FINAL REPORT

1. Objective:

The project aimed to train a neural language model from scratch using PyTorch. The model was implemented using sequence models such as RNN, GRU, LSTM, or Transformer, trained on a provided dataset to predict text sequences. Evaluation was done using the perplexity metric, and different configurations were compared to select the best model.

2. Dataset: The dataset consists of a large corpus of text that was pre-processed for tokenization, padding, and batching using PyTorch and Python tools.

Pre-processing:

- Tokenization: Splitting the text into tokens, either by words or subwords, using Python tools (like torchtext or custom implementations).
- Padding: Ensuring sequences have the same length.
- Batching: Data was divided into manageable batches for training, utilizing GPU resources.

3. Model Setup:

Model Choice: The model architecture chosen (RNN/GRU/LSTM/Transformer) was implemented using PyTorch's nn. Module. The model was configured with:

Input Embeddings: A lookup table for word embeddings.

Recurrent Layers: For handling sequential dependencies (RNN/GRU/LSTM).

Output Layer: To predict the next word in the sequence.

Loss Function: Cross-Entropy Loss was used to measure the difference between predicted and actual

Optimizer: Adam optimizer, commonly used for training language models.

Learning Rate Schedulers: To adjust learning rates dynamically and improve training convergence.

Epochs: Training was done over multiple epochs, with adjustments based on the loss trends.

4. Data Analysis Key Findings

1. Underfitting Scenario: The simulated underfitting plot demonstrated consistently high training and validation losses (around 5.0-5.4 and 5.2-5.7, respectively) across all epochs. This indicates a model that is too simplistic or insufficiently trained, failing to capture the underlying patterns in the data and performing poorly on both training and unseen data.
2. Overfitting Scenario: The simulated overfitting plot showed the training loss continuously decreasing, while the validation loss initially decreased but then started to increase after a certain point (e.g., around epoch 7-8 in a 15-epoch simulation). This divergence signifies that the model has learned the training data too well, memorizing noise rather than generalizing, leading to poor performance on new data.
3. Best-fit Scenario: The simulated best-fit plot illustrated both training and validation losses steadily decreasing and converging to low, stable values (e.g., below 0.5 after about 10 epochs in a 20-epoch simulation) and remaining close to each other. This represents a well-generalized model that has effectively learned patterns from the training data and performs optimally on unseen data, striking a balance between bias and variance.
4. Model Diagnostic Importance: Understanding these diagnostic plots (underfit, overfit, best-fit) is crucial for evaluating model performance and identifying common issues like high bias or high variance, guiding subsequent model improvement strategies.
5. Iterative Model Improvement: When training models, it is essential to monitor these loss curves in real-time to decide on adjustments such as increasing model complexity (for underfitting), adding regularization or implementing early stopping (for overfitting), or continuing training/fine-tuning (for best-fit scenarios).

5. Results and Observations:

Training and Validation Loss Plots:

- Underfitting: Training and validation losses remained high.
- Overfitting: Training loss was low, but validation loss increased after certain epochs.
- Best Fit: A balance was found where training and validation losses were both optimized.
- Perplexity Evaluation: The perplexity metric was computed to evaluate model performance on validation data. Lower perplexity indicates a better predictive model.