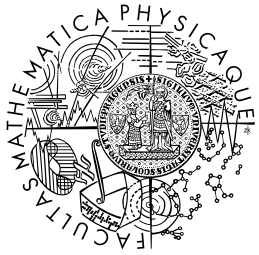

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

David Burian

Document embedding using Transformers

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Jindřich, Libovický Mgr. Ph.D.

Study programme: Computer Science

Study branch: Artificial Intelligence

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Document embedding using Transformers

Author: David Burian

Institute: Institute of Formal and Applied Linguistics

Supervisor: Jindřich, Libovický Mgr. Ph.D., Institute of Formal and Applied Linguistics

Abstract: Abstract.

Keywords: text embedding document embedding transformers document classification document similarity

Contents

Introduction

- what we are aiming to do
- why is it important/useful — where are embeddings used
- long documents — why?
- transformers — why?

1. Document representation

Representing a piece of text by a dense vector, also known as a text embedding, is an ubiquitous concept in Natural Language Processing (*NLP*). Embedding long continuous pieces of text such as documents is, however, substantially more difficult than embedding words or sentences, as the longer and more complex input still needs to be compressed into similarly sized vector. In this chapter we first explore the use of document embeddings in various scenarios. In this context, we then describe the qualities of document embeddings that we deem as beneficial and which create the motivation behind our training method proposed in Chapter 3.

1.1 On the usefulness of document embeddings

Embeddings that capture the semantics of the document in a low-dimensional vector reduce the noise in the raw input while making the subsequent operations more efficient. These are the main reasons why document embeddings are widely used across different tasks such as classification [Cohan et al., 2020, Neelakantan et al., 2022, Izacard et al., 2021, Ostendorff et al., 2022], ad-hoc search [Singh et al., 2022, Zamani et al., 2018], query answering [Neelakantan et al., 2022], visualization [Cohan et al., 2020, Dai et al., 2015] and regression [Singh et al., 2022].

While some models [Singh et al., 2022] can generate different embeddings for a single input depending on the task, most embedding models output a single vector that is effective across many types of tasks [Neelakantan et al., 2022, Cohan et al., 2020, Ostendorff et al., 2022]. This shows that embedding models can substitute several dedicated models, severely saving on time and resources.

1.2 Desirable qualities of embeddings

As is clear from the previous section, the usefulness of document embeddings comes from 3 properties. An ideal document embedding (1) represents the document's text faithfully (2) with a single vector (3) of low dimension.

In this section we focus on faithful document representation, which we view as a composition of two abstract qualities: *structural* and *contextual*. Document embedding with structural quality (or structural document embedding) faithfully models the relationships between word or word sequences. On the other hand, contextual document embedding accurately composes the meaning of all processed words, capturing their overall theme or topic. We view these qualities as scales, and so a document embedding may have high structural quality, but low contextual quality. Such embedding would capture the relationships between words very well, but would fail to assign correct meaning to these relationships due to their ambiguous meanings caused by lacking context.

Since each document embedding is produced by a model, we may attribute similar qualities to the models themselves. In this sense we speak of *structural or contextual capacity* of the model.

In the following subsections we focus on each quality separately, describing each in more detail. Then at the end of this section we compare the two qualities and setup the basic intuition behind our proposed finetuning method described in Chapter 3.

1.2.1 Structural quality of document embeddings

Structural quality defines how well the embedding captures relationships within the input text. The more complex the relationship is the higher is the structural quality of the embedding that captures it. For instance, for an input "Fabian likes playing the guitar, but Rebecca does not.", an embedding with high structural quality would understand that

1. "Fabian" likes something based on words "Fabian likes" ,
2. a guitar can be played based on words "playing the guitar" ,
3. the two sequences of words separated by comma are in a opposition based on words ", but", and
4. "Fabian" likes to play the guitar based on observations 1, and 2 .

Embedding models that compare input words and sequences of input words can produce document embedding with high structural quality. We say that such models have high structural capacity. A good example of model with high structural capacity is Transformer [Vaswani et al., 2017]. Transformer's self-attention layer allows each word to exchange information with other words. Self-attention thus allows not only comparisons of words, but also aggregation of several words into one. Thanks to Transformer's layered architecture, such aggregations can be compared in the same manner on higher levels. An example of a model with low structural capacity is Paragraph Vector [Le and Mikolov, 2014]. Paragraph Vector compares words only in a single fully connected layer. Such architecture prevents understanding of more complex relationships that build on another relationships such as observation 4 in the example above. Additionally, Paragraph Vector ignores position information, so it cannot know how were the words originally arranged.

1.2.2 Contextual quality of document embeddings

Contextual quality of a document embedding defines how well the embedding captures the overall meaning of larger sequences of text. The higher the contextual quality of an embedding is, the longer is the sequence whose meaning the embedding correctly captures as a whole. For instance, let us consider two documents: 1. a description of typical commercial turbo-jet airplane and 2. a recipe for spicy fried chicken wings. A document embedding with high enough contextual quality would reflect that the meaning of a sentence "Left wing is too hot." dramatically differs between the two documents and would accordingly adjust the sentence's contribution to the resulting document embedding.

Provided the document's text is cohesive and continuous, capturing its overall meaning gets easier as the text's length increases. This is intuitive since, if

we consider smaller pieces of text, meaning of some parts may be ambiguous or misinterpreted. However, as the length of the considered sequence grows, the probability of misinterpreting its meaning shrinks as we have more words, whose common theme we are looking for. Consequently, models that are able to process longer pieces of text have higher contextual capacity. Typical example of a model with good contextual capacity is Paragraph Vector [Le and Mikolov, 2014], which can process, in theory, indefinitely long sequences[1]. Additionally, Paragraph Vector stores a single vector per document which is iteratively compared to all words within that document, which gives the model the opportunity to adjust the contribution of individual words to the overall meaning of the document. On the other hand, Transformer [Vaswani et al., 2017] has much smaller contextual capacity as its memory requirements grow quadratically with the length of the input. In practice this prohibits Transformer from processing longer sequences of text.

1.2.3 Combining structural and contextual qualities

Each quality describes different aspect of faithful representations. Structural quality is focused more on local relationships of words, while contextual quality considers mainly the global picture of the document. From another point of view, structural quality is oriented more towards precision, while contextual quality is oriented more towards recall. In a way the two qualities complement each other. Contextual quality brings in the overall document theme, while the structural quality brings in the detailed meaning of a shorter sequence. These two pieces of information can be then aligned to produce precise and unambiguous document embedding. And so we hypothesize that the combination of these qualities is more beneficial for document embeddings than the same amount of either quality alone.

While we predict that mix of these qualities is important, we are unsure which ratio of the qualities would be the most performant. Arguably, structural quality is more important than contextual, since in extreme cases it can model relationships so complex they span the entire document and thereby substituting the role of contextual quality. On the other hand, the number of total relationships found in a document grows exponentially with the length of the document, while the number of topics covered can grow only linearly. Consequently, we can expect that for a given maximum input length an embedding model with contextual capacity to be much smaller than an embedding model with structural capacity.

To find the optimal mix between structural and contextual quality, in Chapter 3 we propose to combine models of the two extremes. Models such as Transformer with high structural capacity and models such as Paragraph Vector with high contextual capacity.

[1] Provided the vocabulary size stays constant.

2. Related Work

In this chapter we go over the research that we consider relevant to the embedding of long pieces of text using the transformer architecture. First we summarize efforts that have gone into making transformer more efficient so that it can process long inputs. These advancements are crucial to embedding documents which are oftentimes much longer than 512 tokens. The next section is dedicated to typical approaches to training embedding models. For completeness, we also mention uncommon approaches to embedding documents.

2.1 Efficient transformers

Though the Transformer [Vaswani et al., 2017] has proven to be performant architecture (TODO: citations) in the world of NLP it has one inherent disadvantage when it comes to longer sequences of text. The self-attention, which is the principal part of the transformer architecture, consumes quadratic amount of memory in the length of input. This significantly limits Transformer's applicability to variety of tasks that require longer contexts such as document retrieval or summarization.

Thanks to the popularity of the transformer architecture, there is a large amount of research that is focused on making transformers more efficient [Tay et al., 2022]. Most of these efforts fall into one of the following categories:

1. Designing a new memory-efficient attention mechanism,
2. Using a custom attention implementation, or
3. Designing new transformer architecture altogether.

We go over each category separately, though these approaches are often combined. In the section dedicated to custom implementation of self-attention we also mention commonly used implementation strategies that make transformers more efficient in practice.

2.1.1 Efficient self-attention mechanisms

As self-attention is the most resource-hungry component of the transformer architecture, it makes sense to focus on it in order to make the transformer more efficient. The core of the problem is the multiplication of $N \times d$ query matrix and $N \times d$ key matrix, where N is input length and d is a dimensionality of the self-attention. Efficient attention mechanisms approximate this multiplication and thereby avoid computing and storing the $N \times N$ resulting matrix.

Sparse attention

Sparse attention approximates the full attention by ignoring dot products between some query and key vectors. Though it may seem like a crude approximation, there is research that shows that the full attention focuses mainly on few

query-key vector combinations. For instance Kovaleva et al. [2019] shown that full attentions exhibit only few repeated patterns and that by disabling some attention heads we can increase the model's performance. Both of these findings suggest that full attention is over-parametrized and its pruning may be beneficial. Child et al. [2019] shown that repeating attention patterns can be found also when processing images and that by approximating full attention using such sparse patterns we can increase the efficiency of the model without sacrificing performance.

Sparse attentions typically compose several attention patterns. One of these patterns is often full attention that is limited only to a certain neighbourhood of a given token. This corresponds to findings of Clark et al. [2019], who found that full attention gives a lot of focus on previous and next tokens. Another sparse attention pattern is usually dedicated to enable broader exchange of information between tokens. In Sparse Transformer [Child et al., 2019] distant tokens are connected by sever pre-selected tokens uniformly distributed throughout the input. In Longformer [Beltagy et al., 2020] every token can attend to every k-th distant token to increase its field of vision. BigBird [Zaheer et al., 2020] computes dot products between randomly chosen combinations of query and key vectors. These serve as a connecting nodes for other tokens when they exchange information. The last typical sparse attention pattern is some kind of global attention that is computed only on a few tokens. Though such attention pattern is costly it is essential for tasks which require a representation of the whole input [Beltagy et al., 2020]. In Longformer some significant input tokens such as the `[CLS]` token, attend to all other tokens and vice-versa. BigBird additionally computes global attention on few extra tokens that are added to the input.

Sparse attention pattern doesn't have to be fixed, but can also change throughout the training. Sukhbaatar et al. [2019] train a transformer that learns optimal attention span. In their experiments most heads learn to attend only to few neighbouring tokens and thus make the model more efficient. Reformer [Kitaev et al., 2020] computes the full self-attention only between close key, query tokens, while letting the model decide which two tokens are "close" and which are not. To a certain degree this enables the model to learn optimal attention patterns between tokens.

Low-rank approximations and kernel tricks

Besides sparsifying the attention pattern, there are other techniques to make the self-attention more efficient in both memory and time. Wang et al. [2020] show that the attention matrix $A := \text{softmax}(\frac{QK^T}{d})$ is of low-rank and show that it can be approximated in less dimensions. By projecting the $(N \times d)$-dimensional key and value matrices into $(k \times d)$ matrices, where $k << N$ they avoid the expensive $N \times N$ matrix multiplication. The authors show that empirical performance of their model is on par with standard transformer models such as RoBERTa [Liu et al., 2019] or BERT [Devlin et al., 2019].

In another effort, Choromanski et al. [2020] look at the standard softmax self-attention through the lens of kernels. Using clever feature engineering, the authors are able to approximate the elements of the above mentioned attention matrix A as dot products of query and key feature vectors. Self-attention can be

then approximated as multiplication of four matrices the projected query and key matrices, the normalization matrix substituting the division by d and the value matrix. This allows to reorder the matrix multiplications, first multiplying the projected key and the value matrix and only after multiplying by the projected query matrix. Such reordering saves on time and space by a factor of $O(N)$ making the self-attention linear in input length.

2.1.2 Implementation enhancements

Transformer models can be made more efficient by using various implementation tricks. As modern hardware gets faster and has more memory, implementation enhancements can render theoretical advancements such as sparse attentions unnecessary. For example, Xiong et al. [2023] train a 70B model on sequences up to 32K tokens with full self-attention. Nevertheless, the necessary hardware to train such models is still unavailable to many and therefore there is still need to use theoretical advancements together with optimized implementation. For instance Jiang et al. [2023] trained an efficient transformer that uses both sparse attention and its optimized implementation. The resulting model beats competitive models with twice as many parameters in several benchmarks.

Optimized self-attention implementation

Efficient self-attention implementations view the operation as a whole rather than a series of matrix multiplications. This enables optimizations that would not be otherwise possible. The result is a single GPU kernel, that accepts the query, key and value vectors and outputs the result of a standard full-attention. Rabe and Staats [2021] proposed an implementation of full self-attention in the Jax library[1] for TPUs that uses logarithmic amount of memory in the length of the input. Dao et al. [2022] introduced *Flash Attention* that focuses on optimizing IO reads and writes and achieves non-trivial speedups. Flash Attention offers custom CUDA kernels for both block-sparse and full self-attentions. Later, Dao [2023] improved Flash Attention's parallelization and increased its the efficiency even more. Though using optimized kernel is more involved than spelling the operations out, libraries like xFormers[2] and recent versions of PyTorch[3] make it much more straightforward.

Mixed precision, gradient checkpointing and accumulation

Besides the above recent implementation enhancements, there are tricks that have been used for quite some time and not just in conjunction with transformers. We mention them here, mainly for completeness, since they dramatically lower the required memory of a transformer model and thus allow training with longer sequences.

Micikevicius et al. [2017] introduced mixed precision training which almost halves the memory requirements of the model as almost all of the activations

[1] `https://github.com/google/jax`

[2] `https://github.com/facebookresearch/xformers`

[3] `https://pytorch.org/docs/2.2/generated/torch.nn.functional.scaled_dot_product_attention.html`

and the gradients are computed in half precision. As the authors show, using additional techniques such as loss scaling, mixed precision does not worsen the results compared to traditional single precision training. In another effort to lower the amount of memory required to train a model, Chen et al. [2016] introduced gradient checkpointing that can trade speed for memory. With gradient checkpointing activations of some layers are dropped or overwritten in order to save memory, but then need to be recomputed again during backward pass. Another popular technique is gradient accumulation, which may effectively increase batch size while maintaining the same memory footprint. With gradient accumulation, gradients of batches are not applied immediately. Instead they are accumulated for k batches and only then applied to weights. This has a similar effect as multiplying the batch size by k, but not equivalent as operations like Batch Normalization [Ioffe and Szegedy, 2015] or methods like in-batch negatives behave differently. Nevertheless gradient accumulation is a good alternative, especially if the desired batch size cannot fit into the GPU memory.

2.1.3 Combination of model architectures

To circumvent the problem of memory-hungry self-attention layer, some research efforts explored combining the transformer architecture with another architectural concept, namely recursive and hierarchical networks. The common approach of these models is to not modify the self-attention, nor the maximum length of input the transformer is able to process, but to instead use transformers to process smaller pieces of text separately and contextualize them separately. Dai et al. [2019] proposes using recursive architecture of transformer nodes, where each transformer receives the hidden states of the previous one. Since gradients do not travel between the nodes, processing longer sequences requires only constant memory. The resulting model achieves state of the art performance on language modelling tasks with parameter count comparable with the competition. Simpler approach was used by Yang et al. [2020], who proposed a hierarchical model composed of transformers. First, transformers individually process segments of text producing segment-level representations, which are together with their position embeddings fed to another document-level transformer. The authors pretrain with both word-masking and segment masking losses and together with finetuning on the target tasks the model beats scores previously set by recurrent networks.

2.2 Training approaches

- How can be (long) text embeddings trained?

2.2.1 Autoregressive Language Modelling

- Standard pre-training for many transformers
- Paragraph Vector [Le and Mikolov, 2014] – just mentioned it. It is already surpassed methodology.

2.2.2 Siamese networks

- SBERT [Reimers and Gurevych, 2019] – did pretty much the same as us (took a model and finetuned it so that the produced embeddings are good), except for sentences
- MPNet [Song et al., 2020] – best SBERT model and the model we are working with

2.2.3 Knowledge distillation

- That paper where SBERT is finetuned on low-resource language with teacher-student training [Reimers and Gurevych, 2020]

2.2.4 Contrastive loss

- OpenAI embedding paper – use contrastive loss, mention large batches and the memory constraints it brings
- Specter [Cohan et al., 2020] – focused on scientific documents, use of contrastive loss based on citations
- Transformer based Multilingual document embedding model [Li and Mak, 2020] – transformer version of LASER, embeds documents

2.3 Unorthodox document embedding approaches

- Self-Supervised Document Similarity Ranking [Ginzburg et al., 2021] – really focused on generating semantically meaningful representations of documents, but with an extra cost and complexity
- Cross-Document Language Modelling [Caciularu et al., 2021] – uses Longformer for cross-document task. Illustrates that Longformer can be flexible and useful for document processing.

3. Finetuning method to increase quality of document embeddings

In this chapter we describe our method of training document embeddings. Our approach is based on teacher-student training approach, where we distil the knowledge of two teacher embedding models into a single student model. In Section 3.1 we explain our training method in detail and roughly define the used loss function. Then in Section 3.2 we describe the two teacher models, which we use in the rest of the thesis. Finally, in Sections 4.3.1 and **??** we walk through each of the two loss components in detail.

3.1 Training methodology

Our training methodology aims to train an embedding model such that its embeddings more faithfully represent the input. As we described in Chapter 1 we distinguish two qualities of faithful representations: structural and contextual. The goal is to instill both of these two qualities into a single embedding model. To do so, we use teacher-student training with two teacher text embedding models, one with high structural capacity, the other with high contextual capacity.

In the following subsections we describe teacher-student training in detail and give high-level overview of the proposed loss function.

3.1.1 Teacher-student training

In teacher-student training we train a single *student* model based on a non-trainable *teacher* model. The goal is to make the student model to imitate the teacher model and thereby digest the teacher's understanding of the input. Teacher-student training is useful in a number of situations. For instance it used to overcome some inherit limitation of the teacher model such as its large size [Sanh et al., 2019]. Also, teacher-student training can be used to enforce some kind of alignment between models' outputs [Reimers and Gurevych, 2020]. The motivation to align the model's outputs often comes from an intuition about language, such as that embeddings of two translations of a given sentence should be the close to each other.

In our setting, we assume two embedding models $\mathcal{T}_S$, $\mathcal{T}_C$ with high structural and contextual capacities respectively. Teacher-student training allows us to instill both of these capacities into a third model $\mathcal{S}$, while also avoiding some of the architectural limitations of both reference models. For convenience we call $\mathcal{T}_S$ *structural teacher*, $\mathcal{T}_C$ *contextual teacher*, and $\mathcal{S}$ *student*.

3.1.2 Abstract loss formulation

Our loss function should align the output of a student model $\mathcal{S}$ with outputs of two teacher models $\mathcal{T}_S$, and $\mathcal{T}_C$. We use two similarity functions $\mathcal{L}_S$, $\mathcal{L}_C$ that compare the student's embedding $y_{\mathcal{S}}$ with structural teacher embedding $y_{\mathcal{T}_S}$ and with contextual teacher embedding $y_{\mathcal{T}_C}$ respectively. As we are unsure

which balance of $\mathcal{L}_S$ and $\mathcal{L}_C$ is optimal we introduce weighting parameter λ that balances the effect of the two losses on the final loss. In the most general form, we can assume λ to be dependent on the input text x, since the performance of the teacher models might vary across different inputs. In particular, we can expect λ to be dependent on the length of the input, since for shorter inputs the context is minimal and therefore expendable. The form of the loss as we have described it is defined in Equation 3.1. We explore concrete options for $\mathcal{L}_S$, $\mathcal{L}_C$ and $\lambda(x)$ in Chapter 4.

$$\mathcal{L}(x, y_{\mathcal{S}}, y_{\mathcal{T}_S}, y_{\mathcal{T}_C}, \lambda) = \lambda(x)\mathcal{L}_S(y_{\mathcal{S}}, y_{\mathcal{T}_S}) + \mathcal{L}_C(y_{\mathcal{S}}, y_{\mathcal{T}_S}) \tag{3.1}$$

3.2 Teacher models

In this section we present the teacher models we use during our experiments in Chapter 4. We highlight why we choose the particular models and how we obtain the their embeddings. We chose Sentence-BERT [Reimers and Gurevych, 2020] (or *SBERT*) as the structural teacher model, and Paragraph Vector [Le and Mikolov, 2014] (or PV) as the context teacher model.

3.2.1 SBERT

Sentence-BERT [Reimers and Gurevych, 2019] is a composition of a BERT-like [Devlin et al., 2019] encoder with a mean pooling layer above its final hidden states. We have chosen SBERT as structural teacher for its Transformer architecture, which as we have discussed in Chapter 1, has high structural capacity. Additionally, SBERT is finetuned on NLI datasets to increase its text understanding and to produce embeddings which are semantically meaningful.

There are several versions of SBERT, that differ in the base Transformer encoder, from which is SBERT warm-started and then finetuned. We use SBERT warm-started from MPNet [Song et al., 2020], that achieves high scores across many sentence embedding benchmarks[1], while being reasonably small. To be exact we use its Hugging Face implementation named
`sentence-transformers/all-mpnet-base-v2`.

We generate the embeddings of any training dataset directly without any additional finetuning.

3.2.2 Paragraph Vector

Paragraph Vector Le and Mikolov [2014] (also known as Doc2Vec) is a simple text-embedding model that views the input as a Bag of Words (or BoW). Paragraph Vector is composed of two sub-models called Distributed Memory (DM) and Distributed Bag of Words (DBOW). In practice one or both models can be used, where the final embedding is the concatenation of all sub-models' outputs. Both models construct embedding of the whole input with which they predict a randomly masked out input word. DM additionally uses embeddings of neighbouring words.

[1] `https://sbert.net/docs/pretrained_models.html`

We choose Paragraph Vector as contextual teacher due to its unique architecture that forces the model to develop a single vector, that summarizes the common theme of the document. Moreover, Paragraph Vector does not have limited maximum input length, and so as a contextual teacher it will always provide some signal to the student regarding the document's context. Also, even though Paragraph Vector cannot match the performance of a substantially more complex models such as Transformers, Dai et al. [2015] show that for larger datasets Paragraph Vector outperforms classical embedding models such as Latent Dirichlet Allocation [Blei et al., 2003] or TF-IDF weighted BoW model [Harris, 1954].

We generate Paragraph Vector's embeddings after training the model on *all* training datasets for which we require a contextual teacher's embedding. There are many hyperparameters that govern how Paragraph Vector is trained. Since there are no universally agreed best-performing values for any of them, we see these as hyperparameters of our teacher-student training method. We explore the effect of some of these parameters on the model's performance in Chapter 4.

TODO: my own graphic here

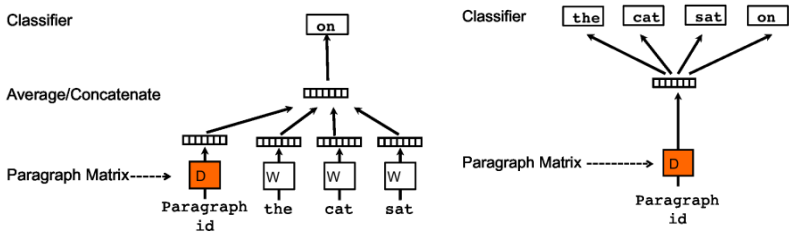

Figure 3.1: PV-DM and PV-DBOW architectures.

4. Experiments

- what metrics we used to evaluate the model
- mention the main hyperparameters
- how we searched – step by step

4.1 Base model

To test our finetuning method we used a model pretrained using Masked Language Modelling on a large corpus. Since our training method relies on the model's basic natural language understanding, we would have to pretrain the model either way. Additionally by using a checkpoint of a pretrained model we dramatically save on resources.

We have considered several transformer encoders: BigBird Zaheer et al. [2020], Longformer Beltagy et al. [2020], (TODO: fill in?). Of these we chose Longformer for the following reasons:

- It is memory-efficient small model that can be trained on a single GPU.
- Its attention implementation is easy to understand.
- Longformer performs above average in comparison to other similar models Tay et al. [2020]

We emphasize that our training method can be applied theoretically to all transformer encoders. We, therefore view this decision as a practical one, rather than theoretical one. In other words we prioritize ease of use and simplicity in order to demonstrate the advantages of our training method.

4.1.1 Longformer

Longformer Beltagy et al. [2020], is a transformer encoder that is able to process input sequences longer than traditional Transformer using its implementation of self-attention.

Self-attention mechanism

Longformer's self-attention is not a full attention, in which each token attends to all other tokens. Instead, some tokens only attend to a selected few other tokens. This makes the attention more sparse and thus allows its computation in $O(n)$ time and memory.

In practice, Longformer's self-attention tokens can attend either just to their local neighbourhood or to all other tokens. These types of attention are called local and global respectively. For a neighbourhood of $\frac{1}{2}\omega$ tokens on each side, the implementation splits the matrix of queries Q and the matrix of keys K into chunks of size ω overlapped by $\frac{1}{2}\omega$ tokens. These are multiplied separately and composed together to form a block diagonal. Global attentions are computed

separately and then are assigned to the result at the correct indices. The neighbourhood size ω can vary across transformer layers. This can be helpful when we try to save on the number of floating point operations per second (or *FLOPS*) while minimizing the negative impact on performance Sukhbaatar et al. [2019].

The attention as it was described above is the basic sparse attention that Longformer supports. Longformer can dilate the local attention pattern by only attending to every n-th neighbouring token. This means that while the number of neighbourhood tokens stays the same, the attended window is effectively increased. However, to use dilated local attention effectively, special GPU kernel is needed.

Training

Longformer is warm-started from a RoBERTa Liu et al. [2019] checkpoint with its learned positional embeddings copied 8 times to support inputs up to 4096 tokens long. Copying RoBERTa's positional embeddings showed to be an effective way how to increase the maximum context length of a transformer with learned positional embeddings. Then it was trained further using Masked Language Modelling (or *MLM*) on long documents with different attention span for each layer as well as different dilatation for each head.

The specially compiled training corpus has some overlap with RoBERTa's pretraining corpus, but is more focused on longer pieces of text. It includes:

- Book corpus Zhu et al. [2015],
- English Wikipedia,
- one third of Realnews dataset Zellers et al. [2019], and
- one third of the Stories corpus Trinh and Le [2018].

4.2 Evaluation metrics

Will I even show them? Isn't it too detailed?

4.2.1 Structural evaluation metrics

4.2.2 Breadth evaluation metrics

4.3 Improving structural quality

4.3.1 Structural loss

The structural loss $\mathcal{L}_S$ is one of the two losses we use in our teacher-student training. The goal of structural loss is to align the student's embedding with the embedding of the structural teacher.

The structural loss $\mathcal{L}_S$ is one of the two losses our teacher-student training will use. For inputs that fit into the maximum context length of the structural teacher, the structural loss should minimize the dissimilarity between the structural teacher's and the student's embeddings as much as possible. This follows

our assumption that for inputs that the structural model can process whole, the structural model produces an embedding with the deepest understanding of the input that is available to us. It is thus important that the similarity $\mathcal{L}_D$ should take into account is the absolute similarity rather than some approximate measure of it.

(TODO: What else to say about this? What dissimilarity we use will be in Experiments chapter as well as deciding "when an input is applicable".)

4.4 Improving contextual quality

4.4.1 Obtaining Paragraph Vector embeddings

4.4.2 Loss selection

The contextual loss $\mathcal{L}_B$ minimizes the disagreements between the embeddings of the contextual teacher model and that of the student model. There are two major differences between the contextual loss and the structural loss. First the contextual loss is always applicable. It might not be the loss that reflects our ultimate goal the most for the given input. Nevertheless we still assume that for *every* input the contextual teacher model offers some information which the structural teacher model does not have. Second the contextual loss is not as exact as the structural loss. Instead of forcing the teacher model to adhere to two possibly very distinct ways how to encode information into a dense vector representation, we give the model a little bit of freedom. We do so by letting the model decide how exactly it should encode the information contained in the contextual teacher's embedding. We give the model more leeway on the breadth side rather than the structural side, because we expect that the precision of the embedding is more important than capturing every piece of the input.

With the above taken into an account we chose to use a variant of *Canonical Correlation Analysis* Hotelling [1992] (or *CCA*) as a base for our breath loss $\mathcal{L}_B$. A variant of CCA fits our needs very nicely as it computes a correlation of outputs after some projection. While the projection gives the model the freedom to restructure its embeddings, the correlation ensures that the two embeddings agree with each other. Before going further let us briefly describe CCA and its variants we considered.

Canonical Correlation Analysis

Canonical Correlation Analysis (CCA) computes linear projections of two vectors such that their projections are maximally correlated. For formal definition reference Definition 1.

Definition 1 (Canonical Correlation Analysis)**.** *For two matrices $X_1 \in \mathbb{R}^{n_1 \times m_1}$ and $X_2 \in \mathbb{R}^{n_2 \times m_1}$, Canonical Correlation Analysis finds $p \in \mathbb{R}_1^m$ and $q \in \mathbb{R}_2^m$ that maximize*

$$\begin{aligned} & corr(X_1 p, X_2 q) = \frac{p^T X_1^T X_2 q}{||Xp||||Yq||} \\ s.t. \quad & ||X_1 p|| = ||X_2 q|| = 1 \end{aligned} \tag{4.1}$$

Definition 1 suggests that CCA gives only a single number as the measure of correlation of two matrices. When the dimensions of the input vectors are large enough, however, often there exists at least one combination of features that results in correlation of 1. As this would make CCA relatively useless in the context of high-dimensional spaces we assume multiple correlations for several mutually orthogonal projections. We name such Canonical Correlation analysis as CCA for more dimensions and define it formally in Definition 2

Definition 2 (Canonical Correlation Analysis for more dimensions)**.** *For two matrices $X_1 \in \mathbb{R}^{n_1 \times m_1}$ and $X_2 \in \mathbb{R}^{n_2 \times m_1}$, Canonical Correlation Analysis for k dimensions finds $P \in \mathbb{R}^{m_1 \times k}$ and $Q \in \mathbb{R}^{m_2 \times k}$ that maximize*

$$\begin{aligned} & \sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) \\ s.t. \quad & P^T X_1^T X_1 P = I_k = Q^T X_2^T X_2 Q \end{aligned} \tag{4.2}$$

If we consider the conditions in Definition 2, the resulting value can be easily reformulated:

$$\sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) = trace(P^T X_1^T X_2 Q) = trace(P^T \Sigma_{X_1 X_2} Q) \tag{4.3}$$

Where $\Sigma_{X_1 X_2}$ is the covariance matrix of X_1 and X_2.

TODO: analytical solution to pure CCA

As we noted above we would like to use CCA as a base for our contextual loss $\mathcal{L}_B$ in order to establish correspondence between the contextual teacher's embedding and the student's. The problem using CCA as it was defined in Definition 2 as a loss is that it is defined in the context of the whole dataset rather than just a minibatch. It is therefore unclear how should be CCA computed using just a pair of minibatches.

Someone (TODO: citation) found that using large enough batch size is sufficient for the training to converge.

Deep CCA

Deep CCA (or DCCA) is an extension of CCA that computes projections using neural networks. As such it is more powerful than plain CCA as it allows for non-linear projections. To compute DCCA the network has to be trained on the pairs of vectors with CCA as its loss.

TODO: graphic of architecture

If CCA is weaker condition than just correlation, DCCA is even weaker since there is no limit to how the projections should look like.

Soft CCA

Soft CCA reformulates CCA and thus allows its straightforward use in the context of minibatches. With constraints from Definition 2 taken into account, CCA can be formulated using Forbenious matrix norm:

$$P^*, Q^* = \underset{P,Q}{\operatorname{argmin}} ||X_1P - X_2Q||_F^2 \tag{4.4}$$

$$= \underset{P,Q}{\operatorname{argmin}} \, trace\Big((X_1P - X_2Q)^T(X_1P - X_2Q)\Big) \tag{4.5}$$

$$= \underset{P,Q}{\operatorname{argmin}} \, -2trace(P^T X_1^T X_2 Q) \tag{4.6}$$

$$= \underset{P,Q}{\operatorname{argmax}} \, trace(P^T X_1^T X_2 Q) \tag{4.7}$$

$$= \underset{P,Q}{\operatorname{argmax}} \sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) \tag{4.8}$$

So, in essence minimizing CCA is the same as minimizing the difference between projections, whose features are decorrelated. This is the formulation Soft CCA builds on. Soft CCA decomposes CCA into to parts:

- minimization of the difference between projections

$$||X_1P - X_2Q||_F^2 \tag{4.9}$$

- decorrelation of each projection P

$$\sum_{i \neq j} (P^T X_{mini}^T X_{mini} P)_{ij} = \sum_{i \neq j} \Sigma_{X_{mini}P}, \tag{4.10}$$

  where X_{mini} is batch-normalized minibatch.

To bring correlation matrix of the current minibatch $\Sigma_{X_{mini}P}$ closer to the true covariance, the decorrelation part is in fact computed from a covariance matrix that is incrementally learned.

In this way Soft CCA incrementally decreases CCA through incrementally learned approximation of the projections' covariances.

Loss selection

- mention which of the CCA variants we chose
- reiterate the formulation in the context of teacher-student training

Mean Squared Error

Vanilla CCA

SoftCCA

4.5 Training with both structural and contextual teachers

4.5.1 Balancing structural and contextual

4.6 Ablation studies

- here should be the ablation studies we did to prove that the main thing we did has some effect (e.g. using MSE instead of Soft CCA)

5. Evaluation

In this chapter we will describe a set of benchmarks, which will test our model and enable us to compare it to other models. First we will describe the tasks — datasets and corresponding evaluation metrics, then we will talk about the models. Results of the benchmarks are discussed in Chapter 6.

5.1 Tasks

Each task aims to test a different aspect of a model. Our aim was to design a set of tasks, which can capture a model's capability to embed whole documents. The major obstacle we faced was the lack of labeled datasets with longer pieces of text (more than 512 tokens).

TODO: how did we solve the issue

TODO: complete list of task types

Classification

Classification tasks test model's capability to separate inputs based on a complex feature. In our settings, classification tasks can tell us what information the document embedding contains.

5.1.1 IMDB Sentiment Analysis

IMDB sentiment analysis task is a simple binary classification task. The dataset contains movie reviews from the Internet Movie Database[1] labeled as either positive or negative. The dataset is commonly referred to as IMDB classification or sentiment dataset Maas et al. [2011].

The dataset is split evenly to test and train set, each having 25000 reviews. The dataset also contains 50000 unlabeled reviews. The label distribution in both sets is uniform, each of of the two labels is represented by 12500 reviews.

As can be seen from the figure Figure 5.1 the reviews are quite short with only 13.56% being longer than 512 RoBERTa tokens.

We included this task to see how our model compares in relatively undemanding settings, while also evaluating its performance on shorter documents.

[1] `www.imdb.com`

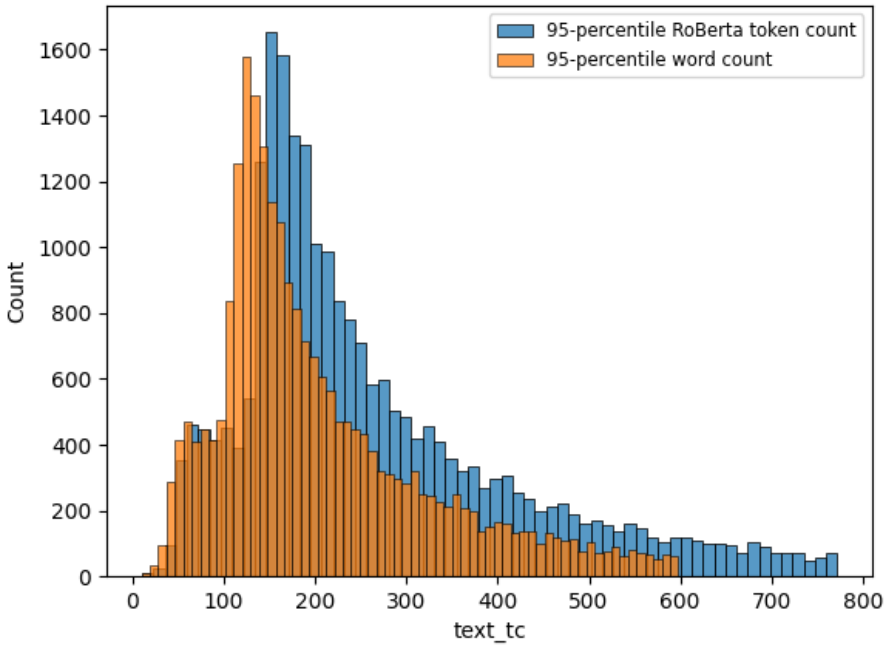

Figure 5.1: Word count and token count distribution of 95-percentiles of reviews. The tokens are generated using RoBERTa's pretrained tokenizer from HuggingFace

6. Results

- tabulated experiment results
- discussion

Conclusion

- what i have done — what are the model's results
- other findings

Bibliography

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Avi Caciularu, Arman Cohan, Iz Beltagy, Matthew E Peters, Arie Cattan, and Ido Dagan. Cdlm: Cross-document language modeling. *arXiv preprint arXiv:2101.00406*, 2021.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S Weld. Specter: Document-level representation learning using citation-informed transformers. *arXiv preprint arXiv:2004.07180*, 2020.

Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Dvir Ginzburg, Itzik Malkiel, Oren Barkan, Avi Caciularu, and Noam Koenigstein. Self-supervised document similarity ranking via contextualized language models and hierarchical inference. *arXiv preprint arXiv:2106.01186*, 2021.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics: methodology and distribution*, pages 162–190. Springer, 1992.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of bert. *arXiv preprint arXiv:1908.08593*, 2019.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

Wei Li and Brian Mak. Transformer based multilingual document embedding model. *arXiv preprint arXiv:2008.08567*, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `https://aclanthology.org/P11-1015`.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Malte Ostendorff, Nils Rethmeier, Isabelle Augenstein, Bela Gipp, and Georg Rehm. Neighborhood contrastive learning for scientific document representations with citation embeddings. *arXiv preprint arXiv:2202.06671*, 2022.

Markus N Rabe and Charles Staats. Self-attention does not need $O(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. *arXiv preprint arXiv:2004.09813*, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Amanpreet Singh, Mike D'Arcy, Arman Cohan, Doug Downey, and Sergey Feldman. Scirepeval: A multi-format benchmark for scientific document representations. *arXiv preprint arXiv:2211.13308*, 2022.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, 33:16857–16867, 2020.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers, 2019.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi: 10.1145/3530811. URL `https://doi.org/10.1145/3530811`.

Trieu H Trinh and Quoc V Le. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*, 2023.

Liu Yang, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. Beyond 512 tokens: Siamese multi-depth transformer-based hierarchical encoder for long-form document matching. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1725–1734, 2020.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 497–506, 2018.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *Advances in neural information processing systems*, 32, 2019.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

List of Figures

List of Tables

A. Attachments