SVCCA and CKA for Transformer Interpretability in Language Models

Andrea Favia

Andrea Favia
STUDENT NUMBER: 2031608

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COMMUNICATION AND INFORMATION SCIENCES
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

Andrew Hendrickson

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands

Contents

Abstract

Interpretability in models which deal with human language is a field rich in questions and potential, and cutting-edge research brings innovation to the tools used to understand how deep learning models reach an interesting level of capability in the classification, prediction and generation of language. The present thesis focuses on the processing of text data by means of two different transformer models, a POS-Tagger and a Language Model (henceforth LM), to investigate whether these go through similar learning phases when given the same data tagged on different levels of abstraction (namely syntax and lexicon). The intuition is that the models will have to undergo a similar learning process as the POS-Tagger one, since syntactic acquisition is a pre-requisite to the acquisition of higher levels of language, which has been researched in the field of deep learning by taking inspiration from language acquisition.

The models have been trained on the WikiText dataset, which is a widely employed dataset in language modeling and POS tagging tasks, often used for benchmarking models.

In order to investigate how the models learn, and whether they might follow similar learning patterns, the SVCCA and CKA algorithms have been applied to measure and compare layers similarity within and between the models, as well as the same layers across epochs during training. A second goal of this thesis was to determine if the two methods yielded comparable results. These algorithms have successfully allowed to find significant similarities between the first two layers of the models, and also gain insights into the LM structure, suggesting that the two sets of layers should share similar information and have learned similar features. Finally, the results of SVCCA and CKA are shown, and a case is made on which algorithm may be more appropriate to use for certain analyses.

1 Introduction

Transformer models have revolutionized the NLP field by applying what are called self-attention blocks before a linear layer (Vaswani et al., 2017). Before them, LSTM-based models were the state-of-the-art choice for most language tasks (Merity, Keskar, & Socher, 2017; Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2015), although they had costly requirements because of their recurrence component. The combination of a transformer model's self-attention block and the linear layer subsequent to it is usually referred to as an encoder, or decoder, depending on the task; however, for language models, an encoder or decoder-only structure can be used. Architectures involving transformers are quite new, and are starting to be implemented also in fields other than NLP (Parmar et al., 2018), which is why there is a necessity and ample space for research on their interpretability, especially when it comes to achieving it by means of equally novel methods such as SVCCA and CKA.

By applying SVCCA in a similar fashion as in Saphra and Lopez (2018)[1], this study intends to check for correlations within and between layers of transformer models trained on language in text form. The final objective is to determine, by comparing the correlation between two models trained on differently annotated data, whether syntactic features are learned at the same time as lexical ones, or differently, if they are acquired at all.

The motivation behind the method chosen for nodes inspection stems from the fact that, since SVCCA is a simple, yet effective way to compare layers, it can accommodate stringent deadlines and yet be based on solid results, as reported in other studies that applied it to models trained on different tasks (these will be discussed in the following section).

The use of CKA as a second similarity measure is motivated by the need to confirm Kornblith, Norouzi, Lee, and Hinton (2019)'s results, which show this method as more sensitive to finding similarities between layers in different models.

Although relevant work has already been done on the interpretability of deep learning models dealing with text, especially LSTM models (Greff et al., 2015), not as much research has been carried out with regards to interpretability in the case of transformer-based models. While the current state of things makes it particularly challenging to find specific literature on a given issue, it also accounts for the presence of wide scope for analysis, a field full of potential for cases and results which have not yet been fully investigated. It is precisely in this space that the present thesis and the choice of its particular tasks, language modeling and POS tagging, have their roots.

By researching how the models learn to perform these tasks, the questions to which this thesis proposes to provide an answer are the following:

- Can two transformer models trained on different levels of abrastraction learn similar features, and therefore share encoded information between some of their layers?
- When inspecting layer activations between different models and within the same models, do SVCCA and CKA return different similarity scores?

An answer to these questions will allow to de-

[1]In this study, layers from two different LSTM models trained on text were compared. Results give indication which suggests language models undergo similar learning patterns as models trained on part-of-speech tags.

termine which of these similarity metrics is more appropriate to obtain some intuitions and additional information on the learning mechanism of these models, giving a small contribution to the very active field of interpretability, which has been already quite engaged in analyzing transformer models.

2 Literature Review

Although the literature on interpretability is rich in implementation of diagnostic classifiers, this thesis takes on the challenge to adopt more novel and less mainstream techniques of encoded information investigation. Because of their widespread use, one would think that diagnostic classifiers would be the most obvious choice for this thesis. As this is not the case, an explanation of their exclusion will here be presented.

Diagnostic classifiers are very successful in testing hypotheses regarding the features learned by models' layers. They usually consist in a linear model trained on the activations of a specific model layer with the task of testing an hypothesis (Hupkes, Veldhoen, & Zuidema, 2017). By way of example, a diagnostic classifier can be trained on the activations of the first layer of a LM to test whether the input sequence consisted in an active or passive phrase; if the accuracy of the classifier is high, it can be inferred that the feature of voice is encoded in such first layer. Diagnostic classifiers are, however, more expensive in computation and time than similarity algorithms like SVCCA or CKA, since they require to be trained for each layer that has to be tested. The reason for choosing these techniques over diagnostic classifiers is not purely practical: as a matter of fact, diagnostic classifiers are based on the assumption that they decode information from a given layer, while SVCCA and CKA do not decode information and allow to test layers between different models trained on specific tasks. This makes these novel and more obscure techniques dataset- and architecture-agnostic (Raghu, Gilmer, Yosinski, & Sohl-Dickstein, 2017). Such flexibility allows to investigate and test interpretability hypotheses in additional ways, which are not possible with diagnostic classifiers.

Previous work by Saphra and Lopez (2018) shows how, in LSTM models, syntactic features are acquired earlier than semantic and/or topical ones. This is allegedly how the network reaches high performance, and, by freezing the layers with the progression of training, it is possible to obtain a model that can generalize better, and have higher accuracy on the testing set. This technique also allows for shorter training time.

In the field of machine translation, Bau et al. (2018) apply different unsupervised techniques to inspect nodes importance and rank them accordingly, with the aim of identifying interpretable information in these nodes. Once tested, these techniques, among which is SVCCA, showed that single neurons can encode grammatical information, and that their manipulation can affect translation quality. Bau et al. (2018) report on how zeroing the top 10% nodes results in poor performance, and, therefore, that the highly correlated nodes found with SVCCA do, in fact, carry important information.

Finally, a similar process is known to be true for, and apply to, children in the case of phonology and language acquisition (Werker, Tees, Best, Maye, & Sebastián-Gallés, 1984): this process of starting from the complete array of possible sounds to culminate with the discrimination of only those phonemes that are useful to the target language prior to production, is some-what similar to what children's brains go through when learning their first native languages.[2]

2.1 Language Modeling

Language modeling is a task which consists in training a model to predict the next item in a sequence of language symbols (words or characters), so that its input is a sequence of symbols, and its output is the same sequence and the next most likely item according to the model weights. There are many datasets available for language modeling, and, although many usually have a considerable size, such as WikiText-103 or, even more, WebText from GPT-2 (Radford et al., 2019), the problem with these datasets is that the training process takes a substantial amount of time, and they are not particularly friendly to early researchers. One of the advantages, however, is that much literature is especially based on smaller datasets such as WikiText-2 or the PennTreeBank corpora, that are more suited for bench-marking and interpretability tasks, rather than surprising performances. Moreover, resources such as paperswithcode[3] are great registrars that allow to check with which model architectures these smaller but widely employed datasets have been used, and what results might be expected.

2.2 Transformers

Transformer models are models that rely only on self-attention encoder-decoder, encoder only or decoder only architectures in order to process sequences. They are mostly known for their employment in NLP tasks such as machine translation, language modeling and more (Parmar et al., 2018), where the task is to process input sequences of symbols to output sequences of symbols, as is the case with both models' architectures in this work, which are made up only by encoder layers.

As mentioned earlier in the introduction, before transformers, LSTM models (Long-short term memory) based on recurrent neural networks (RNN), were the most employed for state-of-the-art results in NLP tasks, such as: language modeling, machine translation, POS tagging, etc... (Merity et al., 2017; Greff et al., 2015). However, because of their recurrence component, they were computationally costly to train and did not offer the same potential as transformers, which get rid of the recurrent component and are more parallelizable (Vaswani et al., 2017).

In this section, the transformer encoder layer will be explained, although a great resource in order to understand well its structure can be found, aside from Vaswani et al. (2017), in the blog article *The Illustrated Transformer*[4].

First of all, the input to the encoder layer, in this study case, is a sequence of symbols which has gone through an embedding layer. Each word vector in a sentence output by the embedding layer will then be an input vector x to the encoder layer. For each of these inputs, a set of query, key and value vectors $< q, k, v >$ will be produced. Each of these will have their own weight vectors $< W_q, W_k, W_v >$ and, after dot products, we will have $< q', k', v' >$.

Second, dot product is performed between q' and k'^T; these are then scaled down by the square

[2]Children are able to discern between all phones producible by the human vocal tract before the age of 4 months for vowels, and 12 months for consonants (Werker et al., 1984).

[3]https://paperswithcode.com/

[4]http://jalammar.github.io/illustrated-transformer/

root of their dimension size (let it be D), so that the outcome is the following:

$$\frac{q' \cdot k'^T}{\sqrt{D}}$$

Subsequently, the result is passed to a softmax function, and dot product between its output and the v' is performed. The below formula shows how attention is computed. The output of one attention head will be a.

$$Attention = softmax(\frac{q' \cdot k'^T}{\sqrt{D}}) \cdot v'$$

It is also possible to have more than one attention head per layer: just by having n sets of weights for the $< q, k, v >$ vectors, n attention heads output vectors of the same sizes can be obtained, which can be concatenated and multiplied by a weight matrix W_a.

Finally, the whole attention block lies within a residual block, after which layer normalization is applied. Its output is passed through a fully connected linear layer, which lies within a second residual block, and after which there is a second normalization layer.

2.3 Encoder-based models

Transformer models such as BERT (Devlin, Chang, Lee, & Toutanova, 2018) are based only on transformer encoder layers followed by a fully connected layer, which practically acts as a single decoder. The main feature of BERT language models, in particular, is that they use MLM (Masked Language Model), a masking technique inspired by the Cloze task, which consists in masking only a small percentage of the input tokens (e.g. 15%), and base the loss function on the classification of those tokens (Devlin et al., 2018; Rogers, Kovaleva, & Rumshisky, 2020). Pre-trained BERT models have been shown to hold more abstract and task-specific information in the last layers, and more low-level features in the first layers, although, when it comes to syntax, it seems that middle layers are responsible the most for witholding syntactic information (Hewitt & Manning, 2019). Analyzing the attention heads is something that has already been investigated, and while this can be difficult from a qualitative approach, quantitative ones as in Kovaleva, Romanov, Rogers, and Rumshisky (2019) have shown that attention heads can be removed without impacting the performance, or can even improve it, and the same applies to whole layers. It was for this reason that it was decided to focus the similarity analysis on the layer activations only, instead of on the single attention heads. Nonetheless, short mentions will be made with regards to some of the attention head activations in the models.

2.4 SVCCA

Singular Vector Canonical Correlation Analysis (henceforth SVCCA) is an algorithm that allows to obtain a similarity metric between nodes activations from two layers. The layers' dimensions do not have to be the same, and, for this reason, SVCCA is quite convenient to use when comparing layers from different models, and this analysis can even be applied to convolutional layers (Raghu et al., 2017). SVCCA can also be used as a tool for pruning nodes which do not carry relevant information. One of the appealing qualities of this algorithm is its simplicity and use of well-known, established concepts from linear algebra and statistics. In explaining this technique, the layers' activations will be simply referred to as vectors.

The algorithm can be broken down into two steps:

1. A singular value decomposition (SVD) is carried out on both vectors, reducing their dimensionalities to a shared one, which can explain 99% of the variance in each vector (although this is not a fixed parameter).

2. Canonical Correlation Analysis is performed to find similarity between the two vectors, let them be a and b. This consists in finding weights W_a and W_b that will align and maximize correlation as much as possible between the vectors, so that $a' = W_a \cdot a$ and $b' = W_b \cdot b$. The output will then be the set of correlations between each direction in the linearly transformed vectors.

It becomes quite clear, then, the potential which this tool has in calculating similarity not only between a model's layer at different training points, but also across different layers of the same model or other models. This allows to obtain intuitions as to what kind of information layers from two models may share. Examples of this can be found in Saphra and Lopez (2018), Kornblith et al. (2019) and Bau et al. (2018) and will be now briefly illustrated.

In Saphra and Lopez (2018), it is highlighted how SVCCA is a better option than a diagnostic classifier, since it can be applied just on the forward pass of the model on the test or evaluation set. This also applies to CKA.

In Kornblith et al. (2019), it is reported how SVCCA can show similarity within the structure of models as other similarity techniques do – CKA by way of example. However, when it comes to identifying corresponding sublayers in transformers' layers, SVCCA seems to be less sensitive in picking up similarities than CKA with an RBF kernel.

Finally, in the case reported in Bau et al. (2018), it is shown how pruning layers which have high correlation drastically reduced the BLEU score of machine translation models.

2.5 CKA

Centered Kernel Alignment (CKA) is a system that has only recently been applied to measure similarity between ANN's layers, starting in Kornblith et al. (2019). In order to obtain CKA similarity, the Hilbert-Schmidt Independence Criterion (HISC) (Gretton, Bousquet, Smola, & Schölkopf, 2005) has to first be obtained from the activation vectors. In this case, too, the activation vectors have shape $V_{S\times F}$, where S is the number of samples and F the number of features, which are the result of the product of the sequence length and output dimension sizes of a given layer. The result of HISC is then made invariant to uniform scaling by normalization:

$$CKA(A,B) = \frac{HISC(A,B)}{\sqrt{HISC(A,A)HISC(B,B)}}$$

In addition to a linear operation, an RBF kernel has been tested as well. As it will be illustrated, results are similar for the two methods, and this has been reported in the literature as well.

CKA was chosen because it is efficient, easy to compute, and seems to find similarity better than other systems like SVCCA, CCA and linear regression (Kornblith et al., 2019).

3 Methods

3.1 Tools and Algorithms

This research was conducted using exclusively the Python programming language, version 3.6.

The libraries that have been used or adjusted according to the research needs will now be listed, and can also be found in the *requirements.txt* file in the main repository[5].

- A forked version of PyTorch1.3.1. Two modules have been modified so that retrieval of the activations would be more accessible from the Transformer Encoder layer.
- TorchText 0.3.0, a package from PyTorch which offers a wide variety of datasets, functions and classes for creating pre-processing pipelines that are tailored for NLP tasks. TorchText also has a good integration with SpaCy, which can be used as tokenizer when pre-processing a dataset. Among the datasets included in the package, there is also WikiText-2.
- CCA_core, a module from Google's SVCCA repository[6]. This module has slightly been modified by using some of the functions from the PyTorch library instead of the ones from Numpy, in order to easily implement computations on GPU and make the process slightly faster.
- CKA implementation from Kornblith et al. (2019), from Google's research Colab[7].
- Seaborn and Matplotlib, for plotting the data.
- MLFlow, to track the training process of the models and obtain ready-to-use plots of the metrics being tracked, such as loss and perplexity scores. MLFlow is a very nice open source tool which can greatly help keeping track of several experiments. It is quite flexible in its implementation, covers most mainstream frameworks (such as scikit-learn, TensorFlow and PyTorch), and, most importantly, it can make reproducibility easier if set up diligently.
- SpaCy, used solely for tokenization in the LM, and for creating the target tags (target values) in the POS Tagger model. SpaCy can be used with several sets of tags, and the one which was chosen for this study was the Universal Dependencies v2 POS tag set.

3.2 Datasets

The dataset selected for this study was the WikiText-2 dataset, designed for language modeling. It contains 720 articles from Wikipedia, for a total of 2,551,843 tokens and 33,278 unique terms. These are divided into the following: training set – 600 articles with 2,088,628 tokens; validation set – 60 articles with 217,646 tokens and finally, test set, with 245,569 tokens (Merity, Xiong, Bradbury, & Socher, 2016).

Another dataset, WikiText-103, is also available, with 28,475 articles, totaling more than 100

[5]https://github.com/andcarnivorous/TestingTransformers

[6]The module can be found here https://github.com/google/svcca

[7]https://colab.research.google.com/github/google-research/google-research/blob/master/representation_similarity/Demo.ipynb

million tokens. Due to its size, however, this dataset would have probably not been feasible to work with, given the hardware restrains concerning this study.

WikiText-2 is available in 2 different versions, raw and pre-processed. The pre-processed version has <unk> (unknown) tokens already implemented in the text, and the only thing which needs to be added are the <sos> (start of sentence) and <eos> (end of sentence) tokens. In this study, the raw dataset was used for the POS Tagger, while the pre-processed dataset was adopted for the Language Model.

The reason behind the selection of WikiText-2, lies in the fact that it has been often used for testing state-of-the-art architectures on language modeling tasks, from LSTM models (Merity et al., 2017) to the more recent GPT-2 architecture (Radford et al., 2019). Moreover, pre-processing of the WikiText-2 dataset is already well implemented in TorchText, as mentioned in the Tools & Algorithms paragraph as well.

3.3 Language and POS-Tagging models

The LM architecture consisted in a first embedding layer, followed by a positional encoding layer (explained in section 3.3.1), and 6 transformer encoder layers followed by a linear output layer. The embedding layer would take a sequence input and output a vector with 240 dimensions, which is then injected with positional clues by the positional encoding layer, which also performs .5 dropout normalization. The 6 transformer encoders have 12 attention heads and are followed by a linear layer with 1024 dimensions each. Every encoder also performs .1 dropout normalization.

The POS-Tagger has the same embedding dimensionality, and a positional encoding layer, which applies the same dropout set at .5. These layers are followed by 2 transformer encoder layers with the same dropout as the LM and the same number of linear nodes at the end of them.

Both layers have attention masks implemented, which obscure the next item in the input sequence in order to prevent the attention block from being able to see the next items in the sequence.

The choice behind having less layers for the POS-Tagger lies in the fact that, as a task, POS tagging is less complex than language modeling, and in Saphra and Lopez (2018) the POS taggers have less layers than the LM.

3.3.1 Positional Encoding & Transformer Encoder

The transformer encoder is made up by a self-attention block followed by a feed-forward layer. Each of these layers is wrapped into residual blocks, and, between the embedding layer and the first encoder layer, there is a positional encoding layer which injects positional information in the vector output by the word embedding layer. The first original type of positional encoding presented, and possibly the most widely used (Vaswani et al., 2017, pp. 5-6), is implemented in this thesis.

3.3.2 Attention mask

The attention mask is a way to ensure that the encoder layers cannot see the item (word vector, in this case) which follows the present one in the sequence. Allowing access to the next token to the attention block would be equivalent to cheating, since the attention block would already know what the next target values will be. Therefore, the attention mask is a square matrix where the entries M_i,j on the diagonal and above it are set to *-inf*.

To further visualize this, in Figure 1, the activations of each single attention head per layer can be observed when a sentence is fed to the model. These are the activations after $dropout(query \cdot key^T)$. Instead, an example of head activations of a model without attention mask can be found in Appendix A.

It must be kept in mind that encoder-based models like BERT normally use MLM, as explained in subsection 2.3. In this thesis, however, a middle ground is found by using an architecture for the LM which consists of only encoding layers as with BERT, while the masking applied to the input is a sequential mask like the one applied to the decoder layers in Vaswani et al. (2017). This choice was mainly inspired by the example of sequence-to-sequence modeling given in the official PyTorch documentation[8], and allows to study in detail a more homogeneous architecture than the one in Vaswani et al. (2017), without having to resort to more complex masking and loss computation.

3.3.3 Training

1. Language Model

   For the LM, training was carried out for 50 epochs, using Stochastic Gradient Descent as optimizer and a scheduler that would reduce by 30% the learning rate every 4 epochs, starting from a learning rate of 5.8. The batch size selected was 64, while the sequence length 32; for the latter, it would be optimal to go with longer sequences, but this has quite an impact on the memory allocated on the GPU, which, for this experiment, was limited to 4GB. Table 1 sums up the parameters of the experiment, which can also be found on MLFlow.

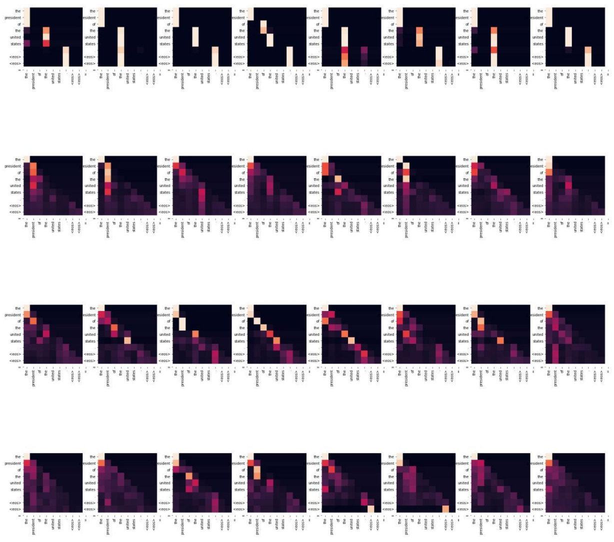
Figure 1: Attention heads' activations from a LM model with 8 heads.

Parameter	Value
batch_size	64
embed_dim	240
epochs	50
eval_batch_size	64
heads	12
layers	6
lr	5.8
seq_len	32
test_data.shape	[3806, 64]
train_data.shape	[32422, 64]
val_data	[3377, 64]

The language model was evaluated based on the perplexity score on the evaluation and

[8]https://pytorch.org/tutorials/beginner/transformer$_t$*utorial.html*

test sets. A perplexity score is just an exponentiation of the loss.

2. POS-Tagger Model

   The POS-Tagger was trained for the same amount of 50 epochs, with the same batch and sequence sizes as the LM model. The parameters that differ are the number of layers and attention heads: where the LM model has 6 layers and 12 attention heads, the POS-Tagger has 2 layers and 8 attention heads for each. Learning rate also differs, starting at 1.8 and being reduced by 35% every 2 epochs. Finally, the target values for the LM were the 28865 possible words, while, for the POS-Tagger, the 50 possible POS tags from the SpaCy Dependencies v2 POS tag set constituted the target values. Table 2 shows the parameters extrapolated from MLFlow.

Parameter	Value
batch_size	64
embed_dim	240
epochs	50
eval_batch_size	64
heads	8
layers	2
lr	1.8
seq_len	32
test_data.shape	[3785, 64]
train_data.shape	[32130, 64]
val_data	[3353, 64]

   The POS-Tagger was evaluated on the accuracy on the validation and test set.

For both models, in order to obtain reliable training loss by epoch metrics, the training loss was computed at the end of each epoch by inference without backpropagation on the whole training set. Layer activations, instead, were saved from evaluation of the validation set, which also occurred after every training run with backpropagation. The loss function employed for both models was cross-entropy loss defined as:

$$Cross - EntropyLoss = -log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right)$$

3.3.4 Activations Collection and Processing

In order to collect activations from the encoder layers of Transformer, the source code from PyTorch *functional.py* and *transformer.py* was changed, so that it would be easy to retrieve activations at the dropout point, at the $softmax(query \cdot key) \cdot value$ point, and at the linear output point of each encoder block [9].

Activations were serialized for every epoch and for every encoder layer, and encoder layers only, at the end of every batch, organizing the serialized activations by model name, layer, epoch and batch number.

In oder to collect the activations in a way that would be feasible, given the hardware limitations, activations were serialized at the end of every batch, creating files easy to manage with limited RAM resources to later be concatenated and processed for the similarity analyses. This, in short, is the reason why the activations are not saved in single files per epoch, but divided even further. However, code that allows to retrieve the activations and concatenate them by several criteria is available in the repo as well.

[9] See commit link here

Since for this study three main types of similarities, listed below, have been tested between and within models, for each one of these processes a different approach has been taken in order to process them without incurring into full memory problems.

- Epoch by epoch against same epoch
- Epoch by epoch against final model
- Final layer by final layer

In the first case, in an iterative fashion, the 50 epochs of activations per layer have been divided by 10, saving the similarity scores from SVCCA and CKA at each iteration. The same was done in the second case, going, however, with a higher number of epochs per iteration, since all activations from the POS Tagger model were compared only with the final epoch of the LM model's same layer. In the third and last case, everything was computed in a single run, as it would easily fit into memory.

3.4 Application of SVCCA and CKA on Models' Layers

3.4.1 SVCCA

SVCCA consists in a canonical correlation analysis (CCA) applied to the nodes' activations of two layers after they have been reduced in dimensionality by means of singular value decomposition (SVD), to a point where their new dimensions can still explain an arbitrarily-chosen portion of the variance of the original vector. In order to apply it, the function get_cca_similarity available from google's core_cca module[10] was used on the collected activations, returning the mean SVCCA score across the nodes of the two layers being compared. The code from the module was slightly changed using PyTorch tensors, and tensor operations, moved to the GPU at certain points in order to save computation time (although the difference has proved to be minimal).

3.4.2 Linear and RBF CKA

In order to carry out the analysis using CKA, the functions defined in Kornblith et al. (2019) Colab's notebook [11] were used. These offer both a linear version of the CKA and one with a RBF kernel implementation. For this module, no changes have been made, especially because computation times for CKA with or without RBF kernel are quite fast and computationally inexpensive compared to SVCCA.

[10]The module is available at the github repo https://github.com/google/svcca

[11]The notebook can be found at this address here

4 Results

4.1 Training results

4.1.1 POS-Tagger Model

The POS-Tagger model converged after few epochs, although training ran for 50 epochs. Figure 2 and Figure 3 illustrate the train and validation loss per batch, and the train and validation accuracy per batch. The model scored 93.2% accuracy on the test set, with a loss score of 0.324

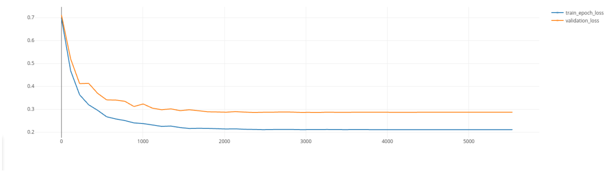

Figure 2: POS-Tagger train loss.

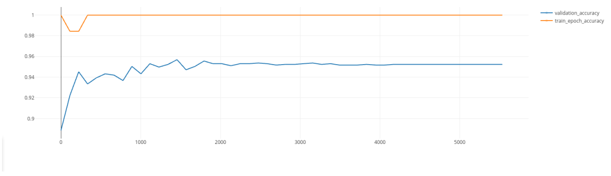

Figure 3: POS-Tagger train accuracy.

The model, therefore, with only 2 layers and 8 attention heads, was able to converge to good standards in a short amount of time. The attention head activations on a sample sentence can be seen in Figure 4. These head activations are not easy to interpret, and although some of them can give hints and clues which allow to manually infer possible embedded information, it cannot be expected to perform such analysis on all of the attention heads in a model across several input samples. This is also the case with other models, likes LSTM's hidden cells, as Hupkes et al. (2017) elegantly put it, such visual analyses are more on the qualitative side rather than on the quantitative. Moreover, as mentioned in subsection 2.3, attention heads that do have information embedded which may appear as essential for key tasks, may not really be essential to the model which could still perform well regardless (Kovaleva et al., 2019).

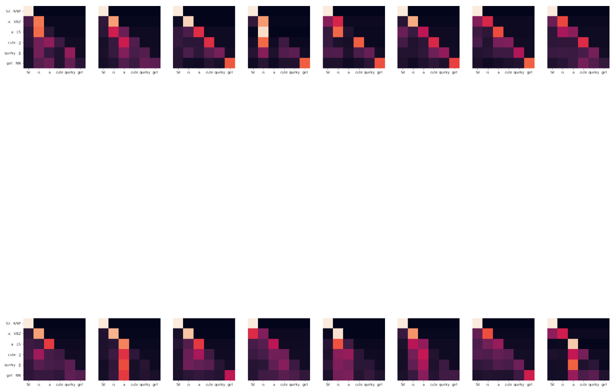

Figure 4: POS-Tagger attention heads activations. Each row is a layer, each column an attention head.

4.1.2 Language Model

The LM was trained for 50 epochs as well, using as evaluation metric perplexity, which is an exponentiation of the loss. After training, the model managed to get to a perplexity of 163.8 on the test set. Figure 5 shows the train and validation loss during training. The result is not state-of-the-art, but it was enough to compare the layers' activations among themselves and against the activations of the POS-Tagger, to obtain meaningful and interesting results, which will be introduced in the following paragraphs.

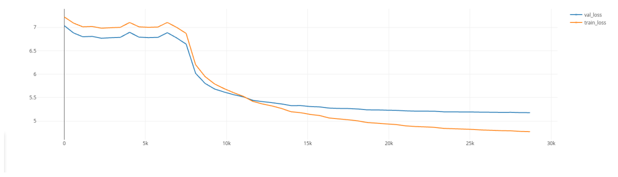

Figure 5: Language Model training and validation loss. Batch number on the x axis.

4.2 Similarity Scores Within Models

The results from the similarity scores between layers within the same models will be presented first. These similarity scores are relevant, in that they give insights into when the layers have started to converge, which, in practice, can be useful to obtain faster training times by means of Freeze Training, as reported in Raghu et al. (2017). These insights are also effective when trying to reproduce patterns which are often found in NLP models, like the fact that lower layers converge faster than higher layers, and that the first layers usually carry similar information (Kornblith et al., 2019).

This pattern was found in the LM model especially thanks to the CKA algorithm in its linear form and with an RBF kernel as well, while SVCCA did not pick up enough similarity to advance any suppositions on the information encoded in the layers, although it did work well to show the converging process in the layers. As Figure 6 and Figure 7 show, by obtaining linear CKA and CKA with RBF kernel similarity scores between all layers from the trained LM model, the first 4 layers and the last 2 layers respectively form clusters of similarities, meaning the two groups are not as similar between themselves as the layers within them are among themselves. This is also confirmed by looking at the similarity between each layer's activations and its converged form's activations (Figure 8), where the last two layers develop similarity differently and with a lower score than the first layers. It is also worth noticing that the CKA score for the second and third layer is higher and raises before than that of the first layer. Even if SVCCA did not perform well when tested between different layers, it performed quite well when testing similarity between the same layer at different points in training and its fully trained version. Figure 9 illustrates how CKA is able to pick up similarity at earlier points in training, although SVCCA outperforms it later in the analysis. It is important to notice as well that, with both algorithms, the point in which the layer's similarity starts increasing is the same, suggesting that both techniques are capable of finding pivotal points in the training process of a layer.

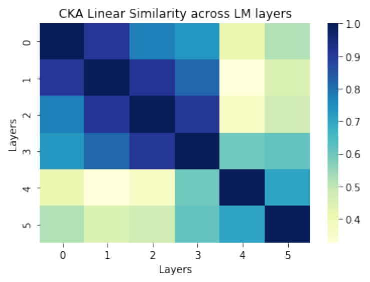

Figure 6: Linear CKA similarity score between layers of converged LM model.

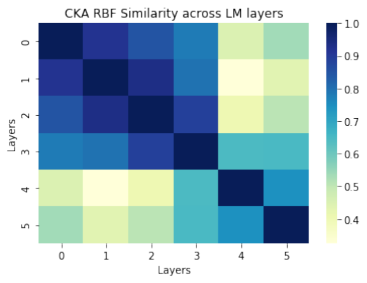

Figure 7: CKA with RBF kernel similarity score between layers of converged LM model.

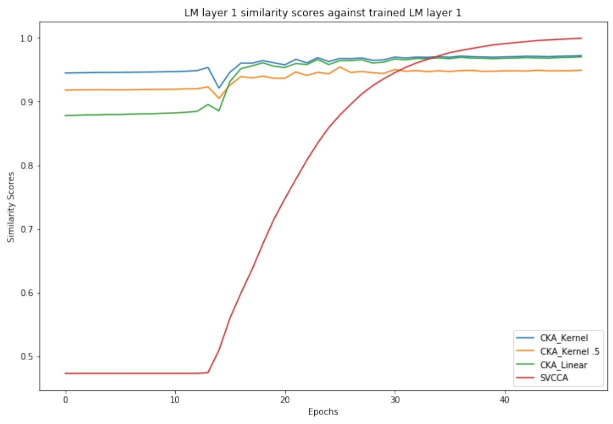

Figure 9: LM layer 1 similarity scores against trained layer.

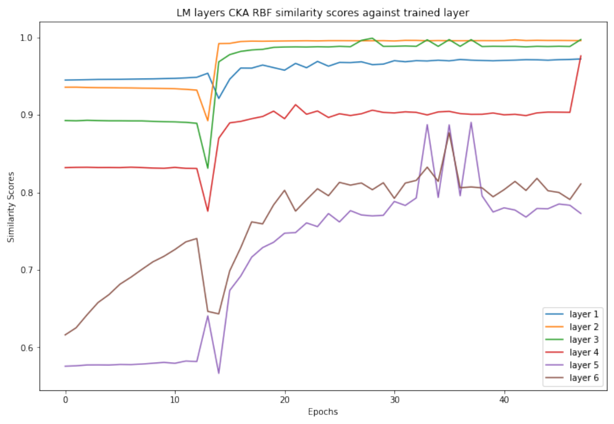

Figure 8: CKA RBF similarity score of every LM's layer and its converged point.

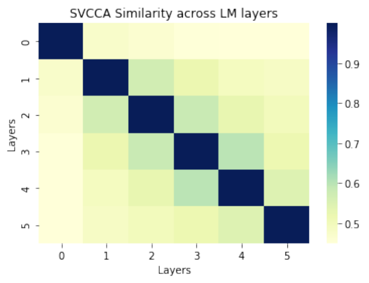

Figure 10: SVCCA similarity score between layers of converged LM model.

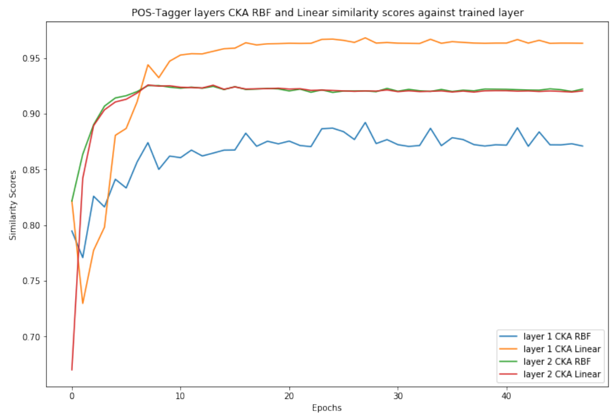

Figure 11: CKA scores for both layers of the POS-Tagger model.

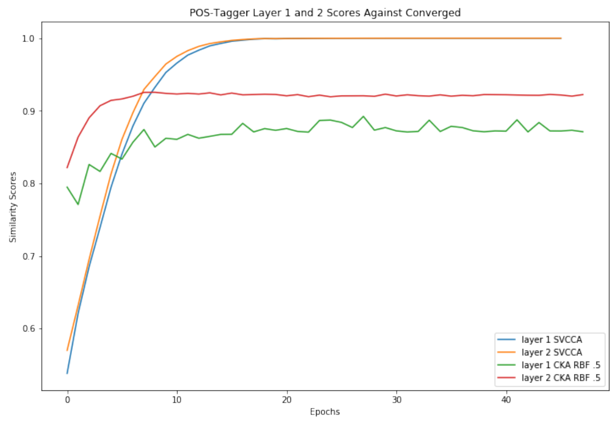

Figure 12: CKA and SVCCA scores on the first and second layer of the POS-Tagger model.

For the POS-Tagger, both SVCCA and CKA have successfully found increasing similarity between the 2 layers of the model and their converged forms. Figure 11 highlights the CKA results while, in Figure 12, the RBF with threshold .5 are compared to SVCCA results. It is worth noticing how SVCCA seems to outperform CKA at later points in training, when analyzing the same layer against its trained activations, and the same is in the case of the LM.

4.3 Similarity Scores Between Models

When similarity scores between the first 2 layers of the POS-Tagger model and the first 2 layers of the LM model were tested, CKA has proved to pick up similarity between activations substantially better than SVCCA. The SVD-based algorithm showed a similarity peaking at around .46 for the first layers, and .44 for the second layers. On the other hand, CKA has consistently provided meaningful results. The algorithm was used with its linear configuration and two RBF kernels, one with a threshold of .5 and the other with a threshold of 1. As Figure 13 and Figure 14 illustrate, interestingly, the second layers' CKA similarity scores have more consistent and higher trends than the ones of the first layers. Additionally, the second layer seems to slightly decrease and plateau after initially reaching high similarity, while the first only shows an increasing trend that eventually plateaus. However, they both seem to reach the point of similarity saturation before the $20^{th} epoch$.

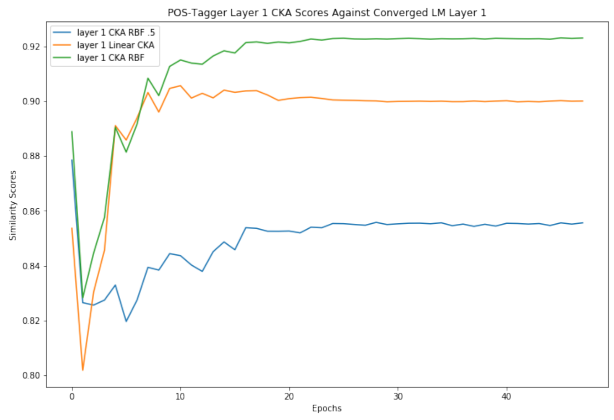

Figure 13: CKA scores: POS-Tagger layer 1 against trained LM layer 1.

Layer	SVCCA	CKA Linear	CKA RBF .5	CKA RBF
1	0.46	0.90	0.85	**0.92**
2	0.44	0.887	0.889	**0.91**

Table 1: POS-Tagger trained layers similarity scores against corresponding trained LM layers.

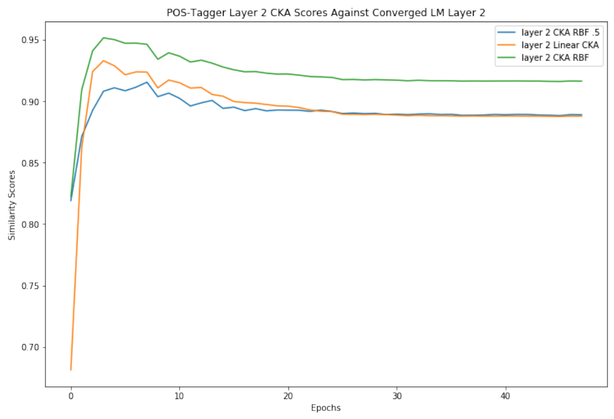

Figure 14: CKA scores: POS-Tagger layer 2 against trained LM layer 2.

Given the results on the corresponding trained layers (Table 1), CKA performs better than SVCCA between layers from transformer models trained with different targets, but same datasets, and different number of encoder layers. The similarity scores from the algorithm may give suggestion that the encoder layers from the two models may contain similar information, making it a valid tool in investigating what information may be encoded in the nodes of different models. In particular, the linear kernel and the RBF kernel with threshold set to 1 have proven to find higher similarity than SVCCA and the RBF kernel with threshold on .5.

By following the example of the sanity tests for similarity indexes performed in Kornblith et al.

	CKA Linear		CKA RBF		CKA RBF .5		SVCCA	
	LM L1	LM L2	LM L1	LM L2	LM L1	LM L2	LM L1	LM L2
POS L1	0.90	**0.92**	**0.92**	0.86	**0.85**	0.73	**0.46**	0.45
POS L2	0.80	**0.88**	0.85	**0.91**	0.86	**0.88**	**0.45**	0.44

Table 2: Similarity scores between the first two trained layers of the POS-Tagger and LM models. Bold numbers represent the highest scoring similarity between the two layers.

(2019), Table 2 shows which layer between the first two of the LM has higher similarity to each of the POS-Tagger layers. Only the CKA with RBF kernels found highest similarity for each layer with the corresponding layer of the other model, while SVCCA and CKA with linear kernel only had one layer correspond to the one at the same index in the other model. These results, especially the one from the RBF kernels, add up to the evidence that the first two layers may carry analogous information.

Finally, it is worth noting how the differences between the SVCCA scores are of .01, while the ones from the different CKA configurations hold wider differences, except in the case of the first POS-Tagger layer with the linear CKA. This is supplemental evidence in favor of CKA as a more sensitive tool to pick up similarity.

5 Discussion

This thesis had set the goal of testing two main hypotheses – whether transformer models trained on different levels of abstraction encode similar information, and whether CKA and SVCCA give similar results when inspecting layer activations.

The results from subsection 4.3 have shown that there is, indeed, high similarity between the first two layers of the POS-Tagger model and the first two layers of the LM. These similarities were successfully tracked during training, and displayed increasing trend until saturation for the first layer. Moreover, testing which layer would score more similar to either of the other model also confirmed that, by using CKA with an RBF kernel, there is correspondence between the indexes of most similar POS-Tagger layers to the first two of the LM, suggesting that they carry similar information. It must be kept in mind that these similarity metrics give intuitions at best as to what kind of information could be encoded in deep neural network layers: it is not possible to determine their contents by these means only. They do provide, however, meaningful insights when used with models trained on different levels of abstraction (syntax and lexical, in the case of this thesis) or different datasets, since, as it is mentioned in Saphra and Lopez (2018, p. 6), a model trained on a specific level of abstraction, such as syntax, should be similar to a model that encodes that levels of abstraction. Given this premise, and by accepting the presuppositions as to what these similarities indicate, it can be stated that transformer models trained on different levels of language abstraction do encode similar information in their lower layers, and since the POS-Tagger is trained on syntactic classification, it is possible that that is the type of information encoded in the first layers of the LM.

With regards to the second hypothesis, it has been determined that SVCCA and CKA clearly differ in the results they give when testing layer similarities. As already reported in Kornblith et al. (2019), CKA finds earlier layers in the network, reaching a level of similarity saturation earlier than higher layers. This was confirmed in other works, such as in Raghu et al. (2017), and in the present research as well. SVCCA seems to work especially well when tracking the similarity of a single layer across its training process; however, in this instance too, CKA has shown to detect similarity earlier than SVCCA. Without taking into consideration the computation times involved in the processing of these two algorithms[12] (CKA being twice as fast as SVCCA partially on GPU), CKA has resulted to be the better tool to be used when investigating transformers models, in order to find similarity between different models, within the same model, or across training. Particularly, it was able to show the structure of the LM model, and which layers were more similar to each other, suggesting they may encode similar information. All of this seems to indicate that SVCCA might be more suited to finding similarities which are more obvious, since it has been shown in subsection 4.3 that similarity started to increase and saturate at the same time with both algorithms, while CKA may be better in finding similarities based on the tails and shape of the distributions, since it centers the points of similarity found, so that comparisons between not only different layers, but also different models or differently initialized models (Kornblith et al., 2019) may return more meaningful results.

6 Conclusion

Research into interpretability of deep learning models, especially NLP ones, is a vibrant and active field which offers many opportunities and potential to find hints or even answers on how these models learn. This thesis has taken on the challenge to gain insights into the shared encoded information between two transformer models trained on different levels of linguistic abstraction, by taking inspiration from previous, although very recent, work, and testing algorithms which have not yet seen wide-spread implementation, especially in NLP tasks. The results have been promising, considering the physical and time-related limitations, confirming literature previous findings and successfully answering the hypotheses put forward. CKA and SVCCA have been proved to give important insights into models' architectures and also test shared information between layers from models trained differently. The first, in particular, has been able to suggest similar features are learned by the first two layers of both models. At the same time, SVCCA has been shown to pick up important points in the learning process of a layer, although it has not been successful in finding similarities between the models.

Further research into transformer models trained on different data and different levels of abstraction is needed, and would allow for a better understanding on how these architectures learn features, and where specific types of information are encoded. While studies of this nature on language models require significant resources, they can shed light on systems which would be more cryptic otherwise. Hopefully, more tests like these will be performed in the near future, and even better systems than the algorithms employed here will surface.

[12] Saphra and Lopez (2018, p. 3263) still make a valid point as to how SVCCA is better than a diagnostic classifier, since there is no training involved in the use of the algorithm, and so is true for CKA as well.

7 Acknowledgements

I would like to thank my thesis supervisor, professor Andrew Hendrickson, for being of such great support and help during such a difficult and tumultuous time in modern history. I would also like to thank my Emacs config and Orgmode, which have made compiling this thesis possible.

8 References

Bau, A., Belinkov, Y., Sajjad, H., Durrani, N., Dalvi, F., & Glass, J. R. (2018). Identifying and controlling important neurons in neural machine translation. *CoRR*, *abs/1811.01157*. arXiv: 1811.01157. Retrieved from http://arxiv.org/abs/1811.01157

Belinkov, Y., Màrquez, L., Sajjad, H., Durrani, N., Dalvi, F., & Glass, J. (2018). Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. arXiv: 1801.07772 `[cs.CL]`

Cortes, C., Mohri, M., & Rostamizadeh, A. (2012). Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, *13*(Mar), 795–828.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv: 1810.04805 `[cs.CL]`

Gong, C., He, D., Tan, X., Qin, T., Wang, L., & Liu, T.-Y. (2018). Frage: Frequency-agnostic word representation. In *Advances in neural information processing systems* (pp. 1334–1345).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. http://www.deeplearningbook.org. MIT Press.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). Lstm: A search space odyssey. cite arxiv:1503.04069Comment: 12 pages, 6 figures. doi:10.1109/TNNLS.2016.2582924

Gretton, A., Bousquet, O., Smola, A., & Schölkopf, B. (2005). Measuring statistical dependence with hilbert-schmidt norms. In

S. Jain, H. U. Simon, & E. Tomita (Eds.), *Algorithmic learning theory* (pp. 63–77). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hewitt, J. & Manning, C. D. (2019). A structural probe for finding syntax in word representations. In *Naacl-hlt*.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

Hupkes, D., Veldhoen, S., & Zuidema, W. (2017). Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. arXiv: 1711.10203 [cs.CL]

Karpathy, A., Johnson, J., & Li, F. (2015). Visualizing and understanding recurrent networks. *CoRR*, *abs/1506.02078*. arXiv: 1506.02078. Retrieved from http://arxiv.org/abs/1506.02078

Kornblith, S., Norouzi, M., Lee, H., & Hinton, G. E. (2019). Similarity of neural network representations revisited. *CoRR*, *abs/1905.00414*. arXiv: 1905.00414. Retrieved from http://arxiv.org/abs/1905.00414

Kovaleva, O., Romanov, A., Rogers, A., & Rumshisky, A. (2019). Revealing the dark secrets of BERT. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 4365–4374). Hong Kong, China: Association for Computational Linguistics. doi:10.18653/v1/D19-1445

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. *CoRR*, *abs/1708.02182*. arXiv: 1708.02182. Retrieved from http://arxiv.org/abs/1708.02182

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *CoRR*, *abs/1609.07843*. arXiv: 1609.07843. Retrieved from http://arxiv.org/abs/1609.07843

Nguyen, H. (2019). Simplebooks: Long-term dependency book dataset with simplified english vocabulary for word-level language modeling. *arXiv preprint arXiv:1911.12391*.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., & Ku, A. (2018). Image transformer. *CoRR*, *abs/1802.05751*. arXiv: 1802.05751. Retrieved from http://arxiv.org/abs/1802.05751

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raghu, M., Gilmer, J., Yosinski, J., & Sohl-Dickstein, J. (2017). Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Nips* (pp. 6076–6085). Retrieved from http://dblp.uni-trier.de/db/conf/nips/nips2017.html#RaghuGYS17

Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). A primer in bertology: What we know about how bert works. arXiv: 2002.12327 [cs.CL]

Saphra, N. & Lopez, A. (2018). Understanding learning dynamics of language models with svcca. *CoRR*, *abs/1811.00225*. Retrieved from http://dblp.uni-trier.de/db/journals/corr/corr1811.html#abs-1811-00225

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Retrieved from https://arxiv.org/pdf/1706.03762.pdf

Vig, J. (2019). Visualizing attention in transformer-based language representation models. arXiv: 1904.02679 `[cs.HC]`

Vig, J. & Belinkov, Y. (2019). Analyzing the structure of attention in a transformer language model. arXiv: 1906.04284 `[cs.CL]`

Voita, E., Sennrich, R., & Titov, I. (2019). The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. *arXiv preprint arXiv:1909.01380*.

Wang, C., Ye, Z., Zhang, A., Zhang, Z., & Smola, A. J. (2020). Transformer on a diet. *arXiv preprint arXiv:2002.06170*.

Werker, J. F., Tees, R. C., Best, C. T., Maye, J., & Sebastián-Gallés, N. (1984). Cross-language speech perception: Evidence for perceptual reorganization during the first year of life. discussions.

Wu, S., Conneau, A., Li, H., Zettlemoyer, L., & Stoyanov, V. (2019). Emerging cross-lingual structure in pretrained language models. *arXiv preprint arXiv:1911.01464*.

Appendices

A Attention Heads

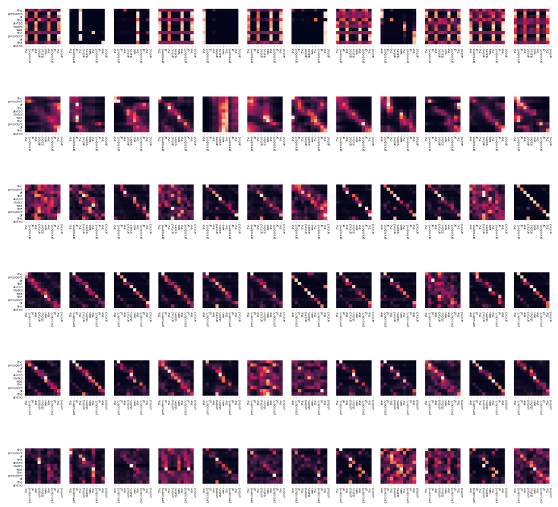
Figure 15: Attention heads activations of LM model with 6 layers and 8 attention heads per layer, trained without attention mask.