# Intrinsic Properties and Biases of Deep Learning Models

Maksim Podkorytov

FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

# INTRINSIC PROPERTIES AND BIASES OF DEEP LEARNING MODELS

By

MAKSIM PODKORYTOV

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2021

Maksim Podkorytov defended this dissertation on November 19, 2021.
The members of the supervisory committee were:

Xiuwen Liu
Professor Directing Dissertation

Adrian Barbu
University Representative

Alan Kuhnle
Committee Member

Peixiang Zhao
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

# TABLE OF CONTENTS

**Appendix**

# LIST OF TABLES

# LIST OF FIGURES

# ABSTRACT

Statistical data models based on the deep learning paradigm have shown remarkable performance in many domains, surpassing human performance in a set of tasks under restricted settings. However, the fundamental reasons enabling these achievements have not been well understood. The understanding of the underlying mechanisms of modern deep learning models is paramount for the following reasons. First, we would like to properly use the models for downstream tasks; for this purpose we need to identify and understand the failure modes of the models. Second, we would like to identify and quantify the biases the models exhibit to facilitate the fair use of the models when protected groups of population are involved. However, illuminating the underlying mechanisms is very challenging due to their size and complexity.

In this dissertation, we study intrinsic properties of two publicly available deep learning models, BERT in the domain of natural language processing, and Glow in the domain of computer vision. Both models have achieved the state of the art for their respective tasks at the time of their publication, and such remarkable performance makes them attractive study subjects. While the natural language processing and computer vision domains are drastically different, common analysis techniques can be applied to both models; the observation also implies that similar techniques could be used in other domains not covered by the current dissertation. In both cases, we analyze the geometric properties of the embeddings and quantify the layer-by-layer transformations through correlations and other analyses. The results lead to novel insights on how the models generate intermediate representations and give rise to their performance, and enable better understanding of the underlying mechanisms.

# CHAPTER 1

# INTRODUCTION

History of deep learning models dates back to 1943 [48] when first multi layer networks of artificial neurons modeling the biological neurons have been designed. Modern deep learning models show remarkably decent performance in many areas, to list a few, machine translation [83], text summarization [68], facial features classification [45], real-time object detection [62], protein folding [35], game of Go [71]. These successes have been attributed to availability of large datasets due to large scale data collection facilities, such as World Wide Web, social networks, street cameras, mobile phone sensors, as well as utilizing specialized hardware, such as Graphical Processing Units and later Tensor Processing Units [34], for more efficient data parallel computation. The remarkable performance and ease of deployment of machine learning models based on the deep learning paradigm [1, 53] lead us to the situation where predictions made by these models and embedded in mobile and web apps and affect everyday life.

On the one hand, most variants of machine learning models based on the deep learning paradigm commonly rely on some form of gradient descent optimizing loss function designed to take arguments as outputs from the neural network, where the gradient is automatically computed using a variant of chain rule for calculating derivatives of composite functions and parameters are updated using backpropagation [37, 66]. On the other hand, modern deep learning models may consist of hundreds of hidden layers containing hundreds of millions learnable parameters (110M in case of BERT-base [16], or 175B in case of GPT-3 [10]). For the reason of scale and connectivity, analysis of emergent properties of deep learning models is complicated.

One line of research concerning understanding of deep learning models takes a theoretical approach to the problem, analyzing collective properties of the models learned with stochastic gradient descent [32]. However, this approach is not applicable to real deep learning models, such as the ones studied in this dissertation, BERT [16] in the domain of natural language processing and Glow [40] as a generative model in the domain of computer vision. In this dissertation, we overcome the large scale issues by analysing properties of distributions of intermediate representations based on two concrete examples, and our approach may be adapted further for analysis of other deep learning models.

BERT is a deep learning model in the domain of natural language processing. It was introduced in 2019 using elements of Transformer architecture [76] and was used as a base to fine-tune state-of-the-art models for General Language Understanding Evaluation [80] that included natural language inference, sentiment classification, question answering, syntactic correctness evaluation. The process that enables good performance on the benchmark consists of two steps: first there is pre-training with large text corpus, followed with fine-tuning with individual smaller dataset for each of the smaller tasks. Since then it has been used as a base for a sequence of related models [44, 41, 27], which change the training procedure, the layer sizes, training objective to achieve better performance than the baseline, so the designated BERT-inspired models being a kind of local optimum for NLP. Such successes have inspired us to study the BERT internals to better understand the reasons for good performance.

Glow is a deep generative model in the domain of computer vision based on discrete normalizing flow. Normalizing flows represent a series of bijective functions, named flow steps, so that there is one to one correspondence between data space and latent space. Latent space distribution is defined to be close-form, for instance, multivariate normal with zero mean and diagonal covariance. Synthesis may be performed by sampling in the latent space and applying inverses of functions in the reverse flow step order. Density estimation may be performed by applying flow steps in their original, forward, order, and, since each flow step is bijective, meaning continuous, and most often by design differentiable, account for density change multiplier with the change of variables formula. We have picked this deep learning model since it has been demonstrated to synthesize photorealistic images, unlike predecessors [17, 18]. Furthermore, as Glow is a normalizing flow, its intermediate representations allow probabilistic interpretation in contrast to other families of deep generative models such as generative adversarial networks [24] and variational autoencoders [39].

Despite their superficial difference, BERT and Glow have one important common property, homogeneous intermediate representations across their hidden layers. This property provides us with the opportunity to study these intermediate representations and analyse the induced distributions; as a result, we develop a better understanding of the internal mechanisms of these deep learning models.

The rest of this document is structured as follows. We start the dissertation with analyzing patterns in BERT embeddings and intermediate representations in Chapter 2. We proceed with analyzing the fragility of token reconstruction by pre-trained BERT in Chapter 3. We introduce language model underlying BERT, design the algorithm for the language model computation and suggest a modification of the algorithm for systematic identifying and evaluating biases in Chapter 4. We switch to study the difference between the training dataset and sampled image distributions in Glow, discover and quantify biases, and suggest a method to control high level features of synthesized images via low level manipulations of intermediate representations in Chapter 5. In Chapter 6, we conclude the dissertation with a spin-off analysis of a theoretical result, allegedly pertaining to a wide class of models including deep learning models, to study its practical implications.

# CHAPTER 2

# GEOMETRY OF BERT EMBEDDINGS

## 2.1 Introduction

With the availability of parallel computation powered by graphical processing units and massive amounts of data, natural language processing techniques have improved the performance of many challenging tasks significantly and surpassed human performance in a number of areas. As vast human knowledge exists in texts and language is the most effective human-to-human communication interface, machines that could understand natural language and communicate with humans naturally could transform many of the services in modern societies. A fundamental problem in natural language processing is to capture rich semantics of natural language computationally. Clearly, the meaning of a sentence depends on its constituents (such as words) and their relationships via syntax rules. By capturing regular patterns in large corpora, computational models can be used to help resolve a number of problems. One such model is the language model (LM) [6], which captures the probability distribution of the next word given its context. The model can be used to predict the next word(s) and resolve ambiguities (such as in speech recognition).

A distinctive feature of language models is the ability to be trained through self-supervision in that one can use the actual next word as the target. This feature enables training models using larger and larger datasets with better performance on a number of benchmarks [50, 16, 44]. Traditional language models are auto regressive; the context has consisted of either the words before the target word or after the target word, but not the entire sentence. However, representations based on the entire sentence or even sentences are useful for at least some NLP tasks[56]. For example, in order to predict the emotion in a sentence accurately, one needs to read the entire sentence as negation and implicit words (such as "not" and "but") can change its meaning completely.

To overcome the limitation, BERT [16] builds on an auto-encoding model, rather than an auto regressive one. More specifically, a BERT model is pre-trained by being able to predict some of the masked words via transformers [76]. Since it was introduced, BERT has set the new state of arts on a number of NLP benchmarks [16]. While there is ample empirical evidence that BERT models work well on a number of NLP tasks, it is not very well understood why they are effective. A common explanation is that BERT computes contextualized representations, where the representations of words depend on their contexts. However, this is not supported by the underlying auto-encoding model. For example, even though the word “bank” may have multiple meanings, the objective function of the pre-training task requires the same word embedding index to be predicted under different contexts. In this part, we attempt to understand BERT by **analyzing the geometry of its word embeddings, and the intrinsic discriminability of the context-dependent representations produced from the embeddings via transformers**. Such analyses provide new insights that are necessary to further improve BERT models.

## 2.2 BERT Architecture and Pre-training

BERT is essentially an encoder from the encoder-decoder architecture of Transformer [76]. By overcoming the limitations imposed by unidirectional processing, BERT and its variations are able to process a bidirectional context by masking the target words with a special [MASK] token, thus adding noise and making the prediction of the target word more difficult. Consequently, the training objective is to predict the original vocabulary id of the masked token based only on its noisy context. The introduction of the [MASK] token causes a mismatch between the pre-training and fine-tuning input. To alleviate this, the training data generator chooses 15% of the token positions at random for prediction; if the token $t$ is chosen for prediction, it is replaced with [MASK] 80% of the time, a random token 10% of the time and kept unchanged 10% of the time. Effectively, the model does not know which tokens it will be asked to predict and is forced to keep a distributional contextual representation for every token, although the cross-entropy loss is calculated only for the chosen 15% of the tokens. The second pre-training objective is the binary Next Sentence Prediction (NSP). An input sequence of BERT consists of two segments $A$ and $B$, 50% of time $B$ is the actual segment that follows $A$ in the corpus and 50% of time it is a random segment. BERT is asked to predict whether $A$ and $B$ are indeed a consecutive segments. The pre-training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood. For a detailed description of the structure and the objective function, we refer the reader to [76] and BERT [16].

By looking at BERT from its input and output relationship, BERT is an auto-encoding model as shown in Fig. 2.1. Given a sentence, consisting of tokenized word pieces [83], the word pieces are first translated into embedding vectors which can be modeled as matrix multiplications. The embedding vectors are first transformed by a number of multi-head attention and feed-forward neural network

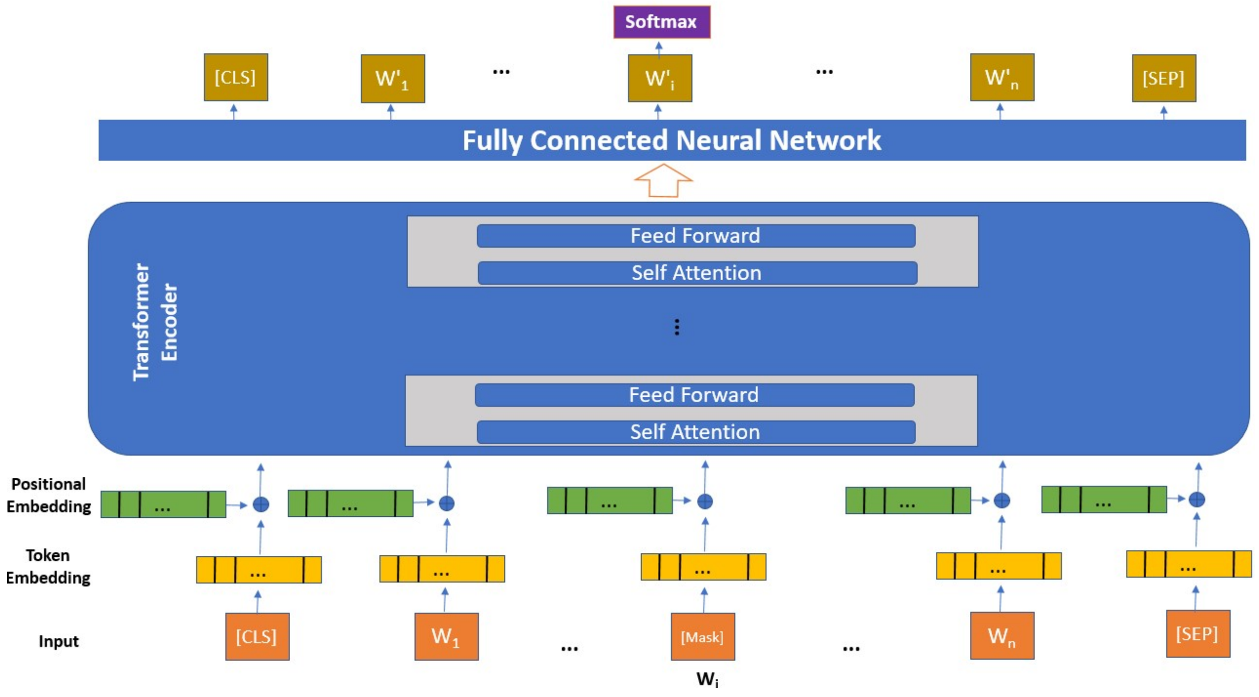

Figure 2.1: Bert pre-training architecture

layers, producing transformed vectors as the output from the BERT architecture. The matching of query and key vectors performed by the attention heads via inner product is inherently pair-wise; this resembles the product of the input and output vectors used in the skip-gram model [49], and may be a source of similarities between the geometry of the representations produced by the two models.

During pre-training, the output vectors are used as “contextualized" representation of the word pieces and a softmax function is applied to the result of multiplying each output vector and the input embedding matrix to produce a probability distribution over the word pieces in the BERT dictionary. Note that while the transformers in BERT encourage contextualized representations through query, key, and value matrices and feed-forward neural networks, the use of cross entropy is to “recover" the indices of the original word pieces, where the optimal solution would be the average vector in different positions and different contexts.

## 2.3 BERT Embedding Geometry and Effects of Transformers

The goal of this section is to study how the BERT embedding vectors are affected by its architecture and pre-training protocol by probing them. The parameters of a BERT model include the embedding vectors for word pieces along with the parameters for self-attention and feed-forward transformer layers. First, the embedding vectors were analyzed, then different layers of transformers were probed using simplified inputs. All of the experiments and analysis in this part were done using the uncased base BERT model, with 12 transformer layers, 12 attention heads, and the hidden dimension equal to 768[1]. When BERT is given an input sequence of tokens, the embedding layer, as well as all 12 transformer layers have a rank-2 tensor as their activations; with the shape of $(len\_sequence, hidden\_dim)$.

### 2.3.1 Embeddings Geometry Analysis

To study the distribution of the representations in the embedding space, pairwise correlations of weights of BERT embedding layer were computed. Since there are 30522 768-dimensional embedding vectors in the pre-trained BERT model, the correlations are $30522 \times 30522$ values in $[-1, 1]$ range. The histogram of correlations in Fig. 2.2 shows that the embeddings tend to have somewhat strong pairwise correlations, implying that the representations occupy a narrow region in the embedding space [8]. This is one of the geometric effects of pre-training, as vectors sampled from a normal distribution would have correlation histogram centered around 0. For comparison, if we compute pairwise correlations for embeddings computed for BERT vocabulary by a different model, ELMo [56], their histogram plot is centered closer to 0, as we may see in Figure 2.3.

Next, the magnitudes of the representations were analyzed. The magnitude of an embedding is important since it directly affects the result of the dot product of the BERT output and the embedding matrix; this product is subsequently used as an input to the softmax function to produce a probability distribution over the model's vocabulary. In order to analyze the magnitudes of the representations, the text8 dataset was used [47], which is a concatenation of English Wikipedia articles text truncated at $10^8$ characters. The dataset was tokenized and the word-part tokens were filtered out. The remaining tokens were sorted by their frequency and the euclidean norm was computed for each of the remaining tokens' embedding. The plot of embedding euclidean norm with respect to token frequency is displayed in Figure 2.4. The norms of 10 most frequent tokens

[1]Pre-trained models based on Transformers [82] are publicly available at https://huggingface.co/transformers/

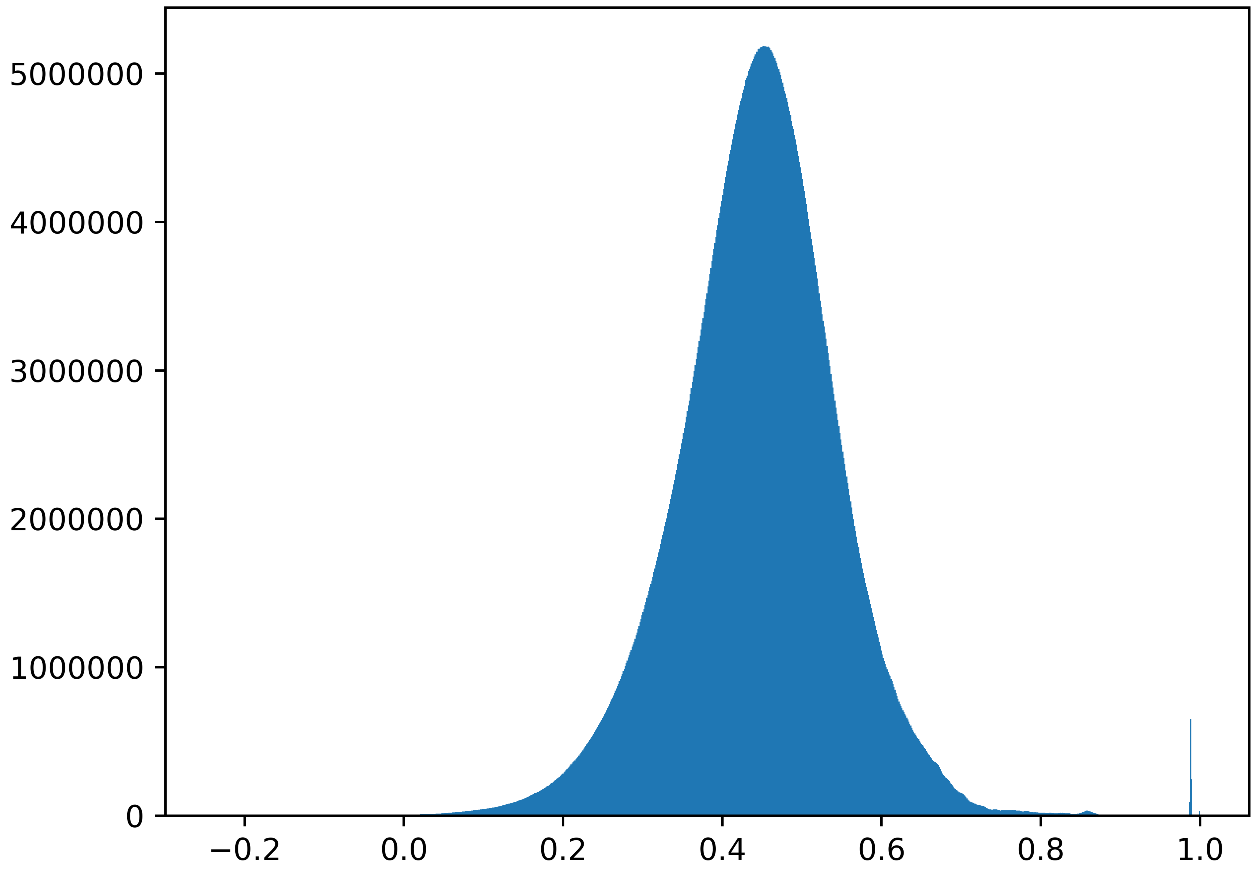

Figure 2.2: Histogram of BERT embedding pairwise correlations. Note that the peak near 1.0 is due to some vectors are similar to each other.

are in the Table 2.1. The more frequent words tend to have smaller norms, with the most frequent word in the dataset ("the") having the smallest norm. Many of the most frequent tokens are *function* words that appear in a large variety of contexts; such a context diversity may be responsible for pushing the tokens embeddings towards the coordinates origin during pre-training.

### 2.3.2 BERT Transformer Analysis

We probed the BERT transformers layers using simplified inputs. In the following experiment, a single token ("cat") was used as an input for BERT in order to compute all hidden states; then, the pairwise correlations of the hidden states were computed. The correlation matrix heatmap is displayed in Fig. 2.5; we observed, that in the matrix there is an emergent block structure with bottom layers and top layers having high pairwise correlations within their group. The correlation with the embedding layer with each of the following layers decreases up to layer 7 and starts increasing after that. Being an auto-encoder model, BERT encodes patterns in the data in the

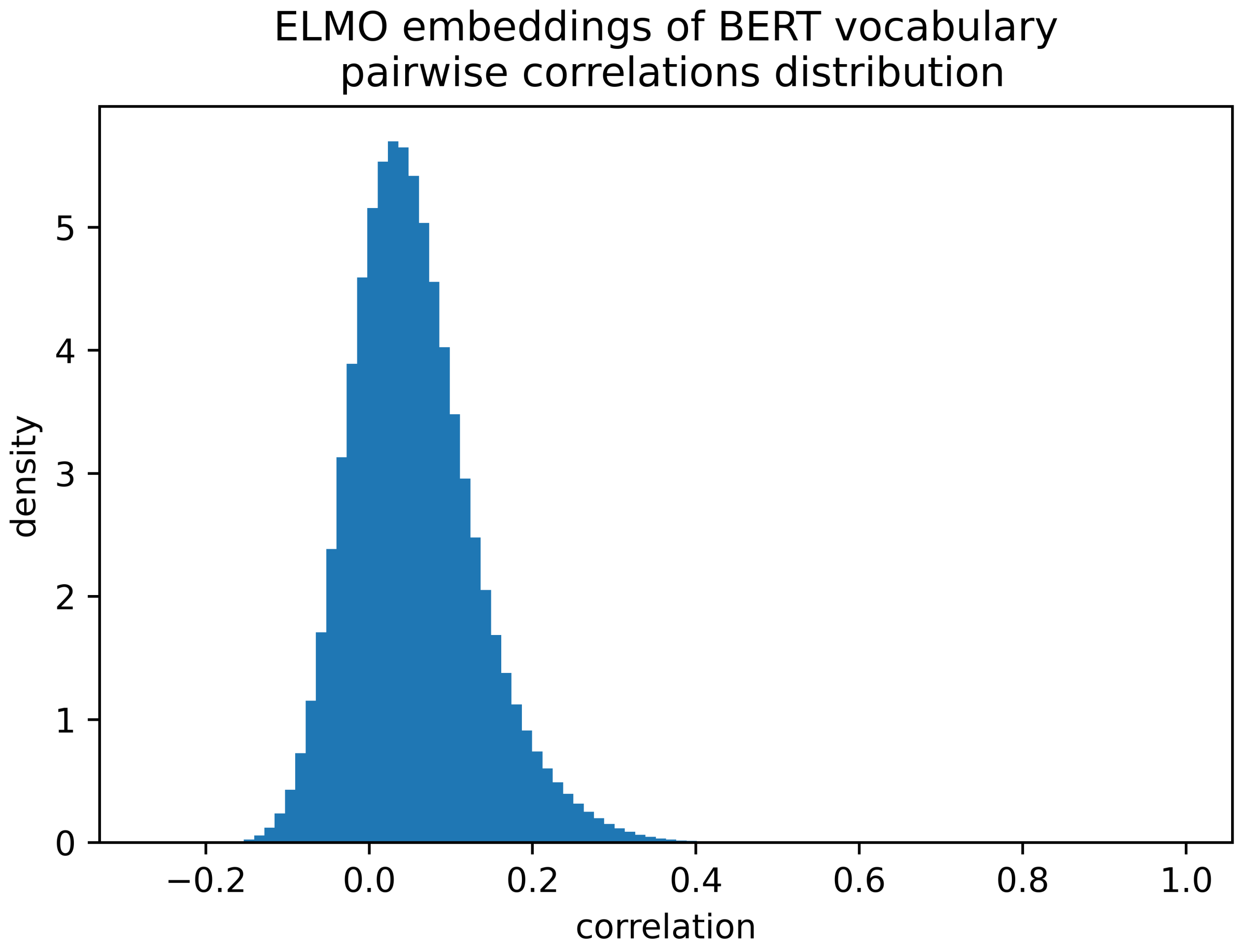

Figure 2.3: Histogram of pairwise correlations for ELMo embeddings of BERT base uncased vocabulary.

first few layers and the upper layers try to “reconstruct" the original masked words, by recovering information from the patterns.

In the next experiment, the analogy dataset[50] was used, where each sample is a quadruple (A, B, C, D) organized so that A is related to B in the same way as C is related to D, e.g. (Paris France Helsinki Finland). The dataset was filtered, so that each word corresponds to a single token. BERT activations were computed when each of these tokens was the only input to the model, and the activations were used for the *analogy* query formulated in terms of vector arithmetic as follows: with respect to (B - A + C), how many tokens embeddings are closer in terms of cosine similarity than the correct answer D? The results are shown in Fig. 2.6; $y$-axis shows the percentage of samples for which the correct answer D belongs to $k$ neighbor embeddings to the *analogy* expression, while $x$-axis stands for the number of neighbors $k$. The results show that the activations in all transformer

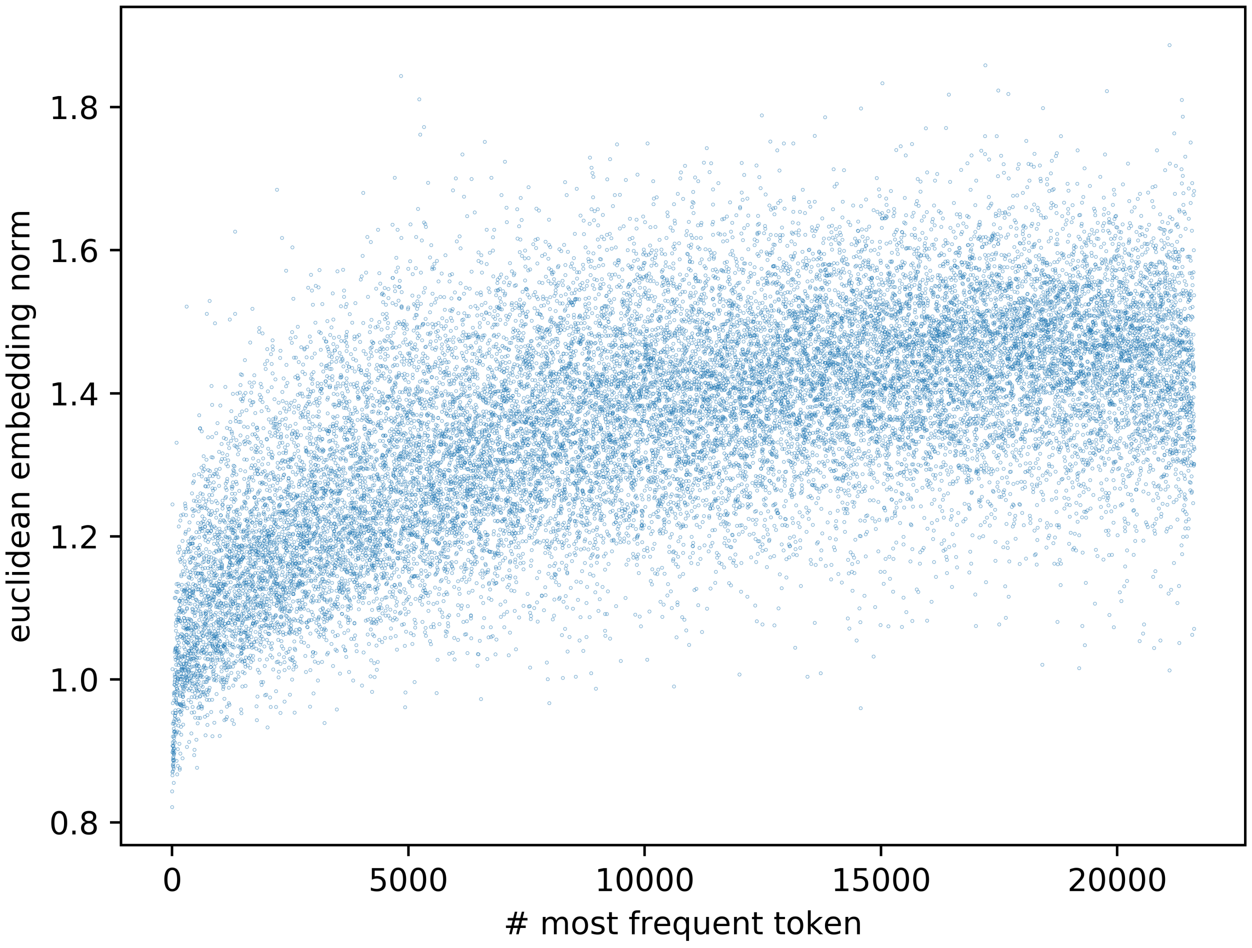

Figure 2.4: Euclidean embedding norms for tokens sorted by their occurrence frequency in Text8 dataset[47]. The more frequent tokens tend to have a smaller Euclidean norm.

layers are able to capture the notion of similarity formulated in terms of *analogy* query. However, this ability is gradually lost along with increasing depth. The rate of accuracy decrease is steepest between the middle layers. [21] show that the activations from higher layer occupy a smaller region in the embedding space, this compressed distribution may negatively impact the performance of the higher layers on this task. Also, surprisingly, the first layer demonstrates a better accuracy than the embedding one. Skip-gram (SG) [49] was used as a baseline. Pre-trained vectors used for this experiment are publicly available at `https://code.google.com/archive/p/word2vec/`. The same set of filtered queries was used; for a fair comparison the vocabulary of SG was limited to the 30,000 most frequent English words and the words needed for the task, resulting in a dictionary of 30,378 words. While SG performs better when only one nearest neighbor is used, it is surpassed by BERT activations when more neighbors are considered. One of the reasons for this may be the

Table 2.1: 10 most frequent tokens in Text8 dataset[47]. The most frequent token ('the') has the smallest Euclidean norm.

| # most freq | token | count | magnitude |
|---|---|---|---|
| 1 | the | 960941 | **0.821649** |
| 2 | of | 537171 | 0.843784 |
| 3 | and | 377328 | 0.898151 |
| 4 | one | 363157 | 1.014323 |
| 5 | in | 340397 | 0.866191 |
| 6 | a | 294527 | 0.870349 |
| 7 | to | 287473 | 0.907536 |
| 8 | zero | 234036 | **1.244975** |
| 9 | nine | 219649 | 0.967104 |
| 10 | two | 170537 | 0.938169 |

higher dimensionality of BERT vectors (768) than the SG vectors (300), resulting in a exponentially larger embedding space and increased difficulty when only one neighbor is used. The experiment was repeated by evaluating (A - B + D) against C. The reformulated task results show the same patterns as the original ones.

## 2.4 Discriminability of BERT Features

As pointed in [16], BERT models can be used both with feature-based approaches and fine-tuning approaches. In this section, the inherent discriminability of representations produced by BERT models is evaluated by analyzing whether words in semantic classes will cluster together first. This forms the basis for such features to generalize well for tasks. Our systematic analysis shows the features are not inherently discriminant.

Then, we investigated whether the sequence using computed embeddings still contains the information necessary for classification if we model the dependencies among the embeddings.

### 2.4.1 Discriminability Analysis

We compute all transformers layer activations of pre-trained BERT on a labeled dataset and compute two distinct metrics that summarize how well the activations are clustered within the same label and separated across different labels.

We borrow the idea for the first metric from [43]. For activation $x \in \mathbf{R}^n$ with label $y$, let $L$ designate the relation between $x$ and $y$ ($L(x) = y$), let $\rho(x)$ be a ratio of the distance to the nearest activation in another class $y' \neq y$ to the distance to the nearest activation in the same class $y$

incremented by a small positive number $\epsilon$ for numerical stability:

$$\rho(x) = \frac{\min\limits_{L(x')\neq y} d(x,x')}{\min\limits_{L(x)=y, x'\neq x} d(x,x') + \epsilon}. \tag{2.1}$$

Intuitively, if an activation $x$ is within a cluster of activations with the same label, its nearest neighbor has the same label; the numerator is greater than the denominator and $\rho(x) > 1$. In a converse case, the nearest neighbor has a different label and $\rho(x) < 1$. Next, we compute the summary of $\rho$ values for the entire dataset; since $\rho$ can range from 0 to $+\infty$, we squash it into a finite range using the following transformation:

$$f(x) = \frac{1}{1 + exp(1-x)} \tag{2.2}$$

After that, we compute the arithmetic mean over the entire dataset:

$$F = \frac{1}{|\boldsymbol{X}|} \sum_{x\in\boldsymbol{X}} f(\rho(x)) \tag{2.3}$$

as well as the empirical distribution of $f(\rho(x))$ and its statistics. Hence we are able to reason about discriminability of the activations; for in-cluster activations, $1 > f(\rho(x)) > 0.5$, whereas for outliers $0 < f(\rho(x)) < 0.5$.

The second metric is based on *silhouette* analysis [65]. Following the convention used during the first metric definition, let $a(x)$ be the mean of in-class distances:

$$a(x) = \frac{1}{|\{x' : L(x') = y, x' \neq x\}|} \sum_{x':L(x')=y, x'\neq x} d(x,x') \tag{2.4}$$

Let $b(x)$ be the smallest mean distance to a set of activations having the same label distinct from $y$:

$$b(x) = \min_{y':y'\neq y} \frac{1}{|\{x' : L(x') = y'\}|} \sum_{x':L(x')=y'} d(x,x') \tag{2.5}$$

The silhouette $s(x)$ is defined as follows:

$$s(x) = \frac{b(x) - a(x)}{\max(a(x), b(x))} \tag{2.6}$$

If $x$ is a sole sample in its class, $s(x)$ is defined to be 0.

Values of $s(x)$ close to 1 imply that $a(x) \ll b(x)$, that is, the mean distance between $x$ and samples within its class is much less than the minimal mean distance between $x$ and samples in other classes, contributing to a high discriminability of chosen feature. Similarly, values of $s(x)$

close to $-1$ mean that the sample $x$ is closer to some set of samples with a label distinct from its own. Values of $s(x)$ close to 0 imply that $x$ is as close to set of samples with its own label as to the samples with another label. The latter two cases contribute to a low discriminability of chosen feature.

Similarly to the first metric, we computed the empirical distribution of the *silhouette* over the dataset and its statistics to analyze the discriminability.

### 2.4.2 Two Approaches for Sentiment Analysis Using BERT

We have a labeled dataset of tweets, where each tweet is a short text and the label is the sentiment expressed in the tweet. For classification, instead of training a model from scratch, we want to leverage knowledge from a pre-trained model. We consider two ways of doing that: The first one is to use BERT activations as input features for a simpler model that is trained *from scratch*, while the second one is to augment the pre-trained BERT model with a classifier subnetwork and *fine-tune* the composite network. Each considered BERT activation is a rank-2 tensor of shape $(num\_tokens, bert\_hd)$, where $num\_tokens$ is the number of tokens in the input to BERT and $bert\_hd$ is the *hidden dimension* BERT hyperparameter. We will refer to each activation as a length-$num\_tokens$ sequence of $bert\_hd$-dimensional vectors in this section. For each model, a cross-entropy loss is computed based on final linear layer outputs (logits) and true data labels; it is optimized using a gradient-descent based optimizer.

**Feature-based classification.** For the feature-based approach, two different network architectures were used. The first architecture is a shallow network that consists of 2 linear layers with CELU nonlinearity between the layers. The input vector is the first element of a BERT hidden state; for such state we use the output of each transformer layer as well as the embedding layer output.

The second architecture is a two-layer BiLSTM with a classifier head on top; the classifier head is a single linear layer that takes as input a concatenation of last forward and backward outputs of the BiLSTM. The BiLSTM input is the *entire* sequence of BERT hidden state elements (as opposed to using only *the first* element of such sequence). We use the same hidden states of BERT as input as for the first architecture.

**Fine-tuning.** In this case, the model consists of a pre-trained BERT model and a classifier head. The classifier head is a linear layer; its input is *the first* element of a BERT hidden state; the hidden states are again the embedding layer output and each of the 12 transformer layer outputs.

## 2.5 Experiments and Results

In this part, we used two datasets: text8 and an emotion dataset. The text8 was used to study the word frequency and the emotion dataset was used for classification.

### 2.5.1 Emotion Classification

In each classification scenario, we perform stratified 10-fold validation and report mean accuracy, balanced accuracy and adjusted balanced accuracy [9] for classifiers based on each BERT hidden state in its according figure.

For feature-based classification using shallow network, we used the entire emotions dataset with 340540 tweets labeled into 6 classes. We use Adam optimizer [38] with learning rate of $2^{-8}$, constant learning rate schedule and no weight decay. We set the batch size to 64 during training and to 1 during validation. The dimension of middle hidden layer activation is 384; the classifier input is 768-dimensional and the network output is 6-dimensional. The classifier is trained for 5 epochs. We report the accuracy metrics in Figure 2.7. The worst accuracy is at the embedding hidden state for the same reason as we have observed in our fine-tuning experiment. In contrast to fine-tuned and BiLSTM-based models, we observe worse accuracy; it seems that this architecture has less capabilities in capturing temporal patterns present in text.

For feature-based classification using BiLSTM as well as for fine-tuning based classification, we used a stratified sample of 8964 tweets of the original emotion dataset as a trade-off between training time and accuracy. We also pad/truncate the input sequences to 32 tokens for enabling batch processing. Having the same dataset size allows to perform a fairer comparison of these models' accuracy scores. For feature-based classification using BiLSTM, we use Adam optimizer [38] with learning rate of $2^{-11}$, constant learning rate schedule and no weight decay. We set the batch size to 64 during training and to 1 during validation; the LSTM hidden dimension is 384, input dimension is 768 and there are 2 bidirectional layers. The dropout probability is 0.2 for dropout layers inside LSTM as well as between LSTM and the linear classifier head. The classifier is trained for 12 epochs. The accuracy metrics are displayed in Figure 2.8.

For fine-tuning based classification, we use AdamW optimizer[46] with learning rate of $2^{-17}$, constant learning rate schedule and no weight decay. The batch size is 16 during training and 1 during validation; the dropout layer between BERT and linear layer is set to have probability 0.1. The classifier is trained for 6 epochs. Figure 2.9 contains the accuracy metrics. The lowest accuracy is at the embedding output. Since the first element is always the '[CLS]' token embedding. the model fails to generalize as it never sees other tokens from the input sentence.

We aggregate the results in Table 2.2. According to the classification results, the performance of shallow network is the worst among the models, while BiLSTM and fine-tuned networks have comparable best performances among BERT hidden states. The original BERT paper [16] also reported that the accuracy of BiLSTM feature-based approach is comparable to fine-tuning one; however, they did not provide accuracies for *all* layers, while we do.

The fine-tuned and shallow networks have almost 0 adjusted balanced accuracy on the embedding layer. In these cases, a model does not take into account all tokens in the input token sequence, as we have set up the model input as the first element of the BERT hidden state; hence, the performance is predictably random. BiLSTM-based network does not have this property as its recurrent subnetwork processes all input tokens before feeding into a linear classifier, and this is reflected in its accuracy. The fine-tuned and shallow networks using the first element of a deeper BERT state as input do not have this property as well, as there is a connection to all tokens in the input through attention [76] layers of BERT.

There is an interesting pattern that we cannot explain yet; the graph of accuracy with respect to BERT hidden state is not monotonous for each classifier architecture. Both feature-based classifiers have the accuracy peak near 4th BERT hidden state. The fine-tuning based classifiers have the accuracy peak in the 4th to last BERT hidden state.

### 2.5.2 Embeddings Discriminability

We study the embeddings of a few selected emotion words and their 10 nearest neighbors (in terms of Euclidean norm) from text8 dataset. A 2-dimensional projection of BERT activations are demonstrated via PCA [54]. The results in Figures 2.10 and 2.11 show that the Bert embeddings do not display well-formed semantic clustering. For instance, in Figure 2.10 "sadness" and "happiness" are very close to each other, while both of them are far from "pleasure" and "joy". In Figure 2.11, "sadness" is somewhere between "pleasure" and "happiness". As we can see, intrinsic discriminability of BERT's pre-trained embeddings is limited for direct usage.

Additionally, we used the emotion dataset to study the raw BERT activations discriminability. We computed two discriminability metrics for each sample in the dataset based on sequence-wise average activation of each BERT transformer layer as well as the embedding layer; the distributions of each discriminability metric are in Figures 2.12 and 2.13. The results show that both metrics values tend to be in the neighborhood of the decision boundary (0.5 and 0, respective), hence, the raw activations in any BERT layer are not clustered based on the dataset labels. In Figures 2.14 and 2.15, we show the same metrics computed for intermediate representations of ELMo [56]. Since ELMo is based on LSTMs, the intermediate representations are not based on transformer block outputs; instead, we use the average of time steps in the word embedding layer (E); in the first forward BiLSTM layer, final memory cell (F1M), final state cell (F1S) and the average of time step outputs (F1TS); same quantities for the second forward BiLSTM layer (F2M, F2S, F2TS accordingly); for the first backward BiLSTM layer (B1M, B1S, B1TS) and for the second backward BiLSTM layer (B2M, B2S, B2TS). We similarly observe both metrics to be in close vicinity of the decision boundary.

## 2.6 Related Work and Discussion

Neural distributed representations of words were initially introduced by [6], who proposed to learn these representation using Neural Network based LM (NNLM). Later, Mikolov et al. [51] introduced RNN based LM (RNNLM) to handle variable length sequences and a better context modeling. Distributed representations of words have gained immense popularity after the introduction of an efficient way of estimating them, in the form of skip-gram and CBOW models by [50, 49] as well as the Glove [55]. However, these approaches use limited contexts. To model dependencies in larger contexts, LSTM networks [30] were used to model the language [73]. The potential of LSTM's memory cell was further utilized by [56] who proposed deep contextualized word representations, called ELMo. In contrast to having a fixed representation, in ELMo words are assigned a representation that is a function of the entire input sentence, where the function is modeled using bi-directional LSTM.

Vaswani et al. [76] introduced Transformer, the first sequence transduction model based entirely on attention (described in detail in the architecture section). Transformer eliminates recursion, allowing for modeling the long-distance dependencies with a constant number of operations. First of the Transformer based LM, GPT [61], is an auto-regressive model based on Transformer decoder (without encoder-decoder attention). At the time of its introduction GPT improved the state-of-the-art result on multi-task NLP benchmark GLUE [81]. Finally, Devlin et al. [16] introduced BERT, further improving the state-of-the-art on GLUE.

Due to BERT's superior performance, a number of studies were performed in order to understand its underlying mechanisms [78, 22]. For example, there are a number of studies on analyzing

Table 2.2: Emotion classification accuracies.

| | Fine-tuning | | BiLSTM | | Shallow NN | |
|---|---|---|---|---|---|---|
| L. | Acc. | Bal. acc. | Acc. | Bal. acc. | Acc. | Bal. acc. |
| E | 0.3382 | 0.1667 | 0.8881 | 0.8511 | 0.3099 | 0.1667 |
| 1 | 0.8551 | 0.8020 | 0.8844 | 0.8412 | 0.5787 | 0.4812 |
| 2 | 0.8859 | 0.8371 | 0.8853 | 0.8434 | 0.6459 | 0.5662 |
| 3 | 0.8976 | 0.8531 | **0.8892** | 0.8448 | 0.6495 | 0.5687 |
| 4 | 0.9003 | 0.8608 | 0.8852 | **0.8464** | **0.6668** | **0.5937** |
| 5 | 0.8978 | 0.8606 | 0.8728 | 0.8292 | 0.6568 | 0.5734 |
| 6 | 0.8986 | 0.8538 | 0.8666 | 0.8245 | 0.6562 | 0.5793 |
| 7 | 0.8993 | 0.8535 | 0.8531 | 0.7969 | 0.6379 | 0.5491 |
| 8 | 0.9006 | 0.8513 | 0.8280 | 0.7740 | 0.6208 | 0.5170 |
| 9 | **0.9028** | **0.8626** | 0.8191 | 0.7515 | 0.6157 | 0.5070 |
| 10 | 0.9003 | 0.8513 | 0.7886 | 0.7080 | 0.6238 | 0.5107 |
| 11 | 0.9022 | 0.8508 | 0.7739 | 0.6858 | 0.6267 | 0.5127 |
| 12 | 0.9016 | 0.8488 | 0.7595 | 0.6819 | 0.6404 | 0.5247 |

attention heads in BERT and their roles of capturing syntax and semantic patterns [23, 52, 20, 31, 14]. In contrast, the focus of this work is on the characteristics of the learned representations, not the patterns captured by the attention mechanisms. The closest to this work are the papers of Voita et al. [77] and Ethayarajh [21]. Voita et al. [77] analyzed the evolution of the word representations across the layers of a masked language model. By using correlation analysis, it reports patterns similar to the one in Figure 2, to show that masked language models are auto-encoders which create contextual representations in the early layers and try to recover the tokens identities in the later layers in order to predict the correct ids of the target tokens. The evolution of the embeddings across the layers is not the focal point of this study; rather, we performed a thorough analysis of key geometric properties learned by masked language models and their roots in the Transformer's architecture, using BERT as a case study. Ethayarajh [21] studied geometry of representations produced by BERT, GPT-2 and ELMo, focusing on evaluating how much context is captured by the respective models using self-similarity - the average cosine similarity between the contextualized representations of the same word in different context across the layers. The author reported that the self-similarities in BERT are consistently higher than that in GPT-2. This can be attributed to the different objectives of the models; while representations in GPT-2 are used to predict the next token in the sequence, those in BERT must predict their own id. Similarly to [21], we found that the words in BERT occupy a narrow cone in the embedding space, however we used a different methodology. In this work we focused on the discriminability of the representations produced by

BERT and their influence on sentence classification tasks; we conducted a broader study of the geometry of the representations grounded in the underlying model's architecture.

While BERT is more powerful than the skip-gram [50] model, there are inherent similarities between the two. Consequently, studies on the skip-gram model have shown similar observations to that of BERT embedding geometry. For example, Schakel and Wilson [67] show a similar inverse relationship between word frequency and the embedding vector length of words using the skip-gram model.

There are also a number of studies which show that the BERT models are capable of discovering features in commonly used NLP pipeline [3, 75]. However, as the BERT models are inherently an auto-encoder model, the underlying mechanisms of the observed features must be rooted in the auto-encoder model and data. An effective multiple-layer auto-encoder model would naturally lead to a hierarchical representation, where more common features are discovered first and then other less common features will be built on the features. Generally, since more complex relationships such as semantic relationships are less common, they are developed in higher layers. This is fundamentally different from the patterns in ELMo [56], where the local dependencies need to form first before longer dependencies.

## 2.7 Conclusion and Future Work

In this part, we have analyzed how the BERT architecture and its pre-training protocol affect its embedding vectors and its induced feature vectors. Due to its similarity to the architecture of the skip-gram and CBOW models, the geometric properties of BERT embeddings are similar in many ways to the vectors given by Word2Vec [49]. Our systematic results show the induced features do not lead to clusters for emotion classification in the original embedding space. However, they are effective for classification when temporal dependencies are modeled explicitly (such as BiLSTM).

More importantly, the results show that the BERT models do not produce “effective" contextualized representations for words and their improved performance may mainly be due to fine-tuning or classifiers that model the dependencies explicitly by encoding syntactic patterns in the training data. This is being investigated further.

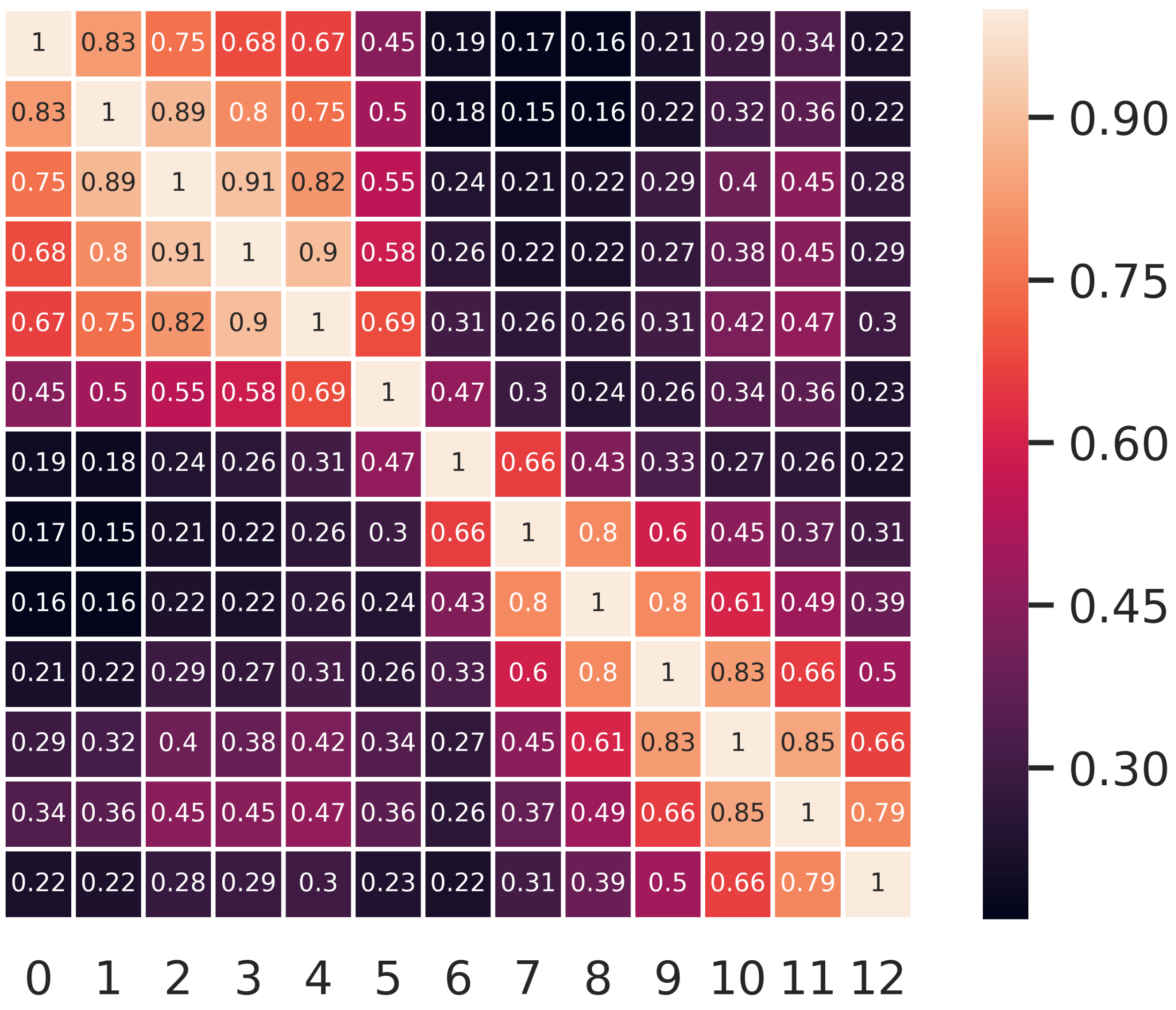

Figure 2.5: The correlation coefficients between outputs of the embedding layer (#0) and all the transformer layers (#1-#12) of pre-trained BERT[16] for the input token sequence consisting of a single token 'cat'.

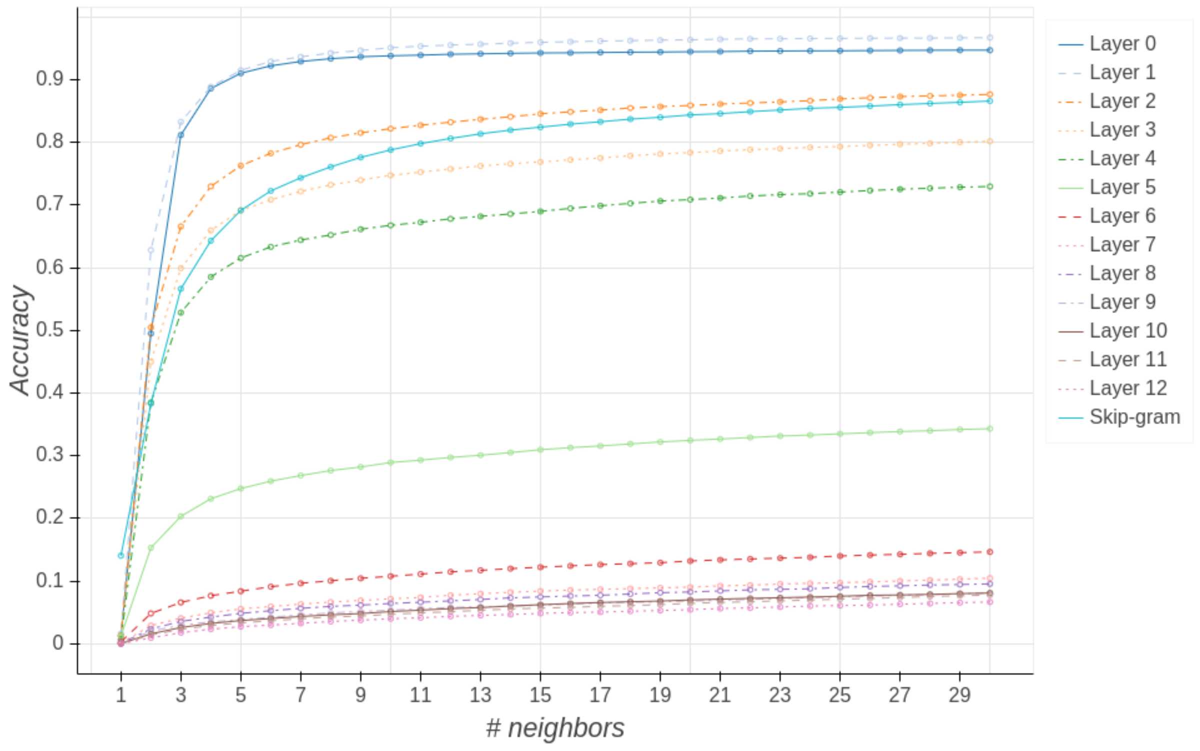

Figure 2.6: Analogy query results: the number of dataset entries for which the answer to analogy query (e.g. *king* - *man* + *woman*) belongs to correct analogy's (*queen*) K nearest neighbors. Different lines correspond to different BERT transformer layers that are used to get embeddings from. Skip-gram is used as a baseline.

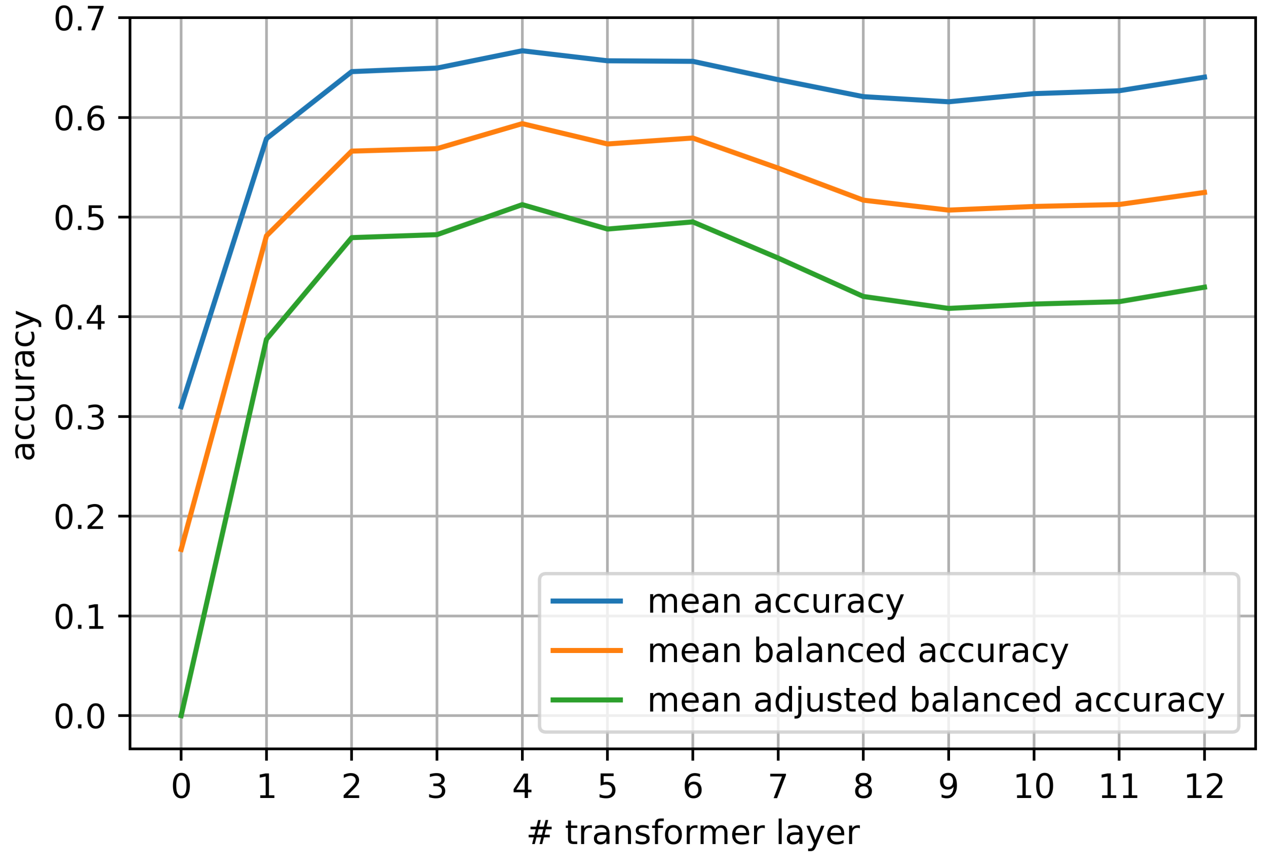

Figure 2.7: Accuracy metrics of feature-based classifier based on a shallow network for each of the BERT hidden states as the classifier input.

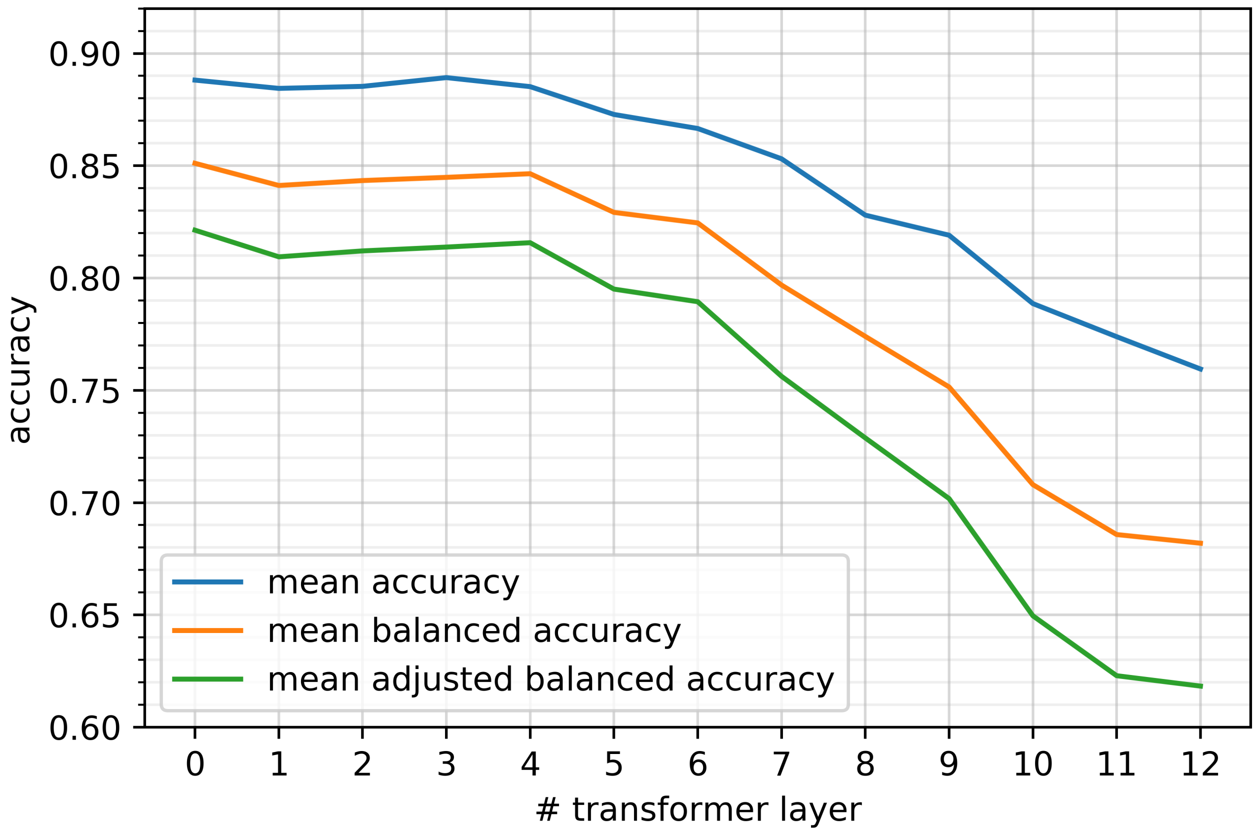

Figure 2.8: Accuracy metrics of feature-based classifier with BiLSTM processing the BERT hidden states. We observe that it is possible to achieve a comparable performance to a fine-tuned model using output of a pre-trained model and a simpler architecture. In contrast to fine-tuned and shallow network based models, here we achieve the best performance in the embedding layer and a few bottom transformer layers.

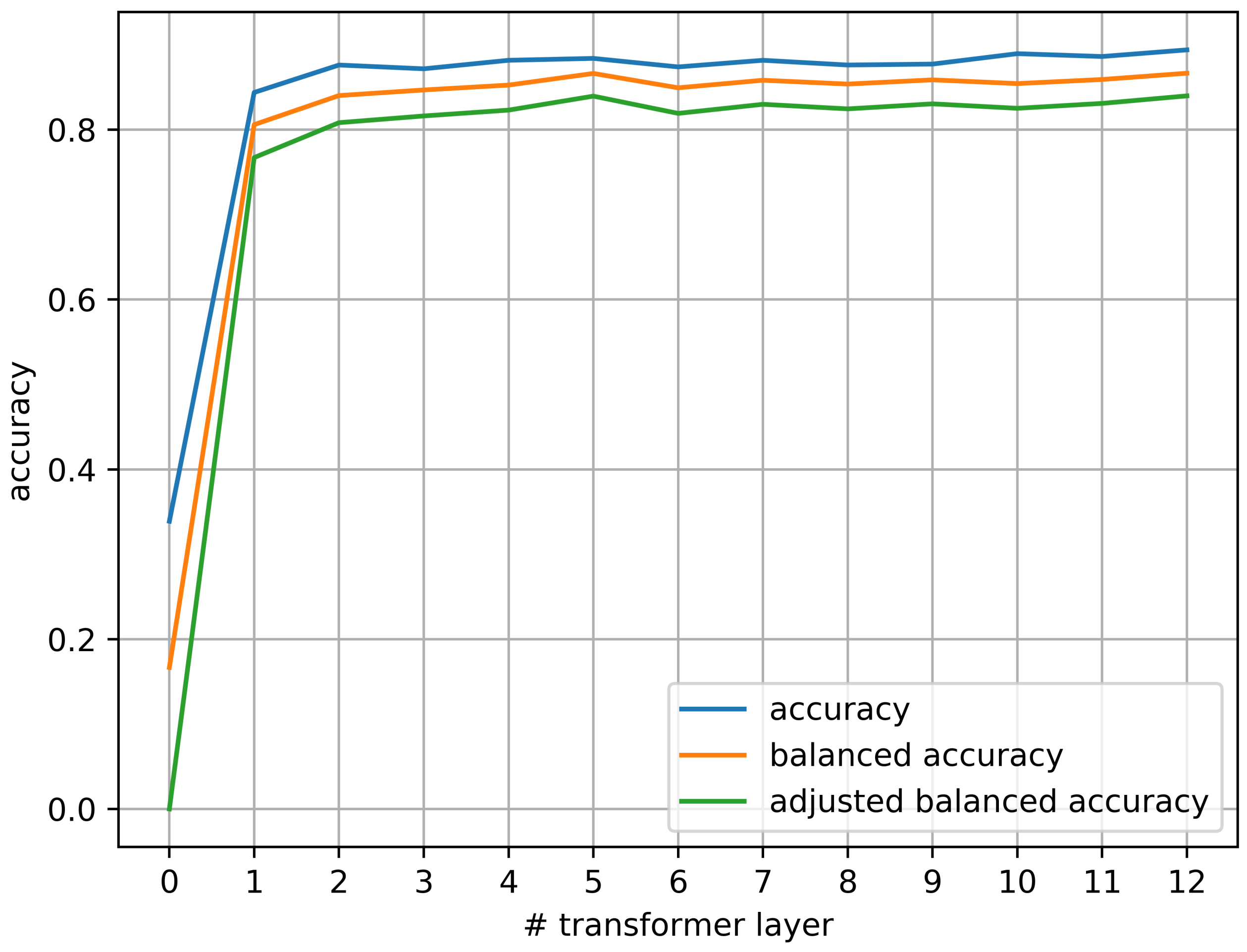

Figure 2.9: Accuracy metrics of fine-tuned model for each of the BERT hidden states as the classifier head attachment point.

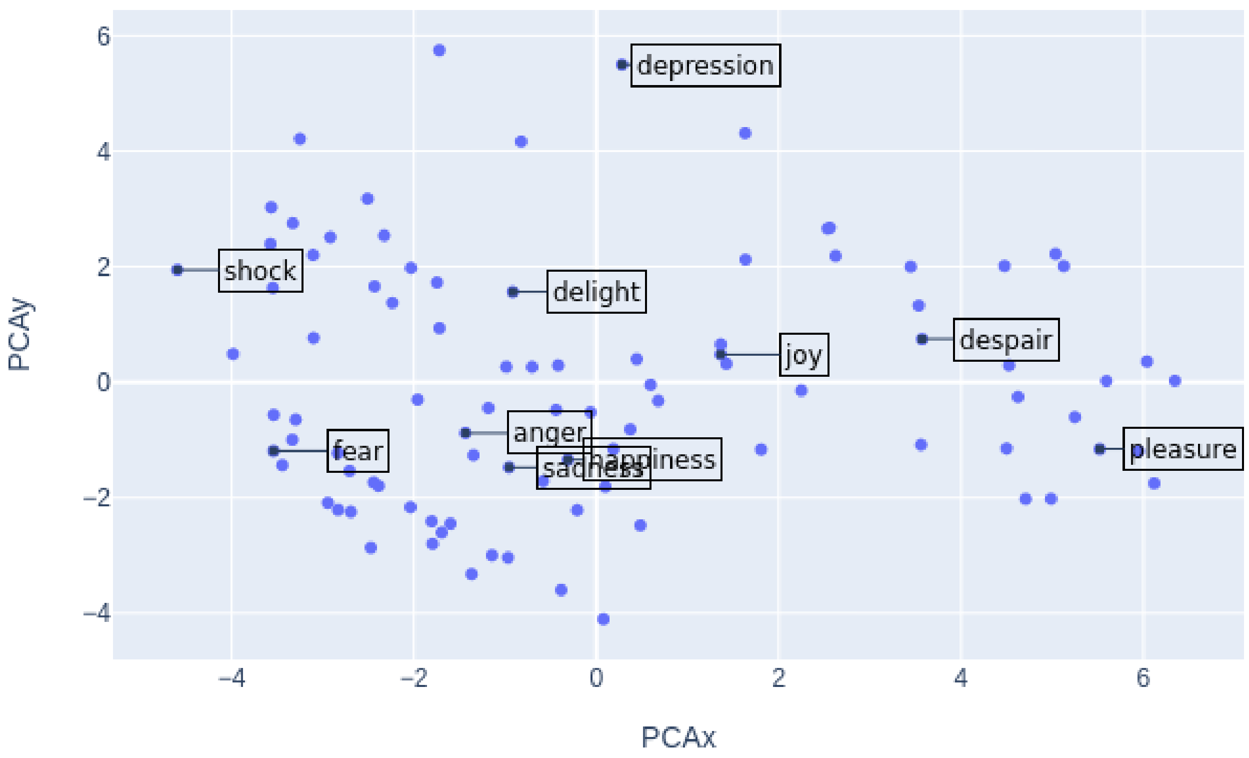

Figure 2.10: PCA of emotion words, averaged through all layers

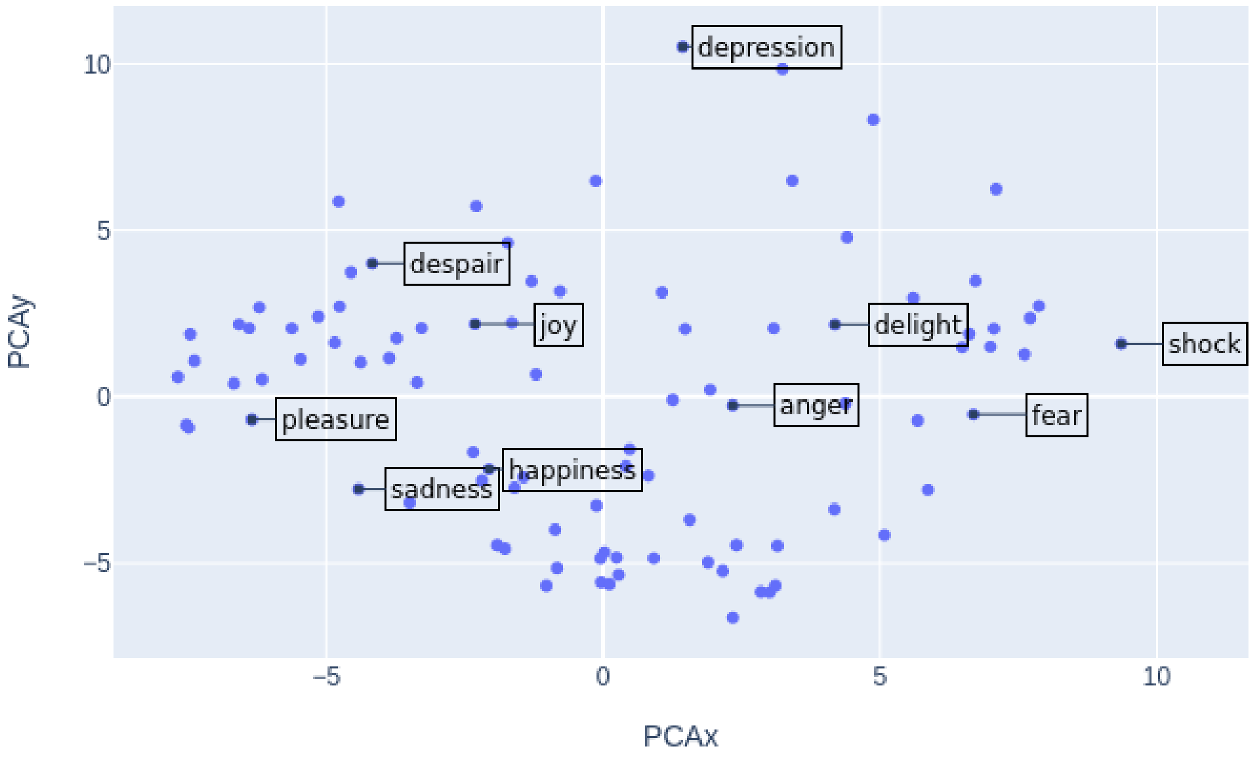

Figure 2.11: PCA of emotion words in the 12th layer

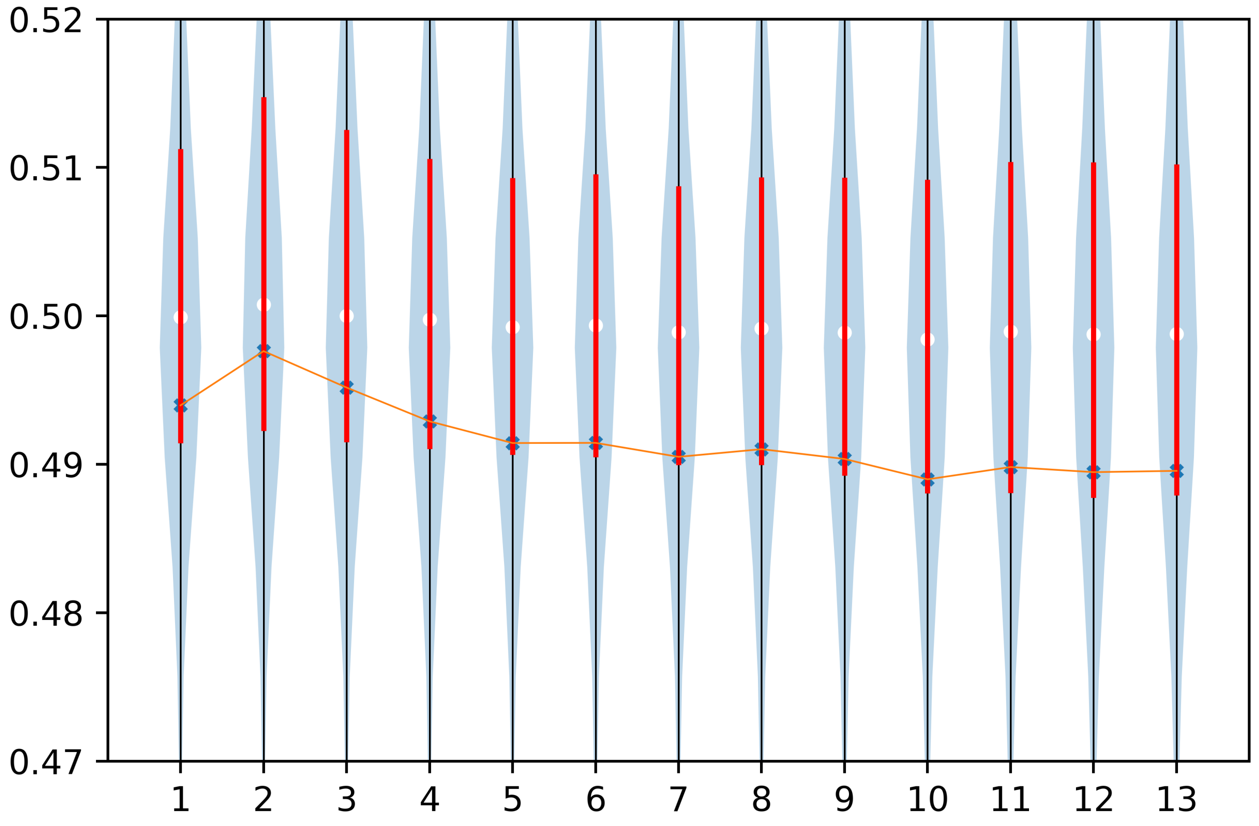

Figure 2.12: Discriminability distributions for each transformer layer.

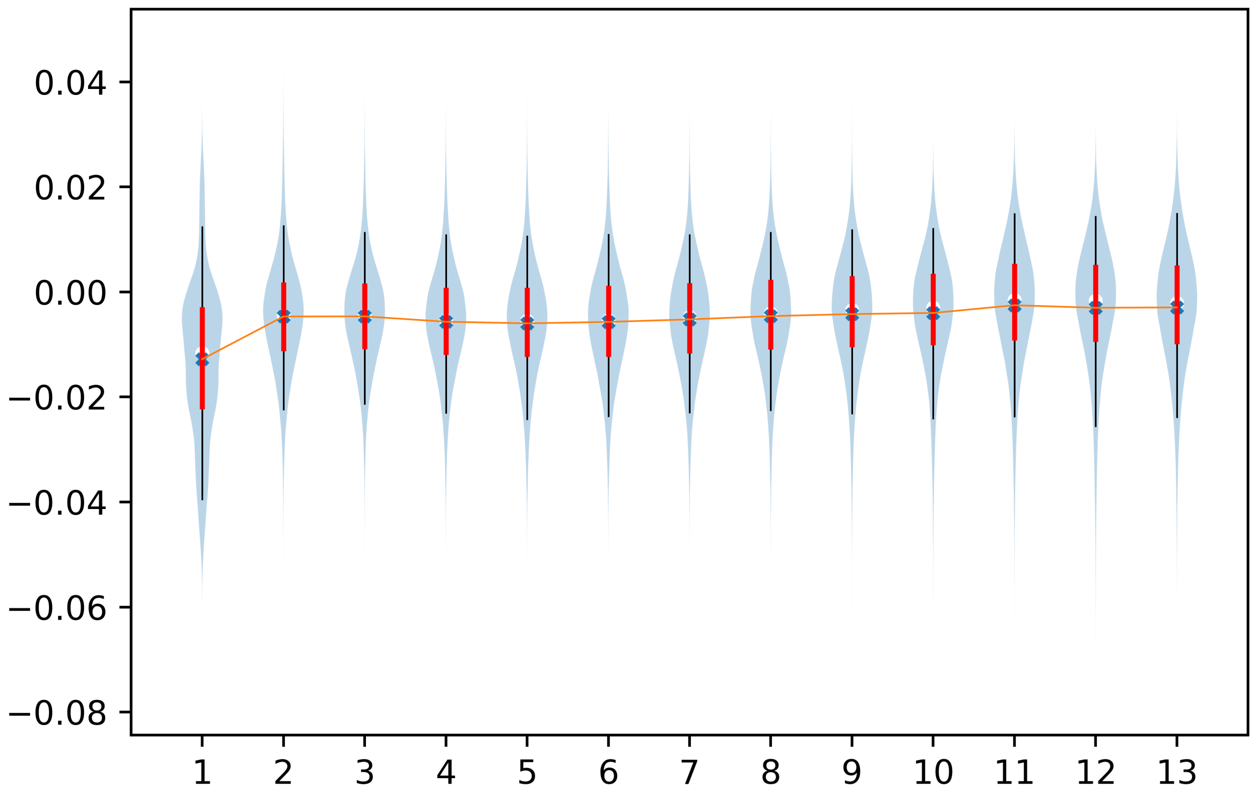

Figure 2.13: Silhouette distributions for each transformer layer.

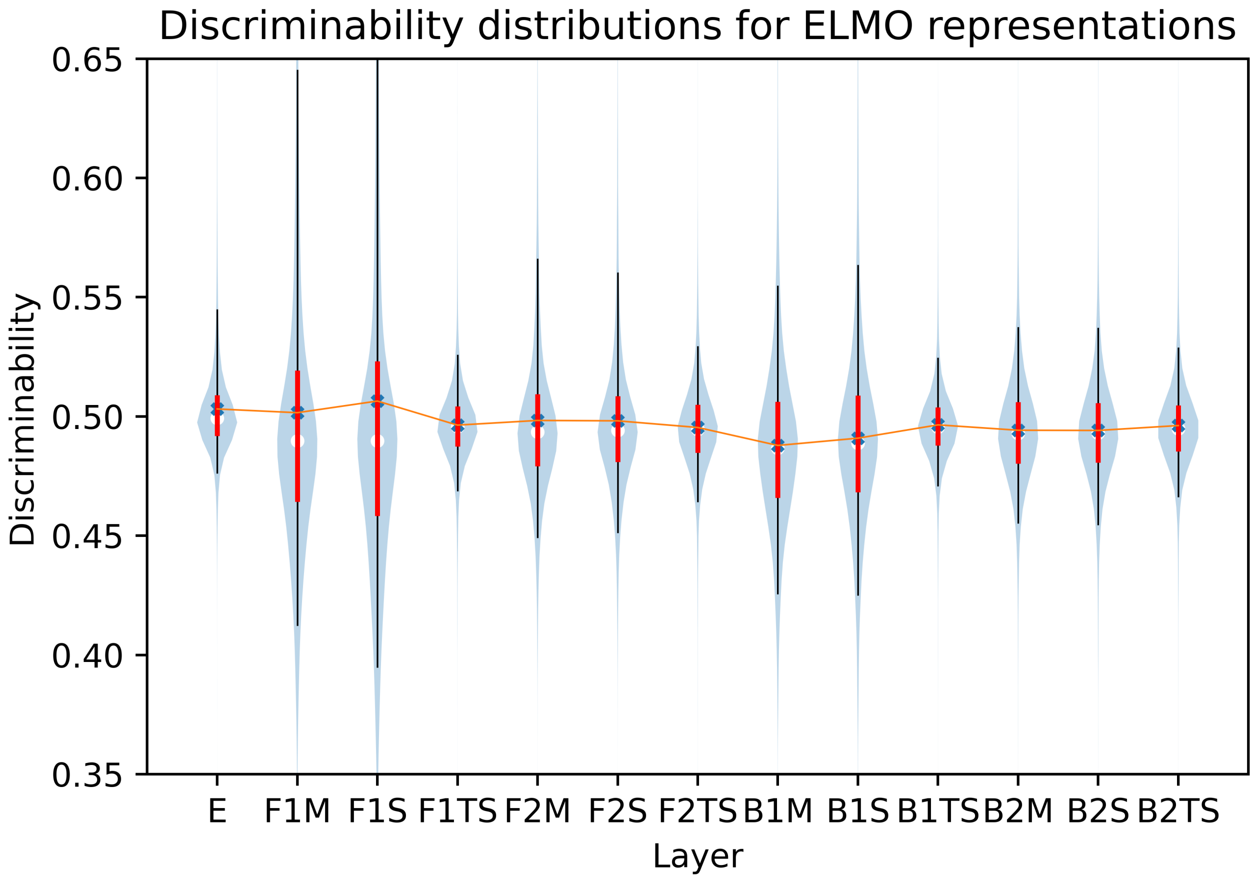

Figure 2.14: Discriminability distributions of intermediate ELMO representations

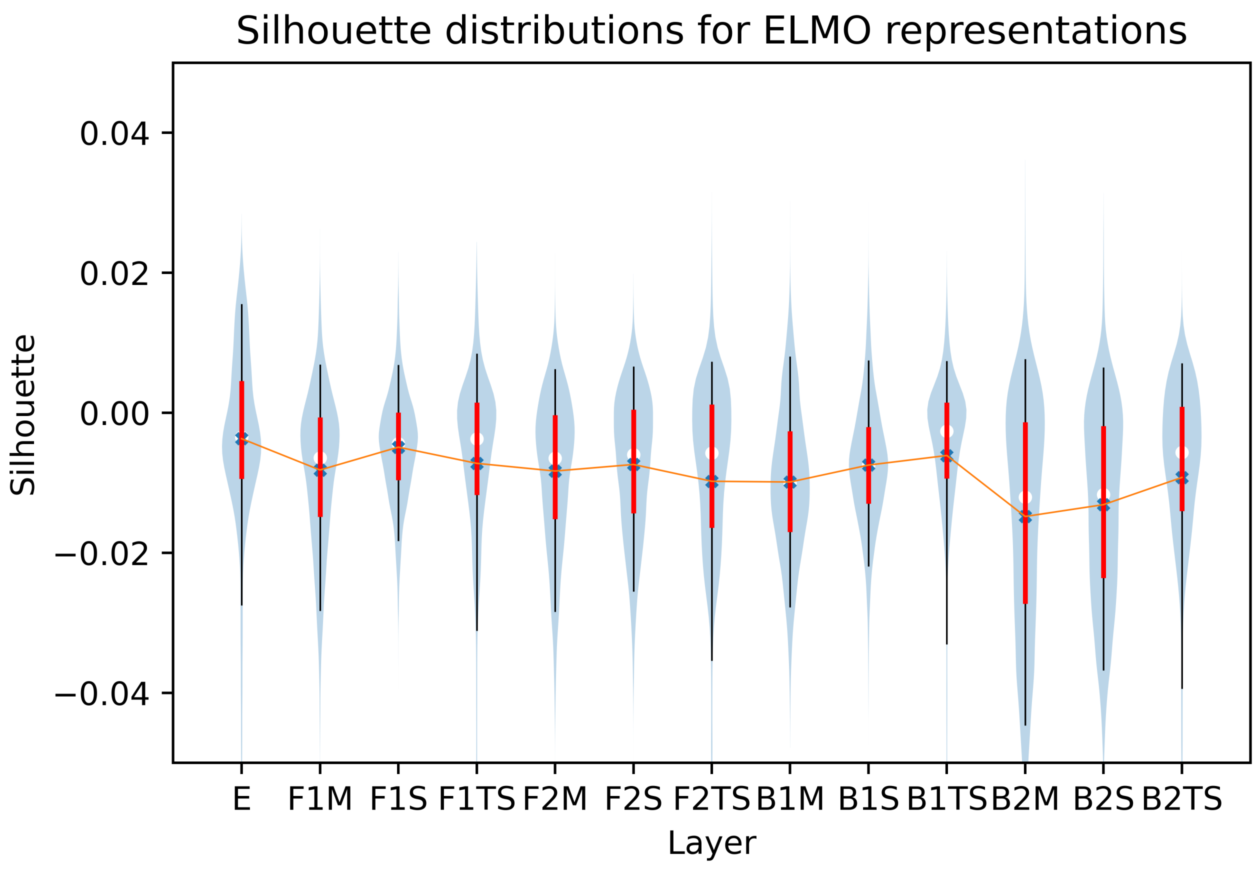

Figure 2.15: Silhouette distributions of intermediate ELMO representations

# CHAPTER 3

# KNOWLEDGE AND CONFUSION IN BERT

## 3.1 Introduction

In the recent years, there has been a surge in natural language processing results that can be attributed to Transformer [76] based models [56, 61, 16, 44]. Studies have shown that the models are able to capture superficial language patterns [14, 23] and queries in natural language can be constructed to retrieve facts about the world from the pre-trained models [58, 33, 57]. While capturing general knowledge in a language model is certainly appealing, it is necessary to understand the source of knowledge and possible confusion to maximize the benefits for downstream applications. Furthermore, while the performance on knowledge benchmarks has progressively improved, there is a lack of understanding of how the Transformer-based models are able to predict facts, what roles different components play, and how robust and transferable the knowledge captured by the model during pre-training is. Addressing these questions is necessary if one likes to use the knowledge in the pre-trained models for critical applications such as the medical domain.

In this part we attempt to fill these gaps in understanding BERT [16], arguably the most popular and the most successful variation of the Transformer-based models, as a case study. While some of the findings may be specific to BERT, the methodology and most findings are applicable to other Transformer-based models. First, we show that BERT's ability to answer queries requiring factual knowledge relies mainly on superficial co-occurrences, rather than generalization and its ability to reason or infer, and can be easily influenced by inconsequential syntax changes (such as removing a dot or [SEP]). For instance, given the input sequence "[CLS] Vaccines cause [MASK]. [SEP]", the top five predictions are: 'cancer', 'disease', 'death', 'diseases', 'mortality'; however, if we change the sentence to "[CLS] Vaccines cause [MASK] [SEP]", the top five predictions become ';', '.', '?', '!', and '|'. Using BERT as a reliable source of knowledge could have catastrophic effects on downstream applications.

Second, we show that the *Layer Normalization* (LayerNorm) [4] module in the language model (LM) head, a classifier layer discarded after pre-training, is responsible for BERT's ability to predict facts. Without the LM head, retrieved facts are dominated by [CLS], a special token used in BERT, due to its large magnitude; after swapping the magnitude of ‘the’ with that of [CLS], ‘the’ dominates in most cases. It is evident that when LM Head is discarded and only the encoder stack is reused, which is the case for most downstream tasks, the knowledge in BERT cannot be transferred. Note that this does not imply the encoder stack is not useful. As the encoder stack transforms the input into contextualized representations that improve the performance of downstream applications, one can replace the LM head by training a customized one to achieve good performance, such as [85]. As the customised ones are only fine-tuned on specific and typically much smaller dataset(s), they would not have the benefits of much larger dataset(s) used during pre-training.

Third, we demonstrate that one word can dramatically alter the predicted answers. We observe that only a few tokens other than the special [MASK] token can be used as a placeholder for the prediction, despite the masking strategy used during pre-training suggesting that the model should be able to correct any token based on its context. We find that most of the words are reconstructed exactly, regardless of the context. This implies the mismatches between pre-training (where [MASK] is used) and applications (no [MASK] token is used typically)[84] could alter the knowledge in BERT substantially. Furthermore, it also implies that BERT knowledge may not generalize well to new input sequences. Overall, the knowledge in BERT lacks the desired robustness and is fragile.

Finally, we have manually constructed a novel misconceptions dataset for evaluation of the robustness and reliability of Language Models trained in a self-supervised manner as knowledge sources; we have shown that BERT does not handle the task well, and it could lead to biases and other negative effects on downstream applications.

The rest of this part is organized as follows. Section 2 contains the related work. In Section 3, we provide background about BERT structure and training objectives. Section 4 is devoted to exploration of LM head’s role in BERT’s token reconstruction capability. Section 5 comprises demonstration of knowledge fragility by using alternative masking tokens. In Section 6, we study the factually wrong knowledge stored in BERT. We discuss our results in Section 7 and conclude with the summary of our results and possible future work directions in Section 8.

## 3.2 Related Work

Dissecting the BERT internals has sparked wide research interests, to the point of forking a deluge of results into a separate subfield coined *BERTology*. [64] provide a comprehensive survey of that research direction. [58] design a probe dataset coupled with a framework for common sense and factual knowledge extraction from pre-trained Transformer models [15, 16, 56, 61, 44]. [74] devise a set of probing tasks involving symbolic reasoning, such that they may potentially be difficult to be captured by the masked language modeling objective, as the latter is designed with the emphasis on capturing tokens' co-occurrences. [85] fine-tune BERT to perform relation prediction and link prediction tasks on knowledge graphs via (entity, relation, entity) triple classification. [33] explore strategies for constructing prompts in natural language to extract knowledge from language models, as arbitrary requests prove to be too difficult to answer correctly. [57] study the question context influence on the precision of the result, showing potential for improvement when the additional context relevant to the question is provided, while arguing that knowledge extraction is robust to adding a noisy or irrelevant context. However, they omit discussing why they observe such effect. We believe that the reason is related to co-occurrences of the original questions and additional context tokens. Note that none of the studies try to understand the underlying mechanisms for the knowledge and confusion in BERT, which is the focus of this part.

## 3.3 Overview of BERT

BERT is a stack of Transformer encoder blocks [76], each composed of two sub-layers: multi-head attention and a position-wise fully connected feed-forward network. In the multi-head attention sub-layer, every input token is linearly projected $h$ times with different, learned parameter matrices for *Query (Q)*, *Key (K)* and *Value (V)*, into $d_h$ dimensional vectors, where $h$ is the number of attention heads and $d_h$ is the attention head dimensionality. The softmax of a scaled inner product between $Q$ and $K$ is multiplied with *(V)* creating weighted, context dependent representations. The representations produced by each head are concatenated and transformed by a linear projection. In the second sub-layer, position-wise, fully connected feed-forward network with a GELU activation [28] is applied. Both sub-layers are wrapped with a skip-connection followed by a LayerNorm [4].

Training objective:. Training BERT consists of two stages: pre-training and fine-tuning. Pre-training consists of two semi-supervised tasks. The first is masked language modeling (MLM) - prediction of randomly masked tokens in a sequence, where 15% of its tokens are masked and the model is asked to predict the indices of the original tokens; out of the 15% tokens, 80% of the time a token is replaced with a special [MASK] token, a random token 10% of the time, and left unchanged token 10% of the time [16]. Masking the target words with a special [MASK] token enables processing a bidirectional context without leaking information between the layers.

Additionally, masking introduces noise to sequences and makes the prediction of the target words more difficult.

The second pre-training objective is the binary Next Sentence Prediction (NSP). An input sequence of BERT consists of two segments $A$ and $B$, 50% of time $B$ is the actual segment that follows $A$ in the corpus and 50% of time it is a random segment. BERT is asked to predict whether $A$ and $B$ are consecutive segments. Note that this part of the objective uses only [CLS], thus affects the embedding vectors and the Transformer stack solely via the gradients through [CLS], which contributes to the unique features of [CLS] (such as its large magnitude).

During fine-tuning, a classification layer is added on top of the encoder stack and a task-specific objective function is used to train the classifier and the Transformer stack. The input pipeline for BERT consists of tokenizing words into wordpieces [83], and adding token, position, and segment embeddings to construct a fixed-length representation. Special [CLS] token is added at the beginning of the input and used for classification predictions in downstream tasks; special [SEP] token is used to separate input segments. [84] pointed out the issue of input inconsistency between pre-training and fine-tuning, as the [MASK] token is usually not used in downstream tasks. We observed a different discrepancy between the pre-training and fine-tuning versions of the model. Specifically, during pre-training a language modeling head, a classifier layer, is used on top of the stack of transformer blocks; later, the language modeling head is discarded and the rest of the model is used for fine-tuning on downstream tasks. Therefore, assuming that BERT indeed captures knowledge, it needs to be reflected in the encoder stack to be directly transferable to the downstream tasks.

## 3.4 Language Modeling Head and Knowledge Retrieval in BERT

A hallmark feature of BERT is that using a masked language model makes computing bidirectional contexts possible. During pretraining, BERT is trained to learn how to replace [MASK] (80% of the time) and other random tokens (10% of the time) with the correct tokens. Since BERT pretraining includes a language model head (LM head) that is discarded when fine tuning downstream applications, it is important to understand if the encoder stack itself learns to perform token replacement. Our experiments and analyses show that it is not the case. To further understand BERT's ability for knowledge retrieval, we dissect the LM head to understand the underlying mechanisms. Our results show the LayerNorm plays an essential role in knowledge retrieval.

### 3.4.1 Knowledge Retrieval Using BERT

According to the masked language modeling framework, the common natural language answer retrieval approach is set up as follows. First, a sentence is extracted from the natural text resources on the Web, such as encyclopedias, manuals, newspapers, books, and so on. Second, the sentence is split into tokens from a model's dictionary. Finally, one of the tokens is replaced with the special [MASK] token to designate the subject of the query. The language modeling head is essentially a classifier that has been trained with the objective of minimizing the masked language modeling loss [16]. Intuitively, the loss rewards correct predictions of tokens that have been replaced with a [MASK] token in the training dataset during pre-training. Therefore, by looking at a few largest logits in the output position corresponding to the [MASK] token, we are able to retrieve the top candidates for the token that the pre-trained model considers to most likely appear, instead of the [MASK] token, in the original input sentence. Also, for each candidate, there is a prediction confidence score, produced by feeding the logits into softmax. While the classifier head accounts for a relatively tiny number of parameters, compared to the base BERT model size (only about 0.57% of the parameters of the entire model[1]), using the output from the encoder stack to predict the tokens directly (by bypassing the processing steps in the LM head) returns dramatically different results. For example, in most cases, [CLS] is the top choice; because inner products are used, magnitudes of the embedding vectors play an important role. Indeed, the embedding vector for the [CLS] token has the largest magnitude. To verify that the magnitude, not the direction of the vector is the reason, we swap the magnitude of [CLS] and 'the' without changing their directions. As expected, 'the' becomes the top choice for most cases. Additional details about this experiment are available in Appendix B. It is evident that the LM head is very important for knowledge retrieval.

### 3.4.2 LM Head Architecture

To understand the underlying mechanism for knowledge retrieval, we zoom into the language modeling head to track the roles of its components responsible for its ability to make sensible predictions. We have used the BERT provided by HuggingFace [82] as our implementation. The language modeling head consists of a sequence of modules: (1) fully connected layer taking $H$-dimensional input (*hidden dimension* parameter in BERT's configuration, equal to 784 for base uncased version) and producing $H$-dimensional output; (2) Gaussian error linear unit (GELU) nonlinearity [28]; (3) LayerNorm [4] with elementwise affine application; and (4) fully connected

[1]BERT for MLM totals $109,514,298$ parameters, out of which $622,650$ are in the LMH.

layer taking $H$-dimensional input and producing $V$-dimensional output (*vocabulary size* parameter in BERT's configuration, equal to 30522 for base uncased version) by computing inner products with the BERT word embeddings and adding bias that was learned during pre-training.

In the LayerNorm layer, for input vector $x$,

$$LrNorm(x) = \frac{x - mean(x)}{std(x)}. \tag{3.1}$$

In a variant with elementwise affine application, shared weight vector $w$ and bias vector $b$ (having the same dimension as the input vector) among all input samples are applied as

$$LrNormAff(x) = LrNorm(x) \otimes w \oplus b, \tag{3.2}$$

where $\otimes$ is elementwise multiplication and $\oplus$ is elementwise addition. To study the inner structure of the language modeling head, we separate the fully connected layers and the LayerNorm module into a sequence of single operations. More specifically, we separate the fully connected layers into weight multiplication and bias addition and the LayerNorm module into LayerNorm normalization only (without elementwise affine application), elementwise weight multiplication ($\otimes$), and elementwise bias addition ($\oplus$). While such splitting reduces the inference speed, it allows us to analyze the intermediate results with a finer precision.

### 3.4.3 LM Head Activation Analysis

We have shown that LM Head is necessary for meaningful knowledge retrieval. Here, we analyze each of LM Head components in order to identify the root of this phenomenon, which should enable conscious usage of BERT in tasks requiring knowledge. In order to do that, we convert a sample sentence to a sequence of tokens and pass it as input to BERT with LM head. We then analyze the activations in the [MASK] token position via each LM head component. To estimate the original token reconstruction ability of these activations, we make a conceptual shortcut to the final head layer, computing the inner products of that $H$-dimensional activation with each of the $V$ token embeddings, and measure the similarity of the $V$-dimensional result with the $V$-dimensional vector produced by BERT with entire LM head in the [MASK] token position. Since the indexes of the largest components of the $V$-dimensional vector are used to retrieve the most likely candidates for the reconstruction from the vocabulary, the absolute values of the components do not matter as much as their sorted order. Hence, we use Spearman's rank-order correlation [72] as the similarity metric. High similarity would indicate the LM head component responsible for high prediction quality. As a concrete example, consider the following probe suggested in [58]: *'Francesco Bartolomeo Conti was born in [MASK].'*. The *[MASK]* token here designates an unknown entity we are trying to retrieve from the pre-trained BERT. As BERT is pretrained to replace [MASK] by the correct token, [MASK] is the natural choice.

We find that while the output of language modeling head is only weakly uncorrelated with the inner products between the BERT last transformer layer and word embeddings, the strong correlation appears after the LayerNorm step. For the example, the top predictions obtained using the LayerNorm activations are 'turin', 'rome', 'bologna', 'pisa', and 'milan', and the ones obtained from GELU layer (immediately before the LayerNorm step) are are '[CLS]', '[MASK]', '[SEP]', 'there', and 'here'. It is evident that the LM head is driven into the correct answer's context after the LayerNorm layer. In particular, we analyzed the LayerNorm weights; while the details are given in Appendix (Figures A.1 and A.2), their effect on most of the input vectors is to multiply them by a factor of around 2.3, except for [CLS] and other tokens similar to [CLS] embedding vectors, whose magnitude is either reduced or multiplied by a factor of around 1.5. Therefore, the ranking orders are changed significantly.

In order to study LM head components' impact on unmasking words systematically, we use the ConceptNet dataset provided with the LAMA probe [58]. It consists of 29774 samples; we used the following fields: (1) a sentence with '[MASK]' replacing the word that appeared in the original text; (2) the ground truth token that we use to estimate the reconstruction accuracy. We study embeddings produced by each LM head component by looking into logits in the output sequence position of [MASK] token; to assess the similarity between each of these embeddings and LM head output, we compute Spearman's rank-order correlation. The average components' histograms of these correlations over the entire ConceptNet dataset are in Figure 3.2. It shows that the similarity significantly increases on average after the LayerNorm component.

## 3.5 Replacement Token Analysis

Here we further investigate the generalization and robustness of BERT models for knowledge retrieval. BERT is pretrained to learn how to replace [MASK] 80% of the cases and 10% of them to replace inconsistent tokens. As pointed out by [84], downstream applications do not use [MASK] typically and it is important to understand how the models generalize to other input sequences. We consider two cases that the models should handle well in order to be useful for downstream

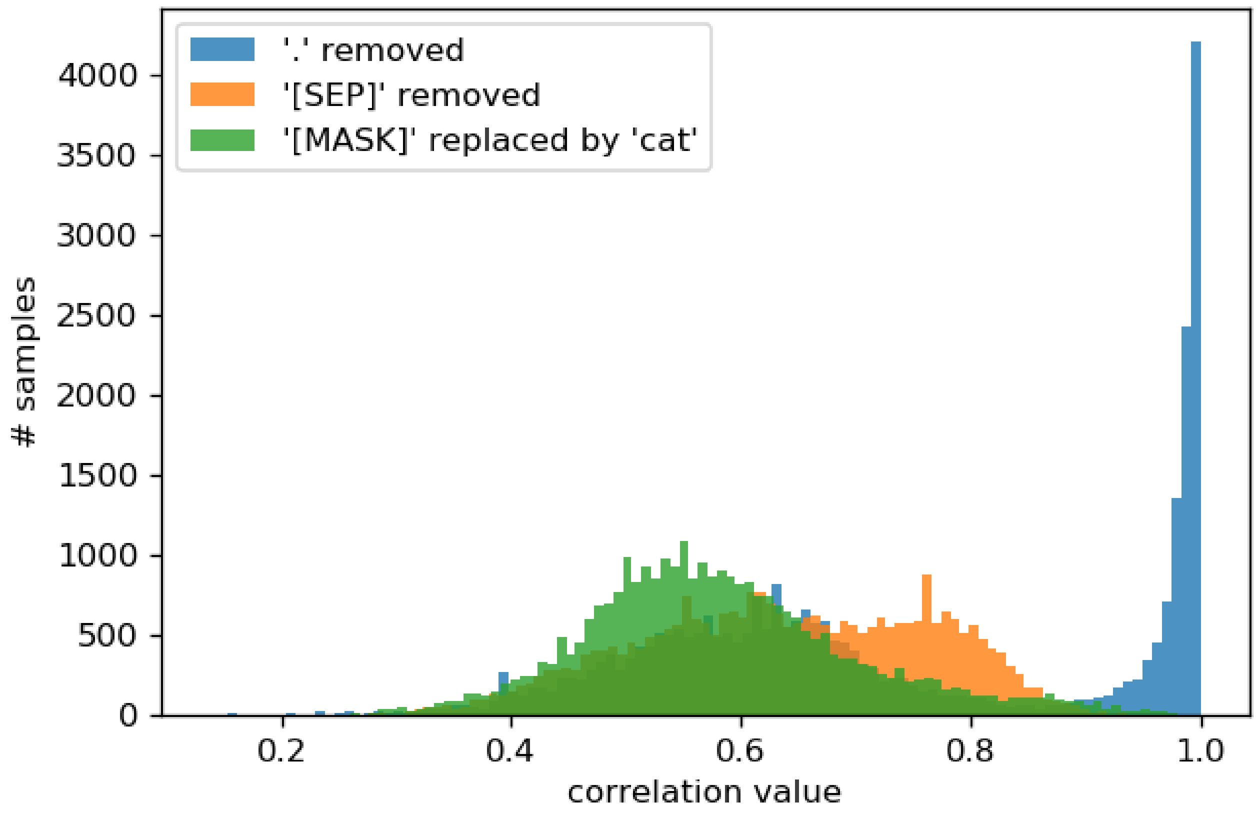

Figure 3.1: Histogram of Spearman's rank-order correlations between logits of the original sample and logits of a perturbed sample, collected over entire ConceptNet dataset used in LAMA. The perturbations are (1) removing the '.' token ending the sample sentence or sentences; (2) removing the [SEP] token ending the token sequence serving as input to BERT; (3) replacing the [MASK] token with the 'cat' one.

Table 3.1: Retrieval counts for BERT base uncased on ConceptNet dataset.

| PROBE TYPE | H@1 | H@2 | H@5 | H@10 | H@20 | H@50 | H@100 | Total |
|---|---|---|---|---|---|---|---|---|
| original | 3710 | 5285 | 7743 | 9460 | 11219 | 13825 | 15843 | 29774 |
| - `[MASK]` at the end | 1216 | 1822 | 2934 | 3839 | 4753 | 6387 | 7697 | 18796 |
| - other | 2494 | 3463 | 4809 | 5621 | 6466 | 7438 | 8146 | 10978 |
| ending `'.'` removed | 2350 | 3327 | 4588 | 6120 | 8798 | 12500 | 14820 | 29714 |
| - `[MASK]` at the end | 6 | 22 | 111 | 766 | 2606 | 5214 | 6806 | 18780 |
| - other | 2344 | 3305 | 4477 | 5354 | 6192 | 7286 | 8014 | 10934 |
| `[SEP]` removed | 1009 | 1649 | 2612 | 3564 | 4632 | 6399 | 8621 | 29774 |
| - `[MASK]` at the end | 52 | 175 | 387 | 608 | 922 | 1609 | 2773 | 18796 |
| - other | 957 | 1474 | 2225 | 2972 | 3710 | 4790 | 5848 | 10978 |
| `[MASK]` / `cat` | 702 | 1373 | 2583 | 3566 | 4722 | 6620 | 8470 | 29774 |

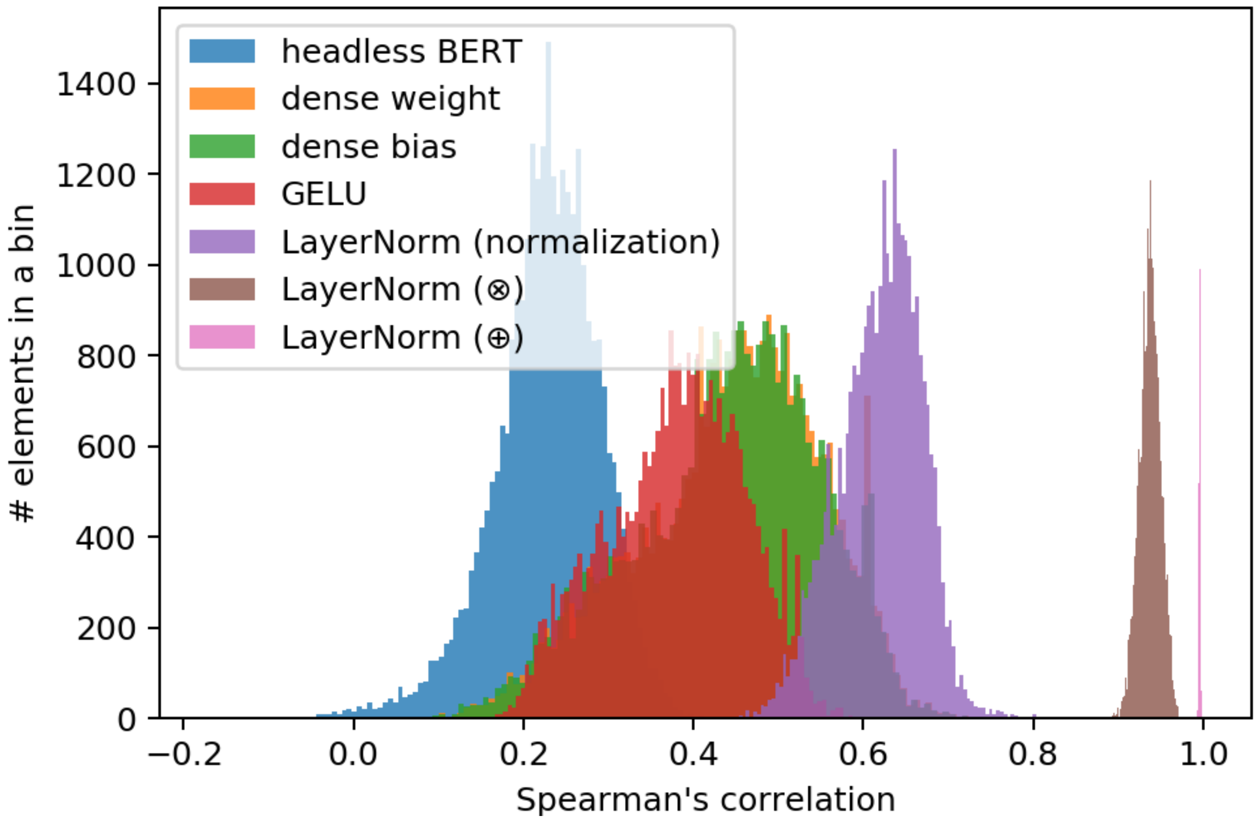

Figure 3.2: Histogram of Spearman's rank order correlations between output of BERT with LM head and vectors produced from headless BERT output and each of the LM head activations using inner product with token embeddings, aggregated over the ConceptNet dataset with 29774 samples.

applications. The first one is the effect of single word replacement on the retrieval results and the second one is the effect of inconsequential syntactical changes on the results. Note that many more similar and other changes can be designed and studied. For example, consider the probe *'Barack Obama was born in X.'*, where we can vary the token $X$ to investigate how the predictions depend on the replacement token itself. When we set $X$ to *'ukraine'*, the top prediction is *'ukraine'* with the highest confidence, followed by *'ukrainian'*, *'where'*, *'crimea'*, and *'delta'*, *'kiev'*. We observe this pattern with a number of probes and replacement tokens. To track the replacement token effect on the prediction quantitatively, we fixed a probe sentence and ran a prediction for each token in the vocabulary used as the replacement token. In approximately 75% of cases, the replacement tokens themselves got predicted as the top prediction, while the distribution of the remaining cases is shown in Table A.2 in Appendix. Furthermore, we also ran experiments by replacing [MASK] using particular tokens on the entire ConceptNet dataset. The results show that the replacement tokens

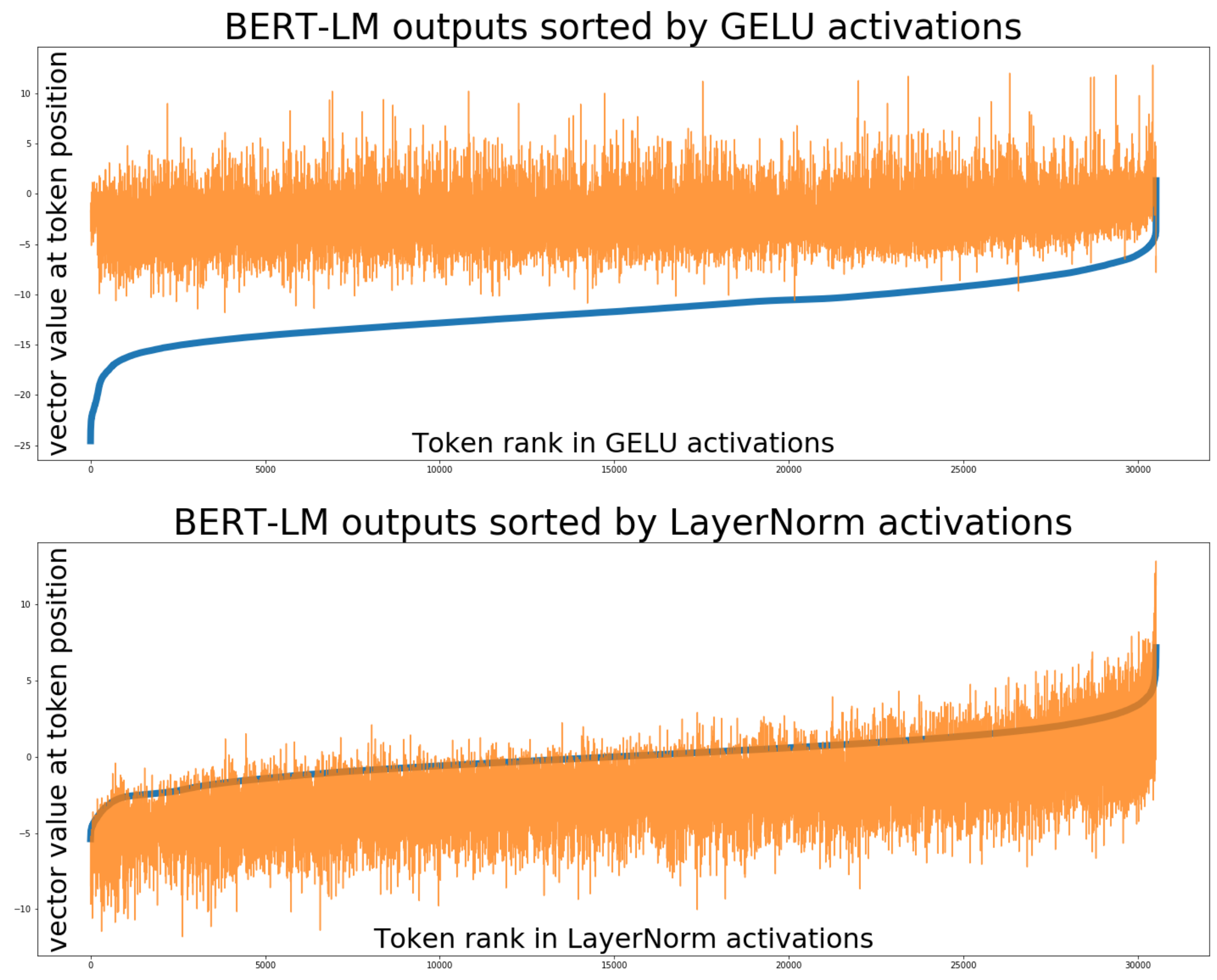

Figure 3.3: BERT with LM head outputs sorted by inner products of token embeddings with LM head components' activations (top: GELU, bottom: LayerNorm).

themselves are retrieved around 73% of time (from 65% to 84%). This experiment demonstrates that while the influence of the context on the output representations improves downstream classification tasks and other applications, it is not sufficient for the model to retrieve knowledge reliably and generalize well to new sentences in meaningful ways.

Furthermore, we quantify the model fragility by crafting perturbations that can be applied to a large number of samples subject to token unmasking. We considered three shallow changes: (1) removing the period ending the sentence (if there is one), (2) removing [SEP] token normally inserted at the end of input token sequence, and (3) replacing [MASK] token with 'cat' while still using ranks of output components in the token position to retrieve the reconstructed token from vocabulary. While such changes are minimal in terms of edit distance [42] and do not change the semantics of input sentences, they affect the predictions significantly both in individual samples and cumulatively. For example, consider the sentence *'One of the things you do when you are alive is [MASK].'* taken from the ConceptNet dataset [58]. The top 5 original predictions of uncased base BERT configuration are *'sleep', 'think', 'cry', 'eat', 'pray'*. With the final period token removed from the original token sequence, the top predictions become *'.', ';', '?', '!', '|'*; with [SEP] token removed from the original token sequence – *'.', '"', 'the', ',', ')'*; with [MASK] token replaced by the 'cat' one – *'cat', 'cats', 'dog', '##cat', 'kitten'*. We systematically evaluated the three changes. In the original cases and the first two cases, we breakdown cases based on whether the [MASK] token is at the end (meaning it is the last token other than '.', and [SEP]) or not. Table 3.1 displays evaluation of retrieval quality of the original and perturbed samples on the entire ConceptNet dataset. It shows that the perturbations decrease the fraction of samples where the correct token is in top N predicted tokens according to logit ranking for each N in (1, 2, 5, 10, 20, 50, 100), and the decrease is at least two-fold. The results show clearly syntactic patterns play an important role in the knowledge retrieved. The experiments show that having [MASK] in the third to last position in the input token sequence accounts for the overwhelming majority of mispredictions both in final period removal and [SEP] removal probes. It is likely due to that the words tend to co-occur much more strongly with their neighboring words [11], a change closer to [MASK] would therefore have a stronger effect. Finally, this evaluation shows that there is a clear difference in the effect among the perturbations. Removing the final period is the least confusing while token replacement by 'cat' causes the most confusion. We also experimented with tokens other than 'cat'; the performance is generally similar. Using 'thereof' reduces the performance even more with 316 for H@1 and 1889 for H@10 respectively.

We also compute a histogram of Spearman's rank-order correlations [72] between the original and each of the perturbed cases' output vectors to probe whether the vectors' geometry affects the prediction results. The results are displayed in Figure 3.1. We observe that the pattern with prediction quality deterioration is reflected also in the correlation plots. While removing '.' still correlates very strongly with the original predictions, the correlation is reduced substantially when the [MASK] is replaced by 'cat'. Additional examples are also available in Appendix (Figure A.6).

## 3.6 Dangerous Factual Errors

In the previous sections, we have explored structural limitations of BERT to alter its knowledge retrieval performance. Here, we aim to highlight a limitation of another kind: due to its MLM

Table 3.2: Results of the misconception experiment. MC stands for the percent of errors that most directly correspond to the common misconceptions.

| | Acc | MC | H@1 | H@2 | H@5 | H@10 | H@20 | H@50 | H@100 |
|---|---|---|---|---|---|---|---|---|---|
| RoBERTa | 31.19 | 73.33 | 15 | 23 | 37 | 53 | 64 | 76 | 85 |
| ALBERT | 33.94 | 70.83 | 8 | 16 | 28 | 46 | 59 | 72 | 82 |
| ELECTRA | 30.28 | 67.10 | 11 | 21 | 31 | 51 | 60 | 81 | 89 |
| BERT | 34.86 | 71.83 | 14 | 22 | 33 | 57 | 66 | 85 | 91 |

Table 3.3: Examples of queries from the misconception dataset. Answers predicted by BERT are in bold, correct answers are underlined.

| Query | Answers |
|---|---|
| Tomatoes are a [MASK]. | **A. vegetable** B. meat C. juice <u>D. fruit</u> |
| Schizophrenia is [MASK] to dissociative identity disorder (having multiple personalities). | **A. equivalent** B. ill C. forget <u>D. different</u> |
| Ostriches stick their heads in the sand to [MASK]. | **A. hide** B. cover C. threat <u>D. dig</u> |

objective, BERT is trained to reconstruct a token that is statistically likely to appear, whether a sentence is factually correct or not. For instance, for the input sentence *"Vaccines cause [MASK]."*, the top 5 predictions are *'cancer', 'disease', 'death', 'diseases', 'mortality'*; more examples can be found in Table 3.3. Such factual misconceptions could cause a disruption in any application that uses a pre-trained model as the source of knowledge.

In order to estimate the possible degree of confusion, we have collected a dataset with common misconceptions. During construction of the dataset, we have used the misconceptions infographic[2] with 100 widely believed incorrect facts. However, we had to reword the sentences, so that there is a blank that could be filled in for each sentence. We have run an open-ended experiment, where any token from the vocabulary can replace the blank, and its output logit is used for ranking. We have also run an experiment with a *limited* number of replacement choices to mimic tests used for human evaluation, where only a few options are available for a given probe. For example, there is a common misconception that rust causes tetanus; the relation is at most correlative, as it is not feasible to distinguish between tetanus-prone and non-tetanus-prone wounds [63]. In the limited-choice blank replacement task, the prompt is *'Rust [MASK] directly cause tetanus.'*, and the options are *'can'*, *'also'*, *'will'* and *'cannot'*; only the last option is correct. There are 109 samples in our misconception dataset. The performance on the dataset was compared among BERT and its three variations: RoBERTa [44], ALBERT [41], and ELECTRA [13]. The results are shown in

[2]`https://www.geekwrapped.com/myth-infographic`

Table 3.2. None of the models achieve an accuracy much higher than chance, with BERT being the best one with an accuracy of 34.86%. Furthermore, error analysis shows that most of the mispredictions, about 70% for all the models, correspond to the answers most directly reflecting the common misconceptions. Somewhat surprisingly, the accuracy of RoBERTa is about 3.5% lower than BERT, while the number of answers reflecting the common mispredictions is 1.5% higher. The main differences between BERT and RoBERTa are that the latter was trained on a larger corpus and for a larger number of steps, resulting in a larger exposure to "common knowledge" from the Web based resources, which are not always reliable. The results suggest the obvious that accumulating more data will not lead to more reliable knowledge and using larger datasets will not fix the inherent issues.

## 3.7 Discussion

By the masked language model, BERT produces representations that depend on the entire input sequence. Empirical results on benchmarks demonstrate the effectiveness of bidirectional contexts. One consequence is that the resulting model is more difficult to analyze, as its outputs heavily depend on the input sequences themselves. While the model should have learned that [MASK] tokens need to be replaced in the input sequences, it is not clear what the model would do when there are no [MASK] tokens. The uncertainty introduced by using random words for the 10% of the replacements renders the situation more complex, as it is constantly unclear which words need to be replaced, on top of the [MASK] tokens. Ideally, the model should first learn if the input sequence is consistent; if it is not consistent, it should then identify the words that need to be replaced. Unfortunately, during pre-training consistent inputs are non-existent. Furthermore, due to its continuous nature, the model can not avoid the influence of the random or [MASK] tokens on the output representations, therefore, cannot make consistent decisions. As a result, given 'Barack Obama was born in X.', the model will likely produce a set of different answers depending on X. Ironically, the model will change the correct answer to other ones influenced by the replacement token. We argue that one should not interpret occasional correct statements of BERT as knowledge sources, which must be robust against noise and can not change due to external inputs. At a minimum, knowledge sources can not give answers that are constantly changing, which is what BERT models do.

## 3.8 Conclusion and Future Work

As the predictions in BERT models depend on the input sequence itself, the knowledge stored in BERT is input-dependent and therefore not reliable. In this part we have performed a number of experiments confirming that. We have explored two approaches to show fragility. The first is making minor syntactic changes that *systematically* perturb the inference results, and the latter is to exploit *the high frequency of incorrect facts*. The results show that one should proceed with caution when using pre-trained language models as source of knowledge in downstream applications, which may include search engines and expert systems.

Besides the two perturbation types we have used, many interesting ones can be designed and studied to reveal the knowledge retrieval properties of BERT. For example, one could reorder sentences based on parse trees without changing the semantics. As BERT contextual representations tend to depend heavily on words in a local neighborhood [11], reordering should influence the knowledge retrieval results and the actual impact can be modeled empirically. Another interesting type is to modify sentences based on their inherent ambiguities and redundancies. Further interesting future work directions include developing automated methods for debiasing language models from most commonly held misconceptions, including ones leading to gender and racial biases, detecting such misconceptions, and making the language models more robust to query syntax changes.

# CHAPTER 4

# BERT AS A LANGUAGE MODEL

## 4.1 Introduction

BERT [16] is a recently released model that has been utilized to achieve state-of-the art results in multiple NLP tasks. Such success has made it commonly used as a part of systems for natural language processing, hence studying BERT is important for ensuring its proper usage. One of the means, in which the systems that depend on BERT may not function properly, is exhibiting racial [2] and gender [7] biases, affecting protected groups of population. We aim to find the source of the exhibited biases in the model itself, namely, in the language model that the pre-trained BERT defines.

This leads us to the language model exhibited by pre-trained BERT. However, it is not immediately clear how to define the language model, since BERT has been pre-trained not with *language modeling*, but with *masked language modeling* objective. The latter does not assign probabilities to input token sequences, as a language model would do, but aims to reconstruct perturbed tokens in the input sequence. We start the current study with working around this limitation by showing how outputs of the masked language model could be aggregated to define a proper language model. Then, we design algorithm for language model computation for any set of n-grams, noting, however, that for large n the space size of all n-grams grows exponentially, and we need to restrict the computation to a subset when we run out of storage for the computed language model. We also note that our algorithm could be modified to search for all n-grams with high probability, and provide such modification, allowing us to explore the regions with highest probabilities. All our algorithms depend on a dataset to define the prior n-gram probability as its occurrence frequency in natural text. We evaluate our algorithms on several large datasets and show that pre-trained BERT exhibits systematic biases, assigning highest probabilities to n-grams containing repeated tokens, some of such n-grams not occurring at all in natural text. We believe these biases originate from mismatch between BERT being pre-trained on large input sequences and our evaluation on sequences of smaller length. Finally, we discover the application of our algorithm to filling multiple blanks in a way that a masked language model does, but relying on the underlying language model instead; we show the way to apply such blank filling approach to a specifically chosen prompt in order to look for gender biases.

## 4.2 Related Work

The first work introducing a language model was [70], where Shannon created a statistical model of English language using markov chains with sequences of letters as states. While it is not trivial to define a language model for a masked language model such as BERT, we know there were previous attempts to accomplish the language model definition. Wang et al. in [79] provide interpretation for BERT as a Markov Random Field language model. It was pointed out later that BERT, while being a generative model, is not a Markov Random Field because of incorrect independence assumption. Goyal et al. in [25] design a sampling scheme from underlying language model based on a Monte Carlo approach; we focus on exact computation of the language model instead. There are studies identifying and quantifying biases in BERT, for instance, [5]. However, these works have been based on prompts, so that biases are evaluated by unmasking the special MASK token in either a manual selection of inputs, or inputs generated according to a set of manually selected templates. Our approach is more rigorous as we are building our reasoning on statistical text properties.

## 4.3 Language Model Definition

A language model assigns a probability to any sequence of tokens from a language. We note, that BERT with attached language modeling head is not a language model as its name implies, as the output probabilities vary along with the input tokens. In order to compute the probability distribution of token sequences independent of any input, we must aggregate the BERT outputs over all possible inputs, applying the law of total probability:

$$P_{LM}\left(x\right) = \sum_{i \in input} P_{LM}\left(x, i\right) \tag{4.1}$$

Using the definition of conditional probability, we obtain

$$P_{LM}\left(x, i\right) = P_{prior}\left(i\right) P_T\left(x|i\right) \tag{4.2}$$

In the last equation, the prior probability must be estimated from a *seed* distribution, of n-gram frequencies in a natural text. Note that different datasets may be used for estimation, and different choices will produce different n-gram frequency distributions.

We can rewrite the equations 4.1 and 4.2 to aggregate over inputs only occurring in natural text, assuming no smoothing technique [12] is used, since a zero prior corresponds to a zero summand, and zero terms do not contribute to the sum:

$$P_{LM}\left(x\right) = \sum_{i \in \operatorname{supp} P_{prior}} P_{prior}\left(i\right) P_T\left(x|i\right) \tag{4.3}$$

In case a smoothing technique is used, the formula is still valid, however, the support of $P_{prior}$ is the entire set of $n$-grams, hence this choice would hinder from computing the exact language model in case $n$ is large. In order to obtain the conditional probability $P_T\left(x|i\right)$, for the target token sequence $x = (x_1, x_2, \ldots, x_n)$, we run BERT with attached language modeling head providing a specific input $i = (i_1, i_2, \ldots, i_n)$. The output $T(i)$ has the array shape $(n, \|vocabulary\|)$, with the first dimension corresponding to the input sequence and the second dimension corresponding to the vocabulary token conditional probabilities. Since the outputs are mutually conditionally independent (but conditionally dependent on the inputs), we factorize the joint output probability conditional on the inputs to obtain:

$$P_T\left(x|i\right) = P_T\left(x_1, x_2, \ldots, x_n|i\right) = \prod_{k=1}^{n} P_T\left(x_k|i\right) = \prod_{k=1}^{n} T(i)_{k,x_k} \tag{4.4}$$

Combining the equations 4.3 and 4.4, we obtain the following formula for the posterior probability of an n-gram estimated by BERT:

$$P_{LM}\left(x_1, x_2, \ldots, x_n\right) = \sum_{i \in \operatorname{supp} P_{prior}} P_{prior}\left(i\right) \prod_{k=1}^{n} T(i)_{k,x_k} \tag{4.5}$$

## 4.4 Language Model Computation

Efficient computation of the language model defined above is challenging; in order to obtain a probability for any single n-gram in the language model, we must compute BERT outputs for *all* n-grams present in the seed dataset. At the same time, we can obtain the probabilities in that language model for *any* number of n-grams simultaneously, by independently storing the partial sums for each n-gram. Based on this observation, and Equation 4.5, we outline Algorithm 1 for language model computation.

Using this algorithm, it is not always feasible to compute the entire distribution, as we would have to store $\|vocabulary\|^n$ probabilities for all n-grams. However, we can compute the probabilities

**Algorithm 1:** Language model computation algorithm

```
Input  : $n$ number of tokens in a sequence (n-gram), $target\_ngrams$ set of n-grams to
         compute language model for
Data: natural text
Result: probabilities assigned by the language model to n-grams in $target\_ngrams$
1  priors := dataset_frequencies(all n-grams in the natural text dataset) ;
2  posteriors := zeros-vector-of(size(target_ngrams)) ;
   // n_gram = n_grams_data[idx_data, :]
3  for idx_ data, n-gram in dataset do
4      lm_out := BERT-LM(n_gram) ;
       // scalar multiplied by vector
5      pv := priors[idx_data] * ones-vector-of(size(target_ngrams)) ;
6      for idx_ gram in range(1, n) do
           // vectors are multiplied element wise
7          pv *= lm_out[idx_gram, target_ngrams[:, idx_gram]] ;
8      end
9      posteriors += pv ;
10 end
11 return posteriors
```

for any set of n-grams of a reasonable size. A good candidate for such set would be the n-grams occurring in the natural text, as it would allow to reason about the difference between the observed distribution and the distribution estimated by the language model.

The algorithm for language model computation could be enhanced to search for all n-grams having large probability estimated by the language model, which would help us extend our analysis of n-gram distribution differences. We will show that we can find k n-grams with the largest probability in a language model, for a reasonable choice of k, using a coarse-to-fine search strategy. In order to find the top k n-grams, we first partition the vocabulary; this partitioning yields a partitioning of n-grams search space, with a single partition in the n-gram space being a Cartesian product of n vocabulary partitions. Next, we note that if we are able to estimate the upper bound of probabilities in each group, we could prune entire partitions if their corresponding upper bound is too small based on a heuristic. As the starting point for the heuristic, we may compute the posterior probabilities for n-grams in the dataset, consider k-th largest of these values as a reasonable approximation of k-th largest posterior of all possible n-grams, and prune the groups whose upper bound is less than that value. After pruning, we can split the remaining groups into smaller partitions, and iterate refining and pruning until all n-grams whose posterior is larger than the cutoff value are found; note that no explicit asymptotic runtime or memory usage bound are derivable, since we don't know exactly how many n-grams we will retrieve as the result, or how many partitions end up having sufficiently high upper bound to be subject for refined search.

In order to estimate the upper bound of probabilities within a partition, we note that

$$\max_{x \in G} P_{LM}(x) = \max_{x \in G} \sum_{i \in \operatorname{supp} P_{prior}} P_{prior}(i)\, P_T(x|i) \leq \sum_{i \in \operatorname{supp} P_{prior}} P_{prior}(i) \max_{x \in G} P_T(x|i) \tag{4.6}$$

Next, we observe that within a partition, instead of examining the entire Cartesian product $G$ of n vocabulary partitions to find the maximum, we can independently search in each vocabulary partition $G_k$, corresponding to a position within an n-gram, as each of the BERT outputs is strictly positive:

$$\max_{x \in G} P_T(x|i) = \max_{x \in G} \prod_{k=1}^{n} T(i)_{k,x_k} = \prod_{k=1}^{n} \max_{x_k \in G_k} T(i)_{k,x_k} \tag{4.7}$$

In Algorithm 2, we modify the Algorithm 1 to search for $k$ n-grams that occur in a natural dataset and are assigned largest probabilities by the language model. The result of evaluation of Algorithm 2 provides us with a heuristic that could be utilized for the efficient search of n-grams with largest probabilities estimated by the language model in the entire space. In Algorithm 3, we describe the efficient search procedure for the likeliest estimated n-grams in the entire n-gram space.

## 4.5 Experimental Results

We study the posterior probabilities estimated by BERT on two datasets from different domains: CNN/DailyMail dataset, and BookCorpus. We expected to observe little difference between priors and posteriors, since the language model should not affect the language it has been trained on; however, our study shows it is not the case for this particular dataset. We suggest, that since such discrepancy exists, there may be n-grams with large estimated posterior that are not present in the dataset (i.e. having zero prior). Finding these n-grams in a naive way is computationally prohibitive, since one would need to compute the posteriors for all possible n-grams, and would require $V^n$ memory just for storing the n-grams' posterior probabilities. We design Algorithm 3 to

```
Algorithm 2: In-dataset top-k n-gram search
    Input  : $n$ number of tokens in a sequence (n-gram), $k$ number of sequences with largest
             estimated posterior
    Data: natural text
    Result: top $k$ $n$-grams in the natural text, according to BERT
 1  priors := dataset_frequencies(all n-grams in the natural text dataset) ;
 2  dataset_posteriors := zeros-vector-like(priors) ;
    // n_gram = n_grams_data[idx, :]
 3  for idx, n-gram in dataset do
 4      lm_out := BERT-LM(n-gram) ;
 5      pv := priors[idx] * ones-vector-like(dataset_posteriors) ;
 6      for i in range(1, n) do
 7          pv *= lm_out[i, n_grams_data[:, i]] ;
 8      end
 9      dataset_posteriors += pv ;
10  end
11  sorted_dataset_posteriors := sort-decreasing(dataset_posteriors) ;
12  return sorted_ dataset_posteriors[:k]
```

work around the memory limitation and search for a reasonably large number of n-grams having largest estimated posterior in the entire search space. We show, that in case of trigrams, the found token sequences are unnatural.

### 4.5.1 BookCorpus

The BookCorpus dataset is a collection of consequent sentences from books. For obtaining the data, we have used the HuggingFace implementation. In order to obtain the prior distribution of n-grams, we concatenate the sentences. We post-process the input trigrams, dropping those having the number of occurrences in text less than 10. There are 8638510 trigrams left for our consideration.

As the first step, we estimate the posteriors for the trigrams in the dataset. We find that only 2.72% of trigrams are overestimated (the posterior is larger than the prior), while the rest are underestimated. The harmonic mean of estimation ratios is 7.5054e-07 which also suggests that BERT is prone to underestimation on the dataset. The total posterior probability on all trigrams in the dataset is 0.19, so the majority of the posterior probability mass is allocated to the trigrams absent from the dataset.

The next question we address, is whether the bulk of that mass corresponds to a few large values, or can be attributed to the large size of the search space of trigrams. After we have obtained the posteriors for the trigrams in the dataset, we set the threshold to search for all trigrams that are estimated to be more frequent than the top-250 estimated trigram in the dataset. We find that there are 488 trigrams in the search result set, having BERT assign a large non-zero posterior to 233 trigrams that are absent from the dataset. The accumulated estimated posterior for this region of trigrams is 0.12, where 0.0852 is contributed by the trigrams from the dataset, and 0.0382 is contributed by the trigrams absent from the dataset.

### 4.5.2 CNN/DailyMail

The CNN/DailyMail dataset is comprised of two groups of stories, taken from the archive of corresponding newspaper, and each story is a post-processed news article. We use these stories for constructing the prior distributions of n-grams for the two groups separately, using the n-gram occurrence frequency as its prior probability. Then, in each individual group, we compute the posterior distribution of n-grams, estimated by BERT for the n-grams present in the dataset, using Algorithm 2.

We study the performance on CNN Stories in the same way as we did with BookCorpus dataset. We denote the subject of our study the distribution of trigrams. As the pre-processing step, we discard the trigrams having number of occurrences in text less than 5. There are 2164378 trigrams left for our consideration.

Then, we obtain the posteriors estimated by BERT on trigrams present in the dataset. We observe that the cumulative posterior for all trigrams present in the dataset is 0.11. We also find that only 1.34% trigrams are overestimated, while the majority of trigrams is underestimated. The harmonic mean of overestimation ratios is 5.0246e-08 which similarly suggests that the model is prone to underestimate.

Next, we search for all trigrams having posterior higher than the top-250 estimated posterior on the dataset. We find that there are 666 trigrams in the result set, hence, a high posterior probability has been assigned to 416 trigrams absent from the original dataset. The aggregate posterior in that region is 0.1157, where the trigrams from the dataset constitute 0.0551.

## 4.6 Introspecting Bias via Language Model Formulation

Previously, we were concerned with computing the context-independent language model underlying BERT. The absence of context during computing the probability assigned by the language model is governed by passing the n-gram as input to BERT-LM in Algorithms 2 and 3. With slight modification, we pass each n-gram augmented with context tokens in the beginning and the end of the original n-gram; for example, original trigram *(apple; banana; milk)* becomes *([CLS]; apple; banana; milk; is; a; doctor; [SEP]; .)*. In such setting, we still will be computing the probability assigned by the language model, but the computation will depend on the augmenting context. We could also interpret such setting as assigning probability to n-grams while filling n blanks in a given prompt.

With aforementioned modification, we investigate whether the augmenting context may result in biased predictions by the language model that we have defined. We have picked the prompt ([CLS]; _; _; is; a; doctor; [SEP]; .) as it is common pitfall in NLP systems to associate the notion of doctor with male themed words and nurse with female themed ones; the prompt has 2 blanks, so we will be evaluating the language model for bigrams.

We evaluate the contextual language model with Text8 used as the dataset for computing natural text bigram occurrences. When we compare the likeliest bigrams according to prior and posterior probabilities, we observe large overlap between the lists (see Table 4.3). We also compute spearman rank-order correlation between the lists, and obtain value of 0.48, which indicates that ranks of the contextual probabilities of bigrams do not differ significantly.

## 4.7 Discussion

We have estimated the Language Model for base uncased variant of pre-trained BERT. This LM depends on a dataset for defining prior n-gram distribution, so we study the results of the model using different datasets for obtaining priors. We find that the posterior distribution is significantly different from the prior in both cases. Although the majority of trigrams were underestimated, the trigrams with largest posteriors do not appear to be natural, that is, they are significantly overestimated.

**Algorithm 3:** Top-k n-gram search in entire n-gram space, employing partitioning

```
Input  : $n$ number of tokens in a sequence (n-gram), $k$ number of sequences with largest
         estimated posterior
Data: natural text
Result: top $k$ $n$-grams in the entire space, according to BERT
1  groups := partition vocabulary into sqrt(size(vocabulary)) chunks ;
2  while group size $> 1$ do
3      group_posteriors := zero-valued array for upper bounds of posterior within a group ;
4      for idx, n-gram in dataset do
5          lm_out := BERT-LM(n-gram) ;
6          pv := priors[idx] * ones-vector-like(group_posteriors) ;
7          for i in range(1, n) do
8              pv *= max-per-group(lm_out[i, :]) ;
9          end
10         group_posteriors += pv ;
11     end
12     new_groups := [] ;
13     for p in group_posteriors do
14         if $p >$ threshold then
15             add corresponding group to new_groups ;
16         end
17     end
18     groups := re-partition each group in new_groups for the same number of groups as we
        had originally ;
19 end
20 return sort-decreasing(flatten(groups))[:k]
```

Table 4.1: Top *in-dataset* trigrams for CNN dataset, sorted by estimated posterior. The overestimation is the ratio of the posterior and the frequency in the dataset.

| top N | tokens | posterior | overest. ratio |
|---|---|---|---|
| 1 | [’.’, ’.’, ’.’] | 0.01356 | 18.55 |
| 3 | ["’", "’", "’"] | 0.005717 | 29.13 |
| 5 | [’-’, ’-’, ’-’] | 0.002646 | 1.813 |
| 6 | [’-’, "’", "’"] | 0.001377 | 139.7 |
| 8 | [’-’, ’.’, ’.’] | 0.00124 | 233.7 |
| 9 | [’.’, ’and’, ’.’] | 0.001172 | 2438 |
| 12 | [’.’, "’", "’"] | 0.0009727 | 0.1807 |
| 15 | [’,’, ’and’, ’and’] | 0.0008901 | 4814 |
| 16 | [’and’, ’.’, ’.’] | 0.0008116 | 105.3 |
| 17 | [’,’, ’.’, ’.’] | 0.0007024 | 263.8 |
| 18 | [’or’, ’‘’, ’‘’] | 0.0006257 | 14.89 |
| 19 | [’,’, "’", "’"] | 0.0005943 | 0.09116 |

Table 4.2: Top *out-of-dataset* trigrams for CNN dataset, sorted by estimated posterior.

| top N | tokens | posterior |
|---|---|---|
| 2 | [’and’, ’and’, ’and’] | 0.01066 |
| 4 | [’[’, "’", "’"] | 0.00493 |
| 7 | [’...’, ’and’, ’and’] | 0.001287 |
| 10 | [’.’, ’and’, ’and’] | 0.001101 |
| 11 | [’...’, ’.’, ’.’] | 0.00102 |
| 13 | [’-’, ’and’, ’and’] | 0.0009679 |
| 14 | [’and’, ’the’, ’and’] | 0.0009393 |
| 20 | [’and’, ’and’, ’.’] | 0.0005834 |
| 21 | [’or’, ’and’, ’and’] | 0.000575 |
| 22 | [’the’, ’the’, ’the’] | 0.0005626 |

Table 4.3: Top 10 likeliest bigrams according to occurrence frequency in Text8 and according to contextual language model based on the prompt `([CLS]; _; _; is; a; doctor; [SEP]; .).` . We observe a large degree of similarity between the bigram ranks.

| Rank | Bigram (text8) | Prior p | Bigram (contextual) | Posterior p |
|---|---|---|---|---|
| 1 | [’of’, ’the’] | 0.0085 | [’of’, ’the’] | 0.0081 |
| 2 | [’one’, ’nine’] | 0.0082 | [’in’, ’the’] | 0.0047 |
| 3 | [’in’, ’the’] | 0.0050 | [’one’, ’nine’] | 0.0029 |
| 4 | [’zero’, ’zero’] | 0.0048 | [’to’, ’the’] | 0.0028 |
| 5 | [’two’, ’zero’] | 0.0030 | [’and’, ’the’] | 0.0026 |
| 6 | [’to’, ’the’] | 0.0026 | [’in’, ’one’] | 0.0021 |
| 7 | [’nine’, ’nine’] | 0.0021 | [’the’, ’other’] | 0.0017 |
| 8 | [’one’, ’eight’] | 0.0021 | [’on’, ’the’] | 0.0016 |
| 9 | [’in’, ’one’] | 0.0020 | [’two’, ’zero’] | 0.0016 |
| 10 | [’and’, ’the’] | 0.0020 | [’the’, ’one’] | 0.0015 |

# CHAPTER 5

# SAMPLING AND GENERALIZATION MECHANISMS OF GLOW

## 5.1 Introduction

Statistical modeling for data analysis is the process of creating a mathematical model that describes observed data. In statistical modeling, there are two main approaches, namely, discriminative and generative. The discriminative and generative approaches model different random variables. The discriminative models assign a conditional probability (in case when the sample space is discrete), or probability density (in case when the sample space is continuous) to target variable given the previously observed data. The generative models assign a joint probability to the target variable and previously observed data, or conditional probability to observed data given the target variable. The distinction has the consequence in how these models may be used. The discriminative models are used for classification, as between several choices of the target variable outcome one has the largest conditional probability assigned by the model. The generative models may be used for likelihood estimation for a given pair of observed data and target variable, or for *generation* of observed data given the target variable (effectively searching in the observed data space for instance that maximizes its conditional probability given the target).

Common to all statistical modeling approaches, there is the problem of estimating the normalization constant. Namely, the sum of probabilities, or the integral of probability density, over entire sample space must be equal to 1. Many models define unnormalized probability or probability density $q(x, \theta)$, so further normalization is necessary to obtain the normalized probability:

$$p(x, \theta) = \frac{1}{Z(\theta)} q(x, \theta) \tag{5.1}$$

$$Z(\theta) = \int\limits_{x \in \mathcal{X}} q(x, \theta) dx \tag{5.2}$$

The normalizing constant $Z(\theta)$ is often intractable to compute because of high sample space dimensionality and uncertainty of the integration region.

A series of methods utilizing deep artificial neural networks for generative modeling have been designed, and these methods are collectively named deep generative models. A few notable examples of deep generative models are deep belief networks [29], variational autoencoders (VAEs) [39], generative adversarial networks (GANs) [24], normalizing flows [17]. Each of these families of models treat the normalization constant problem in a unique way. GANs avoid probabilistic formulation of the model, so there is no need to define the normalization constant. VAEs model latent space distribution as Gaussian with zero mean and unit variance $z \sim \mathcal{N}(0, I)$, and sample space distribution is conditionally Gaussian with mean and variance estimated by neural networks $(x|z \sim \mathcal{N}(\mu_\theta(z), diag(\sigma_\theta(z))))$. The normalizing flows model latent space distribution as a tractable distribution, and provide invertible transformation that bijectively maps the latent space into the original sample space; the probability density in the original sample space is computed using the change of variables formula.

Normalizing flows, in general, assume some tractable probability distribution in the latent space (e.g. standard normal) and learn a bijective and differentiable transformation to represent a more sophisticated distribution in the sample space. The transformation may be modeled as a composition of bijective and differentiable functions (flow steps), or as a solution to a boundary value problem (continuous flows), utilizing black-box differential equation solvers within the framework of weight optimization.

In discrete normalizing flows, the computational bottleneck is in computing the Jacobian determinant of each flow step transformation, as in general case, to compute the determinant of $N \times N$ matrix, we would need to perform $O(N^3)$ operations. The common workaround for this limitation is to consider only classes of transformations for which the Jacobian is upper-triangular matrix, so the determinant computation has linear time complexity. The notable approaches of discrete normalizing flows are planar flows, radial flows, autoregressive flows and coupling flows. The planar and radial flows do not have a closed-form inverse, which hinders their practical applications. The autoregressive flows have a closed-form inverse that needs to be evaluated sequentially because of conditional dependency on past values, so either generative or normalizing directions may be computed efficiently at a time.

The coupling flows use the coupling method to improve expressivity of flow step transformations. The essence of the coupling method is introducing coupling transformations, where the input $x$ is first split into 2 parts $x_A$ and $x_B$; then $x_A$ is supplied as input to bijective function that is conditioned on $x_B$, and $x_B$ propagates unchanged:

$$y_A = h(x_A; \Theta(x_B)) \tag{5.3}$$

$$y_B = x_B \tag{5.4}$$

The expressive power of coupling flow is the consequence of using $\Theta(\cdot)$ function (the *conditioner*), this function can be arbitrarily complex and in practice is modeled using a neural network. $h(\cdot)$ bears the name of the *coupling function*.

The Glow model is a coupling flow; the coupling function in Glow is affine, the conditioner is a shallow ResNet, and a convolutional layer with kernel size $1 \times 1$ as one of flow steps and a means to mix the channels in the input. The samples, computed by running the computation in the generative direction, look relatively photorealistic, and can be produced with high computational efficiency, however, it is not clear whether these images belong to the training dataset distribution. That leads us to asking research question R1: *How different are the original and sampled distributions?* Another research question R2 in context of evaluating biases is: *Does the induced distribution in the sample space have disproportionate treatment of any protected attributes?*

Due to their inherent properties of efficient sampling and computation of the exact log likelihood, deep generative models based on normalizing flows are being studied actively. However, the mechanisms how such models generalize and therefore enable sampling are not well understood. In this study, by focusing on the GLOW model, a recent normalizing flow model, we plan to elucidate the underlying mechanisms.

### 5.1.1 The GLOW Model

GLOW model was introduced in [40]. It is a flow-based generative model [17, 18]. In flow-based models, the approximated function is composed of a sequence of invertible transformations (*flow steps*). As a consequence, the approximated function is itself invertible, so we may compute latent representations taking RGB images as input (*forward* flow), as well as sample from the latent space and compute the representation in the initial image space (*reverse* flow).

For the sampling from the latent space, GLOW model utilizes a distribution with a closed-form probability density, namely, a multivariate normal distribution with a diagonal covariance matrix. Given the latent space distribution as the starting point, under the *change of variable* rule for probability density, we can compute the probability density at each of the intermediate flow steps.

More formally, denoting the sampling that produces the initial latent space representation:

$$\mathbf{z} \sim p_\theta(\mathbf{z})$$

Applying the flow steps, we get a sequence of intermediate representations:

$$\mathbf{h}_0 = \mathbf{z}$$

$$\mathbf{h}_i = f_i(\mathbf{h}_{i-1}), \quad i = 1 \ldots n$$

$$\mathbf{x} = \mathbf{h}_n$$

The last intermediate representation is the generated image.

Under the change of variable rule for any flow step, the probability density is multiplied by the Jacobian determinant:

$$p_\theta(\mathbf{h}_i) = \left|\frac{d\mathbf{h}_i}{d\mathbf{h}_{i-1}}\right| p_\theta(\mathbf{h}_{i-1}) = \left(\prod_{k=1}^{i} \left|\frac{d\mathbf{h}_k}{d\mathbf{h}_{k-1}}\right|\right) p_\theta(\mathbf{z})$$

To avoid incurring numerical computing issues for floating numbers of small magnitude, GLOW deals with *log*-densities:

$$\log p_\theta(\mathbf{h}_i) = \sum_{k=1}^{i} \log\left|\frac{d\mathbf{h}_k}{d\mathbf{h}_{k-1}}\right| + \log p_\theta(\mathbf{z})$$

The Jacobian determinant $\left|\frac{d\mathbf{h}_k}{d\mathbf{h}_{k-1}}\right|$ is computationally expensive in general case; however, GLOW utilizes flow steps with Jacobian matrix having zeroes below the diagonal, so its determinant is the product of the diagonal elements. The scalar value $\log\left|\frac{d\mathbf{h}_k}{d\mathbf{h}_{k-1}}\right|$ is the change in log-density under transformation $f_k$.

### 5.1.2 Flow Step

A flow step in GLOW consists of 3 smaller transformations. Both input $x$ and output $y$ of each transformation are rank-3 tensors of the same dimension $(h \times w \times c)$. **Activation normalization** performs an affine transformation of the activations using scale and bias parameter per channel (i. e. the scale and bias parameters are shared for all pixels in that channel). These parameters are learned; they are independent of the input.

$$\forall i, j : y_{ij} = \mathbf{s} \odot x_{ij} + \mathbf{b}$$

**Invertible 1 $\times$ 1 convolution** mixes the input channels. The sublayer has a learnable matrix $\mathbf{W}$ with dimensions $c \times c$. Treating each pixel in both input and output as $c$-dimensional vector, the transformation is equivalent to a matrix-vector product. Note that the parameter $\mathbf{W}$ is shared between all input pixels and is independent of the input.

$$\forall i, j : y_{ij} = \mathbf{W} x_{ij}$$

**Affine coupling** performs an affine transformation that is conditional on the input. Input is split into 2 parts on the *channel* dimension; *scale* and *bias* of the affine transformation are produced by applying a 3-layer ReLU CNN to the second part of the split; first part of the output is the result of the produced affine transformation applied to first half of the split; second part of the output is the copy of the second part of the input. In the end, the 2 parts of the output are concatenated along the channel dimension. As an affine transformation, it is bijective.

$$(x_a; x_b) = split(x)$$

$$(\hat{s}; t) = \mathbf{NN}(x_b)$$

$$s = sigmoid(\hat{s} + 2)$$

$$y_a = s \odot x_a + t$$

$$y_b = x_b$$

$$y = concat(y_a, y_b)$$

The transformations were picked to have a closed-form formula for the inverse transformation and log-determinant of the Jacobian matrix for the transformation, which can be found in [40].

### 5.1.3 Multi-scale Architecture

The flow steps are further divided into groups. Between the groups, half of the input dimensions along the channel dimension are discarded (during the *inverse* flow, these missing dimensions are sampled). After that, a *squeeze* transformation splits the pixels into $2 \times 2$ regions and reshapes each such region into $1 \times 1 \times 4$ tensor, reducing each *spatial* dimension and increasing the *channel* dimension of the input tensor.

## 5.2 Sampling Efficiency Analysis

Note that a key advantage of the normalizing flow models is that sampling can be done efficiently in the $z$ space. However, as the Jacobian determinants can vary substantially, sampling efficiency in the $z$ space does not translate directly into efficiency of the $x$ space. Here we show the changes of sampling efficiency from the $z$ space to the $x$ space empirically first. Then we provide theoretical explanations of the observed changes.

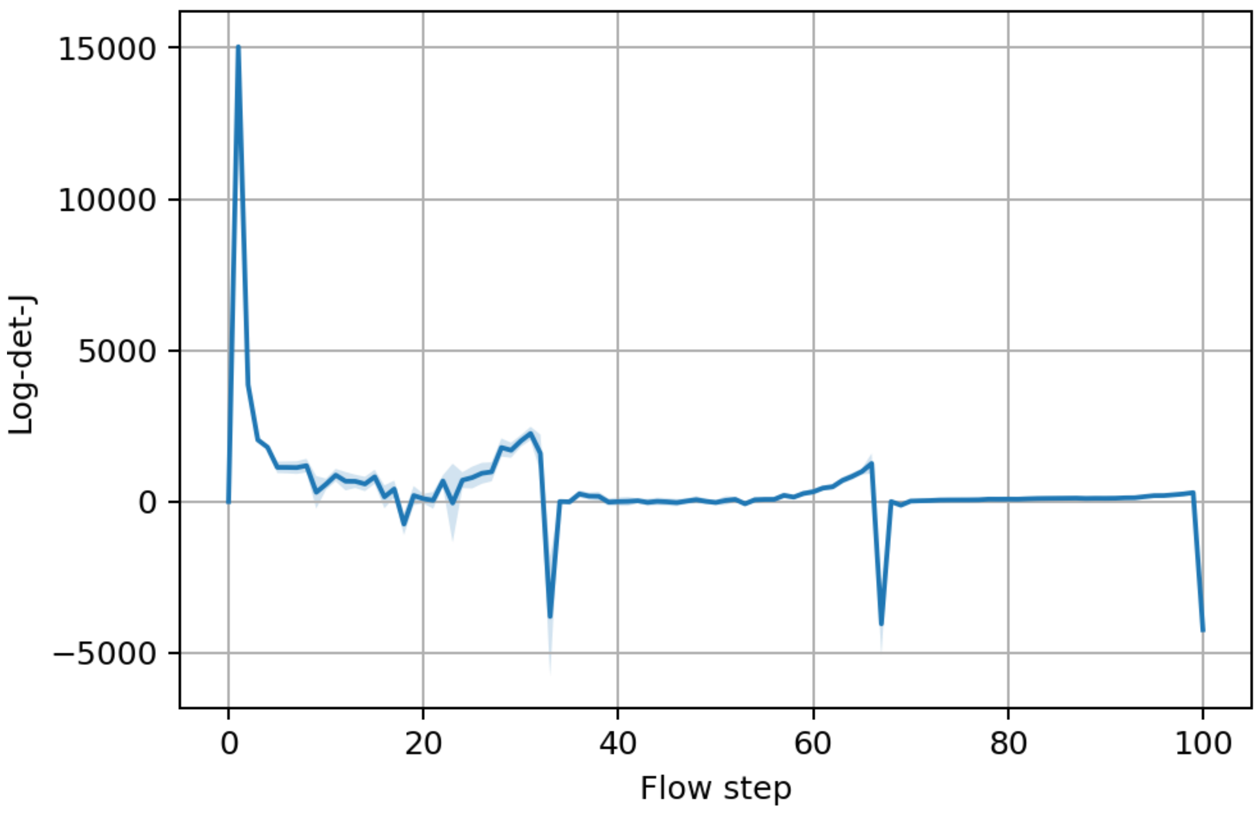

Figure 5.1: Distribution of log-determinant-Jacobian on CelebA dataset measured at each intermediate flow step. Filled area corresponds to span between (mean - 2*variance) and (mean + 2*variance) with mean and variance estimated empirically on the dataset

### 5.2.1 Jacobian Determinants Analysis

Each flow step, being evaluated whether in the normalizing, or in the generative direction, produces 2 results: the output, and the logarithm of the flow step jacobian determinant. The jacobian determinant, from the geometric point of view, denotes the rate of the probability space volume expansion at a given evaluation point. First, we evaluate Glow on a small number of samples from the CelebA dataset, and see that the patterns of volume expansion or shrinking are remarkably similar for the dataset images. We report the aggregate result on the entire dataset in Figure 5.1. From this empirical result we can see that a small number of flow steps contribute a large portion of space expansion, and initial explosive expansion happens in the very first flow step. The major shrinkage, on the other hand, corresponds to the flow steps connecting different scales, that is, the ones discarding half of the input.

### 5.2.2 Discrepancy Between Training Set and Generated Samples

We notice that while the sampled images look photorealistic, there are visual differences between the sampled images and the training set. In this subsection, we quantify the difference between the 2 sets. We compare the sets by computing euclidean norms for each vector, and plotting the histograms for each set. As we see in Figure 5.3, the distributions are distinctly different, the training set norms distribution deviating from the Gaussian bell shape, having different mean and variance.

Alternatively, we collect log-determinant-jacobians for different classes of images. We report the result for log-determinant-jacobians measured on CelebA dataset, on images sampled from Glow with sampling temperature T=0.4, and on images where each pixel is uniformly random generated in Figure 5.2. Each log-determinant-jacobian can be treated as single-value latent representation of an image; the interpretation of this representation is the rate of probability space expansion or shrinkage at the point in space corresponding to the image. We get the following insights from the results of this experiment. First, we note that the distributions for chosen sets of images are drastically different. Alternatively, as larger log-determinant-jacobian values correspond to higher rate of probability space expansion, we see that the sampled images occupy larger region of space than the images from the CelebA dataset that have been used for training the model. Finally, we observe that uniformly generated images have small negative log-determinant-jacobians, so they occupy a very narrow region of probability space, hence the likelihood of uniform noise images being sampled from glow is minuscule.

## 5.3 Biases in the GLOW Model

As generative models are fundamental to classification, regression, and other tasks, understanding their properties is essential in understanding discriminative models that build on them. In recent years, it has been widely recognized that biases are common in deep learning models.

Table 5.1: Confusion matrix for the gender recognition by deepface classifier on a sample from CelebA dataset

| | predicted male | predicted female |
|---|---|---|
| actual male | 1276 | 191 |
| actual female | 14 | 1019 |

Here we analyze the biases of the GLOW model with respect to the protected attributes such as gender, age, race, and other facial features.

In our preliminary experiment, we start with a small sample at a fixed sampling temperature $T = 0.5$. Sampling temperature stands for the variance of images generated in the latent space, and translates into higher variety of images in the original space. The smaller variance leads to generated images having less distinction, since the inputs to the generative pass are all close to the mean image in the latent space, and the entire network represents a continuous function. On the contrary, large variance leads to higher variety of the generated images, however, they start exhibiting artifacts and there is inevitable numerical breakage, as the intermediate representations reach floating point infinity. We pick the sampling temperature $T = 0.5$, so that the generated images exhibit variety, while there are no visible artifacts and intermediate representations remain bounded.

We use sample size of 120 for our manual analysis. The evaluation of the gender attribute showed that the percentage of male images in the generated sample is 64%, while there is only 42% of male images in the CelebA dataset which was used for training.

To facilitate the bias evaluation at a larger scale, we need to automate it, hence we use an external pre-trained classifier. We have used the deepface [69] python package for estimating the percentage of male samples among the generated images. First, we evaluate the trustworthiness of the provided classifier on a representative sample from the CelebA dataset, and obtain the confusion matrix, which we report in Table 5.1. As we observe good performance on real data, we evaluate the gender attribute classification on synthetic images generated with Glow. We sample 2400 images with temperature $T = 0.4$, and observe 2145 samples classified as male and 255 as female, confirming our initial observation of gender bias in sampled images.

The deepface package provides classifiers that can also predict age, race and facial expression; however, the samples from CelebA dataset do not have any attributes that have one to one correspondence to these properties, and it hinders us from evaluating the sampling bias for these protected properties. To that end, we explore another dataset, FairFace [36], where the protected attributes are explicitly specified. While we were able to pre-train Glow on the new dataset, so that photorealistic images could be sampled (see Figure 5.4 for examples), the deepface suite of classifiers does not work well on the Fairface dataset; this leaves the issue of evaluating age, race and facial expression bias for further investigation.

## 5.4 Glow Sampling Controllability

In this section, we investigate a possibility to change high-level features, such as gender, facial expression or pose, of the faces generated with Glow pre-trained on CelebA dataset. We borrow the idea to utilize PCA for editing the face images from [26], who apply this technique to control the features of images generated with GANs.

We start with sampling 14400 images (600 batches with 24 images per batch) at a fixed sampling temperature $T = 0.5$. During sampling, we collect intermediate representations, that is, the outputs of a fixed set of flow steps. Interpreting these outputs as vectors, we perform PCA for each distinct flow step to collect the principal components (unit length vectors, each denoting a direction) and explained variance for each of the components. Then, we perturb the intermediate representations in principal directions with perturbation magnitude depending on the explained variance. Due to large number of variants of perturbations (100 intermediate representations, and *n_pixels* components for each representation), we restrict this study to 10 intermediate representations (flow steps 0 through 90 with the spacing of 10 steps) and 10 principal components exhibiting largest explained variance.

We observe that certain flow steps and respective components allow to control face color, gender, pose angle (yaw), pose angle (pitch), smile, amount of hair, age, hair color, background color, presence of makeup, face shape and lighting. We depict some of the findings in Figure 5.5. We note that in some cases more than one facial property is modified, meaning that a principal direction captures entangled properties. Also, in some cases we were not able to identify the properties being modified with a perturbation, as all images in the series look similar. However, our finding indicates that common techniques could be used to control image generation of *different classes* of generative models, and requires further investigation.

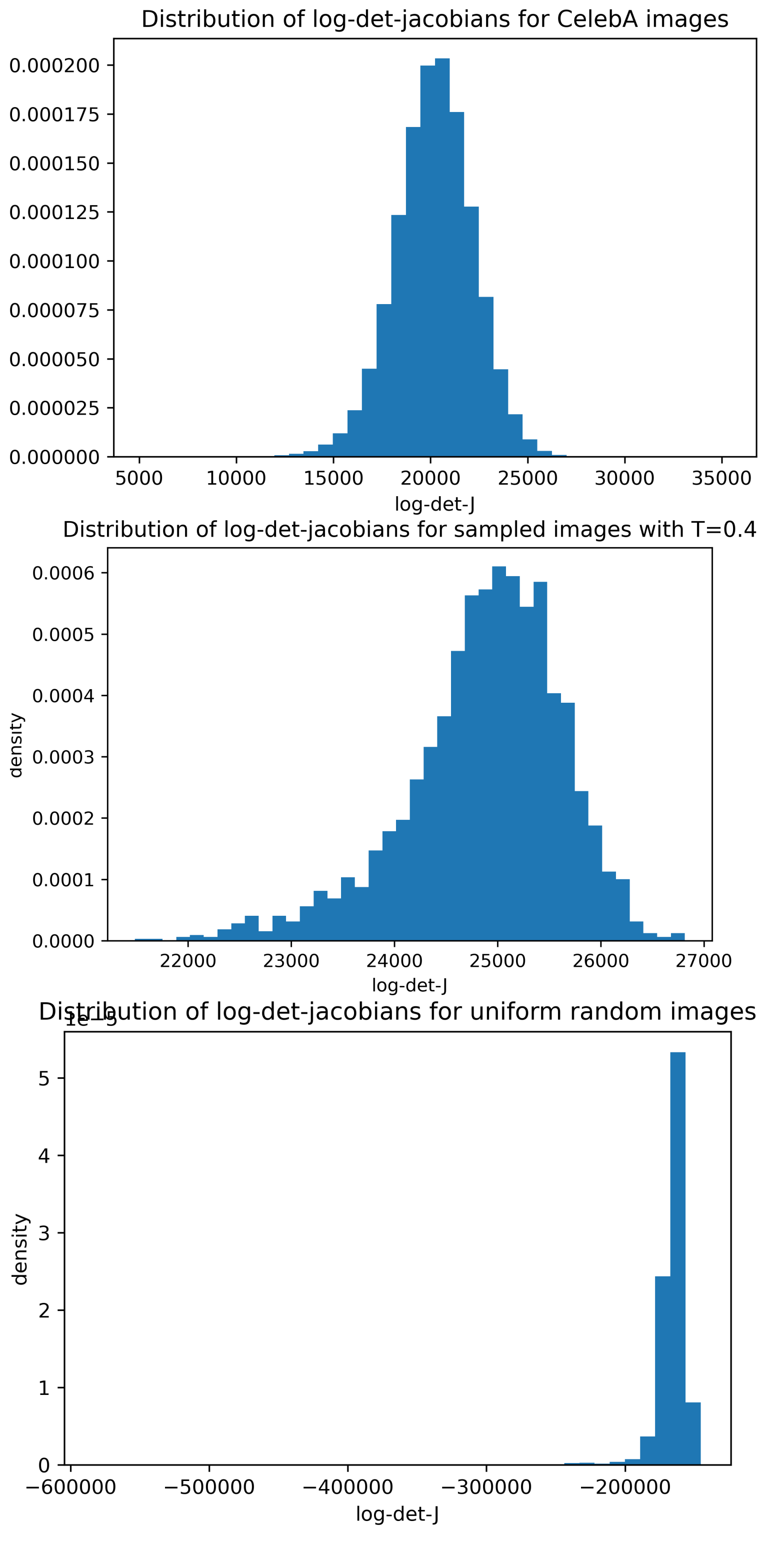

Figure 5.2: Comparison of log-determinant-jacobian distributions for different sets of images.

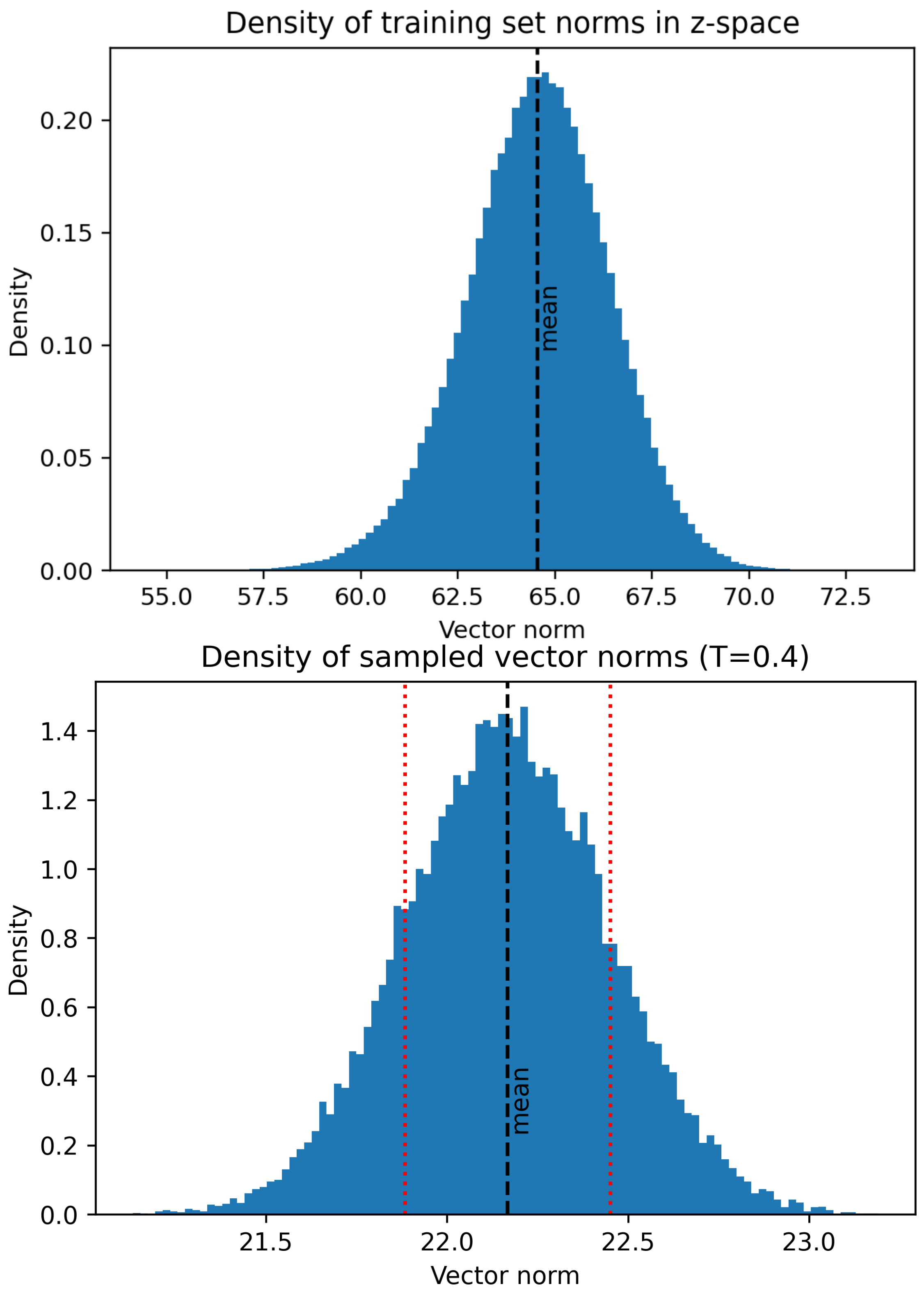

Figure 5.3: Training set representations in the latent space compared to the sampled latent vectors via normalized histograms of their euclidean norms

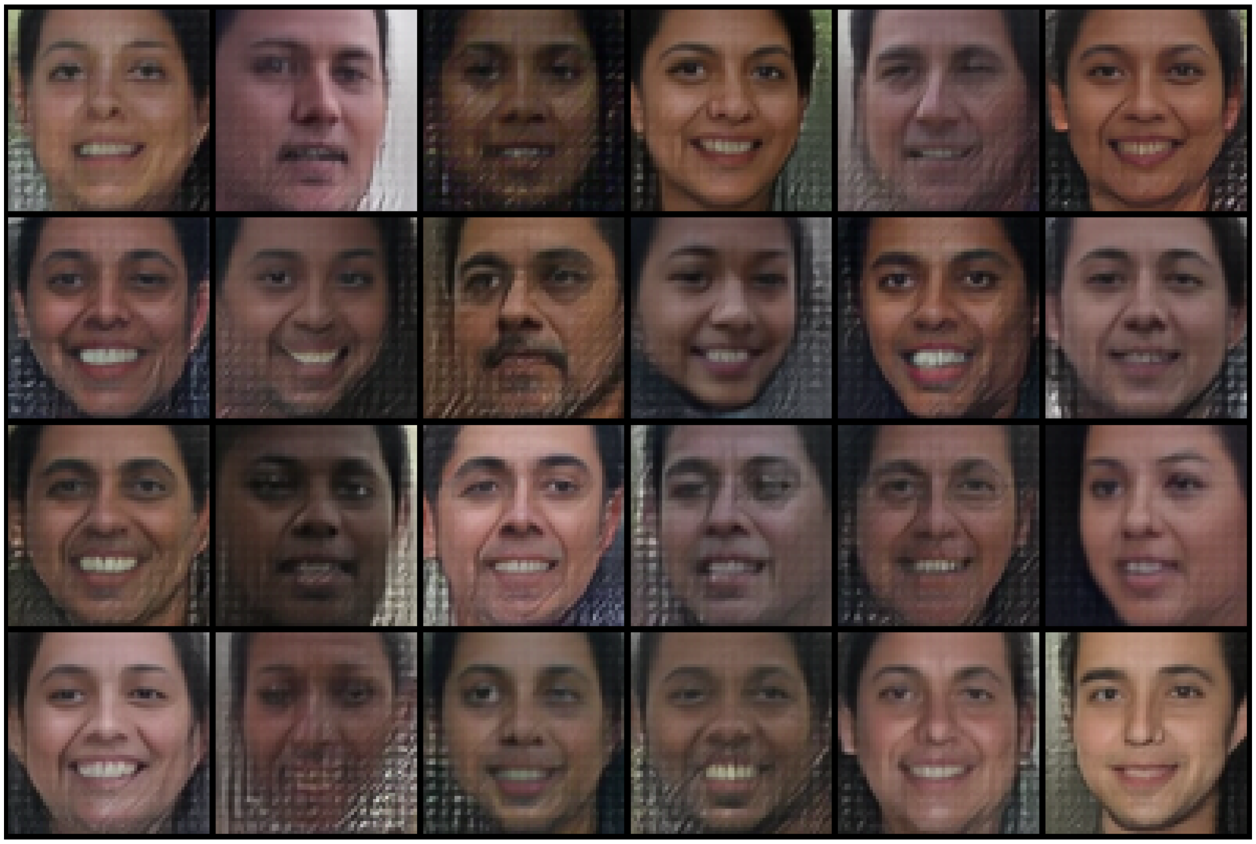

Figure 5.4: A sample of images generated by Glow pre-trained on the FairFace dataset at sampling temperature $T = 0.4$ after 475k minibatch iterations.

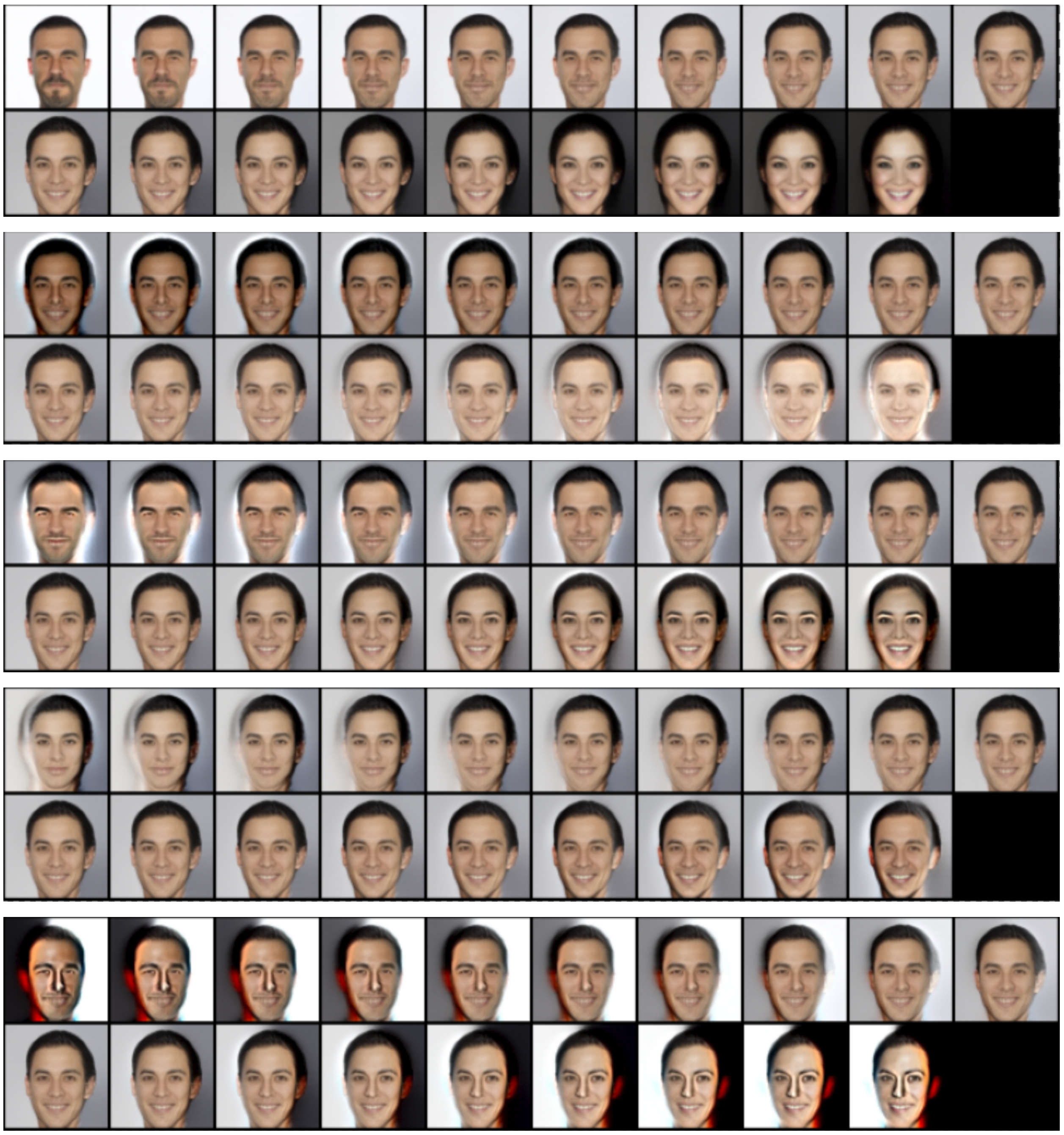

Figure 5.5: Selected results of perturbing the Glow intermediate representations in the principal directions of corresponding layers, showing examples of modifying gender, smile, face color, age and pose.

# CHAPTER 6

# PATH KERNEL ANALYSIS

## 6.1 Introduction

In this chapter, we study an attempt to explain the mechanism of learning that is a property of any model learned with gradient descent. According to [19], any such model is approximately a kernel machine, meaning that for any input sample the prediction done by the network is the kernel interpolation between training samples, where the kernel is defined in terms of an integral over the training path in the parameter space. This idea is coined with the name *path kernel framework*. We have found issues in the proposed derivation, however, we believe the matter deserves to be explored more carefully.

## 6.2 Annotated Derivation

The paper mixes point-free notation where all the elements are functions, with the equations where functions are evaluated at some points, so for clarity it's necessary to separate these 2 notations.

It starts with gradient flow, which is defined as

$$\frac{dw}{dt} = -\nabla_w L_{train}(w(t)) \tag{6.1}$$

where $w$ is the vector of model weights and $L_{train}$ is the total aggregate loss on the training set, that is, sum of losses $l(\cdot)$ evaluated at each individual training sample and corresponding class label.

Then for any function $y$ that depends on $t$ through $w$,

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial y}{\partial w_j} \frac{dw_j}{dt} \tag{6.2}$$

by the chain rule; $d$ is the dimension of vector $w$. In evaluated form, equation 2 looks like

$$\left.\frac{dy}{dt}\right|_{t=\tau, x=\chi} = \sum_{j=1}^{d} \left.\frac{\partial y}{\partial w_j}\right|_{w=w(\tau), x=\chi} \left.\frac{dw_j}{dt}\right|_{t=\tau} \tag{6.3}$$

Combining equations 1 and 2,

$$\frac{dy}{dt} = \sum_{j=1}^{d} \frac{\partial y}{\partial w_j} \frac{dw_j}{dt} = -\sum_{j=1}^{d} \frac{\partial y}{\partial w_j} \frac{\partial L_{train}}{\partial w_j} \tag{6.4}$$

Taking care of evaluated form, that would look like

$$\left.\frac{dy}{dt}\right|_{t=\tau,x=\chi} = -\sum_{j=1}^{d} \left.\frac{\partial y}{\partial w_j}\right|_{w=w(\tau),x=\chi} \left.\frac{\partial L_{train}}{\partial w_j}\right|_{w=w(\tau)} \tag{6.5}$$

The above equations are written with scalar valued $y$ in mind; when $y = (y_1, \ldots, y_c)$, they still hold component-wise, that is,

$$\left.\frac{dy_k}{dt}\right|_{t=\tau,x=\chi} = -\sum_{j=1}^{d} \left.\frac{\partial y_k}{\partial w_j}\right|_{w=w(\tau),x=\chi} \left.\frac{\partial L_{train}}{\partial w_j}\right|_{w=w(\tau)}, \quad \forall k = 1 \ldots c \tag{6.6}$$

The paper then proceeds, applying loss additivity and chain rule,

$$\frac{\partial L_{train}}{\partial w_j} = \sum_{i=1}^{m} \frac{\partial L_{train}}{\partial y_i} \frac{\partial y_i}{\partial w_j} \tag{6.7}$$

**The above statement needs more careful handling** as it reads as taking partial derivative of L with respect to training samples and mixes functions with applications to their arguments. Instead, let us reflect that L is the sum of $m$ loss function evaluations spanning the training set:

$$L_{train}(w) = \sum_{i=1}^{m} l(y)|_{y=y(w,x_i)} \tag{6.8}$$

in point-free notation; both left and right sides of equation are functions of parameter $w$.

Then,

$$\frac{\partial L_{train}}{\partial w_j} = \sum_{i=1}^{m} \frac{\partial \left( l(y)|_{y=y(w,x_i)} \right)}{\partial w_j} = \sum_{i=1}^{m} \frac{\partial \left( l(y)|_{y=y(w,x_i)} \right)}{\partial y} \frac{\partial y}{\partial w_j} \tag{6.9}$$

in case of scalar valued $y$ and in point-free notation. In case $y = (y_1, \ldots, y_c)$,

$$\frac{\partial L_{train}}{\partial w_j} = \sum_{i=1}^{m} \sum_{k=1}^{c} \frac{\partial \left( l(y)|_{y=y(w,x_i)} \right)}{\partial y_k} \frac{\partial y_k}{\partial w_j} \tag{6.10}$$

in point-free notation; in evaluated form it will transform into

$$\left.\frac{\partial L_{train}}{\partial w_j}\right|_{w=\mathbf{w}} = \sum_{i=1}^{m} \sum_{k=1}^{c} \left.\frac{\partial l}{\partial y_k}\right|_{y=y(\mathbf{w},x_i)} \left.\frac{\partial y_k}{\partial w_j}\right|_{w=\mathbf{w},x=x_i} \tag{6.11}$$

Combining that with what we had earlier, $\forall r = 1 \dots c$ :

$$\left.\frac{dy_r}{dt}\right|_{t=\tau, x=\chi} = -\sum_{j=1}^{d}\sum_{i=1}^{m}\sum_{k=1}^{c} \left.\frac{\partial y_r}{\partial w_j}\right|_{w=w(\tau), x=\chi} \left.\frac{\partial l}{\partial y_k}\right|_{y=y(w(\tau), x_i)} \left.\frac{\partial y_k}{\partial w_j}\right|_{w=w(\tau), x=x_i} \tag{6.12}$$

The paper then fast-forwards to the final formulation in terms of integral, which is (doing one step backward),

$$y(t) = y(t_0) + \int\limits_{s(t)} y'(t)dt \tag{6.13}$$

**Which looks a bit odd**, since (1) $t$ is the integration parameter and $s(t)$ is the curve weights travel during training (so $s(t) \equiv w(t)$ ?) and (2) by fundamental theorem of calculus, we have

$$y(t) = y(t_0) + \int\limits_{t_0}^{t} y'(\tau)d\tau \tag{6.14}$$

Let us try to work around this oddity. When we try to compute the last integral numerically, using some approximation formula similar to rectangle rule

$$\int\limits_{a}^{b} f(t)dt \approx \sum_{t_0=a}^{t_n=b} f(t_i)(t_{i+1} - t_i) \tag{6.15}$$

the practical problem is, $t$ is an implicit parameter while we only have access to model's weights after each update. Luckily, $y'(t)$ was expressed purely in terms of weights in Equation 12. We can also approximate

$$w(t_{i+1}) - w(t_i) \approx w'(t_i)(t_{i+1} - t_i) \tag{6.16}$$

$$t_{i+1} - t_i \approx \texttt{avg}\big((w(t_{i+1}) - w(t_i)) \oslash w'(t_i)\big) \tag{6.17}$$

note that $w$ and $w'$ are vectors of dimension $d$, so the "division" must happen elementwise and averaged across $d$ vector elements.

Also remember that we started with gradient flow, and evaluating it at one point will yield

$$w'(t_i) = -\nabla_w L_{train}(w(t_i)) = -\sum_{j=1}^{m} \left.\frac{\partial l(y)}{\partial w}\right|_{y=y(w(t_i), x_j)} \tag{6.18}$$

At this point, we can compute the approximation purely in terms of weights:

$$y_r(t)|_{x=\chi} \approx y_r(t_0)|_{x=\chi} +$$
$$+\sum_{\tau}\left(\sum_{j=1}^{d}\sum_{i=1}^{m}\sum_{k=1}^{c}\left.\frac{\partial y_r}{\partial w_j}\right|_{w=w(\tau),x=\chi}\left.\frac{\partial l}{\partial y_k}\right|_{y=y(w(\tau),x_i)}\left.\frac{\partial y_k}{\partial w_j}\right|_{w=w(\tau),x=x_i}\right)\cdot$$
$$\cdot\texttt{avg}\left((w(\tau+1)-w(\tau))\oslash\sum_{j=1}^{m}\left.\frac{\partial l(y)}{\partial w}\right|_{y=y(w(\tau),x_j)}\right)$$

The triple sum looks like a tensor product,

$$\sum_{j=1}^{d}\sum_{i=1}^{m}\sum_{k=1}^{c}\left.\frac{\partial y_r}{\partial w_j}\right|_{w=w(\tau),x=\chi}\left.\frac{\partial l}{\partial y_k}\right|_{y=y(w(\tau),x_i)}\left.\frac{\partial y_k}{\partial w_j}\right|_{w=w(\tau),x=x_i}=$$
$$=\sum_{i=1}^{m}\sum_{k=1}^{c}\left\langle \nabla_w y_r|_{w=w(\tau),x=\chi}, \nabla_w y_k|_{w=w(\tau),x=x_i}\right\rangle\left.\frac{\partial l}{\partial y_k}\right|_{y=y(w(\tau),x_i)}=\dots$$

Using the equivalence

$$\langle Ab,c\rangle=\sum_j (Ab)_j c_j=\sum_{j,k}A_{jk}b_k c_j=\sum_k b_k\sum_j A_{jk}c_j=\sum_k\langle c,A_k\rangle b_k \tag{6.19}$$

where $A_k$ is the $k$-th column of matrix $A$; also using the definition of jacobian matrix

$$(\mathbf{J}_w\mathbf{y})_{ij}=\frac{\partial y_i}{\partial w_j} \tag{6.20}$$

and using $A=\mathbf{J}_w^T\mathbf{y}\big|_{w=w(\tau),x=x_i}$, $b=\nabla_y\, l(y)|_{y=y(w(\tau),x_i)}$, $c=\nabla_w y_r|_{w=w(\tau),x=\chi}$ in the above equivalence, the triple sum transforms into

$$\dots=\sum_{i=1}^{m}\left\langle \nabla_w y_r|_{w=w(\tau),x=\chi}, \mathbf{J}_w^T\mathbf{y}\big|_{w=w(\tau),x=x_i}\cdot\nabla_y\, l(y)|_{y=y(w(\tau),x_i)}\right\rangle=$$
$$=\left\langle \nabla_w y_r|_{w=w(\tau),x=\chi}, \sum_{i=1}^{m}\mathbf{J}_w^T\mathbf{y}\big|_{w=w(\tau),x=x_i}\cdot\nabla_y\, l(y)|_{y=y(w(\tau),x_i)}\right\rangle$$

Finally, we can get rid of $y_r$ component index, since only the left dot product component depends on it

$$y(t)|_{x=\chi}\approx y(t_0)|_{x=\chi}+$$
$$+\sum_{\tau}\left(\mathbf{J}_w\mathbf{y}|_{w=w(\tau),x=\chi}\right)\sum_{i=1}^{m}\left(\mathbf{J}_w^T\mathbf{y}\big|_{w=w(\tau),x=x_i}\right)\left(\nabla_y\, l(y)|_{y=y(w(\tau),x_i)}\right)\cdot$$

$$\cdot\texttt{avg}\left((w(\tau+1)-w(\tau))\oslash\sum_{j=1}^{m}\left.\frac{\partial l(y)}{\partial w}\right|_{y=y(w(\tau),x_j)}\right)$$

- Is the above derivation correct?
- Could the matrix-vector product inside $\sum\limits_{i=1}^{m}$ be simplified (chain rule)? As in

  $$\left(\mathbf{J}_w^T\mathbf{y}|_{w=w(\tau),x=x_i}\right)\left(\nabla_y\ l(y)|_{y=y(w(\tau),x_i)}\right)=\nabla_w l|_{y=y(w(\tau),x_i)}$$

  In this case, the derived formula folds down even further:

  $$y(t)|_{x=\chi}\approx y(t_0)|_{x=\chi}+\sum_{\tau}\left(\mathbf{J}_w\mathbf{y}|_{w=w(\tau),x=\chi}\right)\vec{\mathbf{1}_d}\,\texttt{avg}\,(w(\tau+1)-w(\tau))\qquad(*)$$

  , where $\vec{\mathbf{1}_d}$ is a $d$-dimensional vector of all ones. But the training samples completely disappear from the equation, so does the "kernel", as there is no interpolation between the training samples.

  *Remark:* The training samples are still *implicitly* present through the summation over $\tau$ as this parameter governs the weights evolution during training, so $\tau$ must depend on the training set.

- It seems that most of the unfolding (introducing loss partial derivatives) and folding back could be skipped; that is,

  $$y(t)=y(t_0)+\int_{t_0}^{t}y'(\tau)d\tau\approx y(t_0)+\sum_{\tau}y'(\tau)\Delta\tau=y(t_0)+\sum_{\tau}\mathbf{J}_w y\,w'(\tau)\Delta\tau\approx$$

  $$\approx y(t_0)+\sum_{\tau}\mathbf{J}_w y\,\Delta w$$

  Or, in evaluated form,

  $$y(t)|_{x=\chi}\approx y(t_0)|_{x=\chi}+\sum_{\tau}\mathbf{J}_w y|_{w=w(\tau),x=\chi}\,(w(\tau+1)-w(\tau))\qquad(**)$$

  Which looks roughly similar to the result (*)

## 6.3 Challenges

The major issue with the derivation is in the kernel form, the coefficients and bias are dependent on the input, making the result less exciting than it looks. For a minor issue, we note that if we try to compute the coefficients for a given input, aiming to find the similarity between the input and the training samples, due to approximating the integral numerically, any variable associated with the training samples disappears from the formula for the kernel coefficients. Overcoming these challenges is a matter of future investigation.

# CHAPTER 7

# CONCLUSION AND FUTURE WORK

In this dissertation, we have investigated intrinsic properties and biases of two deep learning models, BERT in the domain of natural language processing and Glow in the domain of computer vision. While the domains are drastically different, we find that common techniques summarizing distributions of intermediate representations can be applied for gaining new knowledge about mechanisms underlying deep learning models, as we show in two concrete examples.

In chapter 2 [60], we analyze the geometry of BERT embeddings and the intrinsic discriminability of the context-dependent representations obtained from intermediate layers of the BERT transformer stack. We observe that BERT word embeddings have drifted in the common direction during pre-training. We quantify the ability of BERT word embeddings to capture the word semantics based on similarity query and observe quality similar to the word embeddings from the skip-gram model. Our results show that the intermediate representations do not cluster based on emotion classification, however, these representations may be used as input features for relatively unsophisticated classifiers to achieve good emotion classification accuracy. Somewhat surprisingly, we discover that the representations from intermediate layers are better suited than the final representations as inputs for the emotion classification, which implies that the deeper layers learn features less applicable for this particular task.

In Chapter 3 [59], we have analyzed the language modeling head to identify the component responsible for masked token reconstruction. Then, we systematically have probed BERT token reconstruction ability under syntactic changes; we find the predictions are not robust and show that incorrect predictions are likely to be attributed to their statistically high co-occurrence in the text dataset used for training, which may pose a challenge for knowledge retrieval system implementers who use pre-trained BERT as source of knowledge. We have also adapted a common misconception infographic to design a dataset that can be used to systematically measure how language models are prone to selecting wrong answers when filling blanks in a sentence that contains a part of a misconception.

In Chapter 4, we have explored prospects of treating BERT as a language model, i.e. assigning a probability score for any token sequence that such sequence may appear in the natural text. We have computed the language models inherent to BERT based on prior distributions estimated as natural text n-gram frequencies for two distinct datasets. Our results show there are systematic biases inherent to the BERT model. We have extended the language model with a contextual language model that may be used to systematically evaluate gender and other biases, going beyond existing prompt based bias evaluation methods.

In Chapter 5, we have analyzed the inner workings of Glow, a generative model in the domain of computer vision based on discrete normalizing flows, and characterized impacts of different flow steps on the probability density assigned to images. We have uncovered gender biases in the synthesized images. We have discovered a way to control high level features of the images synthesized with Glow by manipulating the intermediate representations adding vectors proportional to principal directions.

In Chapter 6, we have analyzed a recent theoretical result about approximation with a kernel machine of *any* machine learning model having used gradient descent for learning, and highlighted issues that limit the practical applications of this result, namely, the coefficients being dependent on a particular input and computation impossibility of the coefficients due to numerical approximation of the path integral with a discrete sum.

For BERT, a possible extension of our work would apply the language model to the tasks where language models are typically used, e.g. OCR or speech synthesis. Another important issue is not only evaluation of biases but providing means to alleviate them, augmenting the existing pre-trained model. For Glow, we believe there are deep connections between normalizing flows and other classes of generative models, such as GANs, as implied by our study of high level features controllability, and these connections need to be studied in order to understand mechanisms enabling synthesis of photorealistic images.

# APPENDIX A

# LANGUAGE MODELING HEAD COMPONENT ACTIVATIONS

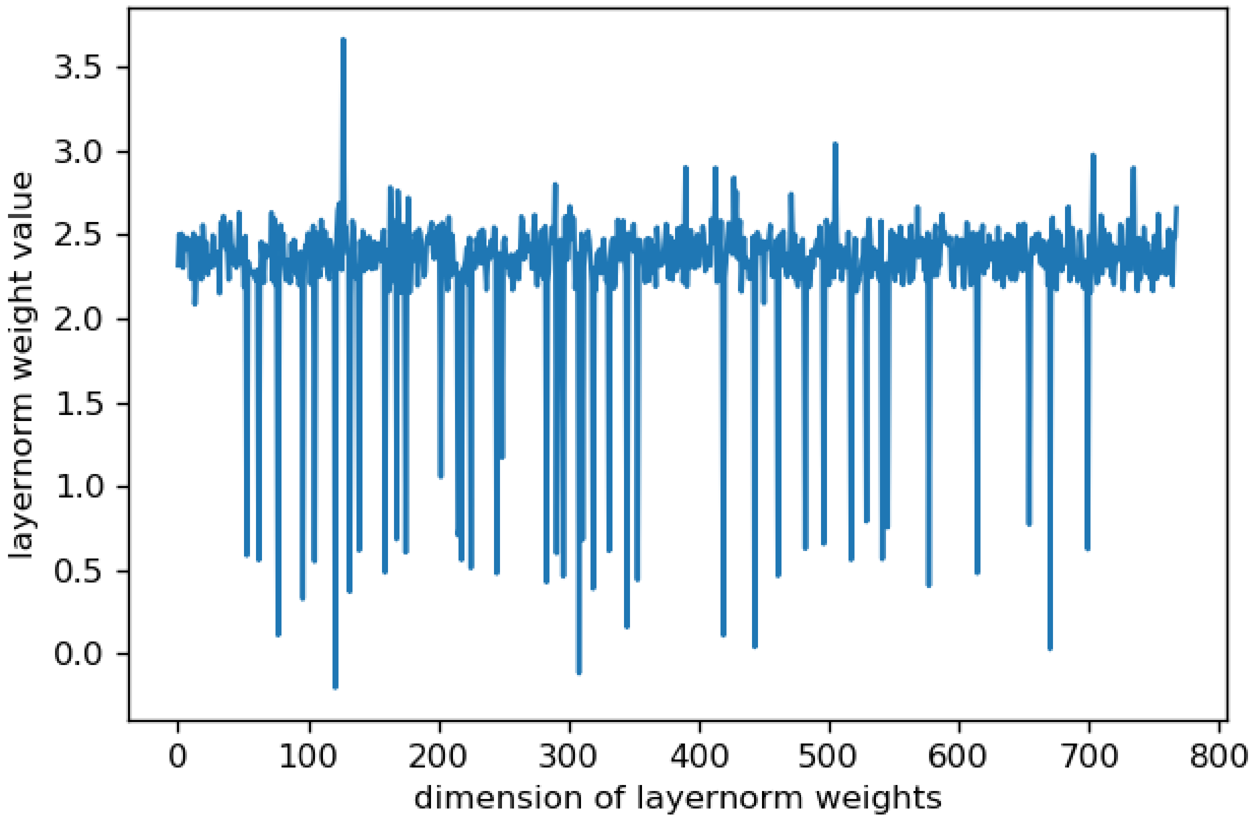

Figure A.1: LayerNorm weights in BERT base uncased.

In this section, we will provide additional detail for the experiment with comparing activation ranks within the LM head. Figure A.1 and Figure A.2 contain LayerNorm weights and biases accordingly. Figure A.3 is similar to Figure 3.3, but lists all of the LM head components. Reiterating, we consider a single input token sequence, where one of the tokens is '[MASK]'; to find the contribution of individual LM head components to the token reconstruction ability, we consequently truncate the LM head *encoder* at each component, computing the inner products between (1) output of the truncated models and (2) word embeddings; thus getting the vocabulary token

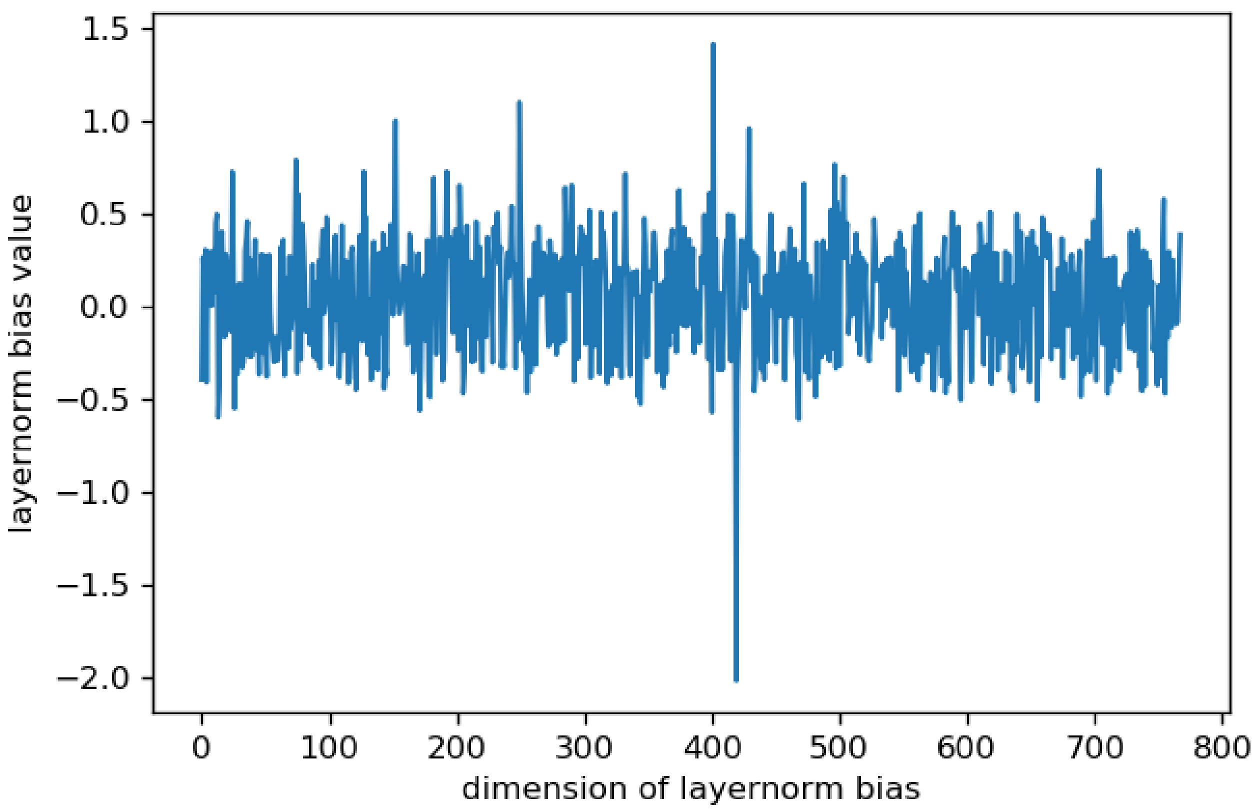

Figure A.2: LayerNorm bias in BERT base uncased.

scores. Then we compare these scores to the scores produced by the entire LM head, sorting both score vectors by the elements of the score vector produced by the truncated model. Since we can truncate the model (1) at the transformer stack, (2) at the dense weights in LM head, (3) at the dense bias in LM head, (4) at GELU in LM head, (5) at LayerNorm normalization in LM head, (6) at LayerNorm elementwise weight multiplication and (7) at LayerNorm elementwise bias addition, we have a total of 7 plots here. Since we sort one sequence by elements of the other one, there are 2 lines in each plot, the blue monotonic line corresponding to the sequence used for sorting, and orange one corresponding to the sorted sequence. We also vary this experiment, swapping the sorted sequences (Figure A.5 contains a column of LM head outputs sorted by component-produced outputs on the left, and a column of component-produced outputs sorted by LM head outputs on the right), and using transformer stack output instead of LM head output to produce a pair of columns in Figure A.4.

Figure A.6 decomposes Figure 3.1 into distinct plots for each sample perturbation strategy (final '.' removal, '[SEP]' removal, replacing of the '[MASK]' token), increasing the visibility of differences

Table A.1: Pairwise Spearman's rank-order correlations (transformer stack output, dense weight output, dense bias output, gelu output, layernorm normalization output, layernorm elementwise multiplication output, layernorm elementwise bias addition output, LM head final output).

| | trf-out | dense-w | dense-b | gelu | lrnorm-n | lrnorm-w | lrnorm-b | out |
|---|---|---|---|---|---|---|---|---|
| trf-out | 1 | 0.126 | 0.124 | 0.064 | 0.089 | 0.220 | 0.257 | 0.257 |
| dense-w | 0.126 | 1 | 0.999 | 0.712 | 0.428 | 0.130 | 0.326 | 0.367 |
| dense-b | 0.124 | 0.999 | 1 | 0.720 | 0.428 | 0.122 | 0.318 | 0.36 |
| gelu | 0.064 | 0.712 | 0.720 | 1 | 0.078 | -0.042 | 0.189 | 0.199 |
| lrnorm-norm | 0.089 | 0.428 | 0.428 | 0.078 | 1 | 0.695 | 0.660 | 0.682 |
| lrnorm-w | 0.220 | 0.130 | 0.122 | -0.042 | 0.695 | 1 | 0.946 | 0.936 |
| lrnorm-b | 0.257 | 0.326 | 0.318 | 0.189 | 0.660 | 0.946 | 1 | 0.996 |
| out | 0.257 | 0.367 | 0.360 | 0.199 | 0.682 | 0.936 | 0.996 | 1 |

between the plots but also taking much more space.

Table A.2 shows the first few most frequent tokens reconstructed in a probe where the '[MASK]' token is being substituted by each token in the vocabulary. As we discover, the factually correct answer (**florence**) is *not* the most popular one, and we can see many tokens in this list have a common *theme* (Italy). Also, in the substantial majority of cases, the replaced token reconstructs itself, thus making queries fragile.

Table A.1 contains the pairwise Spearman's rank-order correlations for the sequences plotted in Figure A.3. The last line of the table contains pairwise correlations between LM head output and sequences produced by each of the truncated models; we can see a significant correlation increase after the LayerNorm normalization layer, becoming even more apparent after the elementwise multiplication.

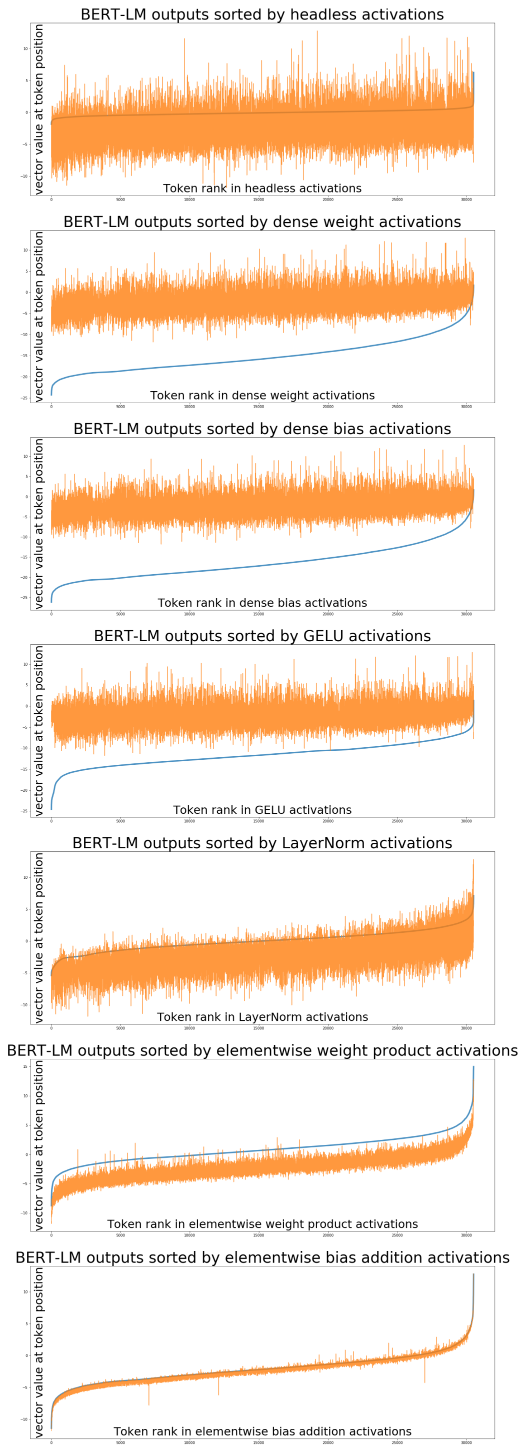

Figure A.3: BERT with LM head outputs sorted by inner products of token embeddings with each of the LM head activations (from top to bottom: encoder stack, dense weight, dense bias, GELU, LayerNorm, elementwise multiplication ($\otimes$) and addition ($\oplus$)). The plots visually align after the LayerNorm layer, and it coincides with reconstruction quality increase that we observe.

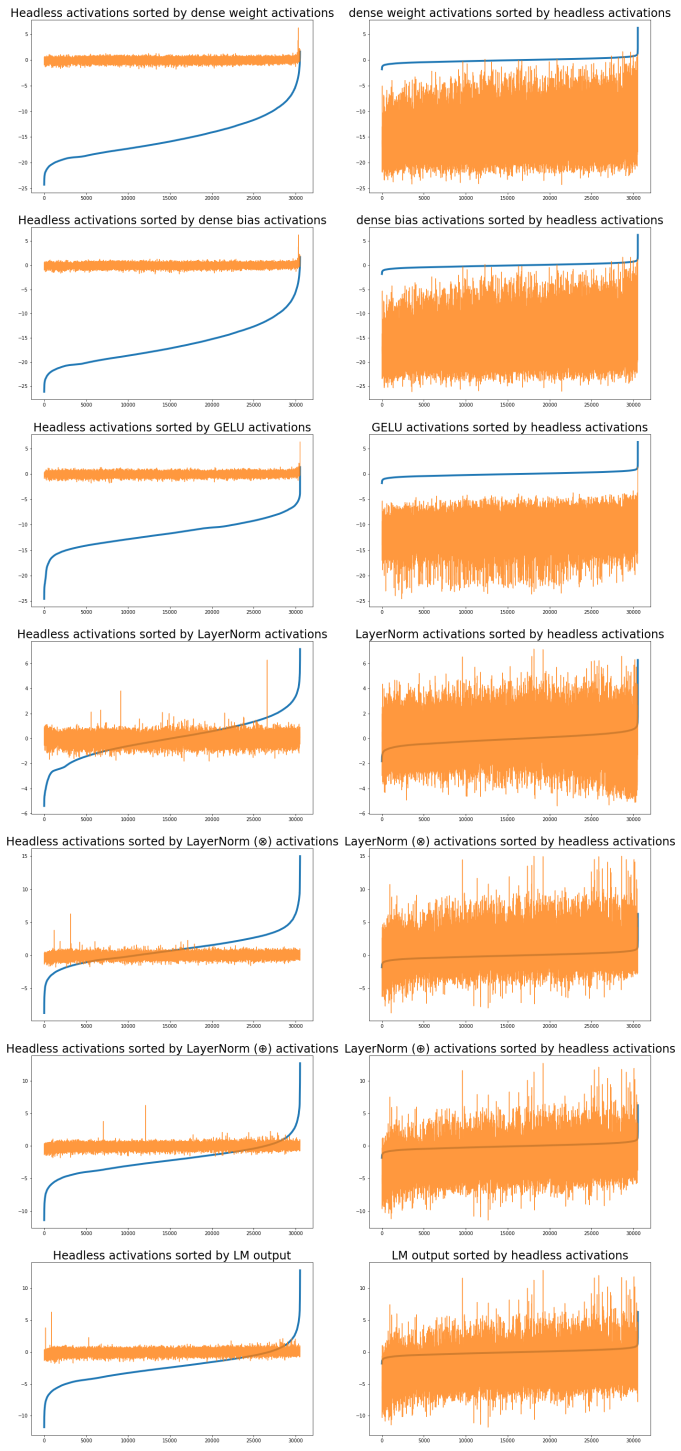

Figure A.4: LM head components compared to transformer output.

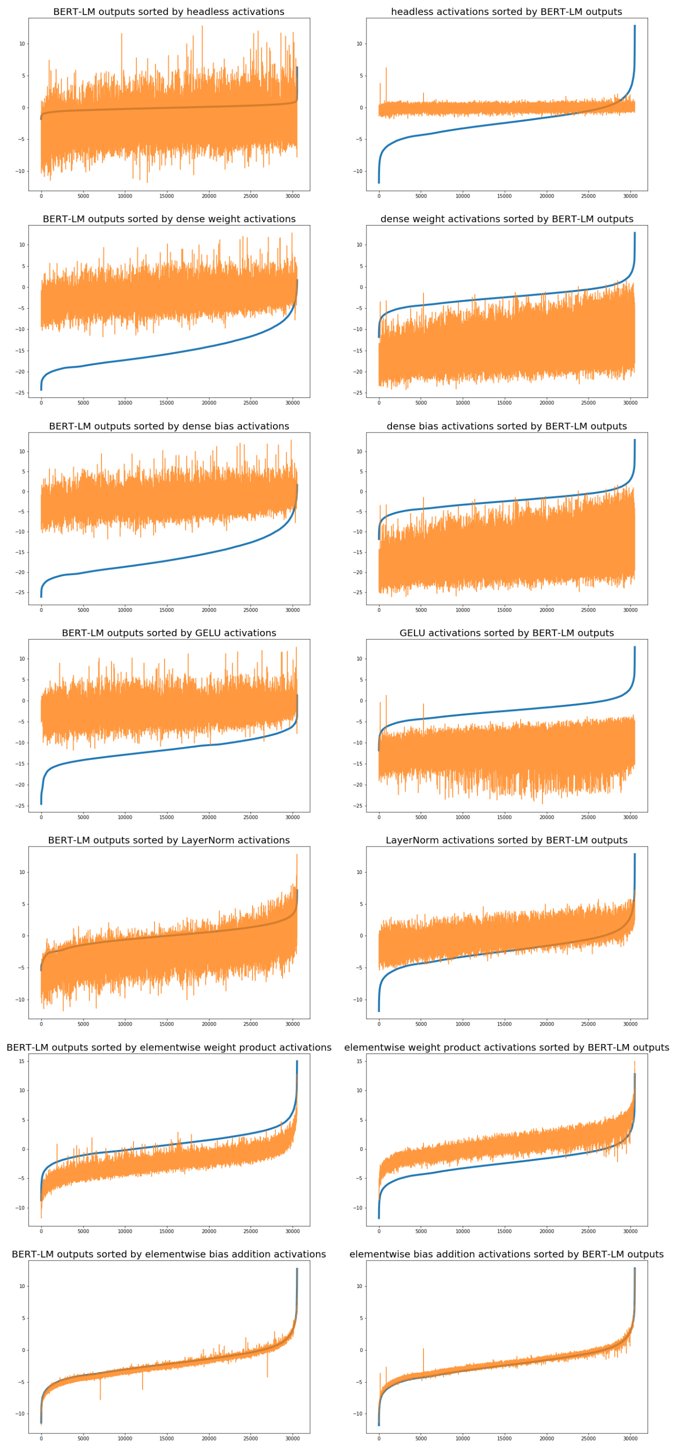

Figure A.5: LM head components compared to LM head output.

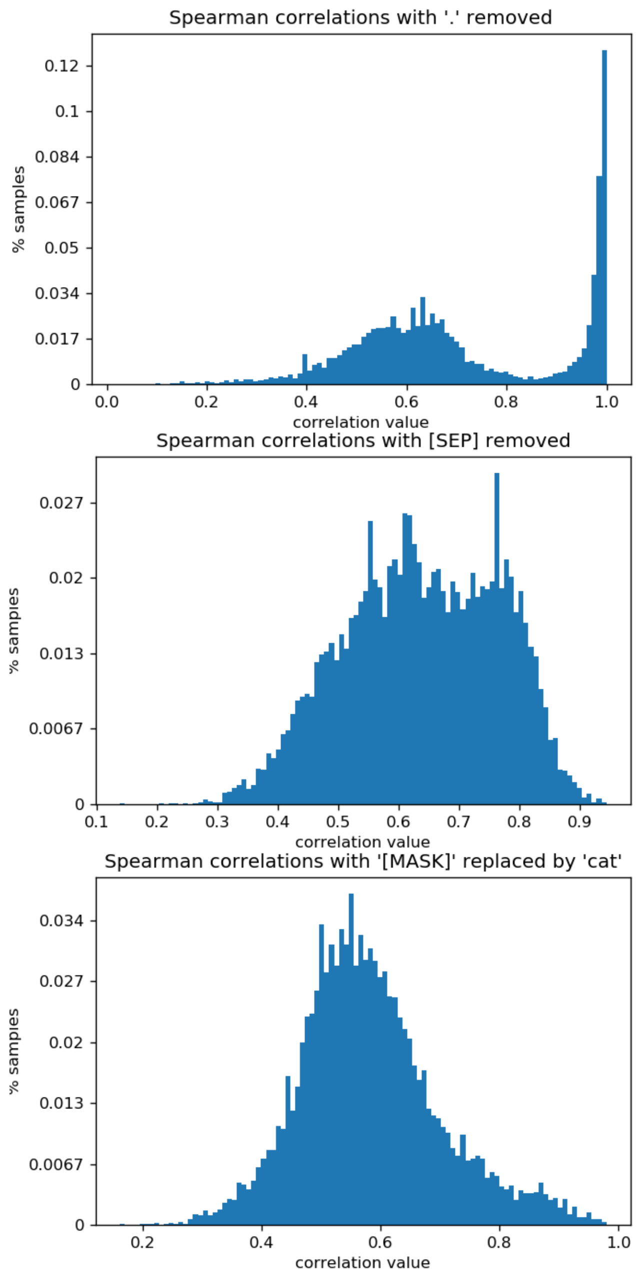

Figure A.6: Histogram of Spearman's rank-order correlations between logits of the original sample and logits of a perturbed sample, collected over entire ConceptNet dataset used in LAMA. In the top figure, we remove the '.' token ending the sample sentence or sentences; in the middle one – we remove the [SEP] token ending the token sequence serving as input to BERT; in the bottom one – we replace the [MASK] token with the 'cat' one. The decrease in reconstruction quality between perturbation strategies is reflected in the correlations decrease as the bell curve shifts to the left, if we compare the plots from top to bottom.

Table A.2: Comprehensive token substitution exploration. We take the sentence *'Francesco Bartolomeo Conti was born in X.'*, substitute X with each token in the vocabulary, and ask BERT with a LM head to guess which original token was substituted. In the majority of cases, X and predicted token turn out to be the same. In the remaining 7608 out of 30522 cases, they are not the same, and the most frequently predicted tokens in this group are in the table above. We can see that the Italian cities prevail, and the ground truth answer from the world knowledge point of view (Florence) has a smaller number of predictions than a few other tokens.

| Token | Prediction count |
|---|---|
| rome | 2519 |
| bologna | 685 |
| genoa | 586 |
| ca | 391 |
| verona | 363 |
| como | 320 |
| **florence** | **273** |
| c | 186 |
| venice | 152 |
| town | 95 |
| padua | 81 |
| villa | 75 |
| ferrara | 69 |
| st | 56 |
| the | 51 |
| turin | 50 |
| est | 49 |
| alexandria | 43 |
| pisa | 42 |
| it | 38 |
| po | 37 |
| milan | 36 |
| san | 36 |
| here | 35 |
| UNK | 31 |
| val | 30 |
| . | 29 |
| this | 27 |
| palazzo | 26 |
| e | 25 |
| italy | 19 |

# APPENDIX B

# HEADLESS BERT'S TOKEN RECONSTRUCTION CAPABILITY

During exploration of the reconstruction capabilities of uncased base BERT without LM head (that is, embedding layer followed by 12 transformer layers), first, we have tried the reconstruction itself. We use the sentence *'[CLS] london is the capital of great britain. [SEP]'* as input and notice that after 1 iteration of reconstruction, the token sequence is transformed into *('[CLS]', '[CLS]', '[CLS]', '[CLS]', 'capital', '[CLS]', 'ned', '[CLS]', '##ann', '##vis')*, and on subsequent iterations, the token sequence converges to a fix-point *('[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]', '[CLS]')*. Then, we have tried rescaling the token's embeddings, so that new embedding of '[CLS]' token has the magnitude of original 'the' token's embedding, and vice versa. After rescaling, we use the same sentence as input and notice that after 1 iteration of reconstruction, the token sequence is transformed into *('##city', 'london', 'the', 'the', 'capital', 'the', 'ned', 'britain', '##ann', '##vis')*, and on subsequent iterations, the token sequence converges to a fix-point *('the', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'the')*. We note that after swapping magnitudes, the 'the' token has taken the place of '[CLS]' token as being the point of convergence.

# BIBLIOGRAPHY

[1] Martín Abadi et al. "TensorFlow: A system for large-scale machine learning." In: *ArXiv* abs/1605.08695 (2016).

[2] Jaimeen Ahn and Alice Oh. "Mitigating Language-Dependent Ethnic Bias in BERT." In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 533–549.

[3] Emily Alsentzer et al. "Publicly available clinical BERT embeddings." In: *arXiv:1904.03323*. 2019.

[4] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization." In: 2016. arXiv: 1607.06450.

[5] Marion Bartl, Malvina Nissim, and Albert Gatt. "Unmasking Contextual Stereotypes: Measuring and Mitigating BERT's Gender Bias." In: *ArXiv* abs/2010.14534 (2020).

[6] Yoshua Bengio et al. "A Neural Probabilistic Language Model." In: *J. Mach. Learn. Res.* 2003.

[7] Rishabh Bhardwaj, Navonil Majumder, and Soujanya Poria. "Investigating Gender Bias in BERT." In: *Cogn. Comput.* 13 (2021), pp. 1008–1018.

[8] Daniel Bis, Maksim Podkorytov, and Xiuwen Liu. "Too Much in Common: Shifting of Embeddings in Transformer Language Models and its Implications." In: *NAACL*. 2021.

[9] Kay Henning Brodersen et al. "The balanced accuracy and its posterior distribution." In: *ICPR*. 2010.

[10] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[11] Gino Brunner et al. "On Identifiability in Transformers." In: *arXiv:1908.04211*. 2019.

[12] F ChenStanley and GoodmanJoshua. "An empirical study of smoothing techniques for language modeling." In: *Computer Speech & Language* (1999).

[13] Kevin Clark et al. "ELECTRA: Pre-Training Text Encoders As Discriminators Rather Than Generators." In: *arXiv:2003.10555*. 2020.

[14] Kevin Clark et al. "What Does BERT Look At? An Analysis of BERT's Attention." In: *arXiv:1906.04341*. 2019.

[15] Zihang Dai et al. "Transformer-xl: Attentive language models beyond a fixed-length context." In: *arXiv:1901.02860*. 2019.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *NAACL-HLT*. 2019.

[17] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. 2015. arXiv: `1410.8516 [cs.LG]`.

[18] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP*. 2017. arXiv: `1605.08803 [cs.LG]`.

[19] Pedro Domingos. *Every Model Learned by Gradient Descent Is Approximately a Kernel Machine*. 2020. arXiv: `2012.00152 [cs.LG]`.

[20] Jiaju Du, Fanchao Qi, and Maosong Sun. "Using BERT for Word Sense Disambiguation." In: *arXiv:1909.08358*. 2019.

[21] Kawin Ethayarajh. "How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings." In: *EMNLP/IJCNLP*. 2019.

[22] Allyson Ettinger. "What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models." In: 2019. arXiv: `1907.13528`.

[23] Yoav Goldberg. "Assessing BERT's Syntactic Abilities." In: *arXiv:1901.05287*. 2019.

[24] Ian J. Goodfellow et al. "Generative Adversarial Nets." In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[25] Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. *Exposing the Implicit Energy Networks behind Masked Language Models via Metropolis–Hastings*. 2021. arXiv: `2106.02736 [cs.LG]`.

[26] Erik Härkönen et al. "GANSpace: Discovering Interpretable GAN Controls." In: *ArXiv* abs/2004.02546 (2020).

[27] Pengcheng He et al. "DeBERTa: Decoding-enhanced BERT with Disentangled Attention." In: *ArXiv* abs/2006.03654 (2021).

[28] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)." In: 2016. arXiv: `1606.08415`.

[29] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: `10.1162/neco.2006.18.7.1527`. URL: `https://doi.org/10.1162/neco.2006.18.7.1527`.

[30] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation*. 1997.

[31] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. “ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission.” In: *arXiv:1904.05342*.

[32] Arthur Jacot, Franck Gabriel, and Clément Hongler. “Neural tangent kernel: convergence and generalization in neural networks (invited paper).” In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2018).

[33] Zhengbao Jiang et al. “How Can We Know What Language Models Know?” In: 2019. arXiv: `1911.12543`.

[34] Norman P. Jouppi et al. “In-datacenter performance analysis of a tensor processing unit.” In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (2017), pp. 1–12.

[35] J. Jumper et al. “Highly accurate protein structure prediction with AlphaFold.” In: *Nature* (2021).

[36] Kimmo Kärkkäinen and Jungseock Joo. “FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age for Bias Measurement and Mitigation.” In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2021), pp. 1547–1557.

[37] Henry J Kelley. “Gradient theory of optimal flight paths.” In: *Ars Journal* 30.10 (1960), pp. 947–954.

[38] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization.” In: *arXiv:1412.6980*. 2014.

[39] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: `1312.6114 [stat.ML]`.

[40] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: `1807.03039 [stat.ML]`.

[41] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations.” In: *arXiv:1909.11942*. 2019.

[42] Vladimir I Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals.” In: *Soviet physics doklady. Vol. 10. No. 8.* 1966.

[43] Xiuwen Liu, Anuj Srivastava, and Kyle A. Gallivan. “Optimal linear representations of images for object recognition.” In: *CVPR*. 2003.

[44] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach.” In: 2019. arXiv: `1907.11692`.

[45] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild.” In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[46] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization." In: 2017. arXiv: `1711.05101`.

[47] Matt Mahoney. *Text8 dataset*. 2006. URL: `http://mattmahoney.net/dc/textdata.html`.

[48] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[49] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality." In: *NIPS*. 2013.

[50] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space." In: *ICLR*. 2013.

[51] Tomas Mikolov et al. "Recurrent neural network based language model." In: *INTERSPEECH*. 2010.

[52] Timothy Niven and Hung-Yu Kao. "Probing neural network comprehension of natural language arguments." In: *arXiv:1907.07355*. 2019.

[53] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *NeurIPS*. 2019.

[54] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space." In: *Philosophical Magazine*. 1901.

[55] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." In: *EMNLP*. 2014.

[56] Matthew Peters et al. "Deep Contextualized Word Representations." In: *NAACL-HLT*. 2018.

[57] Fabio Petroni et al. "How Context Affects Language Models' Factual Predictions." In: arXiv: `2005.04611`.

[58] Fabio Petroni et al. "Language Models as Knowledge Bases?" In: *arXiv:1909.01066*. 2019.

[59] Maksim Podkorytov, Daniel Bis, and Xiuwen Liu. "How Can the [MASK] Know? The Sources and Limitations of Knowledge in BERT." In: *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), pp. 1–8.

[60] Maksim Podkorytov et al. "Effects of Architecture and Training on Embedding Geometry and Feature Discriminability in BERT." In: *2020 International Joint Conference on Neural Networks (IJCNN)* (2020), pp. 1–8.

[61] Alec Radford. "Improving Language Understanding by Generative Pre-Training." In: *preprint*. 2018.

[62] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[63] Peter Rhee et al. “Tetanus and trauma: a review and recommendations.” In: *Journal of Trauma and Acute Care Surgery* 58.5 (2005), pp. 1082–1088.

[64] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. “A Primer in BERTology: What we know about how BERT works.” In: *arXiv:2002.12327*. 2020.

[65] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.” In: *J. Comput. Appl. Math.* 1987.

[66] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors.” In: *nature* 323.6088 (1986), pp. 533–536.

[67] Adriaan MJ Schakel and Benjamin J Wilson. “Measuring word significance using distributed representations of words.” In: *arXiv:1508.02297*. 2015.

[68] Abigail See, Peter J. Liu, and Christopher D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks.* 2017. arXiv: 1704.04368 [cs.CL].

[69] Sefik Ilkin Serengil and Alper Ozpinar. “LightFace: A Hybrid Deep Face Recognition Framework.” In: *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE. 2020, pp. 23–27. DOI: 10.1109/ASYU50717.2020.9259802.

[70] Claude E. Shannon. “A mathematical theory of communication.” In: *Bell Syst. Tech. J.* 27 (1948), pp. 379–423.

[71] D. Silver et al. “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529 (2016), pp. 484–489.

[72] C Spearman. “THE PROOF AND MEASUREMENT OF ASSOCIATION BETWEEN TWO THINGS.” In: *The American Journal of Psychology* (1904), p. 72.

[73] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “LSTM Neural Networks for Language Modeling.” In: *INTERSPEECH*. 2012.

[74] Alon Talmor et al. “oLMpics-On What Language Model Pre-training Captures.” In: *Transactions of the Association for Computational Linguistics* 8 (Dec. 2020), pp. 743–758. ISSN: 2307-387X.

[75] Ian Tenney, Dipanjan Das, and Ellie Pavlick. “Bert rediscovers the classical nlp pipeline.” In: *arXiv:1905.05950*. 2019.

[76] Ashish Vaswani et al. “Attention is all you need.” In: *NIPS*. 2017.

[77] Elena Voita, Rico Sennrich, and Ivan Titov. "The Bottom-up Evolution of Representations in the Transformer: A Study with Machine Translation and Language Modeling Objectives." In: *arXiv:1909.01380*. 2019.

[78] Elena Voita et al. "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned." In: *arXiv:1905.09418*. 2019.

[79] Alex Wang and Kyunghyun Cho. "BERT has a Mouth, and It Must Speak: BERT as a Markov Random Field Language Model." In: *ArXiv* abs/1902.04094 (2019).

[80] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In: *ArXiv* abs/1804.07461 (2018).

[81] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In: *ICLR*. 2019.

[82] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing." In: *ArXiv:1910.03771*. 2019.

[83] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." In: *arXiv:1609.08144*. 2016.

[84] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." In: *arXiv:1906.08237*. 2019.

[85] Liang Yao, Chengsheng Mao, and Yuan Luo. "KG-BERT: BERT for Knowledge Graph Completion." In: *arXiv:1909.03193*. 2019.

# BIOGRAPHICAL SKETCH

Maksim Podkorytov is a Ph.D. candidate in the Computer Science department at Florida State University. He has earned his undergraduate degree in Computational Mathematics from Lomonosov Moscow State University, Russia in 2015. Prior to starting his Ph.D. studies, Maksim has worked at Rock Flow Dynamics as a C++ Software Engineer in a team responsible for the MPI aspect of tNavigator, a scalable distributed oil and gas reservoir simulator; his undergraduate thesis being a part of that experience. Maksim has started Ph.D. in Computer Science at the University of Texas at San Antonio in 2016 and has worked on polystore databases, search and data integration. He transfered to Florida State University in 2018 to continue Ph.D. in Computer Science and has worked on Deep Learning with applications in Natural Language Processing and Computer Vision. Maksim expects to graduate in 2021.