Next-Word Prediction using MLP (Word-Level Text Generator)

Vadithya Harsha Vardhan Nayak
23110349@iitgn.ac.in
IIT Gandhinagar, Palaj
Gujarat, India

Observation and Comparision

1. Comparative Analysis

• Compare your two trained models (Category I vs Category II):
- Dataset size, vocabulary, context predictability
- Model performance (loss curves, qualitative generations)
- Embedding visualizations

• Summarize insights on how natural vs structured language differs in learnability.

Vocabulary

Category 1 – Holmes Dataset:
The total number of tokens (words) in the dataset is **115,481**, representing the complete text from *The Adventures of Sherlock Holmes*.
The total number of words (including duplicates) is **115,481**, and the vocabulary size, including reserved tokens, is **8,171**.

Category 2 – Linux Kernel Code Dataset:
The dataset contains a total of **1,372,952 tokens** obtained from the Linux Kernel source code.
The vocabulary size, including reserved tokens, is **41,324**, which reflects the large diversity of code-related symbols and identifiers.

The Top 10 most frequent and Bottom 10 least frequent words/tokens for both categories are provided in the Jupyter notebook files.

Model Performance: Loss

For **Category 1 dataset (Holmes text):**
Epochs run: **50**
Activation function: **ReLU**
At the end of the 50th epoch, the model converged with:
Train Loss: 0.9695 | **Validation Loss:** 10.6611 | **Validation Accuracy:** 0.1250

Epochs run: **100**
Activation function: **Tanh**
At the end of the 100th epoch, the model converged with:
Train Loss: 0.5393 | **Validation Loss:** 6.8573 | **Validation Accuracy:** 0.2421

For **Category 2 dataset (Linux Kernel Code):**
Epochs run: **50**
Activation function: **ReLU**
At the end of the 50th epoch, the model converged with:
Train Loss: 0.2952 | **Validation Loss:** 6.6400

Epochs run: **50**
Activation function: **Tanh**
At the end of the 50th epoch, the model converged with:
Train Loss: 0.3635 | **Validation Loss:** 5.3241

Across both datasets, the **Tanh activation function** outperformed **ReLU** in terms of validation loss and stability.

For the **Holmes dataset**, Tanh achieved a lower validation loss (6.8573 vs 10.6611) and higher accuracy (0.2421 vs 0.1250), indicating better generalization on natural language data.

For the **Linux Kernel dataset**, Tanh again showed improved performance with a lower validation loss (5.3241 vs 6.6400).
across both categories, the **Tanh activation function** demonstrated more stable learning and better generalization, particularly for datasets with sequential dependencies such as text and code. The **ReLU activation**, while effective in faster

convergence, tended to produce higher validation loss, suggesting mild overfitting and less effective handling of non-linear contextual relationships.

Embedding Visualizations

The embeddings for both datasets were visualized using **t-SNE** to project high-dimensional vectors into two dimensions for interpretation.

For the **Category 1 (Holmes)** dataset, clear **semantic clusters** were observed. Function words like *"the"*, *"and"*, and *"of"* grouped tightly, while nouns, verbs, and descriptive words formed distinct regions. This shows that the model effectively captured **semantic and contextual relationships** among words.

For the **Category 2 (Linux Kernel Code)** dataset, embeddings formed **dense, overlapping clusters** dominated by symbols such as *, (,), ;, and =. Keywords appeared close together, while variable names and identifiers were more scattered, indicating that the model primarily captured **syntactic rather than semantic structure**.

Overall, the Holmes embeddings highlight **semantic organization**, whereas the Linux embeddings emphasize **syntactic regularity**.

How natural vs structured language differs in learnability?

Natural language is statistically easier for models to learn due to its high redundancy, flexible structure, and abundance of recurring patterns. Even when predictions are partially incorrect, the surrounding context often compensates for the error, preserving overall coherence. Its probabilistic nature allows multiple valid word combinations, enabling models to generalize smoothly across similar linguistic contexts.

In contrast, structured programming languages like C++ are significantly more difficult to learn. They follow rigid syntactic and logical rules where each token serves a precise function and can only be followed by specific valid elements. A

single incorrect prediction disrupts the entire grammatical structure. Because programming languages lack redundancy, models must capture exact token-level dependencies rather than relying on broader semantic cues.