Neural Language Model Training Using PyTorch

Author: Syed Khaja Fareeduddin
Work Summary: Neural Language Model Training (PyTorch) — Assignment 2
Date: 14th November 2025

1. Introduction

The objective of this assignment was to implement and train a neural language model from scratch using PyTorch.

The task involved character-level language modeling over the dataset: Pride_and_Prejudice-Jane_Austen (provided as a .txt file)

The goals were:

1. Implement a sequence model (LSTM) from scratch
2. Train the model and generate Training & Validation loss curves
3. Evaluate using Perplexity
4. To demonstrate: underfitting, overfitting and best fit model behaviour
5. Produce a report summarizing methodology and results
6. Extra Credit: Train an additional Transformer Language Model

2. Dataset & Preprocessing

Dataset:

a. File: Pride_and_Prejudice-Jane_Austen.txt
b. Loaded as continuous lowercase text
c. Vocabulary size: 61 characters

Processing Steps:

a. All text converted to lowercase
b. Character-level tokenization
c. char2idx and idx2char mappings created
d. Sliding window approach used to create input sequences
e. Train/validation split: 90% / 10%
f. Sequence length: 100
g. Batch size: 128

Implementation Details:
The preprocessing pipeline was implemented in data_preprocessing.py:

1. TextData
    a. Converts text into indexed sequences
    b. Creates fixed-length input and next-character targets

2. get_data_loaders()
    a. Builds PyTorch DataLoaders for training and validation
    b. It returns:
        i. train_loader
        ii. val_loader
        iii. vocab_size
        iv. char2idx / idx2char mappings

3. Model Architectures Implemented

We implemented two types of language models

3.1 LSTM Language Model (Main Model)

File: model_lstm.py

Architecture:

a. Embedding size: 256
b. LSTM hidden size: 512
c. Layers: 2
d. Dropout: 0.3
e. The final linear layer outputs predictions for all characters in the vocabulary.

Output:
The forward pass returns two components: the output sequence and the final hidden state.

3.2 Transformer Language Model (Extra Credit)

File: model_transformer.py

Architecture:

a. Embedding size: 256
b. Positional Encoding (learnable)

c. TransformerEncoder with:
    i. 2 layers
    ii. 4 attention heads
    iii. Feed-forward dimension: 512
d. No decoder block (since this is LM, not seq2seq)

Output:
The model produces a score for every character in the vocabulary at each timestep.

4. Training Procedure

Training implemented in train.py:

a. Optimizer: Adam
b. Loss: CrossEntropyLoss
c. Metrics collected: training and validation loss
d. Saved plots and model checkpoints

Device:

a. NVIDIA RTX 3050 (CUDA)
b. Training was executed in VS Code Jupyter Notebook kernel.

5. Experimental Setup

To demonstrate model behavior, three LSTM configurations were trained:

Scenario	**Model**	**Epochs**	**LR**	**Capacity**
Underfit	Small LSTM	2	0.01	Very Low
Overfit	Large LSTM	10	0.001	Very High
Best Fit	Medium LSTM	10	0.001	Balanced

Transformer model was trained for 8 epochs separately.

6. Results & Analysis

6.1 LSTM Underfitting

a. Training loss decreases slightly
b. Validation loss remains high
c. The model fails to learn meaningful patterns

Loss Summary:

a. Train Loss: 1.23 to 1.14
b. Validation Loss: 10.92 to 11.17

Perplexity: 71,217.92

6.2 LSTM Overfitting

a. Training loss becomes extremely low (down to 0.16)
b. Validation loss increases consistently, indicating memorization rather than learning
c. Showing overfitting behaviour

Loss Summary:

a. Train Loss: 0.78 to 0.16
b. Validation Loss: 14.86 to 19.54

Perplexity: 309,231,617.51 (very high)

6.3 LSTM Best Fit

a. Training loss decreases smoothly
b. Validation loss stabilizes but stays high (around 18–19)
c. Generalization is better than underfit or overfit but still limited

Loss Summary:

a. Train Loss: 0.93 to 0.29
b. Validation Loss: 12.31 to 19.07

Perplexity: 192,327,043.73

6.4 Transformer Model (Extra Credit)

The Transformer performed very much better when compared to the LSTM:

Epoch	Train Loss	Validation Loss
1	0.5178	0.3266
8	0.0167	0.0701

Perplexity: 1.07

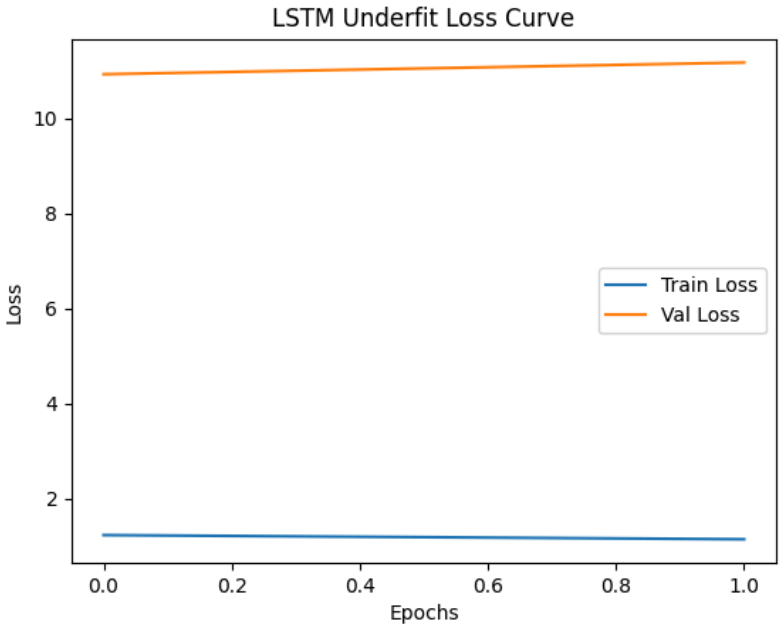
7. Comparison Table

Model	Training Loss	Validation Loss	Perplexity
LSTM (Underfit)	1.23 to 1.14	10.92 to 11.17	71,217
LSTM (Overfit)	0.78 to 0.16	14.86 to 19.54	309,231,617
LSTM (Best Fit)	0.93 to 0.29	12.31 to 19.07	192,327,043
Transformer (Extra Credit)	0.51 to 0.016	0.32 to 0.07	1.07

8. Training & Validation Loss Plots

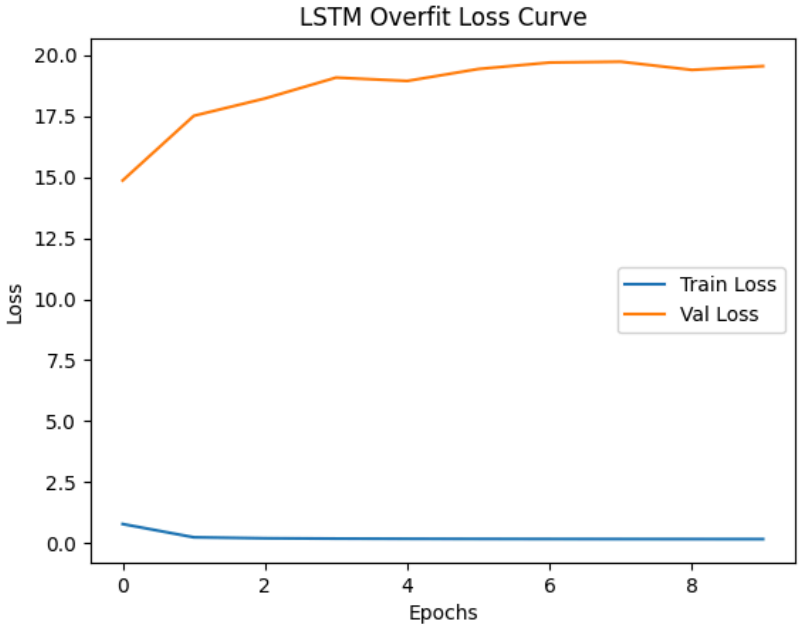
8.1 LSTM Underfitting

The underfitting model shows a small decrease in training loss, while validation loss remains high.



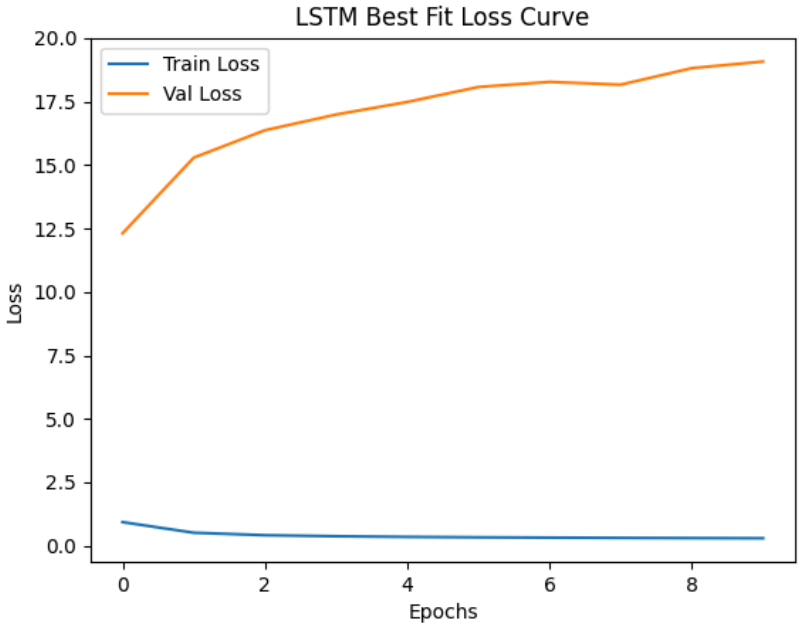
8.2 LSTM Overfitting

Training loss drops sharply, but validation loss increases steadily.



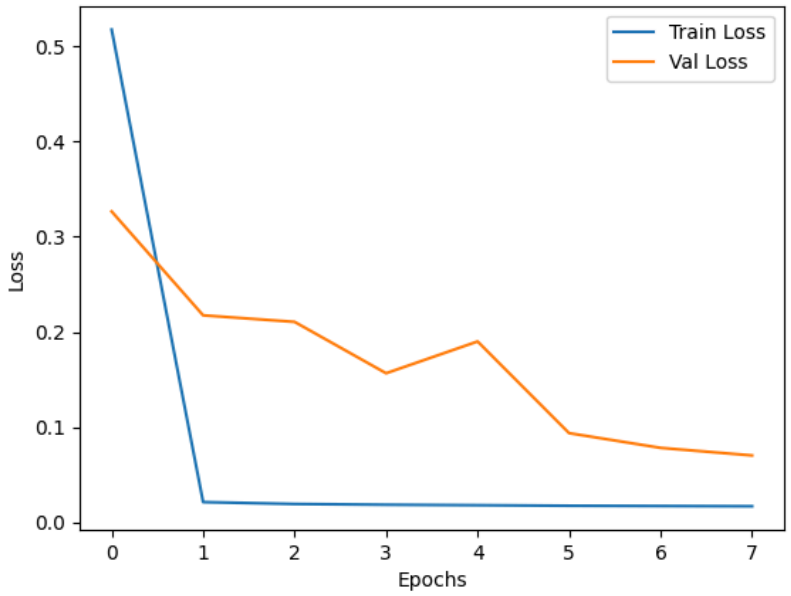
8.3 LSTM Best Fit

The best-fit LSTM shows balanced training, but validation loss remains high due to limited long-range modeling capacity.



8.4 Transformer Model (Extra Credit)

The Transformer achieves very low training and validation loss, showing strong generalization.



9. Key Observations

a. The LSTM models struggled to learn patterns over long text sequences.
b. A larger LSTM reduced training loss but ended up overfitting heavily.
c. Even the best LSTM setup could not generalize well to new data.
d. The Transformer performed much better, with very low loss and perplexity.
e. Self-attention helped the Transformer understand context more effectively than LSTMs.

10. Deliverables

Trained Models: Saved in outputs/models/

1. lstm_bestfit.pth
2. lstm_overfit.pth
3. lstm_underfit.pth
4. transformer.pth

Loss Plots: Saved in outputs/plots/

1. lstm_bestfit
2. lstm_overfit
3. lstm_underfit
4. transformer

Code Files: Saved in src/

1. data_preprocessing.py
2. model_lstm.py
3. model_transformer.py
4. train.py
5. evaluate.py
6. utils.py

Jupyter Notebook: NLM.ipynb

11. Conclusion

This assignment successfully demonstrates:

1. Construction of neural language models from scratch
2. Training for three variations: underfitting, overfitting and best fit
3. Use of perplexity for evaluation
4. How model capacity impacts over/underfitting