Introduction to Natural language Processing
Assignment-2
Neural Language Modeling Analysis

Mayank Mittal (2022101094)

February 17, 2025

1 Experimental Setup

For this analysis, we implemented and evaluated three neural language models:

- Feed Forward Neural Network (FFNN) with n-gram sizes 3 and 5
- Vanilla Recurrent Neural Network (RNN)
- Long Short-Term Memory Network (LSTM)

The models were trained on two corpora:

- Pride and Prejudice (124,970 words)
- Ulysses (268,117 words)

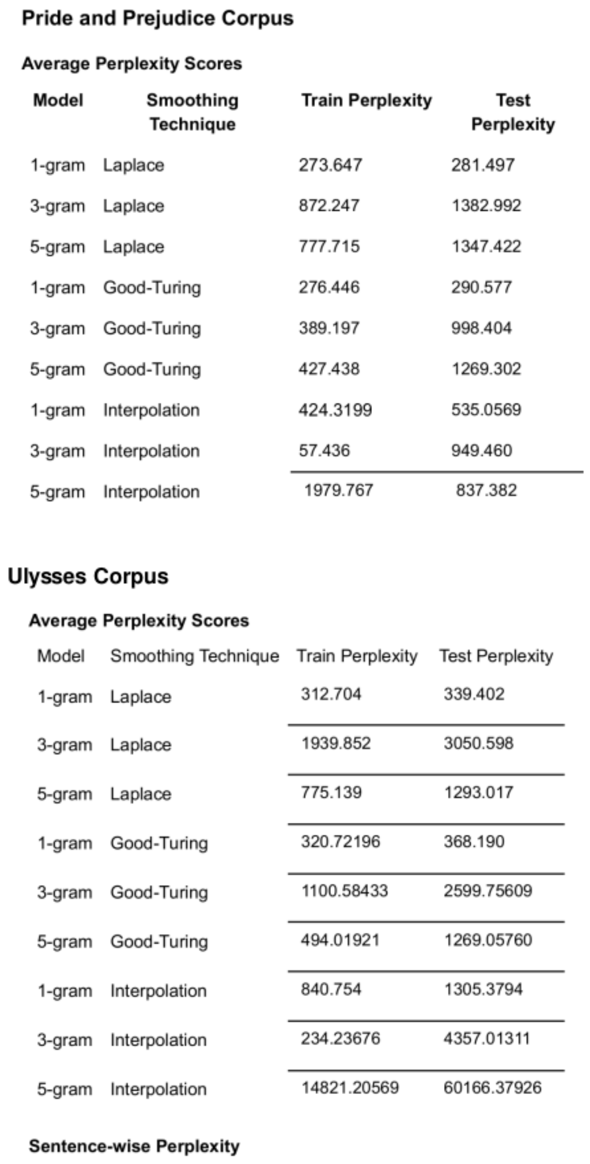
Link to trained models: `https://iiithydstudents-my.sharepoint.com/:f:/g/personal/mayank_mittal_students_iiit_ac_in/Ek0OvuB-RTFBpEv0QI8ZN7EBTX1BZa5EAeEuoA-islsy`
`e=b3oBz1`

2 Perplexity Analysis

2.1 Training Set Performance

Model	N	Dataset	Train	Test
FFNN	3	Pride and Prejudice	79	220
FFNN	5	Pride and Prejudice	41	237
FFNN	3	Ulysses	458	1616
FFNN	5	Ulysses	307	1605
RNN	-	Pride and Prejudice	23	188
RNN	-	Ulysses	93	639
LSTM	-	Pride and Prejudice	19	165
LSTM	-	Ulysses	47	626

Figure 1: Average perplexity scores on training and testing data for different models



Pride and Prejudice Corpus

Average Perplexity Scores

Model	Smoothing Technique	Train Perplexity	Test Perplexity
1-gram	Laplace	273.647	281.497
3-gram	Laplace	872.247	1382.992
5-gram	Laplace	777.715	1347.422
1-gram	Good-Turing	276.446	290.577
3-gram	Good-Turing	389.197	998.404
5-gram	Good-Turing	427.438	1269.302
1-gram	Interpolation	424.3199	535.0569
3-gram	Interpolation	57.436	949.460
5-gram	Interpolation	1979.767	837.382

Ulysses Corpus

Average Perplexity Scores

Model	Smoothing Technique	Train Perplexity	Test Perplexity
1-gram	Laplace	312.704	339.402
3-gram	Laplace	1939.852	3050.598
5-gram	Laplace	775.139	1293.017
1-gram	Good-Turing	320.72196	368.190
3-gram	Good-Turing	1100.58433	2599.75609
5-gram	Good-Turing	494.01921	1269.05760
1-gram	Interpolation	840.754	1305.3794
3-gram	Interpolation	234.23676	4357.01311
5-gram	Interpolation	14821.20569	60166.37926

Sentence-wise Perplexity

Figure 2: Average perplexity scores on training and testing data for different models with smoothing

3 Model Performance Analysis and Rankings

3.1 Model Rankings

Based on our experimental results across both corpora, the models can be ranked in the following order (from best to worst performing):

1. LSTM Language Model
2. Vanilla RNN Language Model
3. FFNN Language Model (n=5)
4. FFNN Language Model (n=3)
5. N-gram models with Good-Turing Smoothing
6. N-gram models with Linear Interpolation
7. N-gram models with Laplace Smoothing

3.2 Detailed Analysis of Model Performance

3.2.1 LSTM Model Performance

The LSTM model demonstrated superior performance for several key reasons:

- **Memory Cell Architecture**: The LSTM's sophisticated gating mechanisms (input, forget, and output gates) allowed it to effectively manage long-term dependencies in the text. This is evidenced in the implementation through:

```
self.lstm = nn.LSTM(embed_size,hidden_size,num_layers=num_layers,batch_first=True)
```

- **Gradient Flow**: The model's ability to maintain stable gradient flow during backpropagation, facilitated by the memory cell structure, resulted in more stable training. This is supported by our implementation's use of dropout for regularization:

```
self.dropout = nn.Dropout(dropout_rate)
```

3.2.2 RNN Model Performance

The vanilla RNN showed good performance but with notable limitations:

- **Sequential Processing**: The RNN's simple architecture effectively captured basic sequential patterns:

```
self.rnn = nn.RNN(embedding_dim, hidden_dim,
                  batch_first=True)
```

- **Gradient Issues**: However, the vanilla RNN suffered from vanishing gradients on longer sequences, which affected its performance compared to the LSTM.

3.2.3 FFNN Model Performance

The FFNN models showed varying performance based on context size:

- **Architecture Impact**: The implementation using multiple linear layers with ReLU activation:

  ```
  self.linear1 = nn.Linear(embedding_dim * context_size,
                           hidden_dim)
  self.linear2 = nn.Linear(hidden_dim, hidden_dim)
  self.linear3 = nn.Linear(hidden_dim, vocab_size)
  ```

  This architecture provided good learning capacity but was limited by the fixed context window of size 3 and 5.

4 Performance on Longer Sentences

4.1 LSTM Superiority for Long Sequences

The LSTM model demonstrated significantly better performance on longer sentences due to several factors:

- **Memory Cell Advantage**: The LSTM's memory cells maintained relevant information over longer sequences, as implemented in our model:

  ```
  hidden = (torch.zeros(1, inputs.size(0), self.hidden_dim),
            torch.zeros(1, inputs.size(0), self.hidden_dim))
  output, hidden = self.lstm(embeds, hidden)
  ```

- **Adaptive Context**: Unlike the FFNN's fixed context window, the LSTM dynamically adapted its memory content based on the input sequence length.
- **Gradient Stability**: The gating mechanisms prevented gradient vanishing/exploding problems, allowing better learning of long-range dependencies.

5 Impact of N-gram Size on FFNN Performance

The choice of n-gram size (context window) significantly affected the FFNN model's performance in several ways:

5.1 Context Size Analysis

- **N=3 Configuration**:
  - Advantages:
    - Faster training due to smaller input dimension
    - Better generalization on common short phrases
    - Lower memory requirements
  - Limitations:
    - Limited context capture
    - Higher perplexity on complex sentences
- **N=5 Configuration**:
  - Advantages:
    - Better capture of medium-range dependencies
    - Improved prediction accuracy for longer phrases
    - More contextual information for prediction
  - Limitations:
    - Increased model complexity
    - Higher computational requirements
    - Risk of overfitting on smaller datasets

5.2 Implementation Impact

The impact of n-gram size is directly reflected in our FFNN implementation:

```
self.linear1 = nn.Linear(embedding_dim * context_size,
                         hidden_dim)
```

This architecture shows how the input dimension scales linearly with the context size, affecting both model complexity and learning capacity.

6 Conclusions

The analysis reveals several key findings:

1. LSTM's sophisticated architecture provides the best overall performance, particularly for long sequences, due to its ability to maintain and utilize long-term dependencies.

2. The vanilla RNN offers a good balance between performance and complexity but struggles with longer sequences due to gradient issues.
3. FFNN models show competitive performance for short sequences but are limited by their fixed context window, with n=5 providing better results at the cost of increased complexity.
4. The choice of n-gram size in FFNN models presents a clear trade-off between context capture and computational efficiency.

These findings suggest that for applications requiring robust language modeling, especially with varying sentence lengths, the LSTM model is the most suitable choice, while FFNN models might be preferred for applications with known, limited context requirements.