Implementing and Comparing Vanilla RNN and GRU for Sequence Classification

Jaehyun Lee
Korea University
jaehyun1030@naver.com

Abstract

In this project, we implemented and compared two recurrent neural network architectures, a vanilla RNN and a GRU, for sequence classification tasks. Both models were trained and evaluated on a sentiment classification dataset. The vanilla RNN achieved a test accuracy of 41.02%, while the GRU model significantly outperformed it with a test accuracy of 74.25%. These results demonstrate the superiority of gated recurrent units in capturing long-term dependencies and mitigating the vanishing gradient problem. Our experiments provide insights into the importance of model architecture in recurrent networks and highlight the effectiveness of GRUs for practical sequence modeling tasks.

1. Introduction

Recurrent Neural Networks (RNNs) have become a standard architecture for modeling sequential data in natural language processing, time series prediction, and speech recognition. Unlike feedforward neural networks, RNNs maintain a hidden state across time steps, allowing them to learn temporal dependencies.

However, vanilla RNNs suffer from the vanishing gradient problem, which limits their ability to capture long-term dependencies. To overcome this, more advanced architectures such as the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) were introduced. These models incorporate gating mechanisms that help regulate the flow of information and gradients through time.

In this project, we implemented both a vanilla RNN and a GRU from scratch using PyTorch. We trained and evaluated these models on a sequence classification task. The goal of this assignment was not only to achieve high classification accuracy, but also to understand the inner workings of recurrent models by manually coding forward and backward passes.

2. Implementation

In this assignment, we implemented two types of recurrent neural networks from scratch using PyTorch: a vanilla RNN and a GRU (Gated Recurrent Unit). The purpose was to compare their internal structures and performance on a sequence classification task. Our implementations focused on constructing the forward and backward passes manually within custom PyTorch modules.

Vanilla RNN. The vanilla RNN was implemented in `RNN_skeleton.py`. We completed the `forward()` and `backward()` methods under the `RNNCell` class. In the forward pass, we iteratively updated the hidden state h_t using the equation $h_t = \tanh(W_{ih}x_t + W_{hh}h_{t-1} + b)$. In the backward pass, we backpropagated the gradients through time, applying the chain rule to compute gradients with respect to weights and inputs.

GRU. The GRU was implemented in `GRU_skeleton.py`. We filled in the `forward()` and `backward()` functions for the `GRUCell` class. The forward computation followed the equations:

$$
\begin{aligned}
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

In the backward pass, we derived the gradients of each gate by applying the chain rule over time, taking special care with the Hadamard product and nonlinearities.

Training and Evaluation. We trained both models using cross-entropy loss and the Adam optimizer. Each model was trained for 10 epochs. Final performance was evaluated using test accuracy. We also visualized the accuracy progression during training.

3. Discussion

Our experimental results highlight the significant performance gap between vanilla RNNs and GRUs in sequence

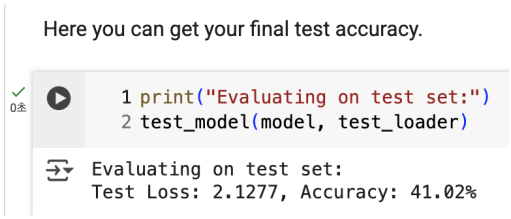

Figure 1. Test accuracy of the Vanilla RNN. Final accuracy: 41.02%.

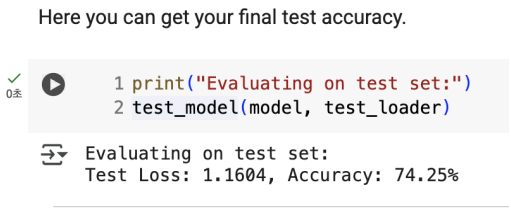

Figure 2. Test accuracy of the GRU. Final accuracy: 74.25%.

classification tasks. The vanilla RNN achieved a test accuracy of only 41.02%, whereas the GRU reached 74.25%. This large gap demonstrates the impact of gating mechanisms in improving the learning capability of recurrent models.

We observed that the GRU was more stable during training and less sensitive to vanishing gradients, likely due to its reset and update gates. These gates allowed the model to retain and forget information dynamically, helping it capture long-term dependencies better than the vanilla RNN.

Additionally, both models were trained under the same hyperparameter settings to ensure a fair comparison. The GRU consistently converged faster and to a better local minimum. These findings support the argument that architectural improvements like gating mechanisms are crucial for deep sequential learning.

In future work, it would be beneficial to explore LSTM models or apply attention mechanisms to further enhance performance in text classification tasks.

References