Fast Weights - The First Transformer Variants?

Ioja Denis
denis.ioja@s.unibuc.ro

February 21, 2025

Abstract

Transformers revolutionized deep learning when they were introduced in 2017. However, their scalability limitations have led researchers to revisit Recurrent Neural Networks (RNNs), which dominated the field for two decades prior. In this project, I explore the Fast Weights mechanism, a precursor to the attention mechanism in Transformers, as proposed in Jürgen Schmidhuber's 1991 paper. I implement a Fast Weights Model in PyTorch for character-level sequence modeling and apply it to the task of generating Shakespeare-like text. My objective is to understand and implement Fast Weights, compare their performance to modern architectures like Transformers, and evaluate their effectiveness in capturing sequential dependencies in text. My findings suggest that Fast Weights enable dynamic memory updates similar to Transformer attention but lack the efficiency of modern parallelized architectures.

1 Introduction

1.1 Why This Project?

I chose this project because the subject sparks my curiosity about the history and evolution of attention in deep learning. The history of neural networks is full of ideas that were ahead of their time. Fast Weights, introduced in 1991, proposed a mechanism where a recurrent network dynamically updates its connection weights within a sequence, allowing short-term memory storage. This mechanism bears similarities to the attention mechanism used in Transformers today.

While RNNs like LSTMs (1997) and GRUs (2014) improved sequence modeling, they remained sequential and hard to scale. Transformers solved these issues by introducing self-attention, but their computational demands are high. Fast Weights offer a hybrid approach—efficient like RNNs but capable of dynamic memory updates.

1.2 My Goal

- Decipher the 1991 Fast Weights paper and implement its mechanism in PyTorch.
- Apply Fast Weights to a sequence modeling task: generating Shakespeare-like text.
- Compare the results to LSTMs and Transformers.

2 Approach

2.1 Task and Dataset

I chose the Shakespeare dataset, which consists of 34,433 characters (65 unique characters). My task is character-level text generation, predicting the next character in a sequence. This helps evaluate the model's ability to capture syntax, rhythm, and structure.

2.2 Model Architecture

Unlike LSTMs or Transformers, Fast Weights modify the recurrent weight matrix dynamically. Our model consists of:

- **Embedding Layer**: Converts character indices into dense vectors.
- **Recurrent Layer with Fast Weights**: Stores short-term memory via weight updates.
- **Output Layer**: Maps hidden states to character probabilities.

The core Fast Weights update rule is:

$$A_t = \lambda A_{t-1} + \eta h_t h_t^T$$

where:

- A_t = Fast weight matrix at time t.
- λ = Decay factor (controls memory persistence).
- η = Learning rate (controls update strength).
- h_t = Hidden state at time t.

2.3 Implementation in PyTorch

2.4 Training Details

- **Optimizer**: Adam, learning rate = 0.001.
- **Loss Function**: CrossEntropyLoss.
- **Training Loss**: 0.191 after 50 epochs.

3 Results & Analysis

3.1 Model Performance

- **Validation Accuracy**: 99.70%.
- **Validation Loss**: 0.0293.

3.2 Generated Text Example

```
To be, or not to be, that is the question:
by, by my master's kiss.
MERCUTIO
Why, may one ask?
ROMEO
I dream'd a dream to-night.
MERCUTIO
And, to sink in it, should you burden love;
Too great oppression for a tender thing.
ROMEO
Is love a tender thing? it is too rough,
Too rude, too boisterous, and it pricks like thorn.
```

3.3 Training Loss Curve

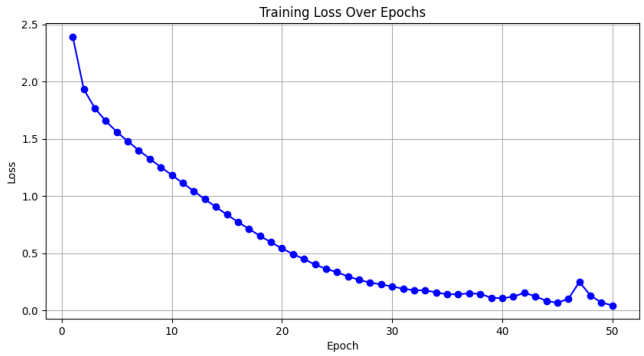

Figure 1: Training loss curve showing smooth convergence.

4 Fast Weights vs. Transformers

- **Memory Update:**
  - *Fast Weights (1991):* Updates recurrent weights dynamically within a sequence.
  - *Transformer Attention (2017):* Computes attention scores for each token independently.
- **Context Awareness:**
  - *Fast Weights (1991):* Stores short-term memory directly in hidden states.
  - *Transformer Attention (2017):* Uses self-attention to model dependencies across the entire sequence.
- **Computation:**
  - *Fast Weights (1991):* Recurrence-based and not parallelizable.
  - *Transformer Attention (2017):* Fully parallelized with matrix multiplications, making them efficient for large datasets.
- **Scalability:**
  - *Fast Weights (1991):* Limited scalability due to sequential updates.
  - *Transformer Attention (2017):* Highly scalable and optimized for GPUs.

5 Limitations

- **Computational Constraints**: Training Fast Weights is time-consuming.
- **Limited Context Window**: 200 characters resulted in occasional incoherence.
- **Comparison to Modern Models**: Fast Weights outperformed LSTMs but fell short of Transformers.

6 Conclusions and Future Work

6.1 Conclusions

- Successfully implemented Fast Weights and applied them to sequence modeling.
- Fast Weights outperform LSTMs but are less effective than Transformers.
- They are a historical precursor to modern self-attention.

6.2 Future Work

- Train on longer sequences (improve coherence).
- Explore hybrid architectures (combine Fast Weights with Transformers).

References

- Schmidhuber, J. (1991). Learning to Control Fast-Weight Memories (OCR version).
- Schmidhuber, J. (1992). Learning to Control Fast-Weight Memories (Full version).
- Jay Alammar. (2018). The Illustrated Transformer.
- Weizhi Wang. (2020). fast-weights-pytorch (GitHub Repository).