxiv.org/abs/1707.02892`.

K. Zhang and S. Bowman. Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 359–361, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5448. URL `https://aclanthology.org/W18-5448`.

Q. Zhang and S. Zhu. Visual interpretability for deep learning: a survey. *CoRR*, abs/1802.00614, 2018. URL `http://arxiv.org/abs/1802.00614`.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27, 2015. doi: 10.1109/ICCV.2015.11.