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

Due to large successes in computer vision, natural language processing (NLP) and many other fields, machine learning (ML) and in particular deep learning (DL) have gained popularity in the research community and among practitioners. Both as object of research and in real world applications these methods have been widely adopted and shown to be effective for several use cases.

One major field, in which ML and DL has been playing an increasingly important role is information retrieval (IR). Not only has "Neural Information Retrieval" [MC18] been acknowledged as promising field of research within the IR research community, but it is already present in modern web IR systems used by google, amazon, netflix and other major companies, which do heavily rely on ML for delivering relevant content to their users.

While these systems most certainly do not solely operate on raw text, when it comes to to finding relevant documents for a given natural language query, the task of text ranking still lies at the heart of the problem.

In 2018 NLP experienced a major breakthrough when [Dev+19]. Core idea pretrain, finetune... While progress in pretraining lms and using for downstream task have been made before this that, this was first time the model achieved sota by large maring on variety of nlp benchmarks. and is often referred to as "the imagenet moment of NLP".

subfields including text ranking of nlp became aware and started to use.

text ranking bert consideredbbreakthrough

- modern ml system often black boxes

- need to understand: mitigate biases, make clearer to use, make better ... we deal with text ranking - advances in nlp, large models, hard to understand

1.1 Motivation

Nowadays, modern approaches for NLP often rely on large language models. These models are first pre-trained on huge amounts of text data and then fine-tuned on a smaller, task specific dataset. Although this approach has shown great effectiveness compared to traditional approaches, due to their size and complexity, these language models are mostly being treated as black boxes.

This is where several issues arise.

goal shine more light on how probing specifically targeted to IR use knowledge to improve

1.2 Problem Statement

I am still alive. - shine light on knowledge dist - research question

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 can be described as 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. 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. 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.

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:

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

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.

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.

Momentum

Adam

2.3.3 Regularization

Weight Decay

Dropout

2.3.4 Learning to Rank

2.4 Transformer Models

2.4.1 Architecture

2.4.2 Pre-Training - BERT

2.5 Probing

3 Previous Work

4 Datasets

4.1 Probing

To assess the distribution of knowledge across layers of a BERT model, we design a set of tasks that require information on different abstraction levels in order to be solved.

For each task, we generate a dataset by sampling instances from the MSMARCO validation and test set and automatically infer targets. In the following we provide a list of all probing tasks and details on the corresponding dataset generation process.

4.1.1 BM25 Prediction

4.1.2 Term Frequency Prediction

4.1.3 Named Entity Recognition

4.1.4 Semantic Similarity

4.1.5 Coreference Resolution

4.1.6 Fact Checking

4.1.7 Relevance Estimation

4.2 Multitask Learning

5 Approach

5.1 Methodology

5.2 Experimental Setup

5.3 Evaluation Measures

5.3.1 MDL

5.3.2 Compression

5.3.3 F1

5.3.4 Accuracy

5.3.5 Ranking

MAP

MRR

NDCG

Precision

6 Results

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

[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`.

[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/`.

[MC18] Bhaskar Mitra and Nick Craswell. "An Introduction to Neural Information Retrieval". In: *Foundations and Trends® in Information Retrieval* 13.1 (2018), pp. 1–126. URL: `https://www.microsoft.com/en-us/research/publication/introduction-neural-information-retrieval/`.

[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`.

[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.

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