Language Modeling with Transformers vs LSTMs

Darian Lee
UC Santa Cruz
Sunnyvale, CA, 94087
daeilee@ucsc.edu

Abstract

This paper compares two models— a Transformer with self-attention and an LSTM— to determine which is more effective for language modeling. Perplexity is used as an intrinsic evaluation metric to measure how well each model captures the statistical properties of language. I find that a transformer is better suited for this dataset and obtain a testing per-sentence average perplexity of 52. This mirrors the results of similar models on this dataset, which show transformers often achieving perplexities between 20 (for state of the art models such as GPT-3) to around 80 (for smaller, less robust transformers), with LSTMs recieving slightly higher perplexities on average.

Due to the method of calculating perplexity in this assignment, which involved averaging sentence-wise perplexities, achieving a significantly lower test perplexity is unlikely and may even lead to a less effective model. The implications of this will be discussed in greater detail later in the report.

1 Introduction

The task was to design a language model to predict the next word in a sequence of text based on a dataset from Penn Treebank dataset. This was a self surprised task where the data is used as its own pseudo-labels during training. The data contained separate datasets for train, val, and test, each containing one column labeled "sentence" where the utterances were scored. The sentences ranged in sequence length, with the longest sequence length found being 84 tokens, however the vast majority of sentences had 50 tokens or less.

The quality of the sentences also varied greatly, with most sounding mostly normal for a formal news setting, to a little weird, to complete gibberish. This is likely due to the diverse sources of the Penn Treebank, its inclusion of domain-specific jargon, outdated constructs, and its focus on being a grammatical corpus rather than a semantic one.

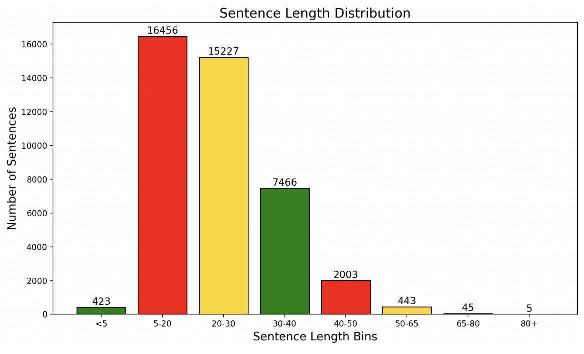

Figure 1: The sentence lengths in the train set

id	sentence
0	aer banknote berlitz calloway centrust cluett fromstein gitano guterman hydro-quebec ipo kia memotec mlx nahb punts rake regatta rubens sim snack-food ssangyong swapo wachter

Table 1: Example sentence in the dataset illustrating the absurdity of certain sentences.

My preprocessing on this data was minimal, however I did remove all punctuation, add a start and stop token to the beginning of each utterance, and lowercased the text.

2 Models and Experimentation

In this section, I will outline the architecture and motivations behind my two most distinct and developed model classes. The first class contains RNN variations and the second contains transformers with self attention.

2.1 Class 1: LSTMs

2.1.1 Motivation

My first choice of model was an LSTM in order to provide a baseline perplexity to use in order to evaluate my transformer model. Although LSTMs have mostly been replaced by transformers for language

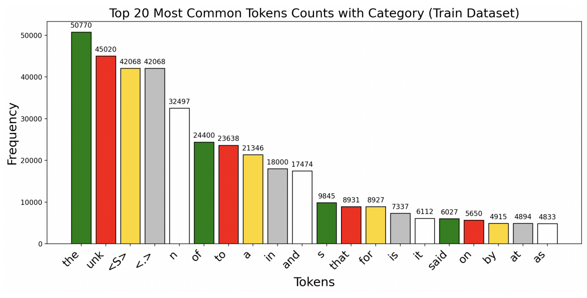

Figure 2: The top 20 most common tokens

modeling tasks, they remain a useful starting point due to their ability to capture sequential dependencies and process data in a step-by-step manner, and are still used for language modeling in many applications where simpler architectures are preferred, and were widely used before the development of transformers, as their ability to store information using a cell state made them better at handeling long term dependencies than a traditional RNN (3).

2.1.2 Performance and Experimentation

The best model in this class achieved a testing average sentence perplexity of 227. Because this was simply a baseline model and not expected to outperform the transformer model, experimentation and efforts to improve its perplexity were limited. However, I did experiment with different embedding dimensions and hidden dimensions, finding the best embedding dimensions to be 300 and the best hidden dimensions to be 1000.

2.1.3 Potential Shortcomings

While LSTMs excel at capturing sequential dependencies and handling long-term relationships in data, they have several limitations compared to transformers. One major shortcoming is their sequential processing nature, which makes them computationally expensive and difficult to parallelize during training, as noted by Vaswani et al. (1). This limitation can lead to slower training times, especially for large datasets. Additionally, LSTMs struggle with very long sequences due to vanishing gradient issues, despite mechanisms like cell states, as highlighted by Hochreiter and Schmidhuber (4). Transformers address these issues through self-attention mechanisms, allowing them to capture global dependencies and process sequences in parallel. Moreover, transformers have been shown to be more scalable and are better suited for leveraging large-scale pretraining, which has become a standard in modern NLP tasks (5).

2.2 Class 2: Transformer

2.2.1 Motivation

My initial model was a language model based on the Transformer architecture utilizing self-attention mechanisms. The motivation behind this approach stems from the seminal work of Vaswani et al. (2017), who introduced the Transformer model as a more efficient and scalable alternative to traditional recurrent neural networks (RNNs) (1). The self-attention mechanism in the Transformer allows the model to capture long-range dependencies in sequences while avoiding the issue of vanishing gradients, resulting in improved performance on language modeling tasks. In constructing this model, I closely followed the original paper (1) in order to implement the entire model from scratch without relying on pre-built PyTorch attention layers. I choose this approach to deepen my understanding of the underlying architecture. Additionally, I referred to a helpful YouTube video (2) for guidance on challenging sections of the implementation.

2.2.2 Performance

The best average sentence-wise testing perplexity I achieved was 51, and it came from a model with around seven hundred thousand parameters, including a vocabulary size of 10001, an embedding size of 30, 5 blocks of 6 attention heads of size 6, and layer normalization and dropout. My final model also uses cross entropy as the loss, due to its relationship with perplexity, and AdamW, which is preferred for its ability to improve convergence and generalization by decoupling weight decay and adapting learning rates for each parameter. Average sentence-wise validation perplexity was calculated at each epoch and used to determine the best model. My final model was saved after 8 epochs.

2.2.3 Experimentation

My first model in this class achieved an average sentence-wise testing perplexity in the thousands. This was because I was measuring global perplexity rather than average sentence-wise perplexity in order to choose my best model. Although the global perplexity was quite low, indicating a confident and highly predictive model, I realized that, because of the way that average sentence-wise testing perplexity is calculated, this model was overfitting on longer sentences and failing to generalize well to shorter ones, leading to poor performance when evaluated on a per-sentence basis. I will discuss

$$\text{Perplexity}_{\text{avg}} = \frac{1}{M}\sum_{i=1}^{M} \exp\left(-\frac{1}{N_i}\sum_{j=1}^{N_i} \log P(w_j|w_1,\ldots,w_{j-1})\right)$$

$$\text{Perplexity}_{\text{standard}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_1,\ldots,w_{i-1})\right)$$

more about the cons of evaluating models in this way in potential short comings.

Next attempts at transformers achieved slightly higher testing perplexities, in the 80s-100s. The main decision that lead my perplexity to drop down to 51 was incorporating drop out to prevet overfitting, shrinking my embedding size to prevent overfitting as well, switching from Adam optimzer to AdamW optimizer, and adding layer normalization, which is recommended in the original "Attention is all you need" paper.

2.2.4 Potential Shortcomings

The biggest shortcoming for this model is the way that perplexity is calculated, which favors less confident models. The sentence-wise approach disproportionately penalizes shorter sequences because the small number of tokens (N) in such sentences fails to balance out the log probabilities. As a result, models that are highly confident in their predictions for shorter sequences can be unfairly punished, while models that distribute probability more evenly (and are thus less certain) may be rewarded.

In contrast, the standard method of calculating perplexity aggregates the log probabilities across all tokens in the dataset, normalizing by the total number of tokens. This approach ensures that each token contributes equally to the overall perplexity score, providing a more balanced evaluation of model performance.

I noticed this in my first model, which achieved a low global perplexity, but a high sentence-wise average perplexity. When looking at the perplexities per sentence, most were very low, however a few perplexities for very short sentences were extremely high, skewing the overall average perplexity.

It is also worth noting that calculating sentence wise perplexity also makes the testing evaluation more sensity to noise and errors in the testing set. As stated earlier, some of the sentences in this dataset were nearly nonsensical. If a model was highly perplexed on a sentence like "aer banknote berlitz calloway centrust cluett fromstein gitano guterman hydro-quebec ipo kia memotec mlx nahb punts rake regatta rubens sim snack-food ssangyong swapo wachter", which is an actual sentence in the dataset, it would have a higher sentence-wise average perplexity, however its global perplexity would likely be evened out by other sentences it predicted better.

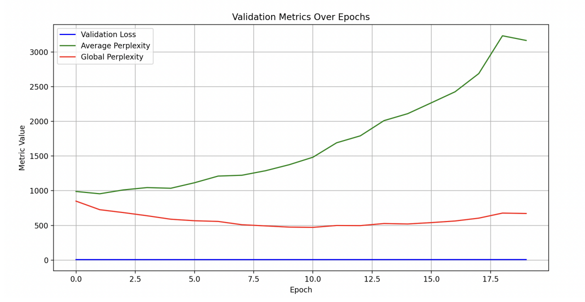

Figure 3: Global vs Sentence-wise Perplexity on a Subset of the Data

2.3 Results

As indicated earlier, my best model was a transformer with a vocabulary size of 10001, an embedding size of 30, 5 blocks of 6 attention heads of size 6, and layer normalization and dropout as well as the optimizer AdamW.

2.4 Linked Models

- **Transformer:** `https://github.com/Darian-Lee-YTKA/Transformer-LM-on-PennBank-Dataset/blob/main/main.py`
- **LSTM:** `https://github.com/Darian-Lee-YTKA/Transformer-LM-on-PennBank-Dataset/blob/main/lstm.py`

Model	**Configuration**	**Sentence-wise Perplexity**
Transformer	No Dropout, Adam Optimizer	81
Transformer	Dropout 0.3, AdamW Optimizer	52
LSTM	-	227

Table 2: Sentence-wise Perplexity of Different Models

2.5 Citations

References

[1] A. Vaswani, N. Shazeer, N. Parmar, L. Uszkoreit, J. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Proceedings of NeurIPS*, 2017.

[2] "How to Implement a Transformer from Scratch in PyTorch", YouTube, 2023, https://youtu.be/kCc8FmEb1nY?si=DyESX7aHB_FL1BnX.

[3] H. Sundermeyer, R. Schlüter, and H. Ney, "LSTM Neural Networks for Language Modeling," *Interspeech 2012*, 2012, https://www.isca-archive.org/interspeech_2012/sundermeyer12_interspeech.pdf.

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[5] T. Brown, B. Mann, N. Ryder, et al., "Language Models are Few-Shot Learners," *Proceedings of NeurIPS*, 2020.