Visual Recognition Part 2 Assignment 1

Vignesh Bondugula - IMT2019092

April 23, 2022

RNN Implementation

1 RNN from Scratch

1.1 Derivation of bias update equations

$$\text{We know Loss function } L(\hat{y}, y) = -\sum_{k=1}^{T} y_k log(\hat{y_k})$$

If the ground truth is at index $k = p, y_p = 1$ and $y_k = 0 \forall k \neq p$.Therefore $L = -log(\hat{y_p})$

1.1.1 Derivation of Gradient Loss Function with respect to b_y

$$\frac{\partial L}{\partial b_y} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial b_y} = \sum_{t=1}^{T} \frac{\partial L_t \partial \hat{y}_t \partial o_t}{\partial \hat{y}_t \partial o_t \partial b_y}$$

$$\text{We know, } \frac{\partial L}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} = (\hat{y}_t - y_t)$$

$$o_t = W_{yh} h_t + b_y$$

$$\Rightarrow \frac{\partial o_t}{\partial b_y} = 1$$

$$\frac{\partial L_t}{\partial b_y} = (\hat{y}_t - y_t)$$

$$\text{Therefore, } \frac{\partial L}{\partial b_y} = \sum_{t=1}^{T} (\hat{y}_t - y_t)$$

1.1.2 Derivation of Gradient Loss Function with respect to b_h

$$\frac{\partial L}{\partial b_h} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial b_h} = \sum_{t=1}^{T} \frac{\partial L_t \partial h_t}{\partial h_t \partial b_h}$$

$$\text{We know, } \frac{\partial L_t}{\partial h_t} = -W_{yh}^T (y_t - \hat{y_t})$$

$$h_t = f(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$

$$\text{Let, } z_t = W_{xh} x_t + W_{hh} h_{t-1} + b_h \text{ ,so}$$

$$h_t = f(z_t) \text{ —}\mathbf{(1)}$$

$$\frac{\partial h_t}{\partial b_h} = f'(z_t)(1 + W_{hh} \frac{\partial h_{t-1}}{\partial b_h}) \text{ —}\mathbf{(2)}$$

$$\frac{\partial L_t}{\partial b_h} = -W_{yh}^T (y_t - \hat{y_t}) f'(z_t)(1 + W_{hh} \frac{\partial h_{t-1}}{\partial b_h})$$

$$\frac{\partial L}{\partial b_h} = \sum_{t=1}^{T} -W_{yh}^T (y_t - \hat{y_t}) f'(z_t)(1 + W_{hh} \frac{\partial h_{t-1}}{\partial b_h}) \text{ , from 1 and 2}$$

This above equation is recursive. We can calculate recursively at time t similar to gradient computation implementation.

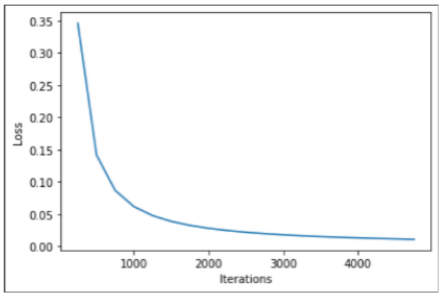
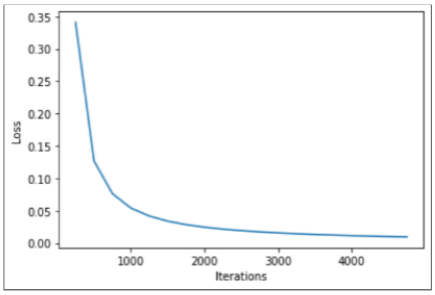
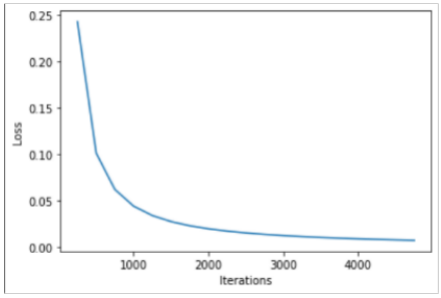
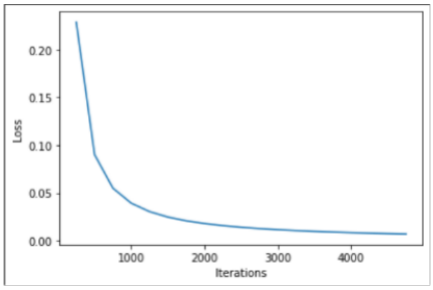
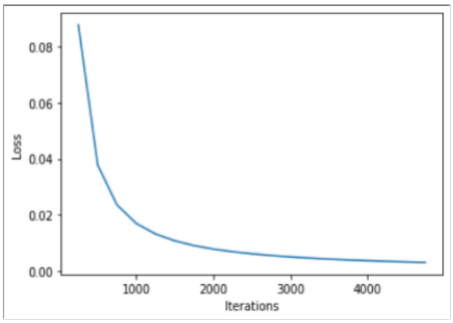
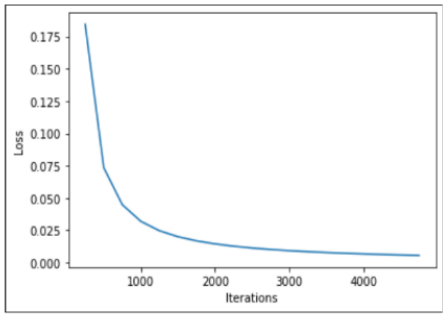
1.2 Implementation

I have implemented RNN from scratch including the derived bias update equations above. I have also tried to plot the losses vs iterations without bias update and with bias update. I trained the model with 5000 iterations and a learning rate of 0.01.

1.3 Results

I have observed the following results,

- With increase in hidden layer size, we can see the loss generally decreases in both the cases of bias update and without bias update.
- Clearly the loss in case of without bias update is less than loss in case of with bias update. But gradually with increase in the hidden vector size, the loss in case of bias update is getting lower than the other case.

Hidden Vector Size	With Bias Update	Without Bias Update
50	 Loss: 0.01	 Loss: 0.009
75	 Loss: 0.0076	 Loss: 0.006
100	 Loss: 0.003	 Loss: 0.0055

The above are the plots observed and losses calculated in various cases.

2 RNN using pytorch

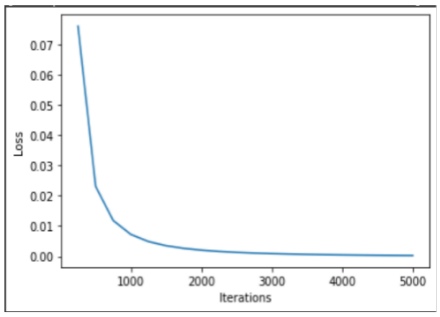
2.1 Implementation

I have implemented RNN in pytorch in following steps,

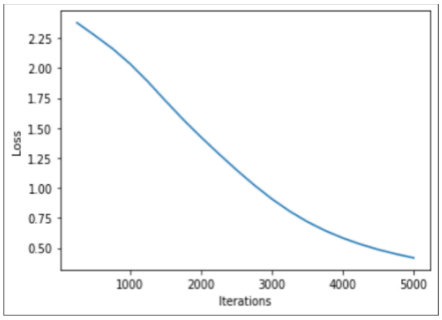
- Created a Datareader class which will read the input words and make it into a inputs and targets.

- Then One hot encoded the inputs in such a way that the rnn pytorch model accepts it.
- Created a RNN model and ran it with different epochs and learning rate. Also tried training with different optimizers.
- Plotted the losses vs iterations plots to compare.
- Tried to train the model with "Acknowledgement" and tried to predict the string with the starting character to be 'A'.

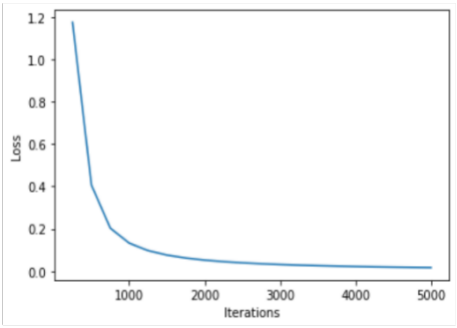
2.2 Results



Epochs: 5000
Learning Rate: 0.01
Optimizer: Adam
Loss: 0.0002



Epochs: 5000
Learning Rate: 0.01
Optimizer: SGD
Loss: 0.417



Epochs: 5000
Learning Rate: 0.01
Optimizer: SGD
Momentum: 0.9
Loss: 0.0169

- Above are few results obtained in different scenarios.
- Adam optimizer was converging faster than SGD optimizer. Even with momentum Adam was performing better.
- "Acknowledgement" was being predicted by the Adam model at 1000 iterations with 0.01 learning rate.

2.3 Predicting the word edgement

To predict the word edgement I have modified the predict function. To get the context for the letter e, I have found out the hidden layer vector till the letter e and passed it to predict the word edgement. I have ran the model for 10000 iterations and 0.09 learning rate. The loss plot looks like,

```
[68] predict('e', 8)

     'edgement '
```