COSC420: Transformer based language model

Jake Norton (5695756)
May 22, 2024

1 Introduction

Transformer-based models have revolutionized the field of natural language processing, offering unprecedented advancements in text generation tasks. These models are lauded for their ability to understand and generate complex language patterns, making them highly effective for a variety of applications, from chatbots to literary analysis. Despite their widespread success, transformer models often encounter challenges, particularly when applied to smaller, stylistically unique datasets. This report explores the performance of such models, specifically focusing on their capability to generalize and produce novel text. Many of the decisions made here were close to arbitrary in nature due to time constraints, and I did not get to explore as much as I would have liked. However, I do believe that the models created are still adequate at creating text that is at least interesting. For this study, the datasets comprised the complete texts of two distinct books, including the voluminous and stylistically rich War and Peace. Using the entirety of these books provided a higher volume of data, although it introduced a potential bias towards the themes, style, and vocabulary prevalent in War and Peace. This was a conscious trade-off, made in the hope that the larger volume of data would enhance the model's performance, even if potentially skewing it towards the narrative style and vocabulary of War and Peace. This skew could influence the model's ability to generate universally applicable text, potentially limiting its generalization capabilities outside the specific contexts of the used texts. Despite this limitation, the aim was to evaluate how well the model could adapt and generate text that was not only syntactically coherent but also rich in varied linguistic expressions and themes derived from the comprehensive datasets.

2 Task 1: Tok2Vec Encoder

2.1 Methodology

Word2vec embeddings can be created using two primary approaches: Skip-gram (SG) and Continuous Bag of Words (CBOW). These methods are essentially opposites; CBOW predicts a word based on the context of surrounding words, while Skip-gram, conversely, predicts the surrounding context from a given word. CBOW is effective for identifying common links among frequently used words, making it ideal for smaller datasets where the focus is on simpler relationships that require less data to discern. However, the method's tendency to average context can lead to a less detailed understanding. In contrast, Skip-gram excels in capturing more complex relationships but requires more data and longer training times. Each position in the context window must be trained separately, unlike CBOW, which integrates all context simultaneously.

I was under the impression that the general consensus holds Skip-gram to be better suited for larger datasets, while CBOW can still perform well with smaller ones. How-

ever, I did not find any conclusive studies to substantiate this, although it's possible I may have overlooked some. I chose to use Skip-gram for its potential to uncover interesting patterns, despite the limitations of our dataset. I also hypothesized that Skip-gram might perform better on smaller datasets given sufficient time, as it processes the data multiple times, equal to the size of the context window. The context window size is important as it determines how many words around a target word are considered when forming predictions, directly influencing the model's ability to grasp broader textual contexts and nuances. Unfortunately, the exploration process was constrained by early corruption in my token-to-vector models. To streamline the process, I saved the encoded vectors of the entire corpus for repeated use, which significantly reduced processing time. However, at some point, I believe the saved dataset became corrupted, as the predictions consistently returned the tokens *i*, *æ*, =. Initially, the t-SNE plots suggested that the models were successfully clustering semantically similar words, so I did not realize there was a corruption issue until I began training the transformer models. Once I reencoded the dataset, this issue was resolved in all versions of the model.

Table 1: Frequency of Words with Different Context Window Sizes

Context window size 3		Context window size 4		Context window size 5	
Token	Probability	Token	Probability	Token	Probability
\s	0.0655	\s	0.0673	\s	0.0911
the	0.0322	,	0.0329	the	0.0446
,	0.0301	the	0.0315	,	0.0419
and	0.0205	and	0.0216	.	0.0272
.	0.0185	.	0.0210	and	0.0260
of	0.0176	of	0.0210	of	0.0238
to	0.0169	to	0.0150	to	0.0175
a	0.0114	in	0.0112	a	0.0135
in	0.0100	a	0.0107	in	0.0121
his	0.0086	his	0.0084	his	0.0097
was	0.0076	was	0.0075	was	0.0083
that	0.0073	with	0.0070	with	0.0076
he	0.0067	that	0.0063	that	0.0071
her	0.0064	he	0.0061	he	0.0069
with	0.0064	had	0.0053	her	0.0063
’	0.0061	at	0.0053	had	0.0059
had	0.0052	on	0.0051	at	0.0056
at	0.0050	her	0.0051	’	0.0054
it	0.0046	’	0.0049	on	0.0052
”	0.0045	for	0.0041	”	0.0051

Regarding the context window for the model, I found that a size below 3 yielded poor results in prediction tests. I experimented with sizes up to 5, but choosing between 3, 4, and 5 proved challenging. The larger the context window, the more it prioritized punctuation, as shown in Table 1. Eventually, I opted for a context window of 4, aiming to balance the difference and reduce training time. I was ultimately satisfied with the punctuation versus text balance, though the model still occasionally

struggles with quotations. Models with a context window of 4 generally showed a nicely distributed clustering of semantically similar words such as "would, should, could, must, shall."

To determine the vocabulary size, I performed a rough count of unique words using a series of Bash commands[1], resulting in approximately 25,000 words. However, this count included many duplicates or near-duplicates due to punctuation, as demonstrated by the list of variations of the word 'your'.

- 'your
- "your
- your
- "you're
- you're

To maximize coverage while managing memory and model size limitations, I capped the token count at 5,000, reasoning that if ChatGPT operates effectively with around 50,000 unique tokens, 10% should be sufficient for a corpus of this size.

I also explored various sizes for the hidden layer dimensions, ranging from 32 to 512. Limited testing made it difficult to determine the best size quantitatively, though it was evident that dimensions of 32 and 64 were insufficient, as they resulted in a noticeable skew in the graphical outputs. While t-SNE reduces dimensionality and may not make a linear skew inherently problematic, my interpretation was that such skewness indicated a simpler embedding representation.

2.2 Hyperparameters

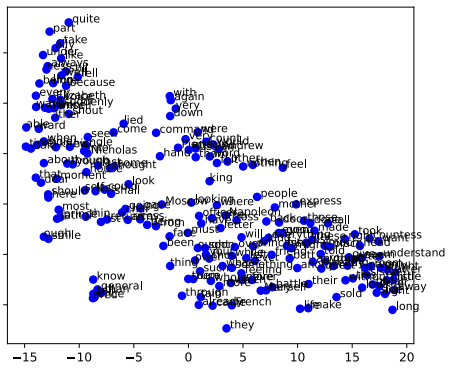
- **Context Window**: 4
- **Hidden Layer Dimension**: 256
- **Vocab Size**: 5000

2.3 Architecture

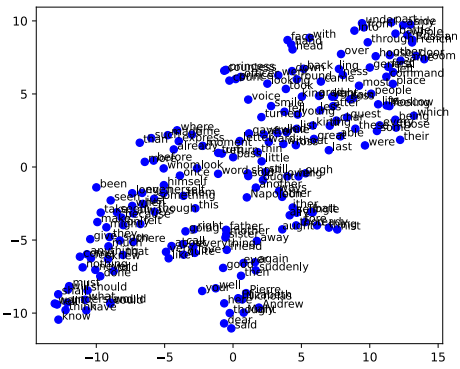
Table 2: Model Architecture Details

Layer (type)	Shape	Params
input (InputLayer)	5000	0
embedding (Dense)	256	1,280,000
output (Dense)	5000	1,285,000
Total params: 2,565,000		

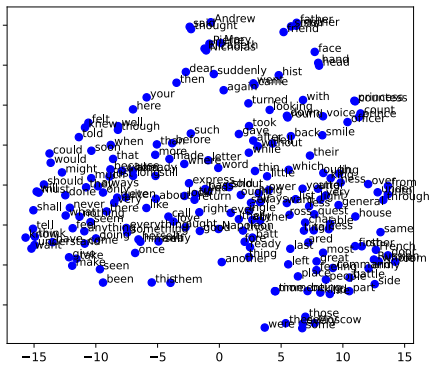
[1]bash command: `tr '[:upper:]' '[:lower:]' < combined_books_text | tr -d '[:punct:]' | tr ' ' '\n' | sort | uniq | wc -l`, in part crafted by ChatGPT



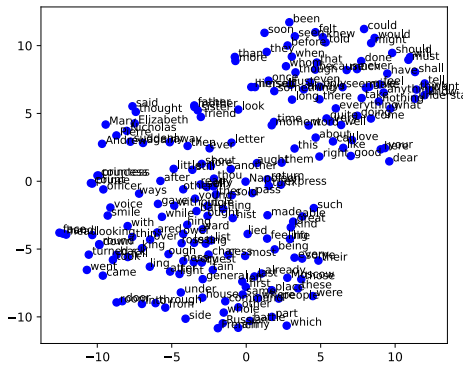
(a) neurons: 32, context window of 1



(b) neurons: 32, context window of 4

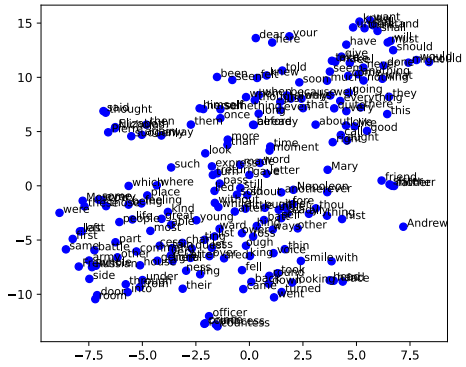


(c) neurons: 64, context window of 3

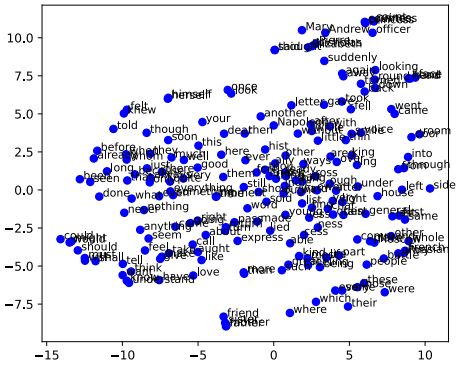


(d) neurons: 64, context window of 5

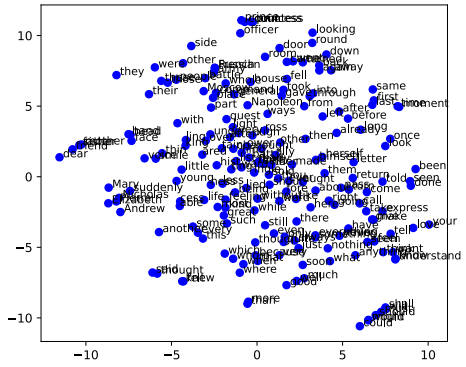
Figure 1: t-SNE plots of embedding space with varying sizes of neurons and context windows



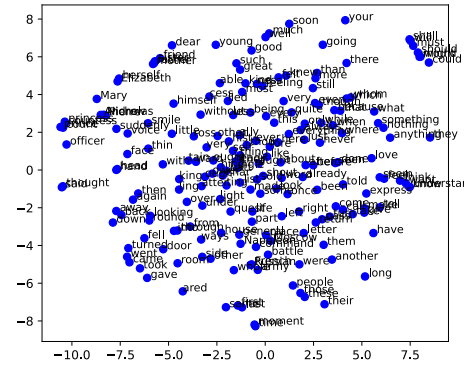
(a) neurons: 64, context window: 4



(b) neurons: 128, context window: 4



(c) neurons: 256, context window: 4



(d) neurons: 512, context window: 4

Figure 2: t-SNE plots of embedding space with varying sizes of neurons and context windows

3 Task 2: Transformer-based Text Prediction

3.1 Methodology

Utilizing the provided transformer, I modified the model to accommodate my 256-dimensional embeddings and to accept one-hot vectors as inputs. Due to time constraints and observing patterns in the training behavior, I limited the training to a single epoch. As detailed in Section 3.1, extending training didn't improve validation accuracy; instead, it worsened the validation loss. Moreover, increases in training accuracy without corresponding improvements in validation accuracy suggested a propensity for overfitting. This often resulted in the model replicating text verbatim, particularly from War and Peace.

I should note that the stagnation in validation accuracy and overall model generalization might be attributed to an excessively high learning rate, which could cause the model to repeatedly settle into local minima. In the final setup, I omitted the validation split and utilized the entire dataset. The quality of the model's outputs was then assessed more qualitatively, rather than relying strictly on quantitative metrics.

Table 3: Performance Metrics for Different Configurations

Transformer with Embeddings				
Attention Heads	**Transformer Layers**	**DFF**	**Loss**	**Masked Accuracy**
4	4	256	2.8427	0.4136
4	4	512	2.6718	0.4373
4	6	256	2.6647	0.4402
4	6	512	2.4718	0.4695
16	4	256	2.3300	0.4995
16	4	512	2.2095	0.5197
16	6	256	2.0522	0.5551
16	6	512	1.9590	0.5725
Transformer using One-hot Representation				
4	4	256	2.2441	0.5154
4	4	512	2.1719	0.5289
4	6	256	2.0241	0.5586
4	6	512	1.9068	0.5823
16	4	256	1.6985	0.6311
16	4	512	1.6726	0.6358
16	6	256	1.4977	0.6802
16	6	512	1.4662	0.6856

Model Validation Results from 5 Epochs

Embedded Model

Epoch	Masked Accuracy	Val Loss	Val Masked Accuracy
1	0.6024	8.5683	0.2430
2	0.8536	9.9641	0.2390
3	0.8941	10.4360	0.2372
4	0.9106	10.6708	0.2369
5	0.9202	10.7672	0.2373

One Hot

Epoch	Masked Accuracy	Val Loss	Val Masked Accuracy
1	0.7033	10.5074	0.2285
2	0.9168	11.5308	0.2275
3	0.9364	11.8598	0.2270
4	0.9448	11.8583	0.2263
5	0.9498	11.6422	0.2252

3.2 Model Architectures

Table 4: Detailed Comparison of Model Architectures

Architecture Details			
	Parameter	**Embedded Transformer**	**One Hot**
Layer (type)	**Output Shape**	**Parameters**	**Parameters**
Fixed/One Hot Embedding	100×256	0	0
Positional Encoding	100×256	0	-
Dense	100×256	-	1,280,256
Transformer Layer	100×256	790,016	790,016
Transformer Layer 1	100×256	790,016	790,016
Transformer Layer 2	100×256	790,016	790,016
Transformer Layer 3	100×256	790,016	790,016
Transformer Layer 4	100×256	790,016	790,016
Transformer Layer 5	100×256	790,016	790,016
Final Dense	100×5000	1,285,000	1,285,000
Total Params		6,025,096	7,305,352

4 Results

In the following section, Section **??**, the examples illustrate significantly fewer direct copies of passages. Nevertheless, the influence of the original texts is evident, albeit

less directly replicated. I was disappointed to find out that "The bloodstained smith" was not of its own making, though "torn from his own wood" is, make of that what you will.

4.1 Observations

Deciding whether the one-hot or embedding-based model performs better presents a complex challenge. The one-hot model seems more inclined to copy text directly, as evidenced by pattern matches, indicating a potential struggle with generalization. Conversely, the overall quality of output from both models points to limited generalization capabilities. Familiarity with War and Peace would likely enhance my ability to detect borrowed themes and characters, further informing this comparison.

It's also noteworthy that the overtrained embedding model handles punctuation with notable proficiency; at a glance, its output might be mistaken for well-composed prose. However, this appearance is deceptive, as much of the text, particularly marked in red[**??**], is lifted directly from the source material. This tendency is even more pronounced in the one-hot model [**??**], where the extent of direct copying is arguably worse.

Bias

The bias is obviously present in the model outputs. It is much more likely to generate text with heavy *inspiration* from War and Peace. As for whether have a more balanced dataset would have improved the model overall is hard to tell.

Further Work

There are several additional experiments I would have liked to conduct if time had permitted, specifically:

- A more comprehensive evaluation of similarities between the generated output and the source material to better understand the model's ability to innovate versus replicate.
- Investigation of various learning rates and optimizers to determine whether rapid convergence was reaching a local minimum or if more optimal minima could be achieved with a gentler learning rate.

5 Conclusion

To conclude, transformer-based models are powerful tools capable of generating text that closely approximates human-like syntax, even when trained on small and highly stylized datasets. However, these models face significant challenges in generalizing their output. Due to the limited range of patterns they can learn, there is a heightened risk of reproducing source text verbatim, particularly when overtrained. Based on my testing, if the objective is to generate new, more original text, the embedding-based model seems to have a distinct advantage, the embedding space allows it produce novel content, whereas the one-hot model often merely replicates existing text, functioning much like a cut-and-paste parrot.

6 References