Assignment 1

Shourabh Payal (MT2020054)

1. **Derive expressions for $\frac{\partial L}{\partial b_h}$ and $\frac{\partial L}{\partial b_y}$ for the RNN discussed in Lectures 2-3. Include the derived bias update equations in the RNN code shared and train the RNN for a word. Record relevant observations during training after adding bias terms.**

   We have

$$L = -y_t \; log(\hat{y}_t)$$

$$\hat{y}_t = softmax(z_t)$$

$$z_t = W_{yh}\, h_t + b_y$$

   (a) Derivative wrt b_y

$$\frac{\partial L}{\partial b_y} = \sum_{i=1}^{T} \frac{\partial L_i}{\partial b_y}$$

$$= \sum_{i=1}^{T} \frac{\partial L_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} \frac{\partial z_i}{\partial b_y}$$

   - $\frac{\partial L_i}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$
   - $\frac{\partial \hat{y}_i}{\partial z_i}$ $\quad = \hat{y}_i(1 - \hat{y}_i)$ when i = k
     $\quad = -\hat{y}_i \;\; \hat{y}_k$ when i =/= k
   - $\frac{\partial z_i}{\partial b_y} = 1$
   - Combining and solving we get

$$\frac{\partial L}{\partial b_y} = \sum_{i=1}^{T} (\hat{y}_i - y_i)$$

(b) Derivative wrt b_h

Let's consider for time t+1 first

$$\frac{\partial L_{t+1}}{\partial b_h} = \frac{\partial L_{t+1}}{\partial h_{t+1}} . \frac{\partial h_{t+1}}{\partial b_h}$$

We know from derivations of $\dfrac{\partial L_{t+1}}{\partial W_{xh}}$ and $\dfrac{\partial L_{t+1}}{\partial W_{hh}}$ that

$$\frac{\partial L_{t+1}}{\partial h_{t+1}} = -{W_{yh}}^T (y_t - \hat{y}_t)$$

Now $\dfrac{\partial h_{t+1}}{\partial b_h} = ?$

Let $z_{t+1} = W_{xh} X_{t+1} + W_{hh} h_t + b_h$

We know $h_{t+1} = \phi_h(z_{t+1})$

$$\therefore \frac{\partial h_{t+1}}{\partial b_h} = \phi_h^{'}(z_{t+1}) . \frac{\partial z_{t+1}}{\partial b_h}$$

$$\frac{\partial z_{t+1}}{\partial b_h} = W_{hh} \frac{\partial h_t}{\partial b_h} + 1$$

$$\frac{\partial h_{t+1}}{\partial b_h} = \phi_h^{'}(z_{t+1}) . (W_{hh} \frac{\partial h_t}{\partial b_h} + 1) \qquad \text{(Reccursion !)}$$

$$\frac{\partial h_t}{\partial b_h} = \phi_h^{'}(z_t) . (W_{hh} \frac{\partial h_{t-1}}{\partial b_h} + 1)$$

and so on...

Total gradient on all losses wrt $b_h = \Sigma_{k=1}^{T} \dfrac{\partial L_k}{\partial b_h}$

(c) Observations after adding bias term

```
for bias in biases:
  plt.plot(bias_dic[bias]['iter'],bias_dic[bias]['loss'], label='bi
  plt.xlabel('iterations')
  plt.ylabel('loss')
  plt.legend(loc="upper right")
```

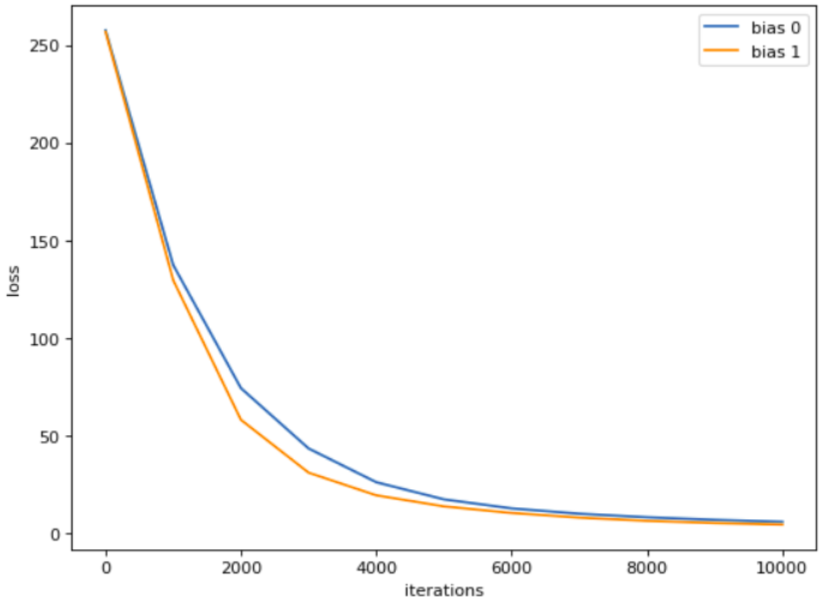

Figure 1: Bias comparison

Did not observe significant difference before and after adding bias.

2. **Replace the basic SGD technique used in the function update_model with any other sophisticated gradient update technique popular in literature. Record relevant observations during training after modifying gradient update method.**

   AdaGrad, AdaDelta, Adam

   Let's consider a simple high level comparison between Basic SGD and Adagrad using their respective update equations.

   Basic SGD:

   $$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

   Here η is learning rate and it is same for all weights and is constant.

   AdaGrad:

   $$W_t = W_{t-1} - \eta_t^{'} \frac{\partial L}{\partial W_{t-1}}$$

   Here $\eta_t^{'}$ is adaptive learning rate and it is different for each weight and changes each iteration.

   $$\eta_t^{'} = \frac{\eta}{\sqrt{\alpha_{t-1} + \epsilon}}$$

   ϵ is a small positive value to avoid division by zero problem.

   $$\alpha_{t-1} = \textstyle\sum_{i=1}^{t-1} \left(\frac{\partial L}{\partial W_{i-1}} \right)^2$$

   Advantages of Adagrad

   - No need of manually tuning η
   - Takes care of right learning rate for sparse and dense features

   Disadvantages of Adagrad

   - α_{t-1} can become very large as t increases as a result convergence can be slowed down.

AdaDelta

To overcome the disadvantage of adagrad we imply a simple technique of exponential decaying averages.

Instead of α_{t-1} we have EDA_{t-1}

$$EDA_{t-1} = (\gamma)EDA_{t-2} + (1-\gamma)\left(\frac{\partial L}{\partial W_{t-1}}\right)^2$$

Growth of the denominator term in $\eta_t^{'}$ is therefore controlled.

Adam (Adaptive moment estimation)

Adam tries to bring together the above methods along with the concept of moments.

- Mean : First order moment
- Variance : Second order moment

Let's define $g_t = \dfrac{\partial L}{\partial W_{t-1}}$

$$m_t = \beta_1(m_{t-1}) + (1-\beta_1)g_t$$

$$v_t = \beta_2(v_{t-1}) + (1-\beta_2)g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^2}$$

$$\hat{v}_t = \frac{v_t}{1-(\beta_2)^2}$$

Here $0 \leq \beta_1, \beta_2 \leq 1$

Typical good values are $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Our final update equation is

$$W_t = W_{t-1} - \eta\left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}\right)$$

Results on code

We will be implementing Adam optimizer. Below are the results of experiment.

```
********************** Optimizer =  sgd  **********

iter num: 0  loss: 256.32902761937254
acJKZS6jZBKLthj6i0VHmpXZFMT97CsSV8RezCa4HBUsWjHZYNE

iter num: 1000  loss: 125.49305610208975
aH90FfMQqjgFKopeRVMmsgdTnkhmT 7qr9UNjSz z1J5RFxWSSh

iter num: 2000  loss: 44.99013560354285
aq9BNfkptudhKVdAOZhmOkJRnktmWMpn5gtr6tFTP64z3J89EQO

iter num: 3000  loss: 21.38854662109456
aKcdKughijcJKVveKcdefghmn5IqbkhijclujkJmjdpmgs2fgpb

iter num: 4000  loss: 13.041387544632776
a9cdefcheqghtfvwefghifghzjMeurhePghiOkpqrohK9MNOPli

iter num: 5000  loss: 9.071217496881347
ajghijklmnopqrst5y 67z9UXyX3W56781 3456709 e056789

iter num: 6000  loss: 6.8450964052604695
aOkhefghijklmnopqrstuv5xyzABCDEQGHIJKLMNOPQ7STUVsX3

iter num: 7000  loss: 5.454772212054938
abcdefghijklmnohqHWtijkpmndpqrhtfghijklmnopqrstuvwx

iter num: 8000  loss: 4.515745324216776
abcdiRcdefgXirghULgtbndefghijklmnopqrstuvwxyzABCDEL

iter num: 9000  loss: 3.8435387761963145
aKcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVMXY

iter num: 10000  loss: 3.3405925170849944
abchefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOSQRSTUVWXY
```

(a) Normal SGD

```
********************** Optimizer =  adam  ************************

iter num: 0  loss: 256.92660123981443
aBJX9qhleO q8FZBB63aFRZeigHqyqDG9CQqtTUjx2CzwZnXB2oFQbdZa00zi6e

iter num: 1000  loss: 17.032066685685322
abcdefghijkCDAB0DJKLnopVRRSTUKxX9Z01X9bcdfqBCDY6BH9JKLt9Z0wWXYZ

iter num: 2000  loss: 0.129939943308596
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 3000  loss: 0.005504777963686187
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 4000  loss: 0.0017524094885176364
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 5000  loss: 0.0009855137710198826
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 6000  loss: 0.0006776010023463483
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 7000  loss: 0.0005142756096922056
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 8000  loss: 0.0004136866833397377
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 9000  loss: 0.0003457091330526938
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

iter num: 10000  loss: 0.0002967732773813102
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

(b) Adam

```python
for optimizer in optimizers:
  plt.plot(optimizer_dic[optimizer]['iter'],optimizer_dic[optimizer]['loss'],
  plt.xlabel('iterations')
  plt.ylabel('loss')
  plt.legend(loc="upper right")
```

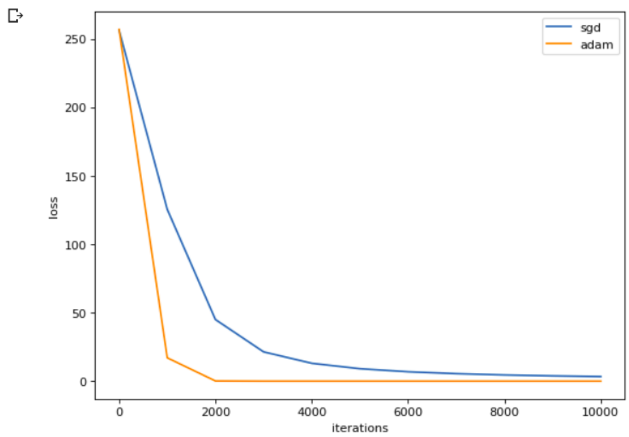

Figure 3: Optimizer comparison

The Adam optimizer converges a lot quickly compared to the regular SGD for this problem.

3. **Experiment with various hidden vector sizes and record your observations**

```
for layer in layers:
  plt.plot(layer_dic[layer]['iter'], layer_dic[layer]['loss'], labe
  plt.xlabel('iterations')
  plt.ylabel('loss')
  plt.legend(loc="upper right")
```

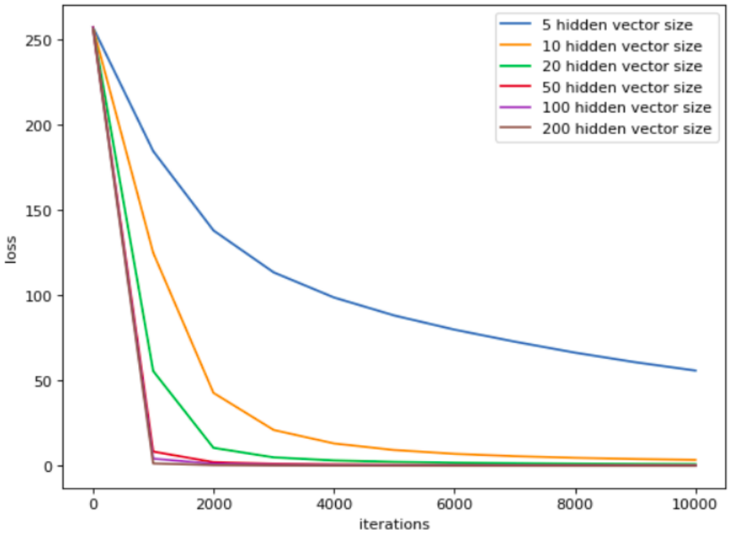

Figure 4: Hidden layer vector size comparison using SGD optimizer

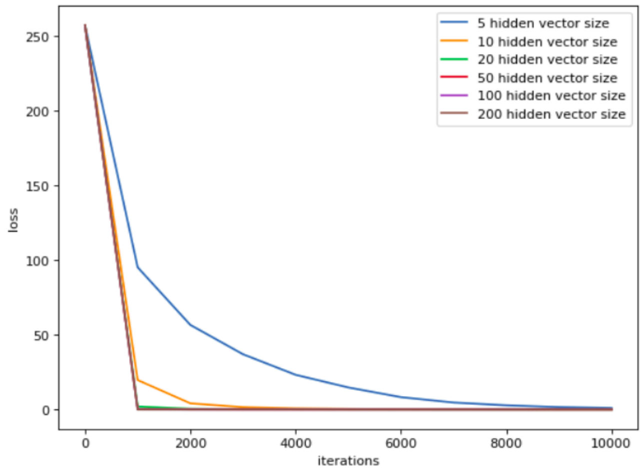

Figure 5: Hidden layer vector size comparison using Adam optimizer