Shakespeare Text Generation with Fast Weights Model

Exploring Dynamic Weights and their Connection to Attention

Ioja Denis

Fast Weights and Transformers

- Background: The concept of fast weights was introduced in a 1991 paper. It involves dynamically updating weights during processing.
- Connection to Transformers: Fast weights share a conceptual similarity with the attention mechanism in Transformers. Both enable adaptive memory updates.
- Key Differences: Transformers use scaled dot-product attention for parallel processing, while the covariance-based fast weight updates are more sequential. Transformers compute attention scores explicitly, whereas fast weights in this implementation use a covariance-based update.

Project Goals

- Understand the theoretical foundations of Fast Weights.
- Implement a Fast Weights model in PyTorch.
- Apply the model to Shakespearean text generation.
- Analyze performance and explore the connection to attention.

Project Components

- The project consists of five Python scripts: config.py, preprocess.py, train.py, evaluate.py, and generate.py.
- Each script handles a specific stage of the workflow.
- This modular design promotes code organization and maintainability.

Configuration Management

- config.py centralizes all hyperparameters and file paths.
- Key parameters include:
    - SEQ_LENGTH: Input sequence length.
    - LEARNING_RATE: Learning rate.
    - NUM_EPOCHS: Number of training epochs.
    - TEMPERATURE: Temperature for text generation.
    - GENERATE_LENGTH: Length of generated text.
- This script ensures reproducibility and simplifies experimentation.

Data Preparation

- preprocess.py prepares the raw Shakespeare text for training.
- Steps include:
    - Loading raw text data.
    - Creating a vocabulary (character-to-index and index-to-character mappings).
- Splitting text into fixed-length input-output sequences using SEQ_LENGTH.
- Saving the tokenizer and processed data to pickle files.
- This script transforms text into a numerical representation suitable for the model.

Model Training

- train.py trains the Fast Weights model. The model architecture consists of:
- Embedding layer: Converts characters to vector representations.
- Fast Weight layer: Implements the dynamic weight.
- Linear output layer: Maps hidden states to character probabilities.
- Training process:
    - Loads preprocessed data.
    - Initializes the model, loss function, and optimizer.
    - Iterates through training data for NUM_EPOCHS.
    - Calculates predictions, loss, and updates model parameters.
    - Tracks and visualizes training loss.
    - Saves the trained model.

Model Evaluation

- evaluate.py assesses the model's performance.
- Steps:
    - Loads the tokenizer, validation data, and trained model.
    - Calculates predictions on the validation set.
    - Computes average loss and accuracy.
    - Evaluation on a held-out validation set is crucial to avoid overfitting and measure generalization.

Text Generation

- generate.py uses the trained model to generate new text. Process:
  - Loads the tokenizer and trained model.
  - Starts with a seed_text.
  - Iteratively predicts the next character using the model.
  - Employs temperature sampling to control randomness.
  - Appends the predicted character to the generated text.
  - Saves the generated text.
  - Temperature sampling adjusts the probability distribution, influencing the creativity and coherence of the generated output.

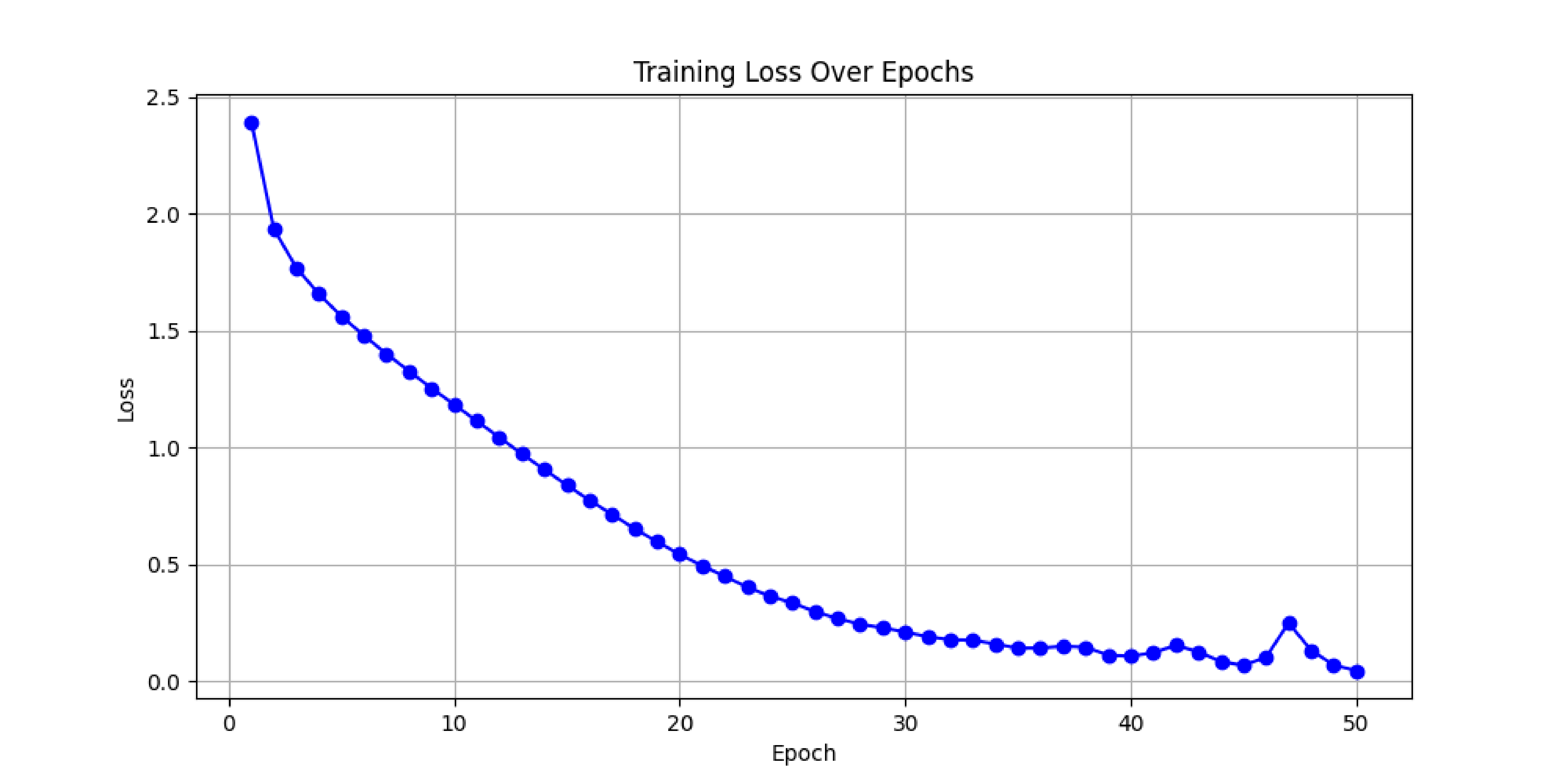
Workflow Overview

- Preprocessing: Raw text is processed, tokenized, and converted into input-output sequences.
- Training: The model learns to predict the next character in a sequence, minimizing the loss.
- Evaluation: Model performance is assessed on a separate validation set.
- Generation: The trained model generates new text based on a seed and temperature.

Results

- Loss Curves: Training loss decreased smoothly and converged, indicating effective learning.
- Validation Accuracy: 99.70% accuracy and 0.0291 loss suggest strong generalization.
- Analysis: Captures Shakespearean rhythm and structure well. Limited SEQ_LENGTH likely causes incoherence in longer phrases. High validation metrics confirm strong learning, but semantic understanding over longer spans is lacking.

Results


Training Loss Over Epochs
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50

Results

Generated Text:

To be, or not to be, that is the question: by, if his mistres sweet.
Farewell, my coz.
BENVOLIO
Soft! I will good cover of the weakest goes
to the wall.
GREGORY
The quarrel is between our masters and us their men's
hands and they unwashed at this fair volume lies
Find written in the margent of his eyes.
This precious book of love, this unbound lover,
To beauty till this night.
Enter CAPULET in his gown, and LADY CAPULET
Speak briefly, can you like of Paris' love?
JULIET
I'll look to like, if looking liking move:
But no more deep will I

Conclusion

- This project successfully implemented a Fast Weights model for Shakespearean text generation.
- The model achieved a validation accuracy of 99.70% with a validation loss of 0.0291 and generated text exhibiting some Shakespearean stylistic elements.
- Limitations in long-range coherence suggest the need for further exploration.

Thank You