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

We dedicate this chapter to the center point of our thesis – representing documents with vectors. The goal of this chapter is to define the desired qualities of our embeddings and illustrate them on examples.

1.1 Desirable qualities of document representations

Since there are many ways how to embed a document we deemed it necessary to describe what are the target qualities of the embeddings. There are two dimensions along which we asses document representations: *depth* and *breadth*. *Depth* dimension defines how deep is the model's understanding of the input text. *Breadth* dimension defines the amount of information the model is able to process. These dimensions are in a sense contradictory to each other. We cannot expect a fully learned model of a fixed size to deepen its understanding of the input while maintaining the same maximum input size. Thus we are searching for a ideal ratio of these two qualities.

1.1.1 Text understanding

In the depth dimension we study how well the model understands its input. For an input *"This is a really nimble horse. You should be cautious when riding him."* we would like the model to understand that *"him"* refers to the *"nimble horse"*. Or that the meaning of *"stick"* in the following sentences is largely different:

- *"You are really good at tennis. You should stick to it."*
- *"Some dogs really like to fetch a stick, some just carry it."*

To produce embedding that would reliably reflect the input's meaning, we expect the model to see the words in context of their relationship to other words. To do so, architectures that enable deep text understanding allow the model to compute with relationships of ordered subwords, words or sequences of words. Such architectures are able to put each word in relevant context and thus recover all the meanings of each sequence of words if they are given enough training data.

To compute with relationships of ordered subwords, the neural network must connect representations of two subwords such that the connections also carry the information about the subwords' ordering. The same holds for computing with relationships of ordered words or sequences of words.

To give some examples we can pick some representative embedding models along the depth dimension going from the ones that show the shallowest understanding of their input to those who show the deepest. At the beginning, there are models that ignore both word ordering and their relationships. An example of such model is TF-IDF. If we go deeper we run into Paragraph Vector Le and Mikolov [2014], that takes word relationships into an account, but ignores word ordering to a certain degree. Next there are models with architectures based on one-directional or two-directional RNNs. These models compute with ordering of words implicitly, but do not have the capacity to consider some word relationships directly. Finally there are models that are based on the architecture of Transformer Vaswani et al. [2017]. Such models not only take the word ordering into consideration, but are also able to compute over all relationships between input's text units. An example of such architecture is SBERT Reimers and Gurevych [2019], which is essentially an instance of BERT Devlin et al. [2019] finetuned on NLI datasets to encourage deeper text understanding.

1.1.2 Maximum input length

In the breadth dimension we study the maximum context length of the model. Put simply, we study the maximum number of tokens the model is able to process. The longer the maximum context length is, the longer text we can coherently embed.

Model's context length is especially important when we expect the properties of the piece of text to vary throughout it. For such texts the embedding of its beginning would be different than the embedding of its end or of the whole text. Comparing embeddings of long texts, thus requires to consider the text as a whole and embed it with a model whose maximum context is larger or equal than the given text's length.

If we revisit the models mentioned in the previous section, we realise the order of the mentioned models flips. The models with the smallest maximum context are Transformers with classical attention mechanisms. They are typically not limited by their architecture, but by the memory they consume for longer inputs. The large memory consumption originates in the classical attention mechanism, which takes into account all of the n^2 relationships among the n input tokens. Though recently there have been many improvements to the implementation of classical attention(TODO: ref) which improves their efficiency, and there are other attention mechanisms which offer more efficient alternative, Transformers are nevertheless costly for longer inputs both in terms of memory and time. Next type of models are those built using RNNs. Though RNNs can in theory process unlimited number of tokens, in practice they are known for "forgetting" past tokens, once the input is long enough (TODO: ref). On the other hand, Paragraph Vector or TF-IDF can both process in theory limitless number of words. In practice Paragraph Vector is limited by the dimension of the document embedding, which can be, however, scaled arbitrarily.

1.1.3 Combining depth and breadth

As we have hinted above, the goal of this thesis is to combine deep understanding with longer maximum contexts. Combination of both qualities would result in a model that can not only register and compute with relationships of words, but do so over large gaps of unrelated tokens. For example, in *"I really like your T-shirt. It almost looks as if it was designed by some really alternative guy living in some cave in Gran Canaria. You should definitely wear it more often."*, we would expect such model to connect *"it"* in the last sentence with the *"T-shirt"* from the first sentence.

To find a compromise, we combine models of the two extremes: models like Transformer that can understand the input very well and models like Paragraph Vector which can process very long sequences.

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 crucial efforts that have gone into making transformer more efficient to be able to process long inputs such as documents. 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 natural language processing (or *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 is only natural 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 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 while not 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 patter 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 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 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 $A := \mathrm{softmax}(\frac{QK^T}{d})$ 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 substituing 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 and the gradients are computed in half precision. As the authors show, using

[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`

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 SOTA 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 finetuning method in detail. As we hinted at the end of Chapter 1 our finetuning method is based on teacher-student training approach, where we distil the knowledge of several teacher models into a single student model. First we present the idea behind the training. Then we describe the teacher models in detail. Finally we go over all components of our training loss in separate sections and define each in concrete terms.

3.1 Training methodology

Our training methodology is based on how we evaluate a text embedding. As we described in Chapter 1 we view the quality of text embeddings in two dimensions: depth and breadth. The depth dimension defines how precise or detailed understanding of the text the embedding shows, while the breadth dimension defines how much information the embedding is able to reflect. We hypothesize that by increasing the amount of detail and of the information that the embedding encodes, we improve the text embedding's usefulness in the plethora of downstream tasks. We, therefore, aim to improve a models' embeddings along each of the two dimensions. To do so we designed a teacher-student training with two teachers, whose embeddings represent an ideal in each of the two dimensions.

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

3.1.1 Teacher-student training

In teacher-student training we train a single *student* model according to a non-trainable *teacher* model. The idea is to make the student model to imitate the teacher model and thereby digest the teacher's understanding of the input. Typically there is an inherit limitation that prohibits straightforward use of the teacher model and requires a model with different kind of qualities. For instance Sanh et al. [2019] used teacher-student training to reduce the size of a model while retaining almost all of its performance. In another experiment Reimers and Gurevych [2020] overcame the insufficient training dataset sizes to train an embedding model on low-resource languages.

In our setting we have two teacher models that correspond to the two aspects of input understanding which we would like to instil to the student. We label the teachers as depth teacher $\mathcal{M}_D$ and breadth teacher $\mathcal{M}_B$. We expect a limitation of both of the teacher models that prevents their straightforward use. As the depth teacher is oriented towards deep text understanding we expect it will be unable to process longer inputs. On the other hand the breadth teacher focuses on ingesting long texts and so we expect shallower understanding of any given text sequence. Particularly for shorter inputs we expect $\mathcal{M}_D$ to reflect their meaning better than $\mathcal{M}_B$ and vice-versa for longer inputs.

3.1.2 Abstract loss formulation

When computing the loss we take an embedding of each of the teacher model, and of the student and define a loss based on their similarity. For each model we have correspondingly labeled similarity-based losses. $\mathcal{L}_D$ compares embedding of the student with that of the depth teacher $\mathcal{M}_D$ while $\mathcal{L}_B$ with that of the breadth teacher $\mathcal{M}_B$. Let us also label the input as x, the student's output as y, the depth teacher model's output as y_D, and the breadth teacher model's output as y_B. Then the overall loss has the following shape:

$$\mathcal{L}(x, y, y_D, y_B, \lambda) = \lambda(x)\mathcal{L}_D(y, y_D) + \mathcal{L}_B(y, y_B), \tag{3.1}$$

where $\lambda(x)$ is a weighting parameter that depends on the input x. This is to reflect our expectation that for some inputs one loss reflects our goal more than the other. For example shorter inputs it can be beneficial for $\mathcal{L}_D$ to be dominant.

3.1.3 Self-balancing out losses

As we noted in earlier the two aspects of the embeddings, which we hope to improve, stand in opposition. To illustrate this opposition on an example, let us say we have a memory of limited size and a text. We can quite easily pack in more detailed information about some of the words while forgetting the rest. We can also forget all the minute little details about each word and make room to remember few extra words. On the other hand doing both – being more specific about the words we already saved while making room to remember few extra ones – seems like an impossible task. This balance between embedding's depth and breadth is important to highlight because it plays role not only in the performance of the final model but also during its training.

During evaluation some tasks may prefer the balance to be shifted one way or the other. So it is crucial to choose tasks that reflect the diversity of text embeddings' applications. During training the opposition between depth and breadth plays to our advantage because optimizing one prohibits the model overfitting on the other. For instance, we expect that if we leave out $\mathcal{L}_B$ the student model would tend to forget the endings of longer inputs. On the other hand by ignoring $\mathcal{L}_D$ the model could quickly loose precision and many of the embeddings would become nearly identical.

3.2 Reference models

In this section we present the teacher models we later use during training. We highlight how we obtain the teacher embeddings and what qualities we can expect based on their previous evaluations and the models' architectures. We chose Sentence-BERT Reimers and Gurevych [2020] (or SBERT) as the depth teacher model, and Paragraph Vector Le and Mikolov [2014] (or PV) as the breadth 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. While BERT-like architecture proved to be performant and versatile useful not only for NLP (TODO:citation) SBERT focuses only on generation of semantic embeddings of shorter text inputs.

The model is not trained from scratch. Instead it is warm started from a checkpoint of BERT-like model such as RoBERTa Liu et al. [2019] or MPNet Song et al. [2020]. After adding a pooling layer the model is trained in a siamese network structure, where two inputs are passed through the model after which their embeddings are compared using a softmax classifier. SBERT is trained on NLI datasets Bowman et al. [2015], Williams et al. [2017] in which the model is tasked to classify pairs of sentences as contradiction, entailment or neutral.

To this day the authors provide a website[1] that offers a selection of pre-trained SBERTs warm-started from different base models. While each model has its advantages and shortcomings, we chose an SBERT warm-started from MPNet, since it is reasonably small while providing the best sentence embeddings according to the website's benchmarks. While the model's maximum input length is only 384 tokens we believe that its performance outweighs the shorter context, particularly given our training method.

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

TODO: siamese networks schematic image here

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) (i.e. it does not acknowledge word order). Architecturally it is an extension of Word2Vec Mikolov et al. [2013] which only embeds word.

Paragraph Vector is composed of two sub-models called Distributed Memory (or DM) and Distributed Bag of Words (or DBOW). In practice one can use one or both models and the final embedding is the concatenation of all used models. Both DM and DBOW are trained on token prediction. To predict masked-out tokens, both models use embedding of the whole input, while DM additionally computes word embeddings. Embeddings of the whole input work as simply as any word embedding: each piece of text (a word or an input) gets an identifier, which is associated with a vector that is trained only for the given input. In practice this means that PV must be trained on the dataset, whose embeddings it is supposed to provide. Consequently there are no pretrained models and the prediction time is rather long.

Though Paragraph Vector cannot match the performance of a substantially larger models such as transformers, Dai et al. [2015] show that for larger datasets it outperforms classical embedding models such as Latent Dirichlet Allocation Blei et al. [2003] or TF-IDF weighted BoW model Harris [1954]. Additionally its ability to embed virtually limitlessly large sequences of text in combination with its

[1] https://sbert.net

small size, is to this day unrivaled between embedding models.

We generate PV embeddings after training the model on *all* datasets for which we require a PV embedding. There are many hyperparameters that govern how PV is trained such as the number of epochs, ignoring words based on frequency or number of negative samples. Since there are no universally agreed best-performing values for any of them, we see these as hyperparameters of our teacher-student training method.

TODO: my own graphic here

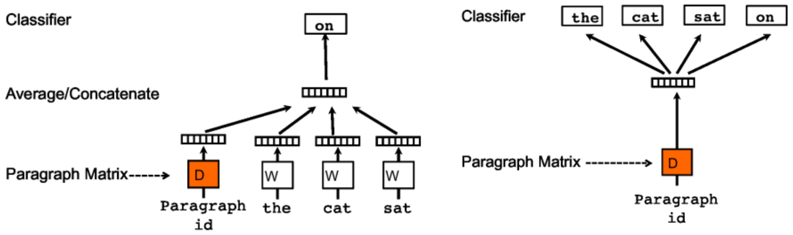

Figure 3.1: PV-DM and PV-DBOW architectures.

3.3 Depth loss

The depth loss $\mathcal{L}_D$ is one of the two losses our teacher-student training will use. For inputs that fit into the maximum context length of the depth teacher, the depth loss should minimize the dissimilarity between the depth teacher's and the student's embeddings as much as possible. This follows our assumption that for inputs that the depth model can process whole, the depth 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”.)

3.4 Breadth loss

The breadth loss $\mathcal{L}_B$ minimizes the disagreements between the embeddings of the breadth teacher model $\mathcal{M}_B$ and that of the student model. There are two major differences between the breadth loss and the depth loss. First the breadth 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 breadth teacher model offers some information which the depth teacher model does not have. Second the breadth loss is not as exact as the depth 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 breadth teacher's embedding. We give the model more leeway on the breadth side rather than the depth 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.

3.4.1 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 $p \in \mathbb{R}^m_1$ and $q \in \mathbb{R}^m_2$ that maximize*

$$\begin{aligned} & corr(X_1 p, X_2 q) = \frac{p^T X_1^T X_2 q}{||Xp|| ||Yq||} \\ s.t. \quad & ||X_1 p|| = ||X_2 q|| = 1 \end{aligned} \tag{3.2}$$

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*

$$\begin{aligned} & \sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) \\ s.t. \quad & P^T X_1^T X_1 P = I_k = Q^T X_2^T X_2 Q \end{aligned} \tag{3.3}$$

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

$$\sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) = trace(P^T X_1^T X_2 Q) = trace(P^T \Sigma_{X_1 X_2} Q) \tag{3.4}$$

Where $\Sigma_{X_1 X_2}$ is the covariance matrix of X_1 and X_2.
TODO: analytical solution to pure CCA

CCA as a breadth loss

As we noted above we would like to use CCA as a base for our breadth loss $\mathcal{L}_B$ in order to establish correspondence between the breadth 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:

$$\begin{aligned}
P^*, Q^* &= \underset{P,Q}{\operatorname{argmin}}\, ||X_1 P - X_2 Q||_F^2 && (3.5)\\
&= \underset{P,Q}{\operatorname{argmin}}\, trace\Big((X_1 P - X_2 Q)^T (X_1 P - X_2 Q)\Big) && (3.6)\\
&= \underset{P,Q}{\operatorname{argmin}}\, -2trace(P^T X_1^T X_2 Q) && (3.7)\\
&= \underset{P,Q}{\operatorname{argmax}}\, trace(P^T X_1^T X_2 Q) && (3.8)\\
&= \underset{P,Q}{\operatorname{argmax}} \sum_{i=1}^{k} corr(X_1 P_{*i}, X_2 Q_{*i}) && (3.9)
\end{aligned}$$

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

$$||X_1 P - X_2 Q||_F^2 \tag{3.10}$$

- decorrelation of each projection P

$$\sum_{i \neq j} (P^T X_{mini}^T X_{mini} P)_{ij} = \sum_{i \neq j} \Sigma_{X_{mini} P}, \tag{3.11}$$

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

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

4.2.1 Depth evaluation metrics

4.2.2 Breadth evaluation metrics

4.3 Training with depth teacher only

4.3.1 Loss selection

Max-marginals losses

4.4 Training with breadth teacher only

4.4.1 Obtaining Paragraph Vector embeddings

4.4.2 Loss selection

Mean Squared Error

Vanilla CCA

SoftCCA

4.5 Training with both depth and breadth teachers

4.5.1 Balancing depth and breadth

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

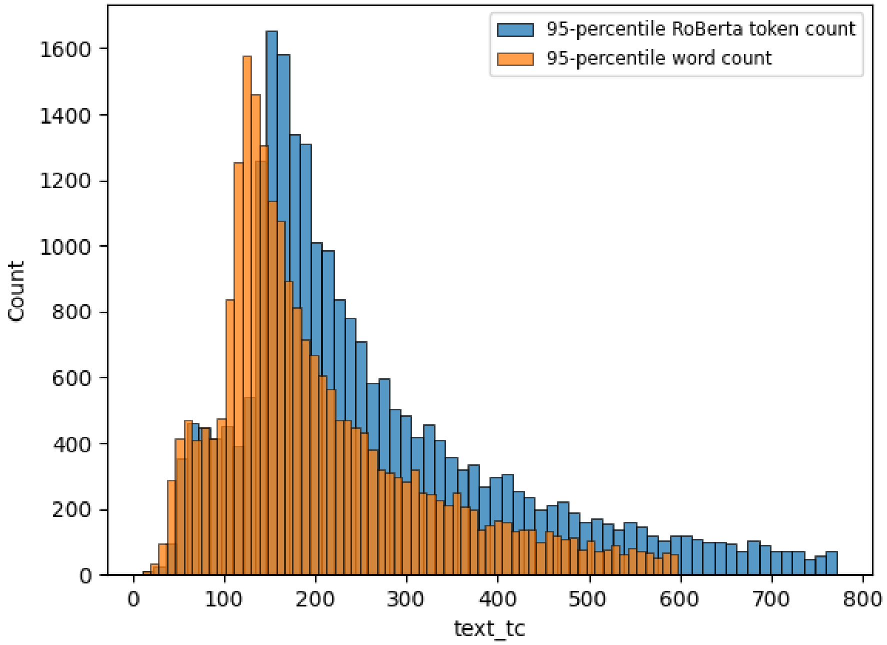

Figure 5.1: Word count and token count distribution of 95-percentiles of reviews. The tokens are generated using RoBERTa's pretrained tokenizer from HuggingFace

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.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

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.

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.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

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.

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.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

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.

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