Deconstructing Ranking Abilities of Language Models

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

Thesis by
Fabian Beringer

First examiner:	xxxx
Second examiner:	xxxx
Advisors:	Abhijit Aanand
	Jonas Wallat

Hannover, xx.xx.2022

Abstract

...

Contents

1 Introduction

general retrieval ?

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* [Vas+17] 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. 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 [NBR21; Ben+21; Kur+19], falsely motivated predictions [RSG16; ZZ18] or simply trying to determine causes of prediction error. In addition, a better understanding might provide insights on model weaknesses and guide model improvement, e.g. when adapting 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. 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 tags from the representations.

while has recently emerged for nlp, but retrieval which important: google guides how we access information etc

we look at bert, initial break through approach, more recent iterations still similar in essence

1.2 Problem Statement

research question:

- given set of tasks, presumably ranking related, which knowledge does bert encode?
- how does ranking trained model differ from base language model
- can we exploit these differences to inform training for ranking?

1.3 Contribution

1.4 Thesis Outline

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 this 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. Further, the retrieved documents should be sorted by relevance w.r.t. the user's information need in descending order, i.e. the documents considered most relevant should be at the top of the list.

While from this formulation only, the task might appear simple, 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. 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 in 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 specified through an exact query from the beginning on.

Further, a query might require additional context information in order for an IR system to find relevant documents. For example, depending on the time at which a query is prompted, the correct answer might change: "Who is 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 which country's president the user is interested in and might depend on their location. In addition, even the corpus might not be static either and change or grow over time, e.g. 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 w.r.t q than d'.

To give the reader a more concrete idea and as we are going to build upon it throughout this work, we will now discuss two traditional approaches to text retrieval which, unlike neural retrieval, are based on exact matching, meaning query and document terms are compared directly. Further, 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 often.
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 frequent relative to the corpus size will receive a high IDF score and those that are more frequent 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:

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

Alternatively, vector space idf vector

2.1.2 BM25

2.2 Machine Learning

Machine learning 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.

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 $C = \{c_i\}_{i=1}^{|C|}$, this is called a classification problem.

Now given the training data and labels, the goal is to find a hypothesis that explains the 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 can act as a proxy for estimating the error on unseen samples.

In contrast, in 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 that, if each X is already a vector, tries to compress them into more compact but still informative representations.

That being said, the separating lines between supervised and unsupervised learning are 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.

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.

$$f(x) = \sigma(Wx + b) \tag{2.4}$$

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 require linear separation. In order to learn more complex, non-linear decision boundaries, multiple layers can be applied in sequence.

An L-layer DNN can be described as follows:

$$\begin{aligned} h^{(1)} &= \sigma^{(1)}(W^{(1)}x + b^{(1)}) \\ h^{(2)} &= \sigma^{(2)}(W^{(2)}h^{(1)} + b^{(2)}) \\ &\vdots \\ f(x) &= \sigma^{(L)}(W^{(L)}h^{(L-1)} + b^{(L)}) \end{aligned} \tag{2.5}$$

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 [NH10])

2.3.2 Optimization

Arguably, the most common way for optimizing a neural network are the gradient descent (GD) 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.

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; \theta), y_i) \tag{2.6}$$

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

Gradient Descent

For GD, 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.

$$\theta_{new} = \theta - \eta \nabla_\theta J(\theta) \tag{2.7}$$

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

- **Cross Entropy Loss**

  $$\mathrm{CE}(y, \hat{y}) = -\sum_{k=1}^{C} y_k \log \hat{y}_k \tag{2.8}$$

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

- **Mean Squared Error**

  $$\mathrm{MSE}(y, \hat{y}) = (y - \hat{y})^2 \tag{2.9}$$

  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 less redundant computations are performed, due to its stochastic nature and high variance, it is more likely to break out of local minima, allowing additional exploration for better solutions. [Rud16]

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 1 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 GD, many algorithms have been introduced in the context of DNNs, that employ further optimizations in order to improve convergence speed and quality. Notable examples include:

- Adagrad [DHS11]
- RMSProp [HSS12]
- Adam [KB14]

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 can consist of multiple layers and thus, is a composition of multiple non-linear functions, computing the gradient w.r.t. to 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 [RHW+88].

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.

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x} \tag{2.10}$$

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 2.5). 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:

$$\frac{\partial J}{\partial a^{(L)}} = \frac{\partial J}{\partial \sigma(a^{(L)})} \frac{\partial \sigma(a^{(L)})}{\partial a^{(L)}} \tag{2.11}$$

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

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

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 to 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 [Pas+19] or Tensorflow [Mar+15] can automatically perform 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 model 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.

$$J(\theta) = \mathcal{L}(\theta) + \lambda ||\theta||_2^2 \tag{2.13}$$

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 [Hin+12]. 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 seen 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 [Vas+17]. The transformer and its variants have set multiple state-of-the-art records in a variety of NLP tasks [Dev+19; Liu+19; Cla+20; Sho+19; Bro+20], and have since then also been adapted to other domains such as computer vision [Dos+20] or audio generation [Dha+20]. 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 [Dev+19].

unlike previous methods like word embeddings, contextualized ...

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 inputs, *positional encodings* are added to the initial embeddings, in order to inject 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, [Vas+17] propose to use sine and cosine waves, as a function of the position:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.14}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.15}$$

Where *pos* and i denote position along in along sequence and hidden dimension respectively. They found this approach to perform nearly identical to learned embeddings in the case of machine translation.

2.4.3 Multi-Head Self-Attention

The Attention Mechanism

Originally, attention mechanisms have been proposed in the context of neural machine translation (NMT) [BCB14; LPM15]. 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) [XZS16] or patches of pixels in an image (inputs) that are important for detecting a certain object type (context) [Xu+15].

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:

$$s_{ij} = a(x_i, c_j) \tag{2.16}$$

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^{N} \exp(s_{kj})} \tag{2.17}$$

$$h_j = \sum_{k=1}^{N} \alpha_{kj} x_k \tag{2.18}$$

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 of 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 to a sequence, given the sequence itself as context. In [Vas+17] 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 $W^{(v)} \in \mathbb{R}^{d \times d_v}$.

$$Q = XW^{(q)} \tag{2.19}$$

$$K = XW^{(k)} \tag{2.20}$$

$$V = XW^{(v)} \tag{2.21}$$

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

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.22}$$

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.

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:

$$\text{MultiHeadAttention}(Q, K, V) = \left[\Big\|_{i=1}^{h} \text{SelfAttention}_i(Q, K, V) \right] W^{(o)} \tag{2.23}$$

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:

$$\text{FFN}(x) = \text{ReLU}(xW^{(0)} + b^{(0)})W^{(1)} + b^{(1)} \tag{2.24}$$

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, it is called a residual connection or skip connection:

$$\text{Residual}(x) = f^{(k)}(x) + x \tag{2.25}$$

Where $f^{(k)}(x)$ is a 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 [BKH16] 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 [IS15]. Layer Normalization tackles this problem by computing mean $\mu^{(k)}$ and standard deviation $\sigma^{(k)}$ over the feature dimension of each hidden layer k:

$$\mu^{(k)} = \frac{1}{d}\sum_{i=1}^{d} z_i^{(k)} \tag{2.26}$$

$$\sigma^{(k)} = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(z_i^{(k)} - \mu^{(k)})^2} \tag{2.27}$$

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 $z^{(k)}$:

$$\text{LayerNorm}(z^{(k)}) = \gamma^{(k)} \circ \frac{z^{(k)} - \mu^{(k)}}{\sigma^{(k)}} + \beta^{(k)} \tag{2.28}$$

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:

$$A = \text{MultiHeadAttention}(X, X, X) \tag{2.29}$$

$$Z = \text{LayerNorm}(A + X) \tag{2.30}$$

$$\text{TBlock}(X) = \text{LayerNorm}(\text{FFN}(Z) + Z) \tag{2.31}$$

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 in 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 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 state-of-the-art on several NLP benchmarks, is BERT (Bidirectional Encoder Representations from Transformers) [Dev+19]. Unlike previous pre-training approaches that optimized for traditional *left-to-right* language modeling [HR18; Pet+18; Rad+18], 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.

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 [Cla+20; Liu+19; Yan+19; Lan+20], MLM still remains an integral part of most of these approaches.

2.5 Probing

formal definition

2.6 MTL

3 Previous Work

tenney, hewitt all the good probing effects of finetuning https://arxiv.org/abs/2004.14448
the idf paper probing different berts bertnesia

4 Datasets

4.1 TREC 2019 - Deep Learning Track

4.1.1 passage level

4.1.2 document level

4.2 Probing

4.2.1 Dataset Generation

For all of our probing tasks, we automatically generate datasets from the TREC2019 passage-level test set. We sample 60k

4.2.2 BM25 Prediction

4.2.3 Term Frequency Prediction

4.2.4 Named Entity Recognition

4.2.5 Semantic Similarity

4.2.6 Coreference Resolution

4.2.7 Fact Checking

4.2.8 Relevance Estimation

4.3 Multitask Learning

5 Approach

5.1 Methodology

edge probing graphic

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. Further, 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 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 across layers, to get a relative measure of how task-specific information is distributed throughout $\mathcal{M}$. To further put our measurements into perspective, we also employ a random baseline model $\mathcal{B}$ which is also probed for each task separately.

performance metric $m(P_i, D_i, H)$?

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

5.1.1 Task Design

We propose a set of classification tasks with ranking properties as target variable. In this section we provide a list of the tasks we've chosen and explain the reasoning behind our selection.

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 heuristic designed around exact term matching, BM25 quantifies query-document pairs on a symbolic level, without the notion of higher level semantics.

Being able to decode BM25 from BERT embeddings might give a hint on whether exact matching properties like term-frequency or even corpus level statistics like inverted document-frequency are encoded by the neural ranking models.

Semantic Similarity

To measure semantic similarity, we compute cosine distance between query and document vectors in the GloVe[PSM14] embedding space:

$$\text{Sim}(q, d) = \frac{q^\top d}{||q|| \times ||d||} \tag{5.32}$$

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.

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.

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 PS 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.

Fact Checking

For fact checking, we leverage the existing FEVER [Tho+18] 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.

Relevance Estimation

Because the TREC2019 dataset provides relevance labels, we can also directly probe what layers contain most information with respect to relevance prediction. This especially becomes interesting when comparing between models tuned for ranking and pure language models.

5.1.2 Models

Subject Models

Subject of our probing experiments is the pre-trained BERT [Dev+19] transformer model. In particular, we use the *bert-base-uncased*[1] 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+15] and text passages from English Wikipedia[2], 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 [Cra+20]. While *bert-msm-passage* is trained to predict relevancy of *passages* given a query, *bert-msm-doc* is trained on *document*-level data. Hence, 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.

Probe

One problem in probing arises when trying to choose a probe of appropriate complexity [HL19a]. 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 is recent debate on whether more complex models are actually problematic for probing [Pim+20], following previous literature [Ten+19; TDP19; HL19a] we decide 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 [Ten+19]:

$$\hat{P}(x) = \text{LayerNorm}(\tanh(xW^{(0)} + b^{(0)}))W^{(1)} + b^{(1)} \tag{5.33}$$

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

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 [Ten+19] we employ the simple attention pooling operator also used in [Lee+17; LHZ18]:

$$\begin{aligned} \alpha_n &= \frac{\exp(w^\top h_n)}{\sum_{k=i}^{j} \exp(w^\top h_k)} \\ \text{pool}(h_i, h_{i+1}, \ldots, h_j) &= \sum_{k=i}^{j} \alpha_k (W h_k + b) \end{aligned} \tag{5.34}$$

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:

$$P(h_1, \ldots, h_N) = \hat{P}\Big(\Big\|_{(i,j) \in S} \text{pool}(h_i, h_{i+1}, \ldots, h_j) \Big) \tag{5.35}$$

5.2 Experimental 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 5.1.2), 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 [Dev+19], using the Adam optimizer[KB14] 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. Hence, we truncate any passages and documents exceeding this maximum length, after applying BERT's word-piece tokenization.

5.2.2 Probing

All three subject models (Section 5.1.2) 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 5.1.1).

As objective function, all probing tasks use cross-entropy loss (Equation 2.8). The Adam optimization algorithm [KB14] with a learning rate of 1e-4 and batch size 32 is employed for updating the probe parameters, while the subject model's parameters stay 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 in 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.

5.3 Evaluation Measures

5.3.1 MDL

As mentioned in Section 5.1.2, 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 [ZB18]. Further, [HL19b] 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, [VT20] propose an information theoretic approach for measuring probe performance, instead of accuracy. 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 for achieving this performance, manifesting in model size or amount of training data required.

Since [VT20] 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 [Ris84]. 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:

$$\begin{aligned} &\mathrm{MDL}(y_{1:n}|x_{1:n}) = t_1 \log_2 c \\ &- \sum_{i=1}^{S-1} \log_2 P_{\theta_i}(y_{t_i+1:t_{i+1}}|x_{t_i+1:t_{i+1}}) \end{aligned} \tag{5.36}$$

where c is the number of target classes. Following [VT20], 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 MDL depends on the number of targets in a probing dataset, it is not reasonable to directly compare it between different tasks. A common way to turn MDL into a relative measure, is by computing *compression*. Compression divides the codelength that would result from a uniform encoding by MDL:

$$\text{compression} = \frac{n \log_2(c)}{\text{MDL}(y_{1:n}|x_{1:n})} \tag{5.37}$$

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:

$$\text{Accuracy}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\hat{y}_i = y_i] \tag{5.38}$$

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:

$$\text{MRR}(Q, C) = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}(q, C)} \tag{5.39}$$

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

$$\text{Precision@}k = \left(\frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}} \right)_k \tag{5.40}$$

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:

$$\text{AP}(q, C) = \frac{1}{|C_{rel}|} \sum_{k=1}^{|C|} \text{Precision@k} \times y_k \tag{5.41}$$

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:

$$\text{MAP}(Q, C) = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}(q, C) \tag{5.42}$$

While MRR indicates the quality of 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:

$$\text{DCG} = \sum_{i=1}^{|D|} \frac{y_i}{\log_2(i+1)} \tag{5.43}$$

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

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}} \tag{5.44}$$

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.

6 Results

- old plots - task to layer to map plot

7 Conclusion

7.1 Future Work

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

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[Ben+21] Emily M. Bender et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (2021).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[Bro+20] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[Cla+20] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *CoRR* abs/2003.10555 (2020). arXiv: 2003.10555. URL: https://arxiv.org/abs/2003.10555.

[Cra+20] Nick Craswell et al. "Overview of the TREC 2019 deep learning track". In: *CoRR* abs/2003.07820 (2020). arXiv: 2003.07820. URL: https://arxiv.org/abs/2003.07820.

[Dev+19] Jacob Devlin et al. "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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. URL: https://www.aclweb.org/anthology/N19-1423.

[Dha+20] Prafulla Dhariwal et al. *Jukebox: A Generative Model for Music*. 2020. DOI: 10.48550/ARXIV.2005.00341. URL: https://arxiv.org/abs/2005.00341.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[Dos+20] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[Hin+12] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[HL19a] John Hewitt and Percy 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)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2733–2743. DOI: 10.18653/v1/D19-1275. URL: https://aclanthology.org/D19-1275.

[HL19b] John Hewitt and Percy Liang. "Designing and Interpreting Probes with Control Tasks". In: *CoRR* abs/1909.03368 (2019). arXiv: 1909.03368. URL: http://arxiv.org/abs/1909.03368.

[HR18] Jeremy Howard and Sebastian Ruder. "Fine-tuned Language Models for Text Classification". In: *CoRR* abs/1801.06146 (2018). arXiv: 1801.06146. URL: http://arxiv.org/abs/1801.06146.

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *University of Toronto, Technical Report* (2012).

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[Kur+19] Keita Kurita et al. "Measuring bias in contextualized word representations". In: *arXiv preprint arXiv:1906.07337* (2019).

[Lan+20] Zhenzhong Lan et al. "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.

[Lee+17] Kenton Lee et al. "End-to-end Neural Coreference Resolution". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 188–197. DOI: 10.18653/v1/D17-1018. URL: https://aclanthology.org/D17-1018.

[LHZ18] Kenton Lee, Luheng He, and Luke 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)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 687–692. DOI: 10.18653/v1/N18-2108. URL: https://aclanthology.org/N18-2108.

[Liu+19] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[Mar+15] Martìn Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[NBR21] Moin Nadeem, Anna Bethke, and Siva Reddy. "StereoSet: Measuring stereotypical bias in pretrained language models". In: *ACL*. 2021.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Pet+18] Matthew E. Peters et al. "Deep contextualized word representations". In: *Proc. of NAACL*. 2018.

[Pim+20] Tiago Pimentel et al. "Information-Theoretic Probing for Linguistic Structure". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 4609–4622. DOI: 10.18653/v1/2020.acl-main.420. URL: https://aclanthology.org/2020.acl-main.420.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[Rad+18] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[RHW+88] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[Ris84] Jorma Rissanen. "Universal coding, information, prediction, and estimation". In: *IEEE Trans. Inf. Theory* 30 (1984), pp. 629–636.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[Sho+19] Mohammad Shoeybi et al. "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism". In: *CoRR* abs/1909.08053 (2019). arXiv: 1909.08053. URL: http://arxiv.org/abs/1909.08053.

[TDP19] Ian Tenney, Dipanjan Das, and Ellie Pavlick. "BERT Rediscovers the Classical NLP Pipeline". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 4593–4601. DOI: 10.18653/v1/P19-1452. URL: https://aclanthology.org/P19-1452.

[Ten+19] Ian Tenney et al. "What do you learn from context? Probing for sentence structure in contextualized word representations". In: *ArXiv* abs/1905.06316 (2019).

[Tho+18] James Thorne et al. "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)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 809–819. DOI: 10.18653/v1/N18-1074. URL: https://aclanthology.org/N18-1074.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[VT20] Elena Voita and Ivan Titov. "Information-Theoretic Probing with Minimum Description Length". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 183–196. DOI: 10.18653/v1/2020.emnlp-main.14. URL: https://aclanthology.org/2020.emnlp-main.14.

[Xu+15] Kelvin Xu et al. "Show, attend and tell: Neural image caption generation with visual attention". In: *International conference on machine learning*. PMLR. 2015, pp. 2048–2057.

[XZS16] Caiming Xiong, Victor Zhong, and Richard Socher. "Dynamic coattention networks for question answering". In: *arXiv preprint arXiv:1611.01604* (2016).

[Yan+19] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[ZB18] Kelly Zhang and Samuel 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*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 359–361. DOI: 10.18653/v1/W18-5448. URL: https://aclanthology.org/W18-5448.

[Zhu+15] Yukun Zhu et al. "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.11.

[ZZ18] Quanshi Zhang and Song-Chun Zhu. "Visual Interpretability for Deep Learning: a Survey". In: *CoRR* abs/1802.00614 (2018). arXiv: 1802.00614. URL: http://arxiv.org/abs/1802.00614.