Harvard John A. Paulson
School of Engineering
and Applied Sciences

Tuned GPT-2: An Exploratory Analysis of Weights, Bias and Features

Advanced Scientific Computing (AM205) Final Project

Ana Vitoria, Adriana Trejo-Sheu, Christopher Gilmer-Hill,
Emma Besier, Hunter Heidenreich

Fall 2021

Contents

1. Objective

Have you ever had a word on the tip of your tongue but not known what it is, despite having the meaning of the word in your head? Perhaps you find yourself saying "I'm looking for a word that means XYZ". What if there was a language model that, given a definition, could spit that word out for you?

Or maybe, you have a word and know the definition but you aren't sure how you'd be able to use that word in a sentence. Sure, sometimes dictionaries are helpful in this context, but what if their examples aren't helpful? Wouldn't it be nice to be able to query a system that can keep generating new examples for you until you feel you understand exactly how that word is used?

In this project, we use a large, pre-trained Transformer model (GPT-2) and adapt it to data from Wiktionary to model the relationship between words, their definitions, and their example usages. The goal is to produce an adjusted model that given any of the above (a word, a definition, and/or an example usage), the model can output a missing piece of information.

After performing this adjustment, we use techniques from class (i.e., matrix decomposition techniques) to better understand the learned word embeddings and features. Finally, we perform an in-depth analysis of the limitations and bias that exists in our model. This section of the project is also an attempt to provide an additional peer review to the broader NLP community (Schoning, 2018).

Code and model weights developed for this project are publicly available on our Group GitHub repository.

2. Materials and Methods

2.1 Data Description, Acquisition and Cleaning

Wiktionary is the Wikipedia of dictionaries, providing a collaborative platform for a community-compiled lexical resource. It is open-source and user-edited/maintained. What makes it a particularly valuable resource is that for a given word/phrase (e.g., run) there are structured data of the different meanings a word can have (often referred to as "senses" of a word in NLP) and examples of how that word is used for that particular meaning. Furthermore, Wiktionary provides structured links for various linguistic relationships between words ranging from basic concepts like synonyms and antonyms (similar and opposite words), to more linguistic oriented links like hyper-/hyponyms (more general or more specific words, respectively), and even esoteric links such as anagrams (words that use the same letters but in a different order).

This project requires the compilation of structured dictionary-like data to tune a large scale language model in order to achieve our objective of producing a model that can supply words given a definition or word meaning, thus Wiktionary, as an open-source dictionary, was selected as the primary data source. To query Wiktionary for our data, we first needed to obtain an appropriate list of words. Corpus Data, a site which contains downloadable, full-text corpus data from ten large corpora of English, provides a free, random sample of words from 95,000 websites (totaling close to 14 million words). We used a subset of this data to generate our potential queries, a filtered list of 277,002 words. This list was narrowed down to only words with a corresponding Wiktionary page that contained at least one definition and example usage.

To perform the queries, we used beautifulsoup and wiktionaryparser. These two packages allowed us to pull the HTML page and filter it such that for each word, we were left with a part of speech, definition, example, and list of synonyms.

After obtaining the data from Wiktionary, we have a set of words, each with a series of definitions and examples of how they are used. For each word, we then flatten the data such that each definition or example consists of a single dataset sample. Specifically, we structure our dataset such that a word precedes the definition or example with key words to help the language model more readily adapt to the data domain. This is exemplified here:

Word: **word** Definition: **definition** $<$ |endoftext| $>$

Word: **word** Example: **example** $<$ |endoftext| $>$

Furthermore, we compile two instances of our dataset: one in which the words precede the definitions or examples (dubbed the `Forward` dataset) and one in which the definitions or examples precede the word (dubbed the `Reverse` dataset). Some examples of the forward split of data can be seen in Figure 2.1.

The reason for the creation of two datasets was to increase the flexibility in subsequent investigations and use cases. Where the `Reverse` data most closely resembles our main objective (to produce a model that can take a definition and produce words), the `Forward` data allows for the exploration of a model that can define new words its never been directly trained on. Though these two formats could be combined in theory, this would likely require more data and computational resources than we had for this project and so we leave such questions for future work.

Word: come Example: Hundreds of thousands of people come to Disneyland every year.<|endoftext|>

Word: upper Definition: That which is higher, contrasted with the lower.<|endoftext|>

Word: follow Example: We both ordered the soup, with roast beef to follow.<|endoftext|>

Word: custom Definition: (law) Long-established practice, considered as unwritten law, and resting for authority on long consent; usage. See Usage, and Prescription.<|endoftext|>

Figure 2.1: Example training data from the `Forward` dataset configuration.

2.2 Introduction to GPT-2

In order to construct a language model that can be used to generate dictionary-style data of the form above, we leverage recent advances in large scale language modeling through the usage of a neural network architecture called a Transformer model (Qiu et al., 2020; Vaswani et al., 2017). Specifically, we utilize a model called GPT-2 (Radford et al., 2019), a unidirectional, causal language model that has been pre-trained on a broad cross section of data obtained from the internet (8 million documents, 40 GB of raw text) and perform a *tuning* on the pre-trained model to adapt it to better handle the data domain in question–Wiktionary.

This is well-aligned with the recent trend in natural language processing (NLP) whereby a large neural network is trained to perform a task thought to be "general" (such as language modeling, described below) and then adapted to many specific problems that leverage the pre-trained weights of the model and the language representations it produces (Qiu et al., 2020). The benefit in this approach has much to do with the upscaling of network size that is the dominant paradigm in the deep learning community. As network sizes increase, so, too, does the need for data so that the network does not overfit to the specific dataset. However, this is directly at odds with the fact that constructing large labelled datasets is often difficult and resource intensive.

Instead, this approach leverages the fact that techniques have been developed that allow one to take a large, un-labelled text corpus and train a large model in a semi-supervised fashion directly on the data. Furthermore, this approach has been incredibly successful, yielding a whole zoo of large, pre-trained models that have different pre-training setups yet, nonetheless, yield powerful representations for language data that have proven useful across NLP, from question answering (Lewis et al., 2020; Radford et al., 2018) to classification (Devlin et al., 2018; Liu et al., 2019) to open-text generation (Lewis et al., 2020; Radford et al., 2019) and more.

GPT-2 comes in several different variations, each with a different number of parameters: small (117M parameters), medium (345M parameters), large (762M parameters), and xl (∼1.5B parameters). Since the goal of this work is largely exploratory and due to hardware limitations, we restrict our attention to the smallest GPT-2 model, the 117M parameter one. Nevertheless, as it has been shown that these models scale well, with the larger models producing language that humans rate to be more "fluent" (Brown et al., 2020; Kaplan et al., 2020), this selection is well justified. If this work produces sufficiently interesting results with the smallest GPT-2 model, then there is evidence that similarly tuning the much larger model (potentially, on more data) will only provide much better and more "fluent" language generations.

2.2.1 GPT-2 Pre-Training

As mentioned above, GPT-2 is a unidirectional, causal language model. Specifically, it is pre-trained to perform the task of *language modeling*: Given a corpus $\mathcal{D} = \{x_1, ..., x_n\}$ (where $x_i \in \mathcal{D}$ is a sequence of discrete text symbols $(s_1, ..., s_n)$, typically in the form of a sentence, paragraph, or other contiguous text sequence), the goal of language modeling is to be able to estimate the underlying probability distribution that generated the data, hence the name of this general family of models: Generative Pre-Training (GPT). Since text is sequential in nature, instead of modeling the full joint probability of discrete symbols in the data, one typically makes the simplifying assumption that the probability distribution may be factored into the product of conditional probabilities of the form (Bengio et al., 2003; Jelinek, 1980):

$$p(x) = \prod_{i=1}^{n} p(s_i | s_1, ..., s_{i-1})$$

Furthermore, due to the difficulty of learning long sequences, one often truncates the past horizon at a fixed length of k (for GPT-2, $k = 1024$):

$$p(x) = \prod_{i=1}^{n} p(s_i | s_{i-k}, ..., s_{i-1})$$

With this phrasing of the probability distribution in-hand and the neural network parameters denoted by Θ, language modeling proceeds by minimizing the negative log likelihood of the loss:

$$\begin{aligned}\mathcal{L}(\mathcal{D}) &= -\sum_{x \in \mathcal{D}} \log p(x; \Theta) \\ &= -\sum_{x \in \mathcal{D}} \sum_{i=1}^{n} \log p(s_i | s_{i-k}, ..., s_{i-1}; \Theta)\end{aligned}$$

As is the typical trend in deep neural networks, this loss is evaluated on batches (random sub-samples of the dataset) and the loss is then backpropagated (Rumelhart et al., 1995) through the network to update the weights using either stochastic gradient descent or more recent variations thereof that use notions like momentum (Kingma and Ba, 2014; Loshchilov and Hutter, 2018) or adaptive learning rates (Duchi et al., 2011) to either speed-up convergence or otherwise provide corrective measures to the optimization procedure.

GPT-2 is a fairly general model largely due to its its training corpus. Where works prior to GPT-2 explored pre-training domains that were fairly scoped (e.g., Wikipedia or a corpus of novels (Devlin et al., 2018)) or were the entirety of an internet scrape (i.e., the Common Crawl (Trinh and Le, 2018)), the researchers behind GPT-2 performed a series of filtration tricks on internet documents in an attempt to gather a diverse sample of language usage from the internet that did not suffer from issues of specificity or low-quality data. The end result is a model that has high performance across many language modeling datasets and can be readily adapted to specific problems downstream (Radford et al., 2019).

2.2.2 Model Architecture

As briefly discussed above, GPT-2 uses a neural network architecture called a Transformer network (Vaswani et al., 2017), but more specifically, just the Transformer decoder architecture (Liu et al., 2018; Radford et al., 2018) as opposed to the full encoder-decoder architecture that was initially proposed. This style of neural network architecture deviates significantly from the previous trend in NLP of using recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997) to perform sequence modeling. Instead, a Transformer network side-steps the recurrence aspect of sequence modeling completely, focusing the majority of its structural aspects on an operation called self-attention (Vaswani et al., 2017). Without specifically encoding a sequential aspect into the model architecture, Transformer networks have additional tricks they must leverage to incorporate information of sequence position, which we elucidate through the rest of this sub-section.

A high-level visualization of the Transformer architecture used for GPT-2 is displayed in Figure 2.2. Mathematically, this corresponds to:

$$\begin{aligned}h_0 &= \texttt{Dropout}(UW_e + W_p) \\ h_i &= \texttt{TransformerBlock}(h_{i-1}),\ i \in [1, n] \\ \hat{h}_n &= \texttt{LayerNorm}(h_n) \\ P(x) &= \texttt{Softmax}(\hat{h}_n W_e^T)\end{aligned}$$

where $W_e \in \mathbb{R}^{|V| \times d}$ is a token embedding, $W_p \in \mathbb{R}^{k \times d}$ is a position embedding, $U \in \mathbb{Z}_{|V|}^k$ consists of the one-hot encoded sequence of k symbols, $n = 12$ is the number repeated Transformer blocks within the network, and $P(x)$ represents the probability distribution the network generates over the next potential token.

Input Representation

In NLP, there is always the fundamental question of how to tokenize (i.e., split a contiguous sequence of text into discrete symbols) a text in a manner that can readily be ingested by a model.

A common level of tokenization is to segment at the level of a word, however, this has several significant issues. One issue is that of training a model to understand each unique token. A typical construction of a neural model in NLP has an initial embedding layer that maps discrete, unique tokens to dense, real-valued vectors. However, to do so, the model must be aware of all the symbols that may need to be potentially embedded. When dealing with words, this is a fundamental limitation as there is an empirical law in computational linguistics called Heap's Law (Egghe, 2007) (or Herdan's Law (Herdan, 1960)) which states that as the size of a corpus grows, so too does the size of the vocabulary without bound. The practical issue

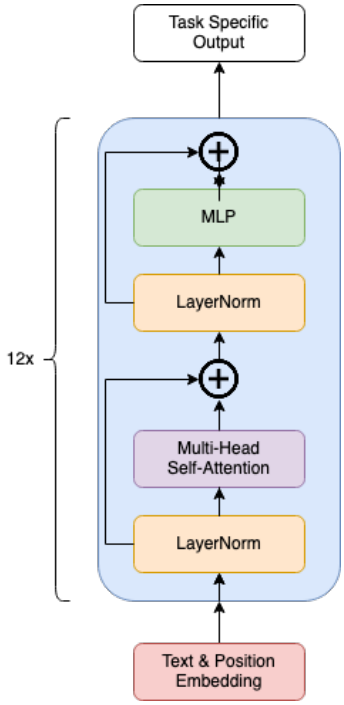


Figure 2.2: A high-level overview of the architecture used for GPT-2. A modification of the original diagram presented in (Radford et al., 2018), with modifications to the architecture from (Radford et al., 2019). After an initial embedding layer (for token and positional information), representations are propagated through 12 identically structured sub-network blocks called Transformer blocks. Within a single block, there is the self-attention portion and the position-wise feed-forward portion, each with its respective LayerNorm operation (Ba et al., 2016) and residual connection (He et al., 2016). As with all Transformer models, one must always train a task-specific "head" for the model, which is often just a dense matrix projection.

with this is that for any NLP model based at the word level, there will always be out-of-vocabulary words encountered by the model in practice. Another key issue is that a word-based model does not utilize any of the sub-word information (i.e., morphology) and thus suffers from efficiency concerns accordingly (Bojanowski et al., 2017).

One the opposite end of the spectrum, one can create a model that tokenizes at the character level. This creates a scenario where it is fairly straight-forward to construct an embedding layer that can handle any character, eradicating the out-of-vocabulary issue entirely. Furthermore, this yields an embedding layer that is relatively small compared to the millions of words that would need to be supported in an embedding layer at the word-level. While this has proven to be a valid option for constructing NLP models (Costa-jussà and Fonollosa, 2016; Ling et al., 2015), reducing to the character level removes knowledge that would otherwise be given to a word-level model *a priori*. Even something as simple as learning that a space separates words in the English language is something that a character-level model needs to discover. Again, the practical result of this strategy is that while it is feasible to train a model from the character-level, it is far from efficient, leading to models that take a long time to train to achieve comparable performance of models that operate at the word-level.

Thus, current practices in NLP strike a compromise between the word- and character-level, motivating something called a sub-word model. As the name suggests, these are models that break contiguous sequences of text not into characters or words, but into character n-grams. Often, these are learned directly from a corpus and may consist of things like prefixes and suffixes (e.g., "pre-" or "-ed") and some high-frequency, short words (e.g., "the" or "of"). The benefit of sub-word models is that they are immune to out-of-vocabulary issues (by design, they can always represent a text sequence as some sequence of character n-grams) and that one can fix a maximal vocabulary size so that the embedding layer remains relatively small (on the order of 10,000 to 100,000 symbols). Several examples of sub-word models include SentencePiece embeddings (Kudo and Richardson, 2018) and Byte-Pair Encodings (BPE) (Sennrich et al., 2016). GPT-2 uses BPE as its text encoding with a fixed vocabulary size of $|V| = 50,257$. Thus, its word embedding matrix W_e is a mapping $W_e : \mathbb{Z}_{|V|} \to \mathbb{R}^d$ where $d = 768$ is its latent dimension size.

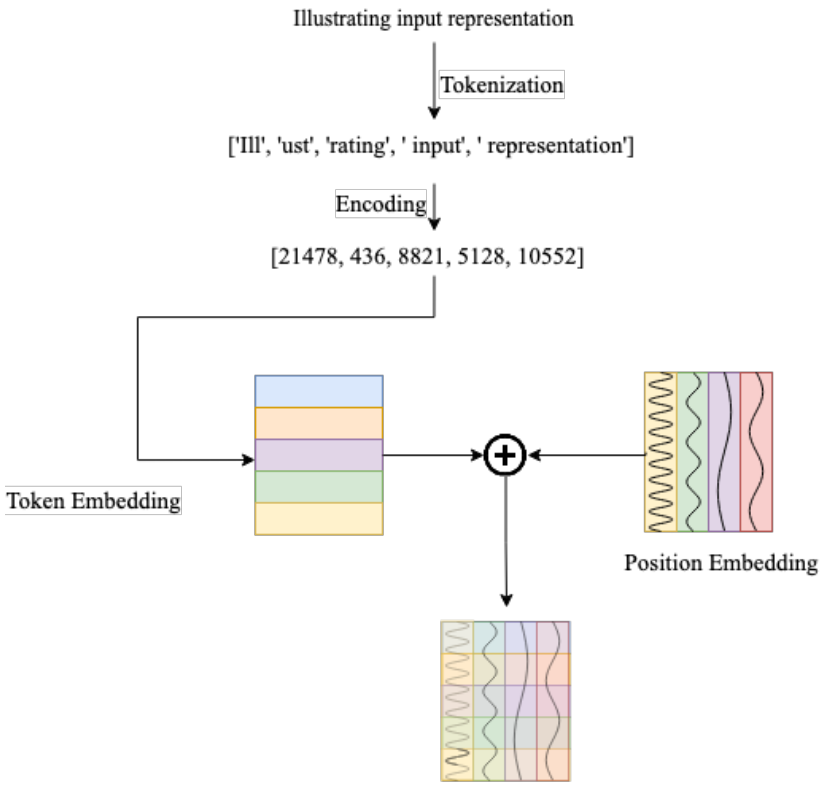


Figure 2.3: A visualization of the token embedding process. First, an input sequence is tokenized into sub-word pieces and encoded as integers. Then, the integers are embedded to obtain context-independent token vectors, which are then offset by the position embedding.

Another aspect of input representation in a Transformer model is that of position information. As will become obvious when discussing the core of a Transformer model, the Transformer block, Transformer models have no notion of positionality encoded in its architecture. In other words, there is no mechanism in a Transformer that helps it to understand where a particular token is positioned within the text sequence. To side-step this issue without altering the architecture involved, Vaswani et al. (2017) proposed using something called a *position embedding* which is added to the token embedding. This results in an initial token representation that is the token embedding slightly offset by an affine projection using the positional information. While Vaswani et al. (2017) explored learning this position embedding directly, they found that it offered no benefit over simply fixing the embedding to a sinusoidal representation that is hypothesized to allow the model to generalize to un-seen sequence lengths. Specifically, the position embedding used for GPT-2 is:

$$PE_{k,2i} = \sin\left(\frac{k}{10000^{2i/d}}\right)$$
$$PE_{k,2i+1} = \cos\left(\frac{k}{10000^{(2i+1)/d}}\right)$$

where k represents the position in the sequence ($k \in \{0, 1, 2, ..., k_{\text{max}} = 1023\}$), i is a dimension of the vector ($i \in \{0, 1, 2, ..., \lfloor d/2 \rfloor\}$), and $d = 768$ is the latent dimension of the model, as before. Thus, the position embedding W_p is a mapping $W_p : \mathbb{Z}_k \to \mathbb{R}^d$. Since the output dimensions of W_e and W_p are the same, their resultant vectors can be readily summed. The full input representation process on an example text is visualized in Figure 2.3.

Transformer Blocks

The core computational unit of a Transformer network is a repeated sub-module called a Transformer block. Typical Transformer constructions stack several Transformer blocks on top of each other, adding residual connections (He et al., 2016) for better gradient flow in backpropagation. While the exact configuration of what a Transformer block should consist of is still an open research question, GPT-2 takes a latent representation $h_i \in \mathbb{R}^{k \times d}$ to the next latent representation $h_{i+1} \in \mathbb{R}^{k \times d}$ through the following operations:

$$\begin{aligned} \hat{h}_i &= \texttt{Multi-Head Self-Attn.}(\texttt{LayerNorm}(h_i)) \\ g_i &= h_i + \hat{h}_i \\ \hat{g}_i &= \texttt{MLP}(\texttt{LayerNorm}(g_i)) \\ h_{i+1} &= g_i + \hat{g}_i \end{aligned}$$

An operation used throughout GPT-2 is one called layer normalization (Ba et al., 2016). This is a regularization layer that helps to normalize and constrain the latent dimension input values at each operation of the network and has been found to be of critical importance in stabilizing the dynamics of hidden states, particularly in recurrent and deep neural networks. It is the transpose operation of a batch normalization (Ioffe and Szegedy, 2015) that instead normalizes with respect to the layer.

Arguably, one of the most important operation that the Transformer block applies is that of multi-headed self-attention. In general, an attention function is one that maps a query vector and a set of key & value vectors into a new vector that is a weighted-sum of the value vectors as an operation of the query vector and the key vectors (Vaswani et al., 2017). Self-attention is the special attention function where keys, values, and queries are all the same sets of vectors: namely, the current hidden state of the text sequence. In other words, let $K = Q = V = h_i \in \mathbb{R}^{k \times d}$ and attention is:

$$\texttt{Attention}(Q, K, V) = \texttt{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where dividing by the square root of the latent dimension size $d = 768$ helps to stabilize computation. Extending self-attention to multi-headed self-attention simply consists of performing multiple self-attention steps (allowing different heads to aggregate different pieces of information across the sequence). Additionally, GPT-2 restricts which tokens are allowed to attend to other tokens by applying a mask within the attention operation. Specifically, GPT-2 uses a triangular mask that restricts self-attention so that tokens are only allowed to attend to themselves or tokens that precede them, but cannot attend to future tokens as this would make next-word prediction trivial. As this self-attention operation mixes token information without a mechanism for determining token position within the sequence, this operation necessitates the positional embedding discussed previously.

Finally, each Transformer block also contains a position-wise multi-layer perceptron (MLP) which consists of a two dense matrix projections ($W_1 : \mathbb{R}^d \to \mathbb{R}^{3072}$ and $W_2 : \mathbb{R}^{3072} \to \mathbb{R}^d$) with Gaussian Error Linear Units (GELU) (Hendrycks and Gimpel, 2016) as non-linear activation functions after each projections. Furthermore, a dropout operation (Srivastava et al., 2014) is applied to the final output of the MLP.

Output Representation

Every Transformer network ends with an output layer that is task-specific. In the case of GPT-2, which is pre-trained to perform language modeling, this final output layer simply consists of a linear mapping from the latent dimension to the vocabulary space: $P(x) = \texttt{Softmax}(h_n W_e^T)$, $h_n \in \mathbb{R}^{k \times d}$, $W_e \in \mathbb{R}^{|V| \times d}$. Oftentimes, this pre-training specific layer is discarded in lieu of a new task-specific layer (e.g., a mapping from sequence to some discrete set of classes $|C|$ for text classification). However, as this work seeks to fine tune using the language modeling task, this layer was retained and adapted to the Wiktionary data as well.

2.3 Model Tuning and Evaluation

The philosophy behind using any large pre-trained language model such as GPT-2 is that it has already been initialized to a weight parameterization that is fairly decent at language modeling and can be readily adjusted to a more specific domain

or NLP objective (Radford et al., 2018). This section details the tuning procedure used and the results of the fine-tuning of GPT-2.

2.3.1 Dataset

In totality, 6,031 unique words were sampled to construct a dataset of dictionary-like entries. As the words were selected such that each word has at least one definition and one example usage, each unique word generates several different data samples for tuning. Expanding the dataset such that each definition and example for a word forms its own entry in the dataset, a total of 65,169 tuning samples are generated. This tuning set is then spit into a train/validation/test split of sizes 80%/10%/10%. This is all summarized in Table 2.1.

Train	52,135 (80%)
Validation	6,516 (10%)
Test	6,518 (10%)
Total	65,169

Table 2.1: A summary of how the tuning dataset was partitioned.

2.3.2 Tuning Procedure

Standard tuning techniques (Liu et al., 2019; Radford et al., 2018) were applied to GPT-2 to tune it on the Wiktionary dataset. GPT-2 was initialized to its pre-trained weights and then was tuned to perform language modeling on the Wiktionary dataset for 3 epochs. The AdamW optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2018) was leveraged to tune the model with an initial learning rate of 5×10^{-5}, a weight decay of 0, a batch size of 8, and a linearly decreasing learning rate schedule (decreasing from its initial value to 0 at the end of the last batch). Every 2,000 batches (16,000 samples), the model is validated on the validation split of the data and the weights are cached only if performance has improved on the validation data to help prevent overfitting to the Wiktionary data.

Two variants of the model were tuned: one that trains on data where the word precedes the example or definition (`Forward`) and one that trains on the data where the example or definition precedes the word (`Reverse`). The `Reverse` model most closely resembles the initial motivation for this project, where one has a meaning of the word and seeks potential word options. The `Forward` model supports sampling new definitions and example usages from the model. While technically it would be feasible to train a model to be able to handle both formats, with limited data, this could end up producing a less powerful model that performs both tasks poorly compared to two models specialized to each ordering. We leave future work to the exploration of a single model that can perform either direction, hopefully with more tuning data and possibly using a larger variant of GPT-2 that has a greater capacity to learn such an invariance.

2.3.3 Results

The results of tuning performance are summarized in Table 2.2.

	NLL			**NLL_{TOK}**		
Model	Train	Val	Test	Train	Val	Test
`Forward`	49.94 $\pm$ 64.61	57.00 $\pm$ 74.08	57.73 $\pm$ 73.86	2.33 $\pm$ 0.83	2.71 $\pm$ 0.98	2.70 $\pm$ 0.98
`Reverse`	48.64 $\pm$ 64.46	55.96 $\pm$ 74.20	56.67 $\pm$ 73.85	2.27 $\pm$ 0.83	2.66 $\pm$ 0.99	2.64 $\pm$ 0.99

Table 2.2: Negative log likelihood (NLL) scores for each split of the dataset. NLL is reported as the median $\pm$ the standard deviation. The median is used instead of the mean since the mean is skewed towards the tail of the distribution, as discussed in more detail in the body. Scores in the left set of columns are NLLs of entire samples (unnormalized by sequence length) whereas the right set of columns are NLLs averaged over the tokens within a sequence.

A couple of key insights about the performance of the two models can be gleaned from the Table:

- The two variants of the model perform fairly comparably, indicating that GPT-2 can readily adapt to either format from its pre-trained weights. It does appear that the reverse model tuned to a slightly better convergence point, as seen by the fact that it has slightly lower negative log likelihood scores. This may indicate that the task of going from a definition or an example to predicting the word is an easier task than going from a word to the generation of one of its definitions or examples, however, more investigation is warranted before one could make such a far reaching claim.
- Both models perform comparably on the validation versus the test splits of the dataset. Of course, the models both fit better on their training set (as would be expected). Although there is a slight difference between performance on the validation and test set, indicating a slight bias towards the validation set, this difference is fairly minimal and seems to indicate that neither model overfit the data to a significant extent.
- As remarked upon in the table description and as seen from the standard deviation of the negative log likelihoods, the distribution of performance across samples is highly skewed. In particular, there are a couple dozen training examples that the model completely fails to handle with any certainty. These are exemplified and discussed further below.

Performance Analysis

Here, we consider the instances that the models performance the best and the worst on (as measured by the negative log likelihood) to gain a sense for what the models handle well and what they fail to capture. Consider, first, the top 3 best and worst training examples ranked by their total sequence negative log likelihood displayed in Table 2.3. Some general trends displayed in these samples:

- **Mastery of Short Usage Gloss:** Considering the best samples, we observe that both models excel at modeling short word usage glosses. This is fairly logical as every word has an example usage gloss like this as a definition, therefore, it is a regular pattern in the dataset. Additionally, since the total sequence negative log likelihood is not normalized with respect to the sequence length, it is natural that the short glosses would score much lower.
- **Difficulty with Old Examples:** Both models exhibit identical worst examples in an identical ordering. Interestingly enough, the top 2 worst instances both are example usages from old sources. The top instance is Shakespeare (specifically, *A Pleasant Conceited Comedie Called, Loues Labors Lost*) and thus is from the end of the 16th century. The second instance comes from a book titled *An embassy from the East-India Company*[1] which was written in Dutch in 1665 and translated into English in 1669. Considering the domain that GPT-2 was originally trained on (a broad cross-section of data from the internet), it seems reasonable that these long and old examples are the ones that the models perform the worst on. English, as used on the internet at present, is very different than Shakespeare, and the unconventional spellings (relative to present English) are likely too far out-of-domain for GPT-2.
- **Difficulty with Rare Unicode:** All three worst instances additionally share a common feature: They use rare Unicode symbols. For the fist two instances, this is largely in the usage of the character ſ. For the third instance, it includes Old English, German, Greek, and Latin intermixed in the definition. Additionally, the lack of usage of such characters in the model's original pre-training may cause it to not often predict these Unicode symbols, which would also drive the negative log likelihood up for these examples.

If we rank the samples by average token negative log likelihood, we get a different set of samples which are shown in Table 2.4. Even with the length now accounted for, we still see similar patterns such as the best reproduced samples all being definitional glosses. However, the worst samples do expose another kind of weakness of the model: poetry and ill formatted examples. This seems to indicate that in future work, more cautious handling of some of the examples is necessary, especially since an example should actually use the word it is supposed to be exemplifying (unlike the 3rd worst example, which is cut-off early due to poor formatting in the original Wiktionary data).

[1] Wikipedia: An embassy from the East-India Company

Type	Rank	Sample
Forward Best	1	Word: involve ; Definition: involve (third-person singular simple present involves, present participle involving, simple past and past participle involved)
	2	Word: cadence ; Definition: cadence (countable and uncountable, plural cadences)
	3	Word: scrape ; Definition: scrape (third-person singular simple present scrapes, present participle scraping, simple past and past participle scraped)
Forward Worst	1	Word: obscene ; Example: [...] I did incounter that obſeene and moſt prepoſterous euent that draweth frõ my ſnowhite pen the ebon coloured Incke, which here thou vieweſt, beholdeſt, ſuruayeſt, or ſeeſt. [...] There did I ſee that low ſpirited Swaine, [...] hight Coſtard, ſorted and conſorted contrary to thy eſtabliſhed proclaymed Edict and continent Cannon; Which with, o with, but with this I paſſion to ſay wherewith: / Clo[wne]. With a Wench.
	2	Word: universe ; Example: Under our new World may alſo be compriſed thoſe vaſt Southern Coaſts and Streights of Magelan, firſt lighted on by Ferdinandus Magelanus in the year 1520, in his Circumnavigation of the Univerſe; which forty five years after Sir Francis Drake, and next Sir Thomas Bendiſh, Engliſhmen, made a furhter inſpection into; and in the Year 1600 Oliver van Noord a Hollander paſt, but of later years a Spaniard, Fedinand de Quier, out-ſhot them all by a more ample Diſcovery then all the former.
	3	Word: death ; Definition: (often capitalized) The personification of death as a hooded figure with a scythe; the Grim Reaper. The pronoun he is not the only option, but probably the most traditional one, as it matches with the male grammatical gender of Old English dēaþ, also with cognate German der Tod. The fourth apocalyptic rider (Bible, revelations 6:8) is male $\theta\alpha\nu\alpha\tau o\zeta$ (thanatos) in Greek. It has the female name Mors in Latin, but is referred to with male forms qui and eum. The following quotes show this rider on a pale horse is his in the English Bible and she in Peter Gabriel's lyrics.
Reverse Best	1	Definition: ambidextrous (comparative more ambidextrous, superlative most ambidextrous) ; Word: ambidextrous
	2	Definition: move (third-person singular simple present moves, present participle moving, simple past and past participle moved) ; Word: move
	3	Definition: copy (third-person singular simple present copies, present participle copying, simple past and past participle copied) ; Word: copy
Reverse Worst	1	Example: [...] I did incounter that obſeene and moſt prepoſterous euent that draweth frõ my ſnowhite pen the ebon coloured Incke, which here thou vieweſt, beholdeſt, ſuruayeſt, or ſeeſt. [...] There did I ſee that low ſpirited Swaine, [...] hight Coſtard, ſorted and conſorted contrary to thy eſtabliſhed proclaymed Edict and continent Cannon; Which with, o with, but with this I paſſion to ſay wherewith: / Clo[wne]. With a Wench. ; Word: obscene
	2	Example: Under our new World may alſo be compriſed thoſe vaſt Southern Coaſts and Streights of Magelan, firſt lighted on by Ferdinandus Magelanus in the year 1520, in his Circumnavigation of the Univerſe; which forty five years after Sir Francis Drake, and next Sir Thomas Bendiſh, Engliſhmen, made a furhter inſpection into; and in the Year 1600 Oliver van Noord a Hollander paſt, but of later years a Spaniard, Fedinand de Quier, out-ſhot them all by a more ample Diſcovery then all the former. ; Word: universe
	3	Definition: (often capitalized) The personification of death as a hooded figure with a scythe; the Grim Reaper. The pronoun he is not the only option, but probably the most traditional one, as it matches with the male grammatical gender of Old English dēaþ, also with cognate German der Tod. The fourth apocalyptic rider (Bible, revelations 6:8) is male $\theta\alpha\nu\alpha\tau o\zeta$ (thanatos) in Greek. It has the female name Mors in Latin, but is referred to with male forms qui and eum. The following quotes show this rider on a pale horse is his in the English Bible and she in Peter Gabriel's lyrics. ; Word: death

Table 2.3: Top-3 best and worst training samples for the `Forward` and `Reverse` models, as evaluated by the full sequence negative log likelihood.

Type	Rank	Sample
Forward Best	1	Word: quibble ; Definition: quibble (third-person singular simple present quibbles, present participle quibbling, simple past and past participle quibbled)
	2	Word: regurgitate ; Definition: regurgitate (third-person singular simple present regurgitates, present participle regurgitating, simple past and past participle regurgitated)
	3	Word: sojourn ; Definition: sojourn (third-person singular simple present sojourns, present participle sojourning, simple past and past participle sojourned)
Forward Worst	1	Word: poker ; Example: Key sense with the super-soakerLongest poker, leave man stressed like yoga
	2	Word: war ; Example: All human tribes glad token see
	3	Word: skill ; Example: And Howell Dha shall goodly well indew
Reverse Best	1	Definition: regurgitate (third-person singular simple present regurgitates, present participle regurgitating, simple past and past participle regurgitated) ; Word: regurgitate
	2	Definition: conjugate (third-person singular simple present conjugates, present participle conjugating, simple past and past participle conjugated) ; Word: conjugate
	3	Definition: quibble (third-person singular simple present quibbles, present participle quibbling, simple past and past participle quibbled) ; Word: quibble
Reverse Worst	1	Example: ...for swenge swat ædrum sprongforð under fexe. ; Word: spring
	2	Example: Key sense with the super-soakerLongest poker, leave man stressed like yoga ; Word: poker
	3	Example: And Howell Dha shall goodly well indew ; Word: skill

Table 2.4: Top-3 best and worst training samples for the `Forward` and `Reverse` models, as evaluated by the average token negative log likelihood.

3. Sub-Projects

3.1 Language Generation Capabilities

3.1.1 Word Guessing Game

One of the initial motivations for this project was to produce a model that, given a definition, can produce the word that corresponds to that definition. This maps onto a user use-case where they have a word on the tip of their tongue and simply can't quite identify the word they seek. Here, we use our produced `Reverse` model to explore its capacity for performing this task.

To evaluate this model in this context, we leverage our Wiktionary dataset and give the model either an example or definition as a seed and then sample potential words using beam search (Freitag and Al-Onaizan, 2017), a method that maintains multiple generation hypotheses (i.e., beams) and sub-selects from them based on largest likelihood of generation. For this work, we take a beam size of 10 which results in the model producing a ranked list of 10 word guesses. From there, we leverage a metric from information retrieval called Mean Reciprocal Rank (MRR) (Voorhees, 1999):

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$$

MRR is most frequently used in information retrieval to evaluate how well a model produces a search ranking. Given a ranked list of options, MRR identifies the rank of a "hit" (the target answer) and assigns that query a score equal to the reciprocal of the rank. MRR, thus, becomes the harmonic mean of the ranks of the target words here. Furthermore, the reciprocal of the MRR then represents the average rank of the true word given a query to the model

Split	MRR	$(\text{MRR})^{-1}$
Train	0.6346 ± 0.4341	1.58 ± 2.30
Validation	0.5158 ± 0.4466	1.94 ± 2.24
Test	0.5242 ± 0.4443	1.91 ± 2.25

Table 3.1: Mean reciprocal rank (MRR) of the target words given their definition or example usage. The reciprocal of the MRR represents the average number of words one would need to consider before observing the target word.

The MRR score for each split of the Wiktionary dataset is shown in Table 3.1. This table indicates that the `Reverse` model constructed performs quite well on this task. While it clearly doesn't perform perfectly, the fact that, on average, the model can produce the target word within 2 or 3 options seems to indicate that it is quite useful at performing this task. While more qualitative than quantitative, upon manual inspection, sometimes the model produces an adjacent word or word fragment to the target word as the top option (e.g., "slimm" when the target word is "slim"). However, more data and perhaps a larger GPT-2 variant would likely perform that much more accurately. Furthermore, expanding the number of beams considered beyond 10, say to 100, would result in a slightly increased model score at the cost of high precision.

To give some examples of how this model performs on this task, see Table 3.2. Barring the example with rutabaga, it seems that even when the model is incorrect, it still produces fairly reasonable results.

Target	Query	Ranked Suggestions
numeral	Definition: (linguistics) A word representing a number.	number, digit, numeral, letter, fraction, word, figure, decimal, complement, exponent
paralysis	Definition: A state of being unable to act.	paralysis, disability, dependency, depression, absence, amnesia, inability, blindness, unemployment, apathy
rutabaga	Definition: (now Canada, US) the edible root of this plant	lettuce, apple, cabbage, mustard, pineapple, strawberry, pepper, cucumber, mango, pepperidge
spontaneous	Definition: Self-generated; happening without any apparent external cause.	spontaneous, natural, self, intrinsic, inherent, external, generated, free, voluntary, selfish
switch	Definition: (transitive) To exchange.	exchange, interchange, swap, convert, trade, sell, take, lend, give, negotiate

Table 3.2: Five randomly selected examples of words generated by the `Reverse` model on the word guessing game.

3.1.2 Definition & Example-Usage Generation

While the `Forward` model cannot be directly applied to predict words from definitions or example usages, it can be used to generate new definitions or exemplify how a word can be used. Here, we exemplify some of the produced outputs from this model on the task of generating definitions from a word.

The best method for generating text from a neural language model like GPT-2 is still very much an open research question (Holtzman et al., 2019). That said, a particular sampling approach called *nucleus sampling* (also, *top-p* sampling) has proven to be effective, particularly if one seeks to produce dynamic and interesting outputs. The intuition behind this approach to sampling is to sample words from the nucleus of the predicted probability distribution, or the most probable region of the distribution.

The core hyperparameter necessary for this sampling algorithm is a cumulative distribution threshold p. Using p, potential tokens are added to the nucleus from most to least likely until the threshold p is reached, at which point all the remaining low-likelihood tokens are discarded. This results in an approach that is dynamic based on the confidence of the model: If the model is highly certain that a particular token should be generated next, then the cumulative threshold is likely to be

exceeded within several tokens. If the model is uncertain, then it has the added flexibility to select from multiple, high-likelihood alternatives. This contrasts with an alternative method for sampling called *top-k* sampling, which restricts to sampling from a fixed number k of high-likelihood tokens, but also suffers from sampling low-likelihood tokens even when the model is fairly confident that one or two tokens should be generated next. For this work, we fix $p = 0.95$ for all sampling of definitions and example usages.

Word	**Generation**
eigenspectrum	(physics) A nonlinear physical force acting on the rotational body of a mass in order to generate an angular velocity with respect to the body.
	The dorsal and ventral regions of a bird's body.
	(nautical) A short-lived, usually curved-slope, sail attached to a ship's mast, and then passed into the bow and stern of a ship, that sails or moves, etc.
matrix	(programming, n-comparative more matrix, superlative most matrix)
	A solid or solid solid-state arrangement.
	(grammar) A syntactical structure that describes, as opposed to a verbal structure.
scientific computing	(uncountable) The application of science to problem-solving and prediction science.
	Having the highest degree of abstraction; open-source. [from 14th c.]
	A science, technology, or technology based on the basic tenets of science, technology, engineering, mathematics, computer science, etc.
tweet	tweet (plural tweets)
	(intransitive, of a verb) To send (an email address or similar).
	(intransitive) To send or express an opinion or message without actually writing it or, generally, without providing proof. [from late 14th century]
jawn	(countable) A small amount of something or someone; especially of a coin or precious stone.
	jawn (plural jawns)
	(transitive, now rare) To make the appearance of resembling a mammal; to imitate.

Table 3.3: Some definitions generated by the `Forward` model when prompted with each word.

Table 3.3 exemplifies the definition generation capabilities of the `Forward` model. on five hand-selected words: eigenspectrum, matrix, scientific computing, tweet, and jawn. For each word, we generate 3 definitions randomly using top-p sampling. A couple of interesting observations include:

- The model's inclination to generate dictionary glosses that demonstrate how a word can be conjugated (shown for matrix, tweet, and jawn).
- Generation of definition glosses that inform how a verb is used (e.g., intransitive) or the topic and usage of a noun (e.g., countable, physics).
- Indications of when such a definition originated (e.g., [from late 14th century]). This is 100% fictional, but is interesting that the model is producing these aspects nonetheless.

While many of these definitions appear somewhat reasonable, some of them are completely incorrect but somewhat topically adjacent to what a true definition might be. This is indicative of some of the associations that the model has either picked up either through its pre-training data or its tuning data, and is emblematic of why models like GPT-2 are often referred to as "stochastic parrots" (Bender et al., 2021). If more accuracy is desired from this model, much more data is likely necessary as well as architectural and training adjustments that numerically encode and penalize with respect to such a constraint.

Word	Generation
eigenspectrum	The eigenspectrum is the small subarachnoid sac in the forehead of a human.
	The second eigenspectrum is for visual acuity. When visual acuity is measured, visual acuity is defined as approximately 100 nanoseconds per second.
	[T]his invention of the eigenspectrum, which was originally intended for the study of the mind and for the development of mechanical devices, was mainly owing to the rapid development of science, and was a method for developing mechanical devices.
matrix	"What matrix?" "Can you pull out the numbers for this?" I didn't know, I didn't know, I had a black box, a black matrix, a black matrix, a black matrix.
	He is the master of the matrix.
	The two groups then worked out a new matrix for each group, a group consisting of three numbers and two groups.
scientific computing	The scientific computing of the last century is coming to an end.
	They call this the 'Scientific Computing' era.
	If you don't give us credit, we'll laugh at you.
tweet	A popular tweet sent to his followers by a user was retweeted more than 3,000 times.
	His Twitter followers' comments section is littered with photos of his fans celebrating his birthday.
	Just to make the worst possible case for his actions, he deleted his tweet on Twitter.
jawn	He saw at once that there were jawns.
	The jawn or sodden part of my head is a part of my throat.
	Synonym: dawn

Table 3.4: Some example usages generated by the `Forward` model when prompted with each word.

Table 3.4 exemplifies some word usage generations for the same set of five words used above (eigenspectrum, matrix, scientific computing, tweet, and jawn). Here, the lack of precise knowledge encoded in the model becomes even more obvious compared to the definition generation. For one, the model struggles to actually exemplify how a word is used in many of the generations (likely as a side-effect of some of the ill-formatting of the data in combination with a need for more data and more training time). For other generations, the model produces text that is loosely related and does exemplify a word's usage. However, the examples generated are not necessarily the most effective at demonstrating how someone should actually use a given word.

3.2 Representation Geometry: PCA/t-SNE/UMAP/VAE Comparison

One approach for the qualitative evaluation of word vector representation is to consider their visualisations under dimensionality reduction techniques. For example, the word2vec algorithm (Mikolov et al., 2013), which helped to popularize neural word embedding techniques, demonstrated that using principal component analysis (PCA) can uncover linear relationships in the word embedding space that describe morphology, analogical relationships, and more. More recently, work has focused on how similar dimensionality reduction techniques may be applied to contextual word representations, like those that are produced by Transformer models, to better understand the internal geometry of the language representations and to reason about the associations that models are encoding (Coenen et al., 2019; Vig, 2019).

In this sub-project, we compare various dimensionality reduction techniques on the embeddings of polysemous verbs (i.e., verbs with multiple meanings, determined by the context of their usage). Questions to be answered include: 1) Do examples with different senses occupy different spaces? 2) Are examples of the same senses grouped in latent space?, and 3) Which clustering technique (if any), best aligns with the human annotation of the senses?

3.2.1 Dimensionality Reduction Algorithms

First, we provide an overview of the dimensionality reduction techniques used:

Principal Component Analysis (PCA)

The goal of PCA (Pearson, 1901) is to find a linear projection of the data into an orthogonal basis system that has minimum redundancy and still preserves the variance of the data. Such a formulation then allows us to reduce the dimensionality of the data from its original dimensionality d to some fixed $k < d$. This transformation relies on SVD, which decomposes a matrix A into three matrices:

$$A = U\Sigma V^T = \underbrace{\begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r\end{bmatrix}}_{\text{Col } A} \underbrace{\begin{bmatrix}\mathbf{u}_{r+1} & \dots & \mathbf{u}_m\end{bmatrix}}_{\text{Nul } A^T} \begin{bmatrix}\sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0\end{bmatrix} \begin{bmatrix}\mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \dots \\ \mathbf{v}_r^T \\ \mathbf{v}_{r+1}^T \\ \dots \\ \mathbf{v}_n^T\end{bmatrix} \begin{matrix}\} \text{Row } A \\ \\ \\ \\ \} \text{Nul } A\end{matrix}$$

To perform PCA, we obtain the first k largest singular values and their singular vectors. Using SVD, this would correspond to $U_k\Sigma_k$.

t-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE (Hinton and Roweis, 2002; Van der Maaten and Hinton, 2008) is a non-linear dimensionality reduction technique that finds patterns in the data based on the similarity of data points. Let $\{x_i\}$ be the set of points in the high-dimensional, original vector space and let$\{y_i\}$ be the set of projection points in a lower dimensional space. Define $P_{j|i}$ to be a Gaussian distribution that characterizes the conditional distribution that point x_i has point x_j as a neighbor:

$$P_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{i\neq j}\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}$$

where σ_i is the standard deviation for the Gaussian distribution associated with point x_i. Since this distribution is asymmetric, the probability that x_i, x_j are neighbors is computed by combing their two condition distributions to yield a symmetric distribution:

$$P_{ij} = \frac{P_{j|i} + P_{i|j}}{2N}$$

for N data points.

Similarly, define Q_{ij} to be the symmetric student-t distribution of the probability that point y_i has point y_j as a neighbor in the low-dimensional space:

$$Q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{k\neq\ell}(1 + \|y_k - y_\ell\|^2)^{-1}}$$

With the probability distributions that two points are neighbors defined in both the high- and low-dimensional spaces, t-SNE then minimizes the difference between these probability distributions using the Kullback-Leibler divergence:

$$KL(P\|Q) = \sum_i \sum_j P_{ij} \log \frac{P_{ij}}{Q_{ij}}$$

Uniform Manifold Approximation and Projection (UMAP)

Unlike other methods, UMAP (McInnes et al., 2018) builds a "fuzzy simplicial complex", i.e. a representation of a weighted graph, with edge weights representing the likelihood that two points are connected. The exact construction of UMAP is well motivated by topology and category theory (the theoretical motivations are discussed in detail in Section 2 of McInnes et al. (2018)), but at a high-level, the algorithm makes 3 assumptions about the data used for UMAP:

- There exists a manifold on which the data would be uniformly distributed over.
- The underlying manifold is only locally connected (i.e., there are no far reaching or global connections within the manifold).
- Preserving the topology of the underlying manifold of this data is the primary goal that UMAP should accomplish.

Specifically, UMAP produces a reduced dimensionality representation of the original data by constructing a k-neighbor weighted graph for each data point (with a transformation kernel to smooth these connections and then a process for introducing symmetry to the graph representation). A low-dimensional representation of this graph is randomly generated and then optimized to minimize the cross-entropy between the edges of the original graph and the low-dimensional one. When visualized, the results are similar to t-SNE, but the algorithm is substantially faster and is sometimes better at preserving global structure.

Variational Autoencoder (VAE)

A Variational Autoencoder is an unsupervised deep learning method that relies on variational inference. Given a dataset x, the encoder network predicts the parameters for a distribution that models the posterior, and backpropagates through the loss to learn the autoencoder weights. It consists of three components—an encoder, a decoder and a loss function.

1. The **encoder** is a neural network that takes in an image x and produces a hidden representation z. The encoder is a variational inference network, which maps observed inputs to posterior distributions over latent space.
2. The **decoder** is a generative network, whose input is the z-representation and who outputs the parameters to the probability distribution of the image.
3. The **loss function** is a negative log-likelihood with a regularizer. The log-likelihood encourages accurate reconstructions, while the regularizer (Kullback-Leibler divergence) encourages diverse representations.

   We use the latent, or hidden, space to visualize the most important features.

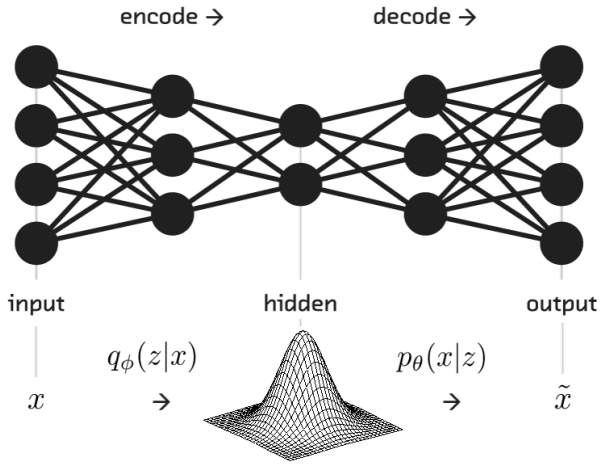


Figure 3.1: Variational Autoencoder

3.2.2 Results

Leveraging a cleaner corpus of linguistic data called WordNet (Miller, 1995), we randomly sample subsets of five verb "synsets" (collections of verbs with synonymous meanings under a particular definition). We made the choice to use this particular dataset due to its readily available, standard collection of polysemous verbs in synonymous cohorts.

Using these verb collections, definitions, and example usages, we passed in definition-example pairs into the tuned model and collected the embedded representations from the final hidden layer output (the output from the 12th Transformer block). We averaged over the first axis of each matrix (over the tokens) to obtain a series of vectors of length 768 for each definition-example instance, and then applied PCA, t-SNE, UMAP, and a trained VAE to reduce the dimensionality from 768 to 2.

After applying the dimensionality reduction techniques, we plotted the results as a series of x and y points on a scatter plot. K-means, an unsupervised learning method, was then applied to each set of points to see if distinct clusters exists. This method, in general, only yielded fruitful results for UMAP and t-SNE. We repeated this analysis five times with different sets of verbs (see Appendix in Section 5), but chose one to highlight one in the main report.

Overall, we obtained reasonable results:

1. The VAE created the least distinguishable clusters. This is likely because in addition to it's goal of minimizing the KL-Divergence, it is minimizing a reconstructive error term.
2. UMAP and t-SNE created the most distinct clusters, with t-SNE having the unique characteristic that all points are almost all equally spaced. The overall performance of UMAP is not best represented in Figure 3.2. Please see the Appendix in Section 5 for more examples.
3. PCA also created clusters, but some elements within a cluster were extremely close together and others were far apart. This made the creation of decision boundaries with k-means difficult.
4. The success of the clustering methods was highly dependent on the randomly selected verbs.

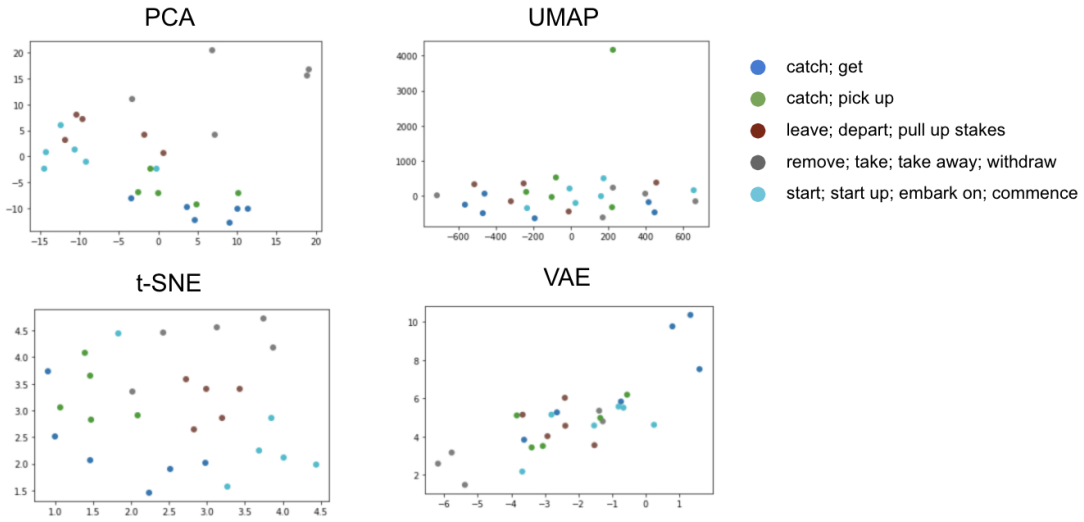


Figure 3.2: Clustering

That said, there are some limitations to the current analysis. For one, more example usages for these verbs would be extremely useful and likely enhance (and possibly draw out) clusters that appear in the data, especially for methods like t-SNE and UMAP.

Another consideration is that of the particular model we used: GPT-2. GPT-2 is primarily a model for language generation, as determined by its pre-training objective. Typically, when one wants a "better" semantic representation of meaning, they need to use a model that processes language bidirectionally, noting the effects of meaning on a given word by considering both the words that come before and after them. Other Transformer-based models like BERT (Devlin et al., 2018) and RoBERTa

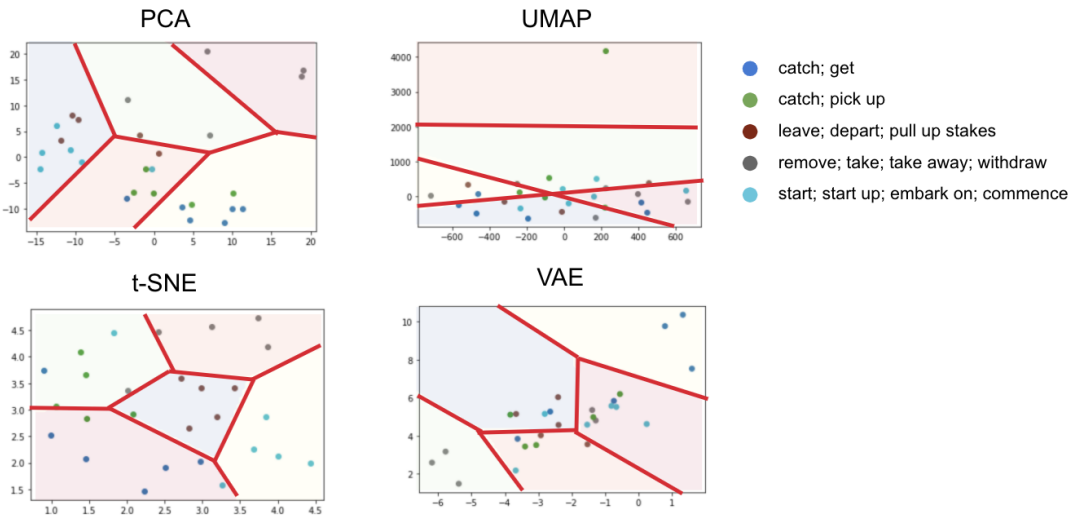


Figure 3.3: Decision boundaries of k-means

(Liu et al., 2019) are bidirectional models that are trained on alternative objectives and have been found to perform slightly better on tasks that involve heavy semantic disambiguation and natural language understanding (NLU). Where GPT-2 excels at language manipulation, it also may not have as clear a distinction in its representation for the different meanings of these verbs leading to clusters that aren't as well-defined. In future work, it may be interesting to consider a BERT-inspired model for Wiktionary data and to perform more visualization and geometric analysis like that of other works (Coenen et al., 2019).

3.3 Model Limitations & Gender Bias

3.3.1 Model Limitations

As described prior, we fine tuned our GPT-2 model on 65,169 set of data points that we parsed from Wiktionary. We want to explore how our fine-tuning has impacted the what the model outputs for words, examples, and definitions. We decided to do a series of tests to see what our model will produce in different scenarios. To have a baseline comparison, we test what the model outputs for examples and definitions on words we know we have trained on. For all of the tests, we used the following parameters in the table below.

Parameter Name	Description	Value
top_p	Restricts to the top p% of the cumulative distribution. Closer to one means more creative the model can be where closer to 0 the more conservative.	0.9
temperature	Higher temperature will make the model sample more "surprisingly". Works similar to how higher temperatures can cause more chaos for electrons in reaction states.	1.0
minlen	Minimum number of symbols the model will add	2
maxlen	Maximum number of symbols the model will add	128
num_samples	Number of samples the model will return	10

Table 3.5: Parameters used for sampling that is use for each of the tasks

Baseline testing:

To get a base comparison, we tested a subset of words that we trained the model on to see what our model outputs for the associated definitions and examples. We want to see if the model will give logical answers for these words so that we can do the comparison for words outside of this baseline, such as foreign words, slightly misspelled words, words with multiple meanings and slang or fake words.

Results: The list of words tested were [old, nervous, performer, eradicate, sauna, processing, steady, shuffle, epistle, cute]. This test was run using our Forward model, meaning it takes a word as an input and gives out 10 definitions. Most of the definitions that came out were similar to some of the inputs in our training data. It was even able to capture the multiple meanings of nervous. To see the results, they are in the baseline_test.txt in the model_outputs folder within Limitations and Bias.

Additional Testing

We identified five tasks to test on to get a preliminary understanding of fine-tuning has impacted our model and what the shortcomings of our model are. These scenarios include foreign words, slightly misspelled words, words with multiple meanings, and slang or 'fake' words. All of these results are found in output_limitations.txt in the model_outputs folder. We also took a refined list of common words for the model to test whether an inputted word can be also outputted if we pass in one of the definitions that was associated with the inputted word.

Foreign words: We are passing words that come from different languages to see how our model will interpret these words and what definitions and or examples will be outputted. It will be interesting to see whether it is able to translate the words correctly. We selected a series of words from the Spanish(ES), French(FR), Chinese(CHN), and Hawaiian(HAW) languages. The words have varying levels of recognition, meaning that they could be words that many people would be able to translate or know based off our assumptions. It is important to note that some of the words tested (chateaux, bao, and cha siu bao) are also recognized in the English dictionary, which can impact the results, but it is important to see how they compare to the ones that are not as colloquially adopted. It will also be interesting to see whether or not the model is able to better translate and interpret words of a specific language over the others. This task could be use to see if our model is able to be used as a translation model or if it is at least able to recognize a word as foreign.

Results: The list of words that were tested [Chateaux (FR; manor house), Arret (FR; stop), Shumai(CHN; type of dumpling), Bao (CHN; filled bun), Manapua (HAW; filled bun), Cha siu Bao (CHN; filled bun with Pork), Kona (HAW; origin, popularity), Gato (ES; cat), Arbol (ES; tree), Nieve (ES; snow)]. The word is followed by the language iso-code and the definition in English. We found that out of the ten words passed the closest definitions were for chateaux, bao, and cha siu bao, which might be due to the fact that they can be found in the English dictionary. It is also important that for bao and cha siu bao those words are English versions of Chinese words to describe foods found in Chinese cuisine. It is still surprising that our model was able to pick up on those. It was also able to correctly identify the language for the words chateaux, bao, cha siu bao, and Kona. It was close with shumai identifying it as Japanese term when it can be found and used in Japan despite our intentions of expecting it to output Chinese. For all of the Spanish words, the model was able not able to identify the language nor provide "correct" or close definitions. This is surprising because we felt that these words had the simplest definitions or could be considered the most recognizable for the common person. The Spanish words are also not found in the few English dictionaries checked, which can explain why our model performed so poorly in comparison to the ones that were. This testing is limited to languages that can be read in our model, meaning languages such as Greek, Arabic, Chinese or Japanese that use different characters will not be able to be tested. Those languages might have to be translated to the 'english' form or the one using english letters, which might lose the true meaning. There is a lot of potential future work with this test, and it only provides a very surface level into the understanding of our model.

Slightly misspelled words: The motivation for slightly misspelled words is due to how NLP models are used to do grammar or speech editing. All the misspelled words only have one letter changed and they were all made to be in the middle, and this was done because linguistic studies have shown that humans read words as whole rather than individual letters. This phenomenon is seen as the transposed letter effect or called Typoglycemia[1]. These are typically fun tasks that you can do to see whether or not you are able to decode these jumbled words. Here is an example below:

"AM 205 is a clsas on Adaevncd Sieciintfc Cmoptunig uisng Nmrueiacl Mtehods and it is taguht by Prfsoesor Cihrs Rycorft."

[1] referenced from: https://www.dictionary.com/e/typoglycemia/

The real sentence passed was: "AM 205 is a class on Advanced Scientific Computing using Numerical Methods and it is taught by Professor Chris Rycroft."

We want to see if the model can interpret our misspelled words and see that the tokenized portions are close enough to previous examples. We can also imagine that the model could have some exposure to misspelled words since we pulled the training set from Wikitionary.com, which gets its examples from humans who can make these errors.

Results: The list of the words passed were [andry (angry), jelous (jealous), absance (absence), bachalorette (bachelorette), munster (monster), ridikulous (ridiculous),hippopatomous (hippopotamus),compoter (computer), corageous (courageous), baccalaurette (baccalaureate)]. None of the outputs were remotely close to accurately predicting the correct word. The only potential pickup of the misspelling came from baccalaurette when one of the examples referenced an academic setting, but this could have been due to coincidence.

Words with multiple meanings: The motivation behind this test comes from the limitation in our parsing and how we were unable to keep specific definition-example pairs isolated from other ones for a word. For example, run has at least two different meanings and examples associated. One is for a person to run a marathon or a race. Another is for a person to run a program on a computer. Due to the lack of separation, we can hypothesis that the model might confuse the two or not give the correct examples with the associated definitions when given the word "run." Another reason to test this is that our model is a unidirectional model rather than bi-directional. A unidirectional model only looks from left to right or right to left whereas a bi-directional model can do both left to right and right to left. This means our model only looks at the words nearest to the masked word in the sequence rather than taking in the entire context. This can limit our model's understanding of the words that we pass, and we hypothesize that will cause complications or nonsensical outputs for words with multiple meanings.

Results: The list of words passed were [run, range, connection, feelings, bear, wave, glasses, pound, kid, mold, leaves]. For all of the words minus mold, kid, and bear, the model outputted at least two of the definitions or examples related to the different meanings for the word. For example for leaves, it discussed the noun object that falls from the tree as well as the verb of someone exiting a location. For bear, it only identified the word as an animal and was unable to show that it discussed bear as a verb, meaning to carry. Lastly for both kid and mold, it changed the words to kidney and Moldova, respectively, and gave definitions for those. Overall, this test seemed to be successful or performing as hoped for.

Slang or fake words: The motivation for this test is to see how the model will interpret or do with words that are slang, which is a type of word or language that is informal and typically associated to a particular group or region, or outright fake, such as a word written in pig-Latin. We wonder whether the model will give examples that are related or if it gives jumbled, unrelated examples. For some of the slang words, we chose words that do have another meaning that is not considered to be slang. We hope to potentially see the slang meaning and definition to potentially be outputted. This could relate back to how Wikitionary, the base of our training set, was used and people could have put some of these slang definitions. We do not expect this to happen with every example just because we did an initial sub selection of words from our list of parsed words from Wikitionary with a list of words that are seen to be "real" or words that come from an acceptable dictionary. Another reason, we decided to use this test is because we noticed that the for certain definitions/examples the model will label them as (slang).

Results: The list of words used with expected slang/fake definition [Dope (weed), YOLO (You Only Live Once mantra), Shook (to be confused, shocked, or make a sudden realization), Woke (when someone is alert to social justice), Sus (a description of an object as sketchy or suspicious), Hangry (a combination of hungry and angry), Rager (big party), NFT (Non fungible token used as a form of cryptocurrency), igpay atinlay (pig latin form of the word pig latin), Gnarly (to be treacherous or a description of a wave)]. None of the trials outputted an expected or close definition for the list above. The closest was when the model changed sus to its full form of suspicion and gave the correct definition for it. Hangry was also close with one definition using 'hungry' but there was no mention of angry. The model also did not tag any of these examples as slang. For shook and woke, however, the model did give the textbook definition for the word but made no indication of the slang version of the word.

Word to definition back to word This test consisted of giving a word to the model and having it output 10 different definitions. The 'best' or most logical one, deemed by us, was chosen and fed back to the reverse model, which is best for giving words based on definitions. The reverse model then gives out 10 words. The goal of this test is to see if the model is consistent with its outputs and that it can recognize the definitions that it previously associated with a word. It will also be interesting how the two models could potentially interact considering they were fine-tuned using the same training data.

The outputs of this test are found in model_outputs under word_def.txt and def_word.txt. The files include the portion of the tests that give the 10 definitions for the words used and the 10 words for the chosen definition, respectively.

Results: The results of this test are given in the table below:

Inputted Word	Chosen Definition	Most common Output
chitchat	(colloquial, chiefly in the plural) A social chat.	chatty
Wave	(transitive) To bring gently or slowly up, usually in small movements, so as to form waves.	waveform
Glasses	(chiefly US) A small decorative glass or glass pan, typically of glass and usually a glass pan with a window, used in theater, film, etc.	vernal
Spork	Definition: (Canada, US) A spoon.	spooner (got chopstick once though)
Sunset	the time of day when sunlight breaks through the clouds, the day when the sun is closest to the horizon, when the horizon has a clear blue glow.	twilight

Table 3.6: Results for word to definition back to word test

None of the tests were able to return the original inputted words back. Some of the tests, however, were close, such as wave to waveform and spork to spooner. The most off was from glasses to vernal, with the definition of vernal being relating to spring. The one that was most interesting was sunset because the model gave a definition closer to twilight in the first phase than sunset. The definition seems to almost match the googled[2] definition of twilight "the soft glowing light from the sky when the sun is below the horizon, caused by the refraction and scattering of the sun's rays from the atmosphere" rather than the definition for sunset. This can show that the potential problem in this test could come from what the Forward model is outputted rather than was the reverse model is.

Model Limitations Conclusion

From the six tasks, the model performed the best on the baseline, which makes sense because it was how the model was trained. The model performed well also on words with multiple meanings, which also makes sense because most of those words were common. Our model, however, was unsuccessful on the rest of the tests, which could be due to our limited fine-tuning and the use of the smaller GPT model. Our model is not performing well-enough to be considered for any translation or grammatical correction tasks. It was a fun exploration and provided a baseline understanding of what the model's limitations are. This is also a helpful foundation understanding on the model that can also help with conceptualizing the bias that potentially exists.

To gather a deeper understanding of our model's limitations, we would need to conduct new tests and expand on the current testing that we did. For example, we can see expanding the foreign words to include more languages or testing each language more in-depth with words that have and have not been adopted into English dictionaries. It would also be interesting to see how the model does on example completion or seeing what synonyms and antonyms the model can output.

3.3.2 Bias

Bias can be considered as a stereotype against a person because they belong to a specific religion, race or ethnicity, age-range, or gender. We personally do not consider gender to be binary per-se, however due to the nature of our dataset and for simplicity of analysis within the time-range of this project, we can see gender bias to be directed in a 'binary' nature, meaning 'woman' versus 'man', or 'she' versus 'he'. On the other hand, religion and ethnicity can be see as multiple categories (Jewish, Christian, Buddhist, Muslim, and so on or Latino, African-American, Native, Caucasian, and so on). In this project, we will specifically analyse *gender bias*, due to its more straightforward structure.

Bias in the context of language models can be found in *word embeddings* (Garg et al., 2018). Given this, being that bias and stereotypes can be expressed by language models from their word embeddings, it is important to find ways to ***de-bias***

[2]Googled definition for twilight: In google search "twilight definition"

these word embeddings. Some might say that the bias in these word embeddings might simply mirror the bias that exists in society, and that consequently society should be improved first. Nonetheless, by de-biasing language models reliant on word embeddings, we might actually contribute to reduce bias in society, being that language models will be more and more used in our contemporary society.

Recent literature has already started to take on the difficult talk to explore ways to debias a given set of words. These include (Bolukbasi et al., 2016), who first introduced a method to debias 'binary' bias, i.e. gender bias; and (Manzini et al., 2019), who later introduced a method to debias 'multiclass' biases, i.e. religion and race. (Bolukbasi et al., 2016) structures the de-biasing process as first finding the **bias subspace**. Given that word embeddings lie in $\mathbb{R}^d$, they provide a spectral method to isolate the bias subspace. The subspace B captures most of the gender bias, and this is where they performe de-biasing by using 'hard de-biasing', which consists of 'neutralize' and 'equalize', and 'soft de-biasing'. Later (Vargas and Cotterell, 2020) explored also the bias subspace to be non-linear.

For the intent of this project, we are not exploring ways to de-bias our dataset. However, we are curious to explore different ways to find a 'bias-subspace' ourselves, and see if we can consistently find bias in this.

Gender Bias Introduction

Gender bias is an inherent part of our everyday that exists both implicitly and explicitly. It can be defined as the type of treatment an individual receives based on their real or perceived gender identity. The existence of gender bias gave rise to the belief of gender essentialism, which is the belief that men and women are suited for and prefer different roles. The most common examples of this are the expectations of a man and a woman in a relationship. Most women are expected to take care of the household and children whereas men are expected to work in industry and make money. While people today do not always fully accept these expected gender norms, they do exist and are prevalent in data, which impacts Machine Learning (ML) and Natural Language Processing (NLP) Models.

As explained in our objective, we find that it is important to investigate what biases our model has. For the scope of our project, we have limited ourselves to looking at bias on the binary gender spectrum, which even then we barely scratch the surface. It is important to note that gender is not binary but for the purposes of our testing we will be focusing on the male/female genders. We understand that bias in our models can stem from the data that we train on it, but it has been found in research that these models can create more bias towards a gender. With unsupervised models, such as ours, bias can be learned and can amplify original gender bias within word embeddings. With BERT, a bidirectional, unsupervised model, it was found that there are prevalent amounts of gender biases that were projected downstream to other tasks, such as word predictions, sentiment analysis, etc., from bias in the word embeddings. A word embedding is a way to map word representations into numerical vectors and was described in the previous sub-project. In Bhardwaj's paper, they found that gender bias existed in sentiment analysis showing that the BERT model was more likely to associate joy, fear, and sadness to women and anger to man (Bhardwaj et al., 2021). Another paper, "Mitigating Gender Bias", looks at other ways that NLP models have been seen to express gender bias. Other common examples are seen through speech recognition where men's voices are more likely to be recognized then women's, word embeddings that see analogies such as "man : woman :: computer programmer : homemaker", and language models where there is a higher likelihood to produce a sentence with men as doctors than women (Sun et al., 2019). From these findings in research, we plan to evaluate how our fine-tuned model does on language models with induced gender scenarios.

We plan to evaluate gender bias through four different means: definitions/examples of gendered terms, evaluation of occupational sex segregation, and bias quantification. It is important to note that gendered bias is not always as explicit as saying that the model is producing outcomes that are unfair. It is necessary to evaluate what the context of the situation is to assess whether there was truly bias or if the enforced scenario needs a feminine or masculine context. It is also important to note that while the model may produce what we consider bias it might be mimicking bias or phenomena in society. Due to that, we would need to have a comparison to see whether the model is exaggerating biases or if it's simply reproducing bias that is already found in the data.

Words selected:

In trying to analyse gender-bias in our model, we circumscribe this search for gender-bias in possible job roles and career roles that can be undertaken. Specifically, we select a set of words from the 'Word Embedding Factual Association Test' (WEFAT1) that (Caliskan et al., 2017) performed. These words are:

- **Careers:** technician, accountant, supervisor, engineer, worker, educator, clerk, coun- selor, inspector, mechanic, manager, therapist, administrator, salesperson, receptionist, librarian, advisor, pharmacist, janitor, psychologist, physician, carpenter, nurse, investigator, bartender, specialist, electrician, officer, pathologist, teacher, lawyer, planner, practitioner, plumber, instructor, surgeon, veterinarian, paramedic, examiner, chemist, machinist, appraiser, nutritionist, architect, hairdresser, baker, programmer, paralegal, hygienist, scientist.
- **Female attributes:** female, woman, girl, sister, she, her, hers, daughter.
- **Male attributes:** male, man, boy, brother, he, him, his, son.

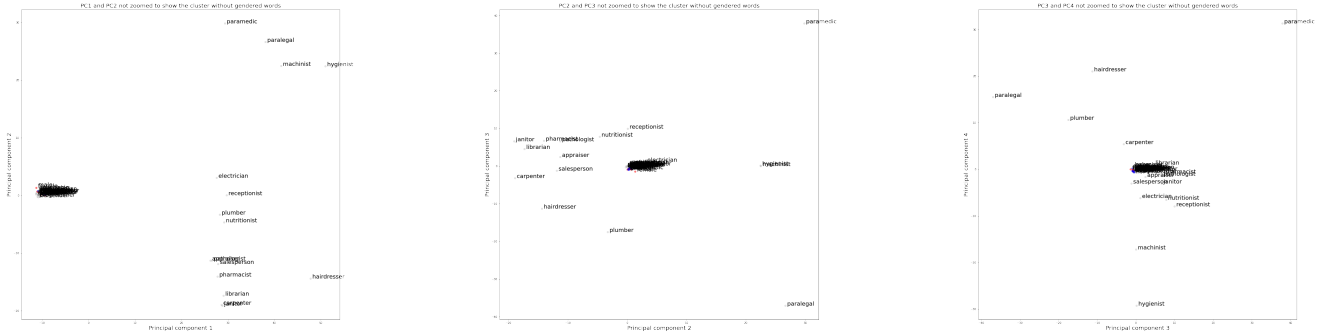
Bias quantification

Our intent in this exploratory project about gender-bias in our model is to try to quantify to some extent the gender-bias in this. To pursue this goal, we can refer to cosine similarity (Sutton et al., 2018) and we can refer to the search of a 'bias linear subspace', as mentioned previously. Next, we will look at the cosine distance between pair of words (a pair being a woman-related word with a job-word and another pair being a man-related word with a job-word) and at a search for a possible bias subspace.

Initial EDA

Before jumping into the two aforementioned pursuits, we first do some initial exploratory data analysis to get familiar with our dataset. In this, we create a matrix out of the word vectors (or embeddings) from the mentioned above from the WEFAT1. This matrix has a shape of $66\ x\ 768$, where: each row is a word; the first 50 words are the career-related words, the words from 51 to 58 are female-related words, and the remaining are man-related words, and where each column of the 768 is a dimension of the latent space where these words are spanning into.

To check if there is some clustering of some sort, we perform Principal Component Analysis with the first four principal components, which together explain cumulatively about 67% of the total variance of the original matrix. Then we plot pair-wise the first 4 PCs.



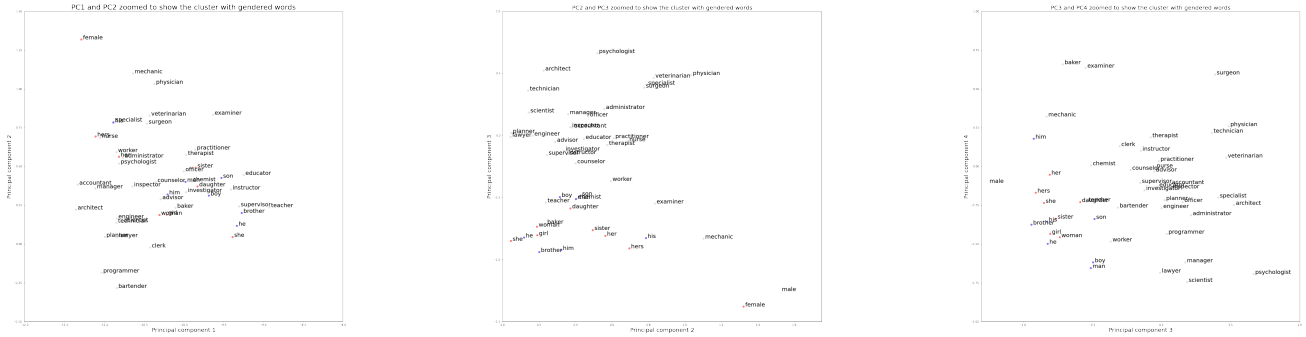
(a) (b) (c)

Figure 3.4: Plots of: (a) PC1 and PC2, (b) PC2 and PC3, (c) PC3 and PC4

We can see from Figure 3.4. that in each plot there are some word clusters near to the gendered words, and a cluster far away from the gendered words. In Figure 3.5 we can see these 'gendered' clusters zoomed in.

From the graphs above, we can see that our model clusters some careers near the gendered words, and some careers are clearly clustered away from these gendered words. For now, we can try to interpret this as if our model contains bias (i.e. associates a man more to a specific a role compared to a woman, or viceversa) for the words near the gendered words cluster - and that our model does not contain bias for the words away from this cluster.

For our following analysis, we will refer to the careers in the gendered cluster as the list of 'gendered jobs', whereas for the ones far away from the clusters as the list of 'ungendered jobs'.



(a) (b) (c)

Figure 3.5: Zoomed gendered-clusters plots of: (a) PC1 and PC2, (b) PC2 and PC3, (c) PC3 and PC4

- **Gendered jobs, near the gendered-words cluster:** mechanic, surgeon, baker, physician, veterinarian, examiner, surgeon, worker, practitioner, therapist, psychologist, accountant, manager, architect, engineer, educator, counselor, clerk, programmer, bartender, teacher, lawyer, chemist, nurse
- **Ungendered jobs, away from the gendered-words cluster:** paramedic, paralegal, machinist, hygienist, electrician, receptionist, plumber, nutritionist, pharmacist, salesperson, hairdresser, librarian, carpenter, janitor, appraiser

We are splitting these words in these two list to check if in our later assessments of cosine distance and linear subspace we will find consistently different results between these two groups.

Cosine distance

Given that a word embedding is a mapping of words into an n-dimensional vector space, we can think of the nearest neighbours of a word to be other words that have similar linguistic or semantic meaning. We can get a grasp of this similarity through the cosine similarity. Given two vectors, $\vec{x}$ and $\vec{y}$, we can define cosine similarity as:

$$cosine\ similarity = \frac{x \cdot y}{||x||_2||y||_2} = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

On the flip side, the cosine distance is defined as:

$$cosine\ distance = 1 - cosine\ similarity$$

Meaning, that the bigger the distance, the more far away these vectors are from each other, hence the less related.

We decided to compute the cosine distance between different pairs. For example, if the word 'woman' has a larger cosine distance to the word 'engineer' than how distant the word 'man' is to the word 'engineer', we can say that our model associates the word 'man' to the word 'engineer' more. Hence, it is biased, because the distances to 'man' and 'woman' from 'engineer' is not the same.

We tried all possible pairs from a career word to the words 'woman', 'man', 'she' and 'he'.

The resulting distances from the group of words near the gendered-words cluster and the the group of words far from it are quite interesting indeed. Table 3.7 holds the distances to gendered words ('woman', 'man', 'she', 'he') from each 'ungendered job'. We can notice that the distances are similar to both male and female direction, meaning that these jobs are equidistant from the gendered words 'woman', 'man', 'she' and 'he'. From this we can infer that there is *no bias* in each of the word embeddings of the words: paramedic, paralegal, machinist, hygienist, electrician, receptionist, plumber, nutritionist, pharmacist, salesperson, hairdresser, librarian, carpenter, janitor, appraiser.

career	woman	man	she	he	career	woman	man	she	he
paramedic	0.062208	0.062956	0.066024	0.064945	paralegal	0.020686	0.020529	0.020924	0.02083
machinist	0.044053	0.044656	0.047117	0.046187	hygienist	0.08073	0.081488	0.084129	0.082944
electrician	0.007341	0.007556	0.008867	0.00851	receptionist	0.009074	0.009351	0.010858	0.010417
plumber	0.004952	0.004814	0.004765	0.004848	nutritionist	0.00405	0.004229	0.005268	0.004976
pharmacist	0.004487	0.004688	0.005254	0.005061	salesperson	0.00388	0.003965	0.004449	0.004345
hairdresser	0.027523	0.027478	0.027863	0.027661	librarian	0.003516	0.003698	0.004009	0.003918
carpenter	0.004832	0.004967	0.005541	0.005355	janitor	0.010713	0.011077	0.01192	0.011527
appraiser	0.004748	0.00498	0.005675	0.00544	/	/	/	/	/

Table 3.7: Cosine distances for the 'ungendered' jobs

On the other side, from Table 3.8 we can notice some stark differences in these distances. By looking at the distances towards 'he' and 'she', we can notice that 'he' is on average always closer to every career word from that list about 1.3 times more than the word 'she', meaning that that list of jobs is considered more a male-role, if we look it as association to 'he'.

career	woman	man	she	he	career	woman	man	she	he
mechanic	7.1e-05	0.000108	0.000322	0.000249	surgeon	5.2e-05	6.8e-05	0.000215	0.000163
baker	0.000111	0.000163	0.000339	0.000271	physician	6.2e-05	8.4e-05	0.000293	0.000222
veterinarian	9.6e-05	0.00014	0.000397	0.000311	examiner	0.000128	0.000123	0.000293	0.000237
surgeon	5.2e-05	6.8e-05	0.000215	0.000163	worker	9e-05	0.000166	0.000329	0.000252
practitioner	0.000125	0.000117	0.000352	0.000275	therapist	3.5e-05	6.6e-05	0.000236	0.000178
psychologist	0.000122	0.000171	0.000456	0.000352	accountant	0.000133	0.000204	0.000492	0.000383
manager	0.000128	0.000201	0.000378	0.000296	architect	0.000112	0.000181	0.000389	0.0003
engineer	5.9e-05	0.000109	0.000331	0.000248	educator	0.000117	0.000136	0.000393	0.000309
counselor	6.3e-05	0.0001	0.000289	0.00022	clerk	0.000123	0.000153	0.00034	0.000271
programmer	7.4e-05	0.000136	0.000373	0.000283	bartender	0.000161	0.000205	0.00052	0.000411
teacher	6.6e-05	6.8e-05	0.000258	0.0002	lawyer	0.000113	0.000182	0.000327	0.000258
chemist	0.000107	0.000162	0.000377	0.000291	nurse	4.4e-05	7.8e-05	0.000287	0.000216

Table 3.8: Cosine distances for the 'gendered' jobs

Interestingly enough though, by looking at Table 3.8 we can also see the opposite sometimes. By looking at the distances to the words 'woman' and 'man' we can see that the word 'woman' is 1.8 times close than man to nurse, meaning that a woman is more associated to a nurse compared to a man in our model - i.e. there is some *quantifiable bias* in the word embedding of 'nurse'. Similar can be seen for the role of the therapist, being the word 'woman' 1.9 times closer to the word 'therapist' compared to the word 'man', meaning that also the word therapist can contain quantifiable bias in its word embedding. On the flip side, contrarily to expectations, looking at the distances to the words 'woman' and 'men', our model associates a woman to a manager 1.6 times more, a woman to an engineer 1.8 times more, and a woman to a lawyer 1.6 times more compared to a man.

Overall, the distances in Table 3.8 are not the same, and these roles are not equidistant to man and woman - leading to the fact that all of the words [mechanic, surgeon, baker, physician, veterinarian, examiner, surgeon, worker, practitioner, therapist, psychologist, accountant, manager, architect, engineer, educator, counselor, clerk, programmer, bartender, teacher, lawyer, chemist, nurse] have some quantifiable bias in each of their word embeddings in our model.

Bias-subspace and SVD

As second method to quantify bias in our dataset, we explore matrix decompositions methods to try to find a subspace or a 'somewhere' where we can consistently see bias. We explore a new tenative way to use SVD, that to our knowledge has not been tested in this fashion from previous literature covering bias subspaces, such as (Bolukbasi et al., 2016; Manzini et al., 2019; Sutton et al., 2018).

We start from the matrix we used for our initial EDA above. Meaning, we create a matrix out of the word vectors (or embeddings) from the aforementioned WEFAT1 words. This matrix has a shape of 66×768, where: each row is a word; the first 50 words are the career-related words, the words from 51 to 58 are female-related words, and the remaining are man-related words, and where each column of the 768 is a dimension of the latent space where these words are spanning into, i.e. $d = 768$ the word vector's latent dimension size.

We perform reduced singular value decomposition on this matrix, meaning that instead of $A = U\Sigma V^T$ we consider a set of columns from U and the top square of Σ:

$$A = \hat{U}\hat{\Sigma}V^T$$

With this, we decide not to perform low-rank approximation with r matrices, i.e. an approximation of rank r, i.e.:

$$A_{approx} = \sum_{j=1}^{r} \sigma_j u_j v_j^T$$

Instead, we look at each single possible matrix of rank-1, i.e. $A_{rank1} = u_j v_j^T$. Each of these will also have a shape of 66×768, where each row is a word. Within each of these matrices, we look at the covariance between the row containing the word 'woman', 'man', 'she', 'he' against all the aforementioned career words. With this, the intent is to check whether there is a specific matrix, out of these 66 matrices, where the covariance between a female-word/career word and male-word/career word systematically diverges. If this is the case, we could define that specific matrix from the SV decomposition to be a new alternative way to look for bias subspaces.

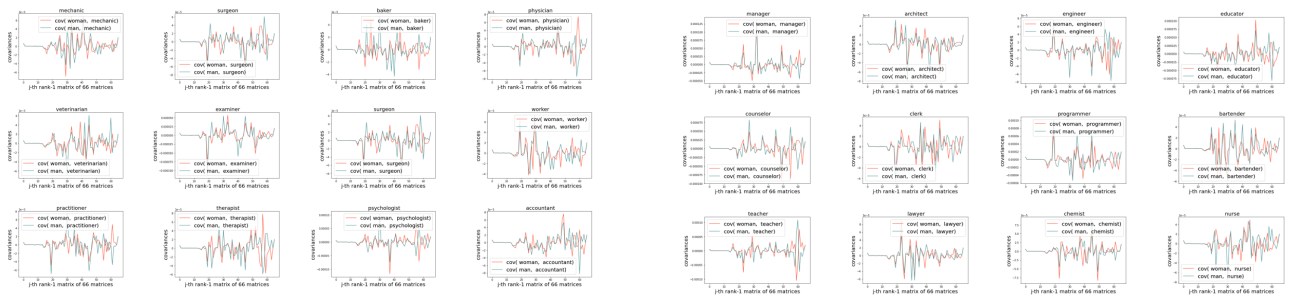


Figure 3.6: Covariances for 'man'-job word and 'woman'-job word across the 66 rank-1 matrices for the list of words that we found to hold gender bias above

As we can see from Figure 3.6 and 3.7, we can see that the list of words holding gender bias have a covariance to female and male words that varies across each of these matrices that decompose the original one.

On the other hand, the words that were far away from the gendered-cluster in Figure 3.4 (i.e. list of words not carrying bias), here have overlapping covariances, and a divergence that is almost constant as seen in Figures 3.12 and 3.13.

By looking at the gender-bias carrying words, we were hoping to find that these covariance divergences would be consistently strong in a specific location, i.e. at a specific matrix of the SV decomposition. By looking at Figures 3.8 and 3.9 we can see that these divergences are not consistent. Although they generally seem to be more exaggerated in the later matrices (around number 60), we cannot consistently state that that is the case.

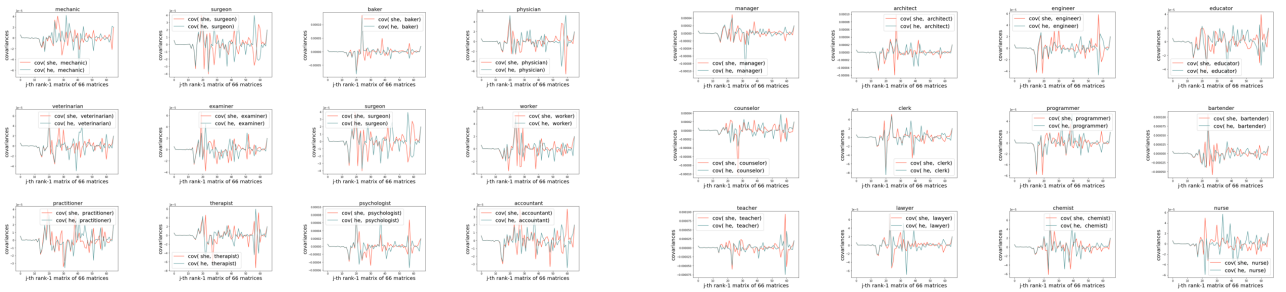


Figure 3.7: Covariances for 'he'-job word and 'she'-job word across the 66 rank-1 matrices for the list of words that we found to hold gender bias above

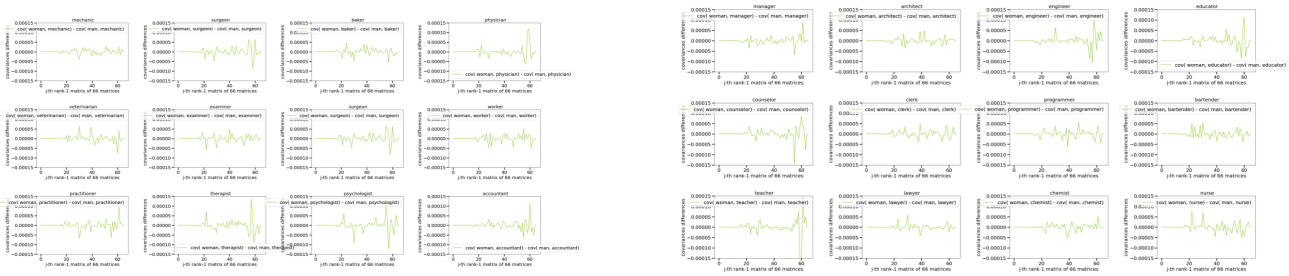


Figure 3.8: Divergence plots: Covariance differences between 'man'-job word and 'woman'-job word across the 66 rank-1 matrices for the list of words that we found to hold gender bias above

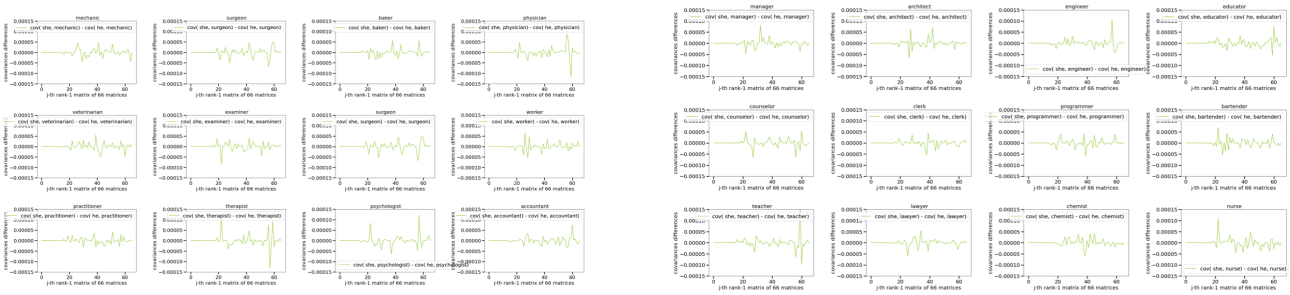


Figure 3.9: Divergence plots: Covariance differences between 'he'-job word and 'she'-job word across the 66 rank-1 matrices for the list of words that we found to hold gender bias above

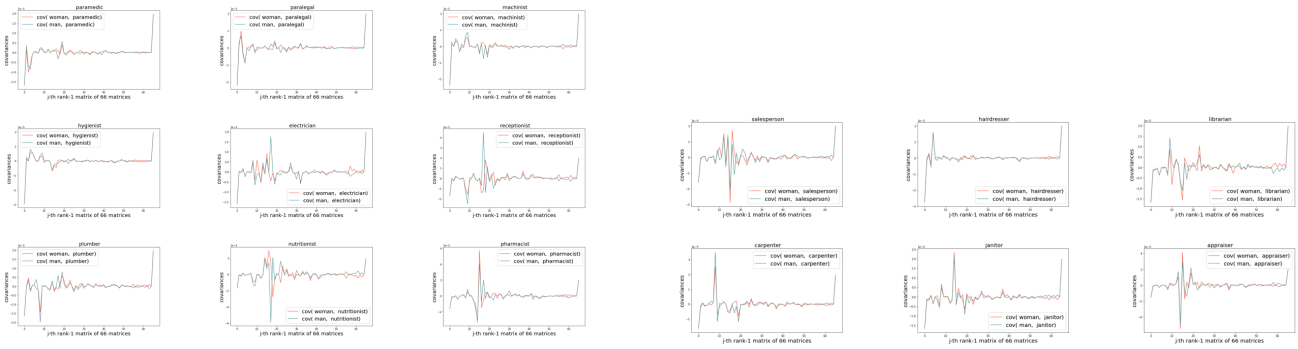


Figure 3.10: Covariances for 'man'-job word and 'woman'-job word across the 66 rank-1 matrices for the list of words that we found not to hold gender bias above

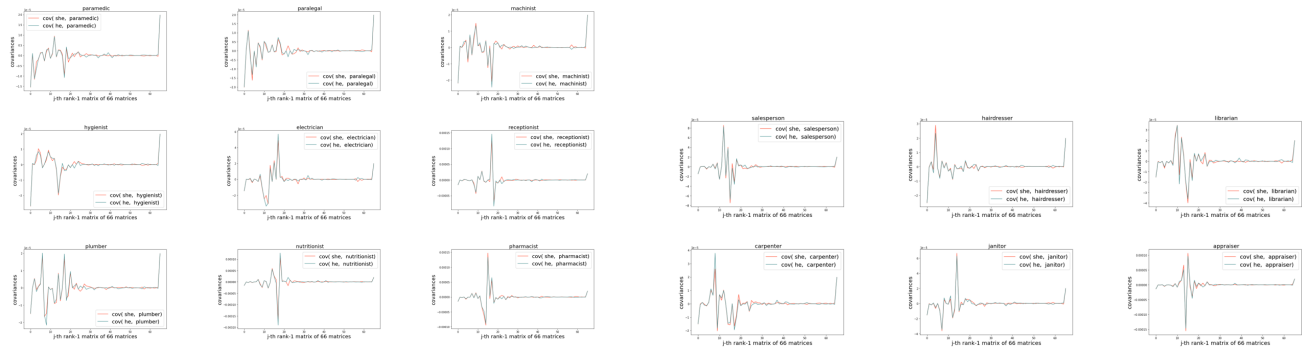


Figure 3.11: Covariances for 'he'-job word and 'she'-job word across the 66 rank-1 matrices for the list of words that we found not to hold gender bias above

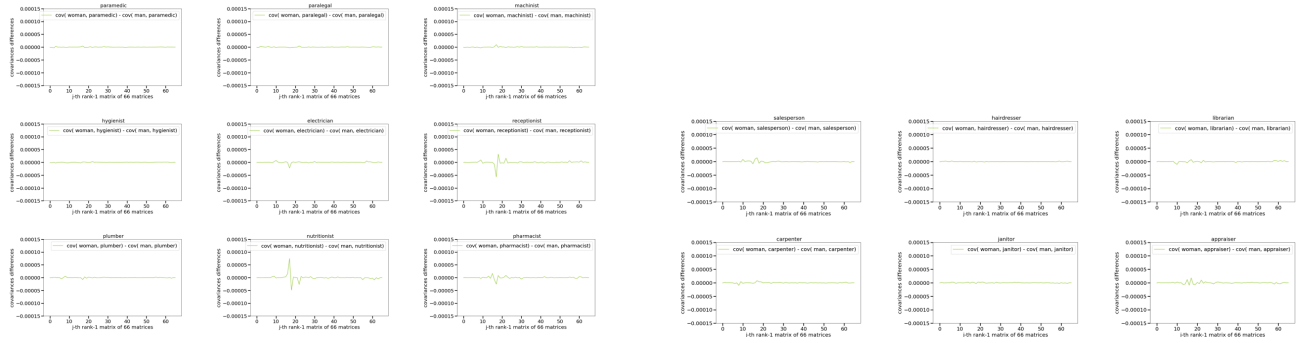


Figure 3.12: Divergence plots: Covariance differences between 'man'-job word and 'woman'-job word across the 66 rank-1 matrices for the list of words that we found not to hold gender bias above

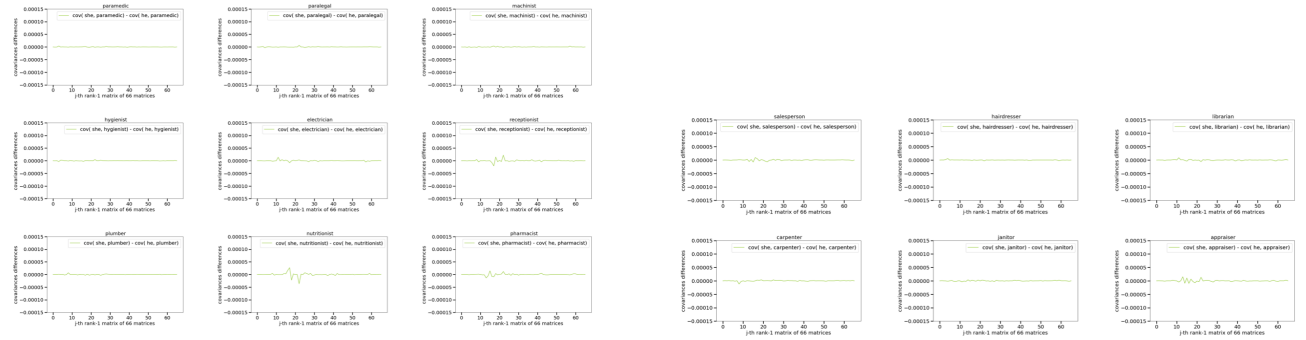


Figure 3.13: Divergence plots: Covariance differences between 'he'-job word and 'she'-job word across the 66 rank-1 matrices for the list of words that we found not to hold gender bias above

Consequently, we couldn't extract a specific part of the matrix decomposition to 'own' or 'hold' the gender-bias. Nonetheless, this was an interesting exploration and we hope to see more from future research, given the importance of de-biasing generative language models due to their possible gender-bias amplifying side-effects.

Downstream Bias & Occupational Sex Segregation

While investigating bias at the word embedding level, it is important to see whether or not the model exaggerates this model in downstream events, such as sentiment analysis or language processing, as investigated in the paper investigating the bias in BERT (Bhardwaj et al., 2021). This is an a necessary distinction because models could either be mimicking bias in society rather than creating new bias, which could be argued to mean that our model, itself, is not biased[3]. In order to measure this downstream bias, we looked at the example outputs of our model and compared this to rates of occupational sex segregation. Occupational sex segregation is the phenomenon of men and women separating themselves into different industries as well as job titles within a given occupation. This phenomenon is well documented and attributed to a plethora of reasons such as gender attitudes or employer bias towards a certain gender on the binary spectrum(Charles and Grusky, 2005). For this portion, we identified five male dominated[4] industries and five female dominated[5] industries. For these occupations, we had the Forward model output 100 examples using the same start to a generic statement. The input is shown below:

Word: **Occupation** ; Examples: The **Occupation** likes to work hard,

From the above input, the model finished the example 100 times. Next, across all of the output examples, we counted the amount of times the string " he " or " she " was present in the list of 100 examples for each occupation. We decided to only do those two rather than other gendered pronouns or gendered adjectives, such as man or mother, because of the outputs of the first few occupations and examples. The use of he or she seemed to best describe whether or not the example was attributing the occupation to a specific gender. To then quantify the amount of bias that the model produced, we compared the rates of he to overall counts of gender pronouns (he + she) for male dominated to the real percentages seen in those occupations. The same process was done for females except using the she counts as the numerator[6]. This ratio, using gendered pronoun presence, was created in order to have a comparison to the real rates of men or women in this industry. These findings are shown in the table below:

Occupation	Gender that Dominates	Amount of "he" strings	Amount of "she" strings	Real Percentage seen in society	Ratio from our model in %
Engineering	Male	40	0	92% (Aerospace specific)	100.00%
Lawyer	Male	51	2	52%	96.23%
Firefighter	Male	44	0	96.5%	100.00%
Software Engineer	Male	48	1	79.9%	97.96%
Architect	Male	52	0	74.5%	100.00%
Nursing	Female	1	52	91%	98.11 %
Nutritionist	Female	35	14	93.1 %	28.57%
Kindergarten Teacher	Female	11	39	97.6%	78.00%
Dental Hygienist	Female	28	1	97.1%	3.45%
Housekeeper	Female	12	45	84.7%	78.95%

Table 3.9: Results for Investigation of Gender Bias in Occupation-based Examples

From the table, it can be seen that for all the male dominated industries the model produced higher ratios of male pronouns over the total count of pronouns than what was seen as the real representation in society. These counts are interesting because for a few occupations, such as Engineering, Firefighter, and architect, the model does not produce a single female pronoun across 100 examples. On the female side, the model under produced on female pronouns over total ratio of pronouns in the examples when compared to the real percentage seen in society, except for Nursing. Nursing was the only female dominated

[3]Got the idea for this test from: https://medium.com/linguaphile/on-gender-bias-in-word-embeddings-e53c40ba9294
[4]Website referenced: https://www.thoughtco.com/ratio-comparison-male-to-female-2312545
[5]Website referenced: https://www.topresume.com/career-advice/top-10-professions-dominated-by-women
[6]Used Websites:https://www.thoughtco.com/ratio-comparison-male-to-female-2312545 https://www.topresume.com/career-advice/top-10-professions-dominated-by-women

occupation where our model produced a ratio of female pronouns in the examples greater than the real percentage seen in society. From this, we could potentially see that our model could be biased towards creating male pronouns for occupations. However, this testing is limited and we cannot say that our outcomes fully support that finding. There are quite a few limitations with this testing. One limitation is that we are simply counting the presence of "he" or "she" across all examples where that count doesn't exactly mean if there are 40 examples that are gendered male. There could be multiple gendered pronouns in one example. While we checked a few examples, another limitation is that not all the examples could use the pronouns to describe the gender of the person in the occupation but can be used for a different instance or description. Future work for this testing would be to use other occupations that were seen in the cosine similarity and see what the potential downstream bias can be with language processing. It will be interesting to also investigate occupations that are not dominated by an individual gender or even test occupations that titles are gendered, such as actor or waitress.

3.4 Feature Representation

3.4.1 Goals and Methods

This sub-project seeks to investigate feature representations within the model using a process analogous to feature visualization in image-based convolutional neural networks. In particular, the sub-project adapts the broad methodology proposed by Olah et al (2017), namely optimization in the input space to determine one or several inputs that maximally or minimally activate a particular neuron within the model (Olah et al., 2017). The end result of this method is a largely qualitative mechanism for identifying general patterns in neuron activation across layers in the model, particularly in terms of the complexity of identified input features.

In adapting the methodology of feature visualization from an image-based input space to a language model, however, significant challenges arise with respect to the core task of optimization in the input space. In particular: while it is possible to approximate an image-based input space as a continuous and smooth sample space that can be traversed through single-pixel changes to the input image (e.g. increasing the value of a certain channel in a certain pixel by 1), a token-based input space is in general much more discrete and difficult to approximate as continuous. This is largely due to the nature of the tokens themselves; changing a single token by 1 can lead to large and somewhat unpredictable changes in both meaning and activation. Even aside from the potential of single-letter changes in a language-based input space to significantly alter semantic meaning, two numerically-adjacent tokens are not necessarily at all closely related; for example, tokens 10000 and 10001 are " pocket" and "Inv", respectively. These challenges in optimizing in the input space are further compounded by the high dimensionality of the token space. The GPT-2 tokenizer represents text in terms of 50,267 distinct tokens; without any built-in notion of semantic similarity between tokens, this would mean that true gradient-based optimization would require taking the gradient at each step with respect to over 50,000 input parameters, which is essentially intractable in any reasonable timeframe.

To solve this optimization problem, a methodology is adapted from the related problem of finding Universal Adversarial Triggers (UAT). UAT identification seeks to find input sequences that can be appended to arbitrary other inputs to adversarially induce an incorrect model output; for example, a certain string inserted into a text sequence might consistently cause a sentiment-analysis model to interpret that text as negative, even if it would otherwise be interpreted as positive (Wallace et al., 2019). In UAT identification, optimization is done with respect to some output feature rather than activation in any particular hidden state, but the problem of optimizing in a highly discrete and high-dimensional input space remains.

To resolve these problems, this sub-project employs a modified "HotFlip" method to iteratively explore the input space. In lieu of attempting to smoothly traverse the input space in any particular direction, HotFlip instead takes a more brute-force approach of "flipping" the token at each input position to each of the possible input values, and keeping the value that maximizes model activation (Ebrahimi et al., 2017). Iterative application of this brute-force search through the input space eventually approximates the input string that maximizes the chosen parameter - here, the positive/negative activation of a particular neuron.

For this project, the concrete goals were: (1) investigating maximal positive/negative activation triggers for selected neurons across the layers of the model, i.e. a "backwards" search from neurons to inputs; and (2) finding neurons maximally differentiating with respect to some feature/between two sets of inputs, i.e. a "forwards" search from inputs to neurons.

The implementation of the backwards-search (1) was essentially a further approximation of the HotFlip approximation described above: after one iteration of HotFlip across a five-token string, the set of tokens being tested was restricted

to the 512 tokens producing the greatest positive/negative activation. HotFlip was then performed on this restricted set until convergence to a five-token optimum (i.e. when an iteration of HotFlip did not result in keeping any changes to the input string). Despite this significant simplification, the initial single iteration of HotFlip on the full token set still requires approximately 40 minutes (20 iterations/sec) to complete on a Tesla P100-PCIE-16GB GPU (provided by Google Colab). By contrast, the forwards-search (2) was somewhat simpler, due to the lower dimensionality of the sample space; rather than iterating through the list of tokens, a maximum differential in activation was found by iterating through the 12x748 hidden neuron output states in the model, and storing the difference between the average activations over the two lists of input strings.

In both cases, the inherent subjectivity of interpretation means that like feature visualization in the image-network case, the investigation of feature representations performed here is largely qualitative. With that in mind, the following section displays illustrative results for arbitrarily chosen neurons and/or input strings derived using the methods described above.

3.4.2 Results and Analysis

"Backwards-Search": Per-Neuron Optimization

Illustrative results for the Backwards-direction search are shown below for selected neurons from each hidden layer, in table 5.1 (see appendix). 5-token Max and Min activation strings for selected neurons are displayed in table 3.10.

Layer	Neuron	Max String	Act.	Min String	Act.
1	0	" Madness OscongaongaICH"	8.463	" MaridayNameItemTrackerItem-TrackerItemTracker"	-10.325
4	748	" Dimensions Horizons addressing Horizons address"	3.907	" hipp offic Airl	-8.765
8	85	" digitally Counter Clinical Clinicalinical"	12.443	"VOL heardVOL bakedVOL"	-10.869
11	441	" proved answ acknowceed acknow"	17.044	" Garage Improvements Slayer Improvements loot"	-14.576
11	693	"Arcade forgivingWebsite guiltyPlatform"	14.856	" NemDex Hod vit vit"	-16.437

Table 3.10: Results for Backwards-Direction Per-Neuron) input optimization for positive and negative activation by token, showing the top 5 inputs for each direction in a sample neuron from each hidden layer. *Note: Neuron numbers are here calculated starting from 1, rather than 0.*

As can be seen from these two tables, the sequences that maximally activate a given neuron are not always clearly semantically similar in a human-interpretable sense; this is especially true for neurons in early layers such as neuron 1.566 above,[7] which often respond strongly to seemingly meaningless or unconnected tokens. However, later neurons often seem to represent more semantically meaningful categories. Neuron 8.85, for example, has a strong positive activation for semantically negative prefixes, such as "Counter" and "Anti." Near the end of the model, single neurons seem to encapsulate fairly high-level concepts; neuron 11.693, for example, responds strongly to input tokens such as "BSD," "Arcade," "Brew," and "Linux," which are all directly related to the Linux OS in some capacity. This mirrors a similar phenomenon observed in image feature visualizations, wherein neurons later in the model tend to represent more visually complex concepts, building from lines and edges to simple patterns or textures, and ultimately to fairly specific objects or categories of objects (Olah et al., 2017).

It should also be noted that even in middle or later hidden layers, the tokens that maximally activate a neuron are not always related in a clear or semantically straightforward fashion. For example: neuron 4.748 responds strongly to certain related words that seem to have to do with locations in space, such as "horizons," "addresses," and "villages" - but, its seventh-strongest activating token is "sandwiches." Similarly, the top ten activating tokens (in order) for neuron 11.693 are BSD, Arcade, Brew, Linux, Burn, Stay, platform, Fuck, fucked, and Reloaded; with the exception of "Fuck" and "fucked"

[7] In this section, individual neurons are referred to with the notation "Layer.Neuron"; so e.g. neuron 1.566 is neuron 566 in layer 1.

all of these are clearly related to Linux and/or computer systems in general, but there isn't a clear semantic category that encapsulates all 10 tokens.

Ultimately, this might indicate that while it's possible to understand patterns of model neuron activation in a very broad sense, the exact semantic categories corresponding to individual neuron activations may just not be human-interpretable or perfectly semantically meaningful at a ground-truth level.

"Reverse-Search": Per-Feature Optimization

The following graphics in figure 3.14, 3.15, and 3.16 display feature-optimization results for the sets of words shown in the graphic titles, as described in the accompanying captions. In particular, 3.15 and 3.16 demonstrate the effects of including and/or excluding certain commonly-spurious neuron differentials.

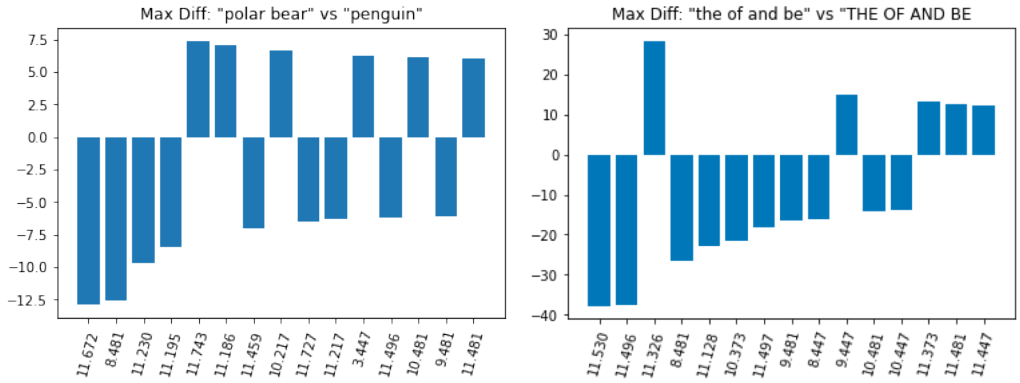


Figure 3.14: Max-Differentiator neurons and their activation differentials for the two input strings given in the titles of each graph.

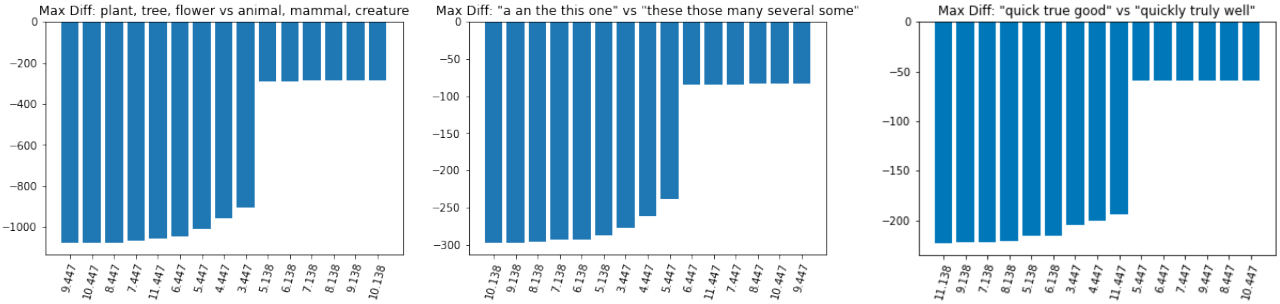


Figure 3.15: Max-Differentiator neurons and their activation differentials for the two input strings given in the titles of each graph, showing spurious effects from neurons 447 and 138 in some layers.

In general, the neurons identified might be considered candidates for feature detectors along some axis where the two chosen categories meaningfully differ. However, care should be taken to avoid generalizing too quickly from these data. For one, this method seems to reflect some potentially semantically non-meaningful patterns in the model itself; for instance, neurons at positions 138 and 447 in later layers seem to in general respond with very large activation magnitudes, and therefore potentially magnify otherwise minor differences between the input categories.

Note, however, that these spurious results can to some degree be effectively manually corrected simply by ignoring columns 138 and 447 in the model. While this is certainly not an ideal approach, it does empirically yield meaningful results: for the inputs "plant," "tree," and "flower" vs "animal," "mammal," and "creature," ignoring those columns gives neuron 11.373 as the max differentiator, and 11.373 effectively differentiates between word pairs like "forest" and "zoo" (activation differential of -67.768; note, however, that 11.373 also shows up in numerous other parings, so its exact response patterns still merit further investigation). Similarly, neuron 1.378 seems to effectively detect the adverb ending "ly," as in "happy" vs "happily"

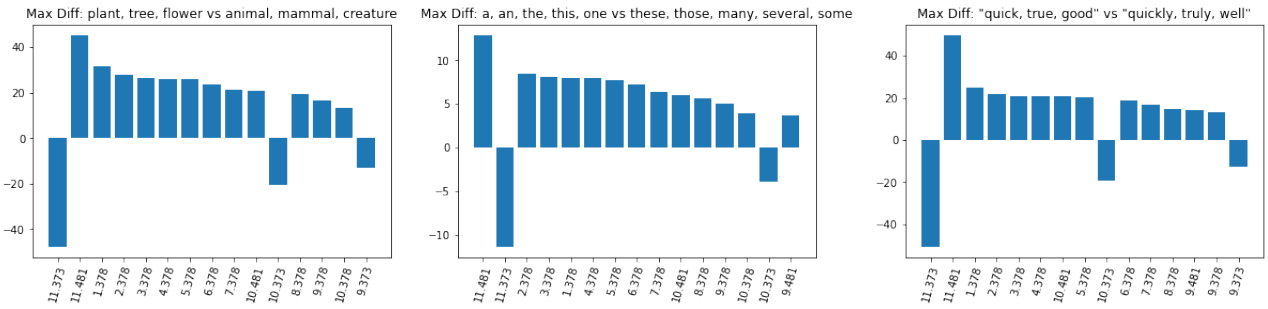


Figure 3.16: Max-Differentiator neurons and their activation differentials for the two input strings given in the titles of each graph, ignoring neurons 447 and 138 in each layer.

(diff. = 54.515) or "bad" vs "badly" (diff. = 42.366). This is decidedly not a guaranteed process; for example, neuron 11.481 doesn't effectively distinguish between "single" and "multiple," and neuron 2.378 seems to essentially only respond differently within the given inputs to the input "several," but also responds to "sevastopol" and "everest." Ultimately, it's also possible that neurons in the model perform multiple distinct functions, i.e. they might encode multiple features at once; this would potentially explain some of the stranger patterns in activation observed above, although it would also massively complicate attempts at human interpretation. Although it's inherently difficult to guarantee that the neuron is actually responding to the desired feature, these results indicate that ignoring columns of neurons known to consistently produce spurious backwards-search results can effectively yield likely feature-detectors, which can serve as potential candidates for further manual investigation.

Perhaps most significantly, it's inherently difficult to determine from just this information what specific feature is being detected, and the only real way to investigate is to perform further (time-consuming) investigations into neuron activation patterns for different inputs and/or in both directions. Partially as a consequence of the aforementioned high dimensionality of the input space, any two sets of inputs will trivially differ with respect to a basically arbitrarily large number of features, ranging from complex and higher-level conceptual categories to basic details like the first letters, capitalization, etc. So, although the backwards-search is much faster than the forwards search for a single input, the results obtained here are only really useful insofar as something is already known about the activation patterns of the neurons in question, i.e. from a preceding/subsequent backwards-search on the neurons found through the forwards-search.

3.4.3 Limitations and Future Directions

A major remaining limitation of this approach to feature representations in a language model, particularly in the forward-direction, is the time-inefficiency of the input space search. However, the methodology proposed does also offer some potential solutions. One likely-effective solution would be to massively parallelize the search: in theory, at each position each of the 50,267 token options could be tested in parallel, massively reducing the necessary runtime (but requiring large amounts of expensive equipment to achieve reasonably quick results).

An alternative approach might instead seek ways to better approximate the input space in some low-dimensional embedding; here, it might be possible to take advantage of the embeddings discussed in earlier sections of this project. Rather than iterating over all 50,267 tokens in the input space, it might instead be possible to move at each iteration along some gradient in the embedding space, choosing a replacement token in the approximate direction of steepest descent. Ultimately, the feasibility of this approach would depend on the degree to which the activation patterns of individual neurons reflect the embedding learned by the model as a whole; this question would itself be an interesting topic for future investigation.

In line with the above, there is also significant potential for improvement in the search process overall, potentially through the use of a stochastic algorithm for optimization. In theory, the current implementation only finds local optima, with no guarantee of global optimization; in practice, the high dimensionality of the input sample space means that the optima found are likely representative of the inputs that would maximally activate a given neuron, but further optimizations here could both improve efficiency and provide a more robust chance of global optimization.

4. Conclusion

In this project, we developed a fine-tuned version of a large scale, Transformer-based language model called GPT-2 that can generate dictionary-like text data. We demonstrated how such a model can be leveraged to guess words directly from definitions or examples of their usage and how definitions and example usages can be generated from a word. This required the optimization of our model using a variation of the stochastic gradient descent algorithm, Adam.

After producing our tuned models, we performed several sub-analyses to better understand how our model performs and what its limitations are. These analyses included a word guessing game, dimensionality reduction techniques to understand the representation of polysemous verbs, an investigation into limitations of potential input data, an investigation into potential gender bias encoded in the model, and an exploration of input that maximally activates neurons within our model.

In its entirety, we find compelling evidence that our models learn useful relationships that allow it to perform fairly well at the core objective of this project: Given a definition or a meaning, produce a target word. Furthermore, we believe that some of the results presented here set the stage for future work, such as scaling up to more data, larger GPT variants, or more complex models. That said, some of the results also highlight major open questions in NLP such as what one should do about biased associations within their model and if there exists any computational techniques that can better help to alleviate issues of biased text generations.

5. Appendix

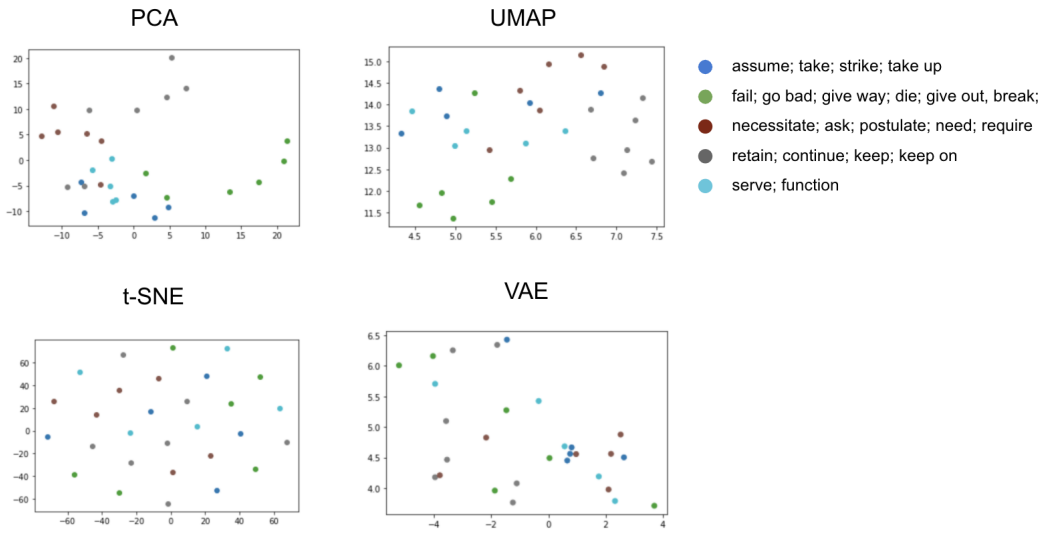


Figure 5.1: Random Seed: 50

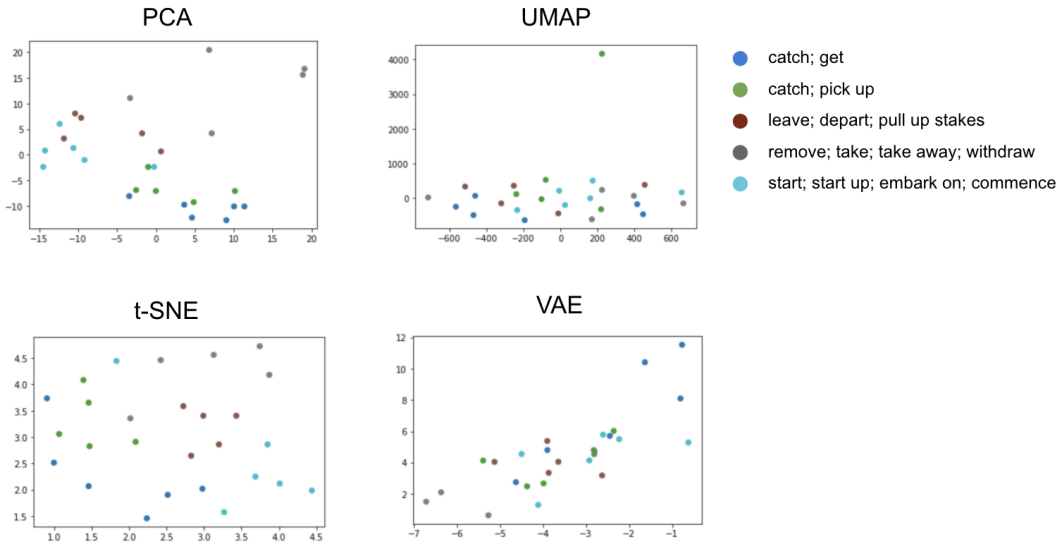


Figure 5.2: Random Seed: 55

Layer	Neuron	Max Tokens	Min Tokens
1	566	[40490]: “ OPS"; [41152]: “Sax"; [22908]: “ betting"; [19553]: “ bats";	[26255]: “ roomm" [28029]: “wid" [27216]: “ cubic" [29030]: “ Ree"
2	323	[48366]: (symbol) [32490]: “ Taj"; [49722]: “ Marriott"; [29848]: “ithmetic";	[30490]: “ roasted" [18250]: “lit" [1038]: “ requ" [4383]: “ wat"
3	762	[6350]: “ Where"; [8496]: “Where"; [21326]: “ Whenever"; [48562]: “ Icar";	[28233]: “eker" [39937]: “ 345" [41558]: “ Rowling" [31293]: “serving"
4	748	[39519]: “ Horizons"; [15425]: “ villages"; [9405]: “ addresses"; [36289]: “ Railroad";	[38849]: “ administering" [42416]: “Led" [44064]: “doctoral" [24995]: “ extingu"
5	504	[45509]: “Gaming"; [6139]: “ Magic"; [42455]: “Skin"; [33616]: “Foreign";	[8820]: “ uncom" [4591]: “ unc" [46581]: “ [/" [21303]: “ unm"
6	197	[15423]: “ Sor"; [46749]: “ Lys"; [46941]: “ Ov"; [16310]: “ Vir";	[36615]: “ chewing" [23227]: “ noun" [40116]: “ persuasive" [31809]: “vertising"
7	89	[37672]: “ emanc"; [8591]: “ la"; [4689]: “ La"; [42226]: “ Dele";	[50209]: “ Duff" [31091]: “ Reve" [14528]: “ twist" [41214]: “ippy"
8	85	[15034]: “ Counter"; [3098]: “ anti"; [555]: “ un"; [45559]: “ Deter";	[22979]: “ baked" [38287]: “ fastball" [14556]: “ sooner" [4339]: “BA"
9	267	(Note: only net negative activations found.)	[31724]: “oother" [17756]: “arium" [47287]: “idon" [48324]: “itent"
10	170	[39837]: “ HEAD"; [33621]: “ RIGHT"; [46405]: “ HIT"; [40282]: “ ACTION";	[8484]: “ Turk" [18867]: “ Maur" [1897]: “ Dem" [41392]: “Tue"
11	693	[21800]: “BSD"; [43763]: “Arcade"; [44029]: “Brew"; [19314]: “Linux";	[11667]: “ Ru" [45220]: “ Sut" [18288]: “ grim" [614]: “ year"

Table 5.1: Results for Backwards-Direction Per-Neuron) input optimization for positive and negative activation by token, showing the top 5 inputs for each direction in a sample neuron from each hidden layer. *Note: Neuron numbers are here calculated starting from 1, rather than 0.*

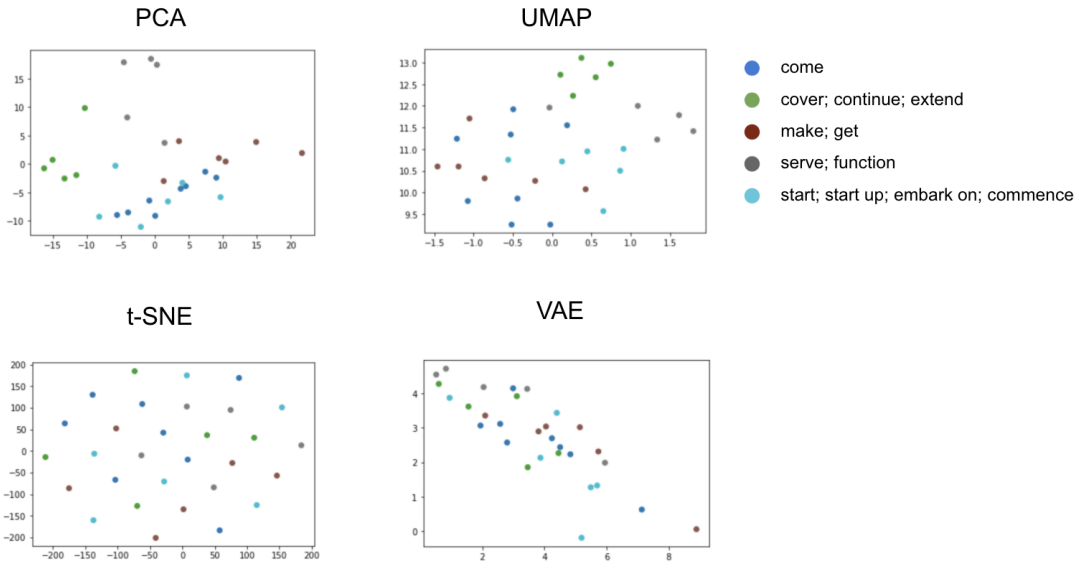


Figure 5.3: Random Seed: 60

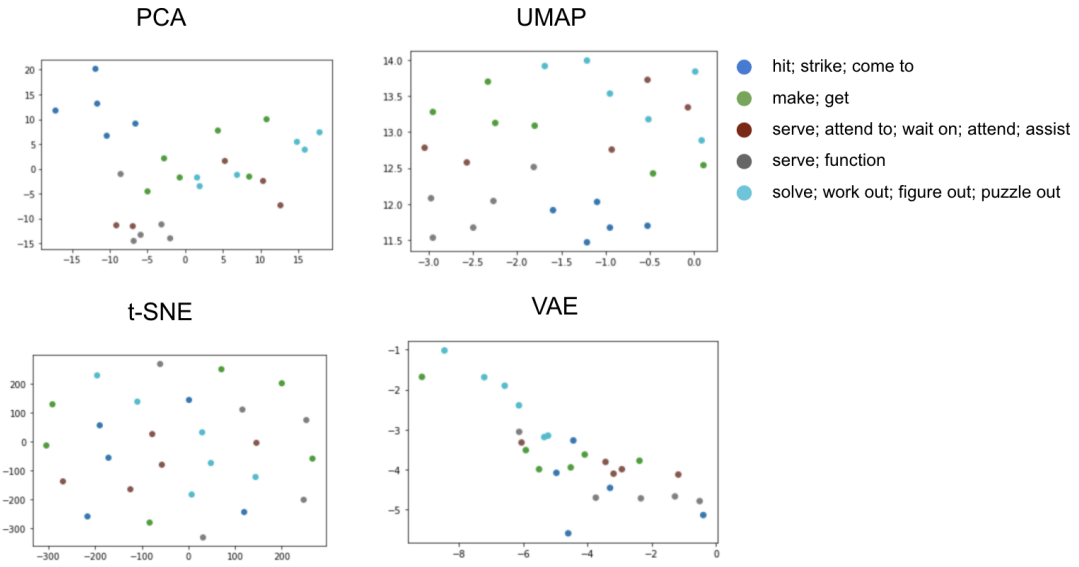


Figure 5.4: Random Seed: 65

Bibliography

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bender, E. M., T. Gebru, A. McMillan-Major, and S. Shmitchell (2021). On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 610–623.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003). A neural probabilistic language model. *Journal of machine learning research 3*(Feb), 1137–1155.

Bhardwaj, R., N. Majumder, and S. Poria (2021). Investigating gender bias in bert. *Cognitive Computation*, 1–11.

Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics 5*, 135–146.

Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems 29*, 4349–4357.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from language corpora contain human-like biases. *Science 356*(6334), 183–186.

Charles, M. and D. B. Grusky (2005). *Occupational ghettos: The worldwide segregation of women and men*, Volume 200. Stanford University Press Stanford, CA.

Coenen, A., E. Reif, A. Yuan, B. Kim, A. Pearce, F. Viégas, and M. Wattenberg (2019). Visualizing and measuring the geometry of bert. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 8594–8603.

Costa-jussà, M. R. and J. A. Fonollosa (2016). Character-based neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 357–361.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*(7).

Ebrahimi, J., A. Rao, D. Lowd, and D. Dou (2017). Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

Egghe, L. (2007). Untangling herdan's law and heaps' law: Mathematical and informetric arguments. *Journal of the American Society for Information Science and Technology 58*(5), 702–709.

Freitag, M. and Y. Al-Onaizan (2017). Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*.

Garg, N., L. Schiebinger, D. Jurafsky, and J. Zou (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences 115*(16), E3635–E3644.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hendrycks, D. and K. Gimpel (2016). Bridging nonlinearities and stochastic regularizers with gaussian error linear units.

Herdan, G. (1960). *Type-token mathematics*, Volume 4. Mouton.

Hinton, G. and S. T. Roweis (2002). Stochastic neighbor embedding. In *NIPS*, Volume 15, pp. 833–840. Citeseer.

Hochreiter, S. and J. Schmidhuber (1997). Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, 473–479.

Holtzman, A., J. Buys, L. Du, M. Forbes, and Y. Choi (2019). The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR.

Jelinek, F. (1980). Interpolated estimation of markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice, 1980*.

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kudo, T. and J. Richardson (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71.

Lewis, M., Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer (2020). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880.

Ling, W., I. Trancoso, C. Dyer, and A. W. Black (2015). Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*.

Liu, P. J., M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer (2018). Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Loshchilov, I. and F. Hutter (2018). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Manzini, T., Y. C. Lim, Y. Tsvetkov, and A. W. Black (2019). Black is to criminal as caucasian is to police: Detecting and removing multiclass bias in word embeddings. *arXiv preprint arXiv:1904.04047*.

McInnes, L., J. Healy, N. Saul, and L. Großberger (2018). Umap: Uniform manifold approximation and projection. *Journal of Open Source Software 3*(29).

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM 38*(11), 39–41.

Olah, C., A. Mordvintsev, and L. Schubert (2017). Feature visualization. *Distill 2*(11), e7.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science 2*(11), 559–572.

Qiu, X., T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang (2020). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 1–26.

Radford, A., K. Narasimhan, T. Salimans, and I. Sutskever (2018). Improving language understanding by generative pre-training.

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

Rumelhart, D. E., R. Durbin, R. Golden, and Y. Chauvin (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, 1–34.

Schoning, J. (2018). It s time to do something: Mitigating the negative impacts of computing through a change to the peer review process.

Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*(1), 1929–1958.

Sun, T., A. Gaut, S. Tang, Y. Huang, M. ElSherief, J. Zhao, D. Mirza, E. Belding, K.-W. Chang, and W. Y. Wang (2019). Mitigating gender bias in natural language processing: Literature review. *arXiv preprint arXiv:1906.08976*.

Sutton, A., T. Lansdall-Welfare, and N. Cristianini (2018). Biased embeddings from wild data: Measuring, understanding and removing. In *International Symposium on Intelligent Data Analysis*, pp. 328–339. Springer.

Trinh, T. H. and Q. V. Le (2018). A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Van der Maaten, L. and G. Hinton (2008). Visualizing data using t-sne. *Journal of machine learning research 9*(11).

Vargas, F. and R. Cotterell (2020). Exploring the linear subspace hypothesis in gender bias mitigation. *arXiv preprint arXiv:2009.09435*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008.

Vig, J. (2019). Visualizing attention in transformerbased language models. *arXiv preprint arXiv:1904.02679*.

Voorhees, E. (1999). Proceedings of the 8th text retrieval conference. *TREC-8 Question Answering Track Report*, 77–82.

Wallace, E., S. Feng, N. Kandpal, M. Gardner, and S. Singh (2019). Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*.