CS7.501 : Advanced Natural Language Processing
Assignment - 1

Prakhar Jain
2022121008

1 Analysis of Models

We considered three different language models—Neural Network, LSTM, and Decoder-only—in this assignment. These models were trained and evaluated on a dataset of 35,000 sentences, where special characters were removed during preprocessing This preprocessing step helps the model handle unknown words. The performance of each model was measured using perplexity, which quantifies how well a probability distribution or probability model predicts a sample. A lower perplexity score indicates better model performance, meaning the model can better predict the sequence of words in a sentence.

The trained model can be found here.

1.1 Model Architectures

Neural Network Language Model: The Neural Network Language Model (NNLM) is a feedforward network that learns to predict the next word in a sentence based on a fixed-size context window. In this case, we limit the context to the previous 5 words, meaning the model processes only a small part of the input sequence to predict the next word. This model uses embeddings as input, which are dense vector representations of words that capture semantic relationships.

While NNLMs are relatively simple and computationally efficient compared to more complex architectures like LSTMs or Transformers, they lack an inherent mechanism to handle sequential or time-dependent data. The model cannot store or remember information beyond the fixed window size, limiting its ability to model long-range dependencies or more complex patterns in natural language.

Advantages:

- Simpler and faster to train compared to more complex models.
- Can be useful for tasks where the focus is on local context rather than long-term dependencies.

Drawbacks:

- The lack of recurrent structure makes it inefficient for capturing long-term dependencies.
- Since it only looks at a few previous words, its understanding of sentence context is limited, especially for longer sentences.

LSTM Language Model: Long Short-Term Memory (LSTM) is a specialized form of Recurrent Neural Network (RNN) designed to combat the vanishing gradient problem commonly encountered in standard RNNs. LSTMs introduce memory cells that can store information over long sequences, making them suitable for tasks that require understanding of context spread over multiple time steps. The ability to remember and forget information is governed by three gates: input gate, forget gate, and output gate, which control the flow of information through the network.

In the context of language modeling, LSTMs are particularly effective because they can maintain context across long sequences, allowing them to capture syntactic and semantic relationships between words that are far apart. For instance, LSTMs can understand that "he" refers to a person mentioned earlier in a long sentence or paragraph, which a basic neural network might struggle with.

Advantages:

- Designed to handle long-term dependencies, making it ideal for sequence tasks like language modeling.
- The gating mechanism helps in managing information flow, preventing issues like the vanishing gradient.

Drawbacks:

- LSTMs are slower to train due to their complex structure and sequential processing.
- They require more memory and computational resources compared to feedforward networks.

Decoder-only Language Model (Transformer): The Decoder-only Language Model refers to a variant of the Transformer architecture, which relies solely on the decoder component, originally part of the encoder-decoder structure. Unlike traditional recurrent models, Transformers use a self-attention mechanism to model dependencies between input tokens, regardless of their distance in the sequence. This allows the model to process the entire input sequence in parallel, rather than step-by-step, making it more efficient and scalable for large datasets and longer contexts.

Self-attention works by allowing the model to attend to different parts of the sequence, assigning different weights to different words based on their relevance to the current word prediction task. For example, when predicting the next

word in a sentence, the model might assign higher importance to words that are semantically or syntactically related to the target word, even if they are far apart in the sequence.

The Decoder-only model is powerful in handling long-range dependencies due to its self-attention mechanism, and it has become a foundation for state-of-the-art models like GPT. This model also incorporates positional encoding to retain information about the order of words, which is critical for understanding the sequence of language.

Advantages:

- Captures long-range dependencies efficiently, thanks to the self-attention mechanism.
- Can process the entire sequence in parallel, leading to faster training times on modern hardware.
- Better suited for large-scale data and tasks requiring the model to understand global context.

Drawbacks:

- Requires a large amount of data to perform optimally, as it has many parameters to train.
- Computationally intensive, especially in terms of memory usage, due to the self-attention mechanism.

2 Perplexity Results

Neural Network Language Model:

- **Train Perplexity:** 217.4456
- **Test Perplexity:** 188.6686

The high train and test perplexity values suggest that the neural network model struggled to effectively capture the underlying structure of the language data. However, the lower test perplexity compared to the training perplexity suggests that the model did not overfit, but rather, struggled with both the training and test data. The architecture likely has limitations in handling sequential data with long-range dependencies, which impacted its overall performance on language modeling tasks.

LSTM Language Model:

- **Train Perplexity:** 53.98
- **Test Perplexity:** 78.29

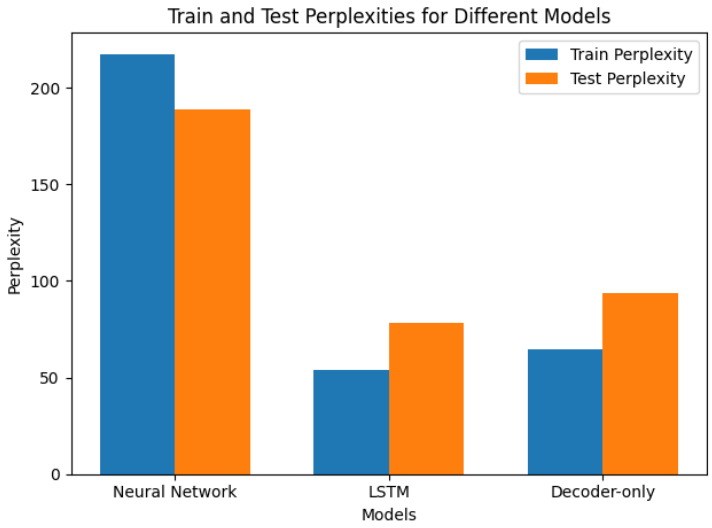

Figure 1: Train and Test perplexities for all the models

The LSTM model demonstrated a significant improvement over the basic neural network model, with much lower train and test perplexities. The LSTM's ability to capture temporal dependencies in sequences made it better suited for this task. However, the increase in test perplexity compared to train perplexity indicates some degree of overfitting, which could be due to the relatively small dataset (35k sentences). Despite this, the LSTM outperforms the neural network model by learning more effectively from sequential data.

Decoder-only Language Model:

- **Train Perplexity:** 64.6486
- **Test Perplexity:** 93.4371

The Decoder-only model showed a competitive performance, though its test perplexity is higher than the LSTM's. The relatively close values for train and test perplexities suggest that the model avoided significant overfitting, but the higher test perplexity indicates it didn't generalize as well as expected. The self-attention mechanism allows the model to capture dependencies across longer contexts, which helped in capturing some aspects of language data, but it may require more data to fully utilize its capabilities.

3 Bonus

Optimal Hyperparameters:
Dropout: 0.1

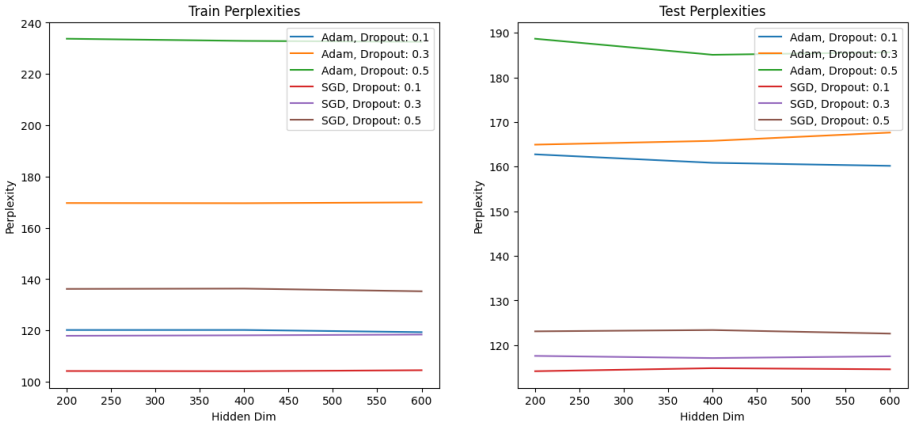

Figure 2: Results for Different parameters for NNLM

Hidden Dim: 200
Optimizer: SGD
Test Perplexity: 114.15992736816406

3.1 Effect of Dropout

Dropout helps prevent overfitting by randomly dropping units during training. As the dropout rate increases, the model becomes more regularized, which may lead to a poorer fit to the training data.

- **Dropout 0.1**:
  - Perplexity is relatively low, especially when using Stochastic Gradient Descent (SGD). The model learns effectively with minimal regularization.
  - The Adam optimizer results in higher test perplexity compared to SGD, suggesting that Adam may overfit when the dropout rate is low.
- **Dropout 0.3**:
  - Perplexity increases slightly. The model is more regularized, leading to a slight degradation in training performance.
  - SGD maintains better test perplexity compared to Adam, indicating that SGD is more robust with moderate dropout, while Adam's performance declines.
- **Dropout 0.5**:
  - Perplexity increases significantly for both optimizers. The model is heavily regularized and struggles to fit the training data.

  - Both Adam and SGD perform worse in this scenario, though SGD still achieves consistently lower perplexity compared to Adam.

3.2 Effect of Hidden Dimensions

The hidden dimension represents the size of the model's hidden layers, which controls its capacity to learn complex patterns. Larger hidden dimensions allow the model to capture more complex information but may lead to overfitting without adequate regularization.

- **Hidden Dimension 200 vs 400 vs 600**:
  - For **dropout 0.1** and **SGD**, increasing the hidden dimensions does not drastically affect test perplexity, though there is a slight increase.
  - **Adam** shows higher perplexity with larger hidden dimensions, likely because it is more sensitive to overfitting.
  - For **dropout 0.3** and **dropout 0.5**, increasing the hidden dimensions results in minor changes in perplexity. However, larger hidden dimensions generally lead to worse generalization, especially with Adam, as indicated by increasing test perplexity.

3.3 Effect of Optimizer

Two optimizers are compared: Adam and SGD. Adam is an adaptive learning rate optimizer, while SGD uses a fixed learning rate and is typically more stable in the long run but slower to converge initially.

- **SGD** consistently exhibits **lower perplexity** than Adam across different settings, indicating better generalization, especially with lower dropout rates and larger hidden dimensions.
- **Adam** tends to converge faster during training, but often shows higher test perplexity, indicating a tendency to overfit, especially with lower dropout rates (e.g., dropout 0.1).

3.4 Summary of Trends

- **Lower dropout (0.1)** generally yields better performance, though Adam tends to overfit more, resulting in a larger gap between training and test perplexity.
- **SGD** shows better generalization, especially at **moderate dropout rates (0.3)**, and consistently achieves lower test perplexity across different hidden dimensions.
- **Increasing hidden dimensions** can lead to slight overfitting, particularly with Adam, though the effect is less pronounced compared to changes in dropout rates and optimizer choice.

In conclusion, **SGD** with **moderate dropout (0.3)** tends to generalize better, whereas **Adam** is more prone to overfitting, especially with smaller dropout rates and larger hidden dimensions.