T2-20-AI 825 / Assignment 1 Report

1st Shashank Reddy Chirra
IMT2018069
IIIT Bangalore
shashank.reddy@iiitb.org

I. Introduction

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. A plain vanilla RNN has already been implemented in class.

In this assignment, we add bias terms(bh and by) to the RNN, we then try using different optimization algorithms such as Momentum and Adagrad, we finally observe the affect of the size of of the hidden matrix on the RNN.
I used the training loss as a measure of model performance.

II. Bias

The calculations for the backward pass for the terms is given below.

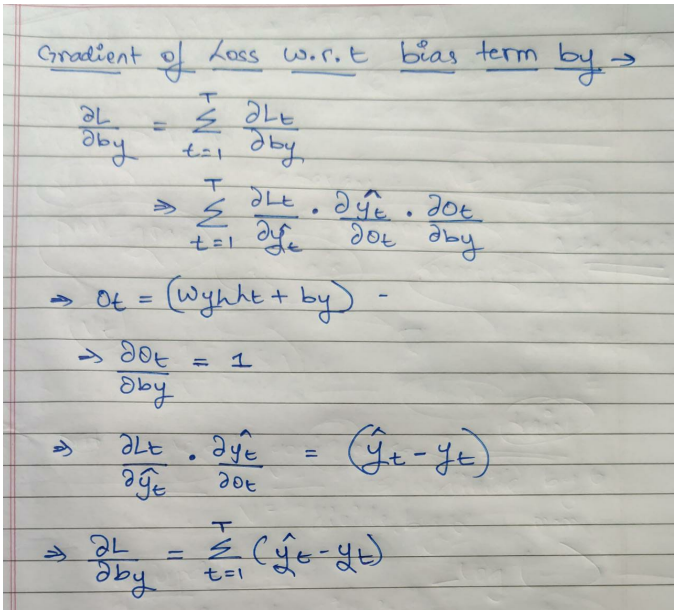

Fig. 1. Gradient w.r.t by

We will now discuss the effect of bias terms on the RNN. Bias allows you to shift the activation function by adding a constant (i.e. the given bias) to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value. Since, the bias offers more flexibility to the RNN, one would expect that the RNN would perform better when add the bias terms as well.

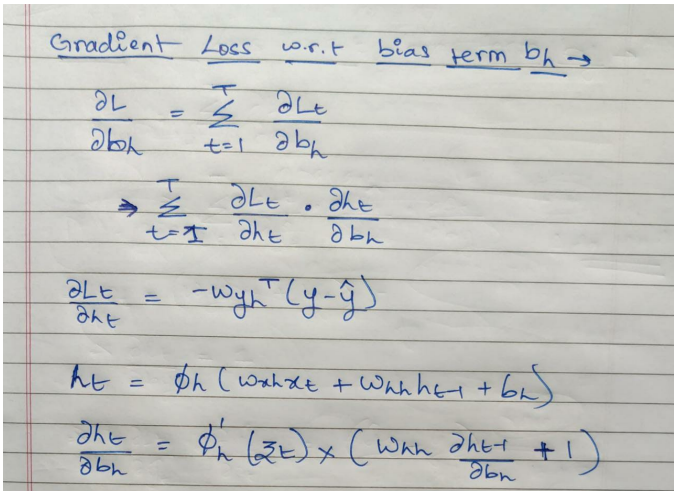

Fig. 2. Gradient w.r.t bh

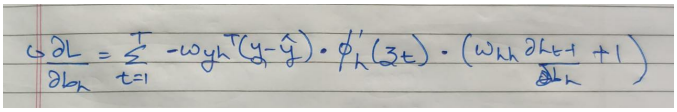

Fig. 3. Gradient w.r.t bh

My observations support this claim, although the model had a lesser training loss when the bias terms were present, I found that the difference between the loss was low (order of 1e-3). I thought this was because the bias terms were being set close to 0 after training, however when I checked the values, I observed that this was not the case. Thus, I came to conclude that adding the bias terms helped fine tune the model in this case.

III. Gradient Descent Algorithms

I tested the performance of a model using 3 different gradient descent algorithms. The first being plain vanilla gradient descent (SGD), gradient descent with momentum and the Adagrad algorithm. Although Adagrad and Mometum algorithms are designed to effeciently traverse complex manifolds, I found that the SGD algorithm performed the best. This, was probably because the manifold was quite smooth and ease to traverse in this case.
The training loss in the case of the Momentum algorithm was very close to that of the SGD algorithm, and for a very high number of iterations, they converge to almost the same value.

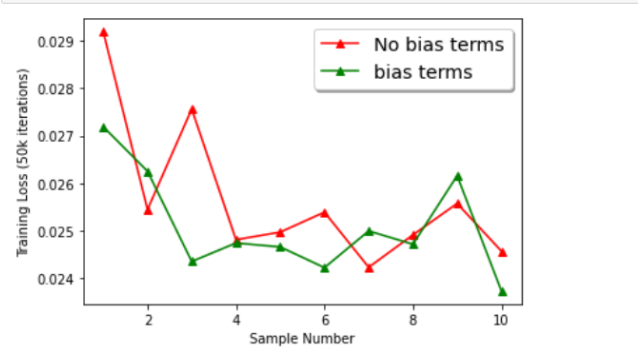

Fig. 4. Effect of bias terms.

However as expected, it takes time to build momentum and thus has a higher loss for lower number of iterations.

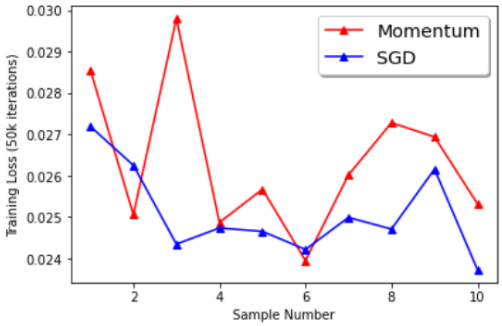

Fig. 5. Momentum vs SGD

In the case of Adagrad, although it eliminates the need to manually tune the learning rate, it has a major weakness that it causes the the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge. Thus, I found that the algorithm failed to converge after a certain number of iterations, as the learning rate had diminshed. Thus, the training loss in Adagrad was significantly higher than the previous 2 cases.

IV. Size of hidden matrix

Increasing the size of the hidden matrix increases the number of parameters of the Neural Network. Thus, it is expected that the training loss will reduce with an increase in the size of the hidden matrix, however one would expect that this would come at the cost of over fitting and a high validation loss.

My observations support this theory, and thus I found that the training loss was reducing when I increased the hidden size, when the hidden size was 20, the training loss was very close to 0 indicating over fitting.

Acknowledgment

I would like to thank Prof. Vishwanath G for such an interesting assignment, that gave us the oppurtunity to build an RNN from the ground up.

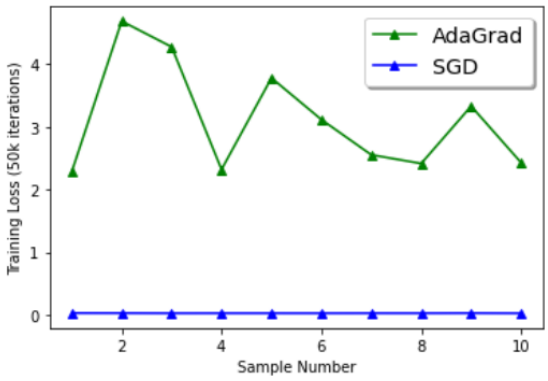

Fig. 6. Adagrad vs SGD

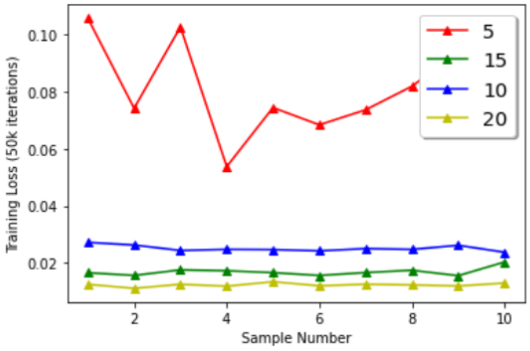

Fig. 7. Effect of Hidden Size