EE5179:Deep Learning for Imaging

Programming Assignment 3: MNIST Classification using RNN

Intoduction

To demonstrate how to use recurrent neural network to predict the famous handwritten digits “MNIST”.

Libraries

```
import numpy as np
import torch
import matplotlib.pyplot as plt
from torchvision.datasets import MNIST
import torch.nn as nn
from tqdm import tqdm
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam #Adam for GD
import time
```

Data Loading

The most important argument of DataLoader constructor is dataset, which indicates a dataset object to load data from. PyTorch supports two different types of datasets:

- map-style datasets,
- iterable-style datasets. \ This is an iterable-style datasets, So we next use the function iter() and next()

Parameters:

input_size — The number of expected features in the input x

hidden_size — The number of features in the hidden state h

num_layers — Number of recurrent layers. E.g., setting num_layers=2 would mean stacking two RNNs together to form a stacked RNN, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1

nonlinearity — The non-linearity to use. Can be either 'tanh' or 'relu'. Default: 'tanh'

bias — If False, then the layer does not use bias weights b_ih and b_hh. Default: True

batch_first — If True, then the input and output tensors are provided as (batch, seq, feature). Default: False

dropout — If non-zero, introduces a Dropout layer on the outputs of each RNN layer except the last layer, with dropout probability equal to dropout. Default: 0

bidirectional — If True, becomes a bidirectional RNN. Default: False

Ref : https://medium.com/@nutanbhogendrasharma/pytorch-recurrent-neural-networks-with-mnist-dataset-2195033b540f

```
hidden_dim = 128
input_dim = 28
output_dim = 10
num_layers = 1
```

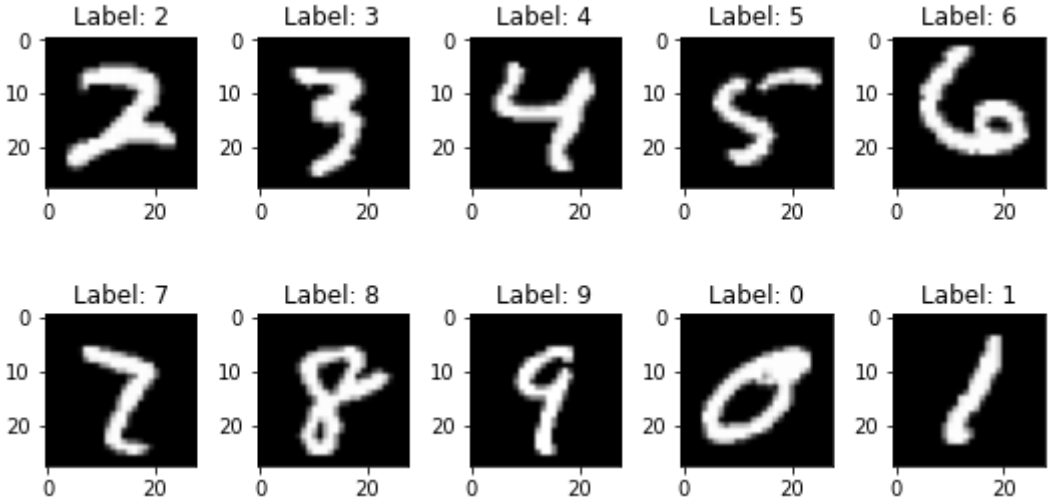
Parameters Used

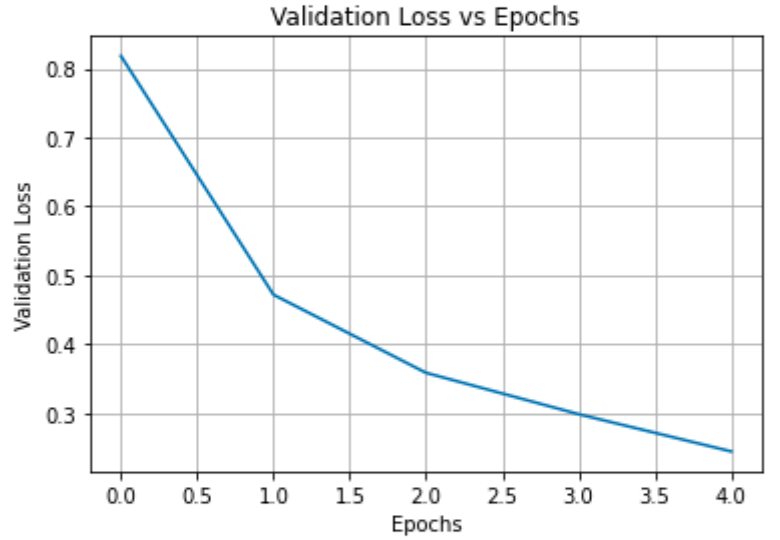
```
learning_rate = 0.002
num_epochs = 5
lamb=0.01
criterion1 = nn.CrossEntropyLoss()
optimizer1 =
torch.optim.Adam(Vanilla_RNN(input_dim,hidden_dim,num_layers,outp
lr=learning_rate)
```

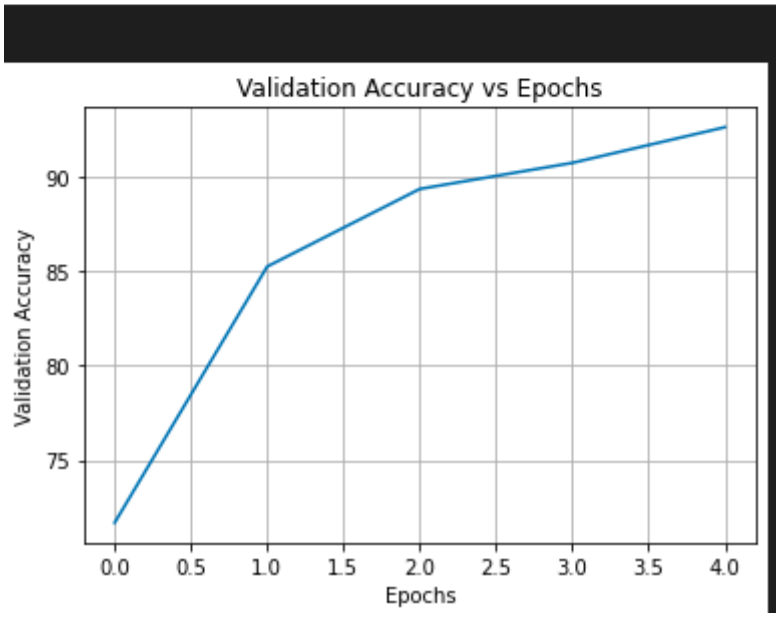
Vanilla RNN

Output

```
Epoch 1 out of 5 epochs are over
Epoch 2 out of 5 epochs are over
Epoch 3 out of 5 epochs are over
Epoch 4 out of 5 epochs are over
Epoch 5 out of 5 epochs are over
Predicted tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])
real-Label tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])
Test Accuracy of the unregularized model on the 10000 test
images: 93.08999999999999 %
```







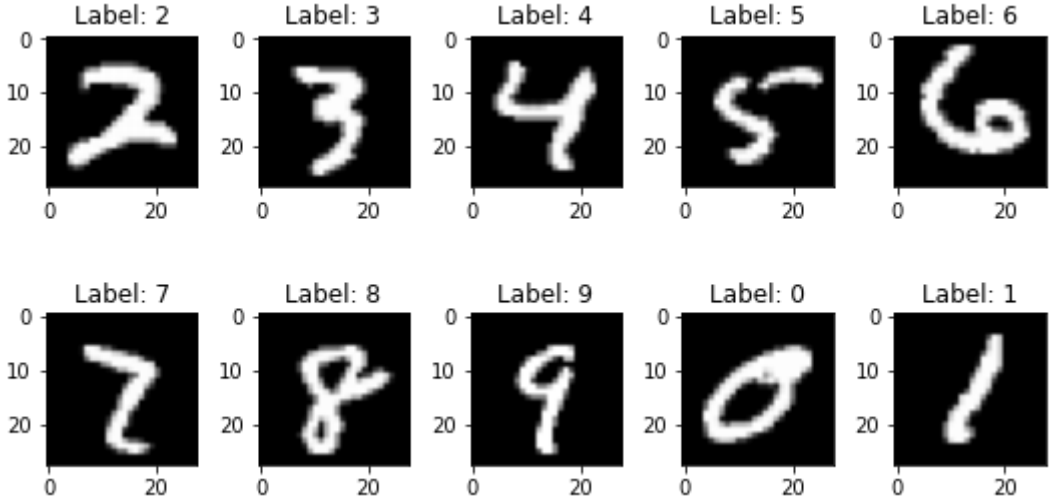
Graphs for Vanilla RNN

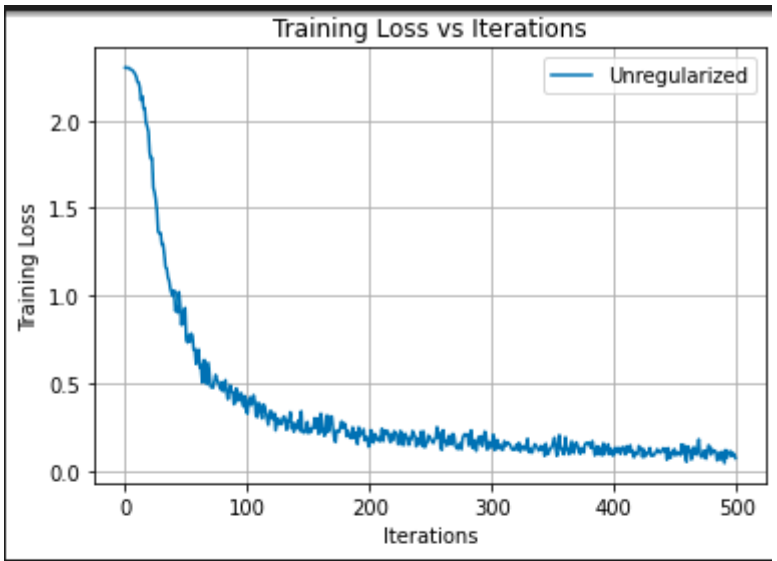
L2 regularization is added in the code , we can turn it on bu giving True as the last argument

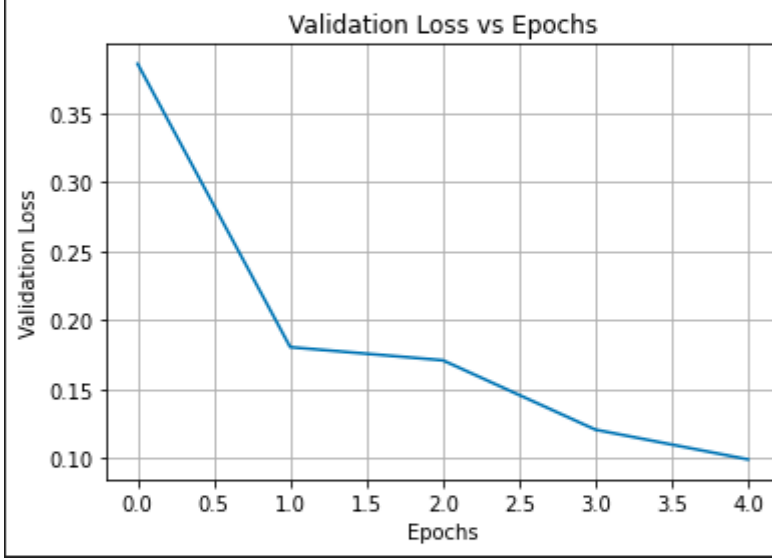
Vanilla LSTM

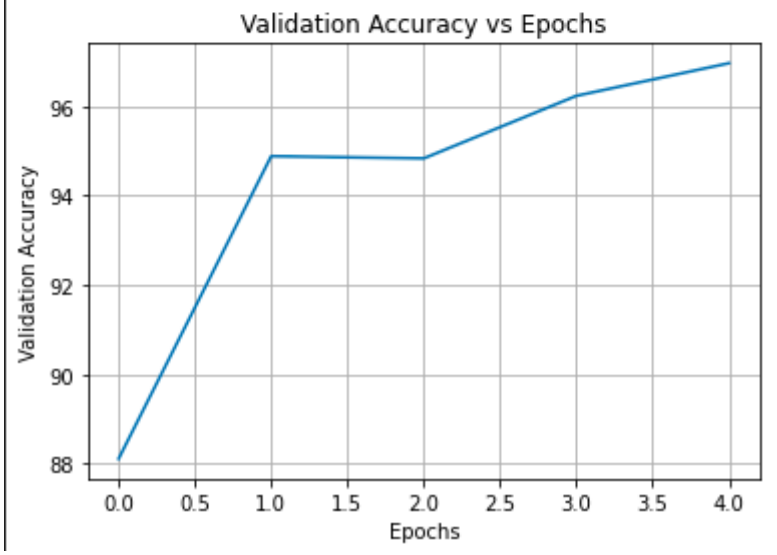
Output of running to 5 epochs

```
Epoch 1 out of 5 epochs are over
Epoch 2 out of 5 epochs are over
Epoch 3 out of 5 epochs are over
Epoch 4 out of 5 epochs are over
Epoch 5 out of 5 epochs are over
Predicted tensor([7, 4, 5, 6, 7, 8, 9, 0, 1])
real-Label tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])
Test Accuracy of the unregularized model on the 10000 test
images: 96.98 %
```




Training Loss vs Iterations
Unregularized
Training Loss
Iterations
0.0
0.5
1.0
1.5
2.0
0
100
200
300
400
500


Validation Loss vs Epochs
Validation Loss
Epochs
0.10
0.15
0.20
0.25
0.30
0.35
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

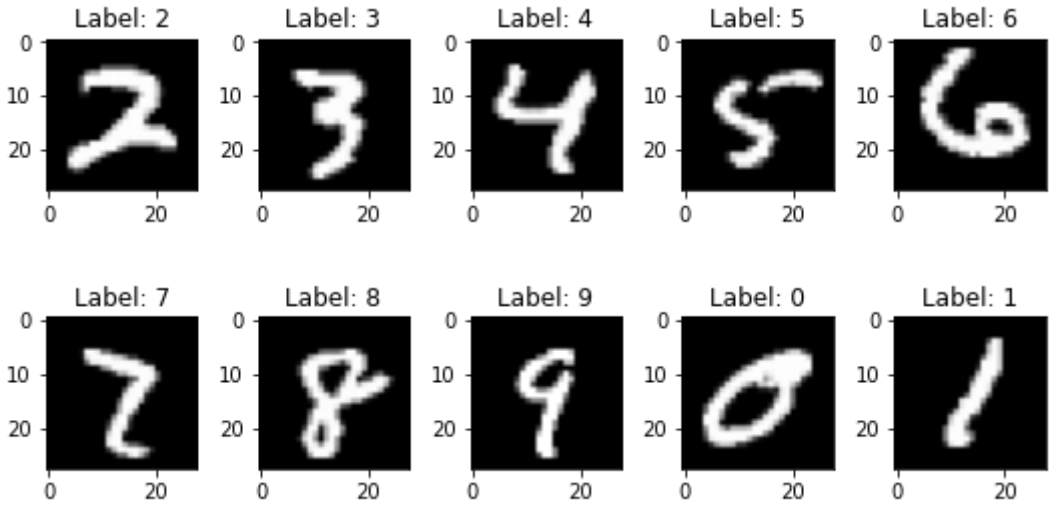

Validation Accuracy vs Epochs
Validation Accuracy
Epochs
88
90
92
94
96
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

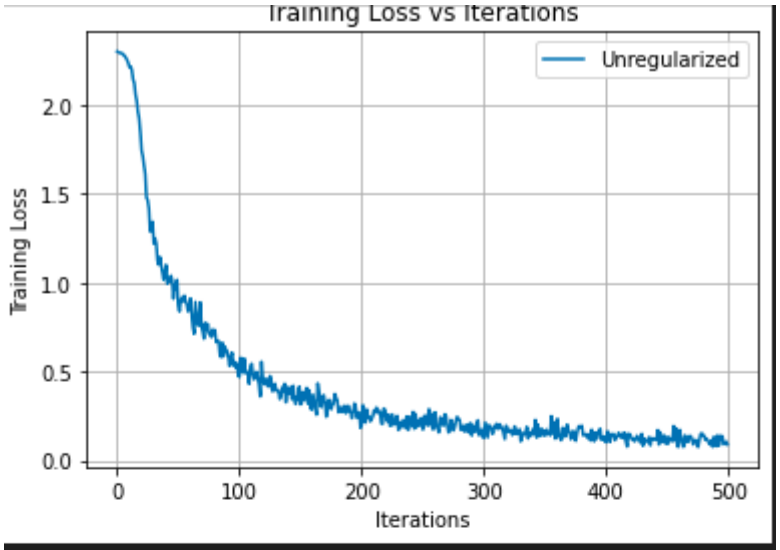
Vanilla GRU

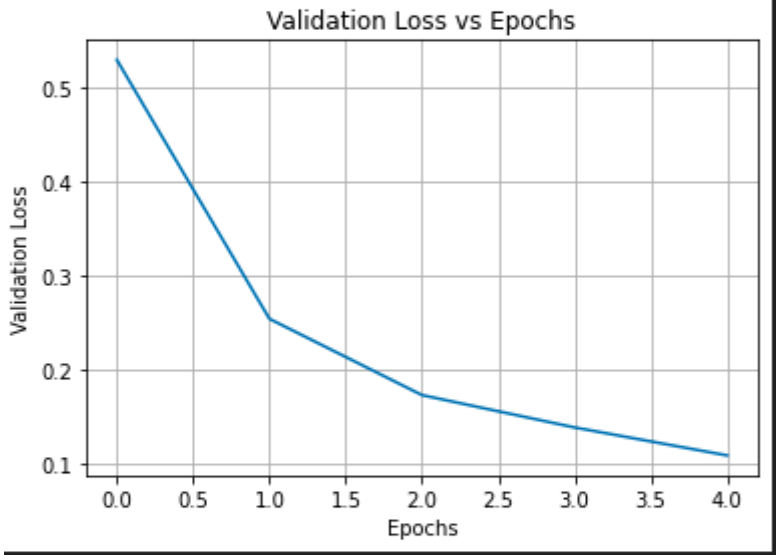
Output :

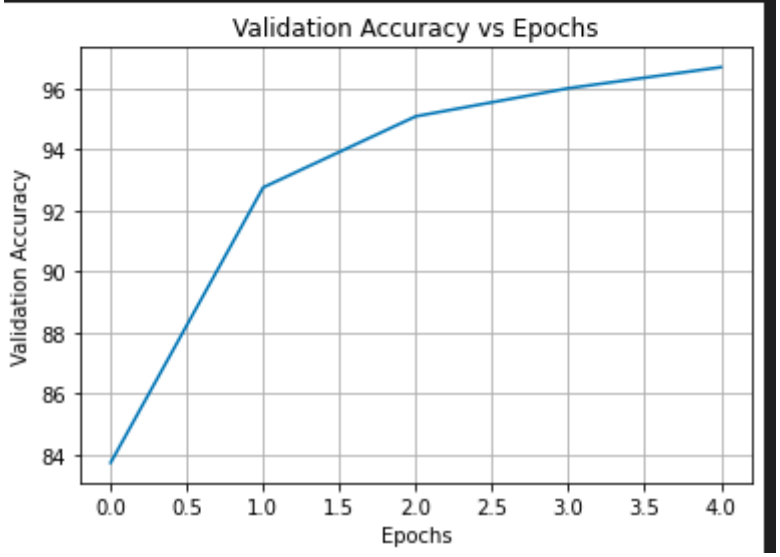
```
Epoch 1 out of 5 epochs are over
Epoch 2 out of 5 epochs are over
Epoch 3 out of 5 epochs are over
Epoch 4 out of 5 epochs are over
Epoch 5 out of 5 epochs are over
Predicted tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])
real-Label tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])

Test Accuracy of the unregularized model on the 10000 test
images: 96.8 %
```







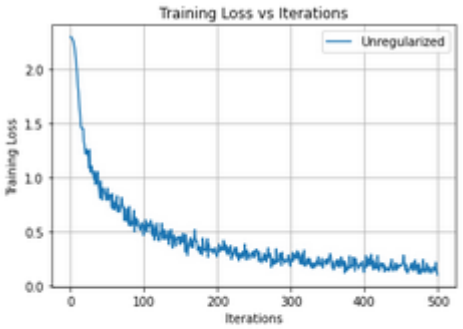


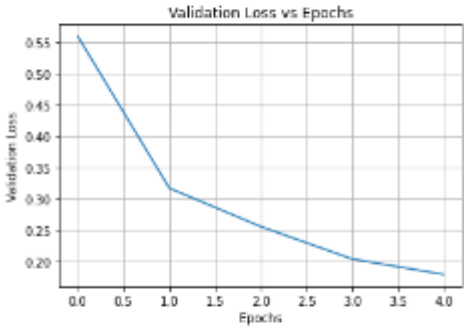
Bidirectional RNN

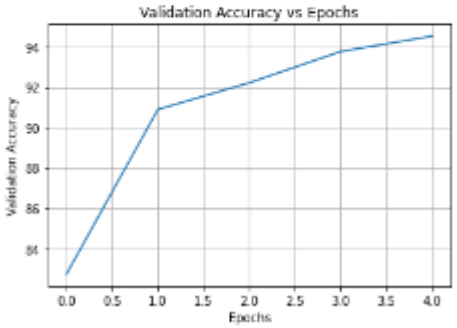
Output

```
Epoch 1 out of 5 epochs are over
Epoch 2 out of 5 epochs are over
Epoch 3 out of 5 epochs are over
Epoch 4 out of 5 epochs are over
Epoch 5 out of 5 epochs are over
Predicted tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])
real-Label tensor([3, 4, 5, 6, 7, 8, 9, 0, 1])

Test Accuracy of the unregularized model on the 10000 test
images: 94.75 %
```







Observations

- First, comparing the RNNs, LSTMs and GRU, we see that for this particular experiment, LSTMs and GRUs outperform the RNN while they train for just half the number of Epochs. Between GRUs and LSTMs there is not much different in the accuracies.
- It is interesting to see that the GRU took longer to train. However, this trend did not carry forward on repeating the experiments.
- We see that the one with a smaller hidden size takes slightly lesser to train.
- We see that the bidirectional LSTM didn't improve the accuracy by a lot while taking almost double the time to train.
- The regularized LSTM trains much much faster compared to the other models and has a comparable accuracy too, due to the low value of the regularization coefficient.This brings to light the tradeoff between model complexity and computational complexit

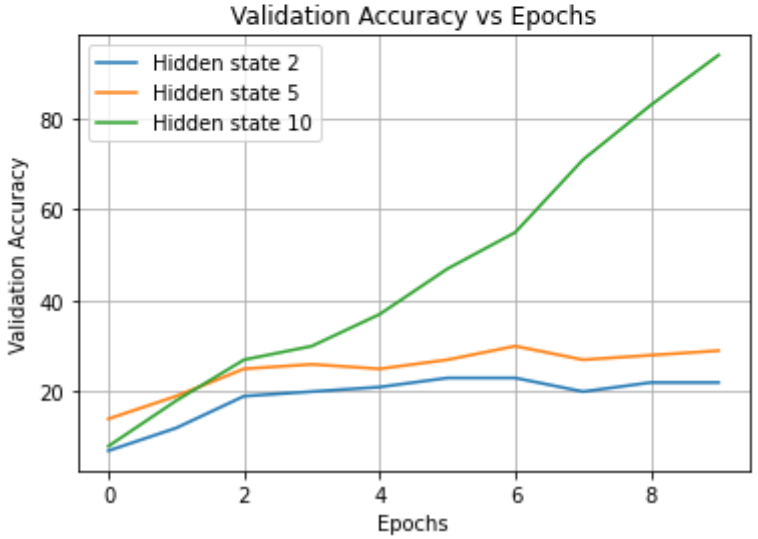
2 Remembering the number at a particular index in a given sequence

1. You will need to create a function that can create a random input-output pair . Write a function that takes a sequence length L as input and returns a training sample to be fed to the RNN. For a given length K, a training sample is a 2-tuple of input-x and output-y where input is a tensor of size L×10: Second dimension is 10 since each integer is represented as a one-hot vector of dimension=10 (since we consider only sequence of integers in [0-9]). Output is just a single number . Its the number at the Kth position which need to be remembered. So y is a tensor with just one element of size 10

```
def Random_creator(L):
  #takes input of length L
  return np.random.randint(0,10,L)

def one_hot_encoded(x):
  x_new = np.zeros((len(x),10))
  x_new[np.arange(len(x)), x] = 1
  return x_new
```

(a) Train 3 different RNN models: with hidden state size 2, 5, 10. For each of the experiments, you are required to show the plot of training error and prediction accuracy over the training progress. For each of the experiments, at the end of training, report the average prediction accuracy for the test set you created.



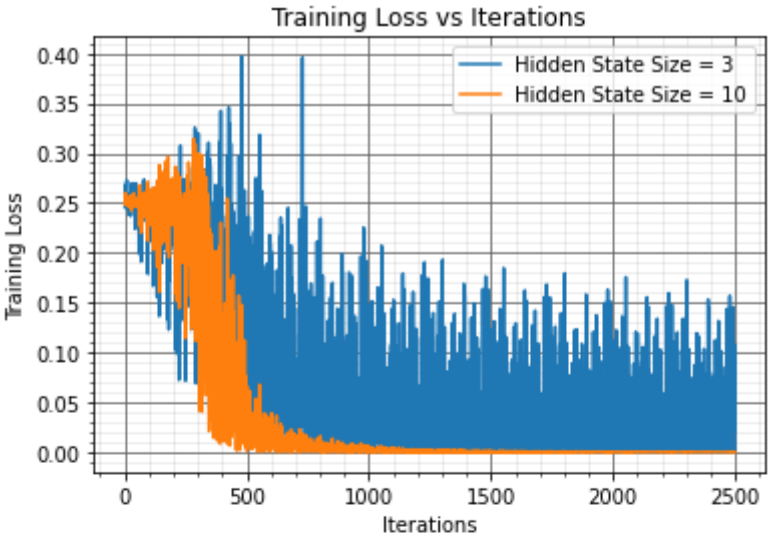

```
Test Accuracy of the model with 2 hidden states  16.0 %
Test Accuracy of the model with 5 hidden states 30.0 %
Test Accuracy of the model with 10 hidden states 96.0 %
```

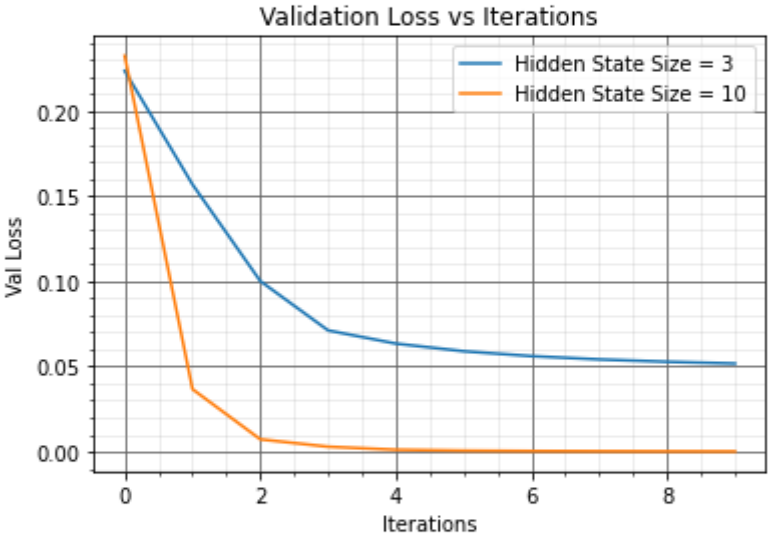
```
For Length 3
Array: [4 9 9] Prediction at position 2: 9
Array: [0 6 0] Prediction at position 2: 6
Array: [2 4 5] Prediction at position 2: 4
For Length 4
Array: [2 1 5 2] Prediction at position 2: 1
Array: [9 4 7 7] Prediction at position 2: 4
Array: [8 1 4 0] Prediction at position 2: 1
For Length 5
Array: [6 4 1 6 4] Prediction at position 2: 4
Array: [8 7 3 0 2] Prediction at position 2: 7
Array: [7 8 4 3 6] Prediction at position 2: 8
For Length 6
Array: [0 1 5 5 1 9] Prediction at position 2: 1
Array: [4 3 1 1 2 7] Prediction at position 2: 3
Array: [3 0 2 3 7 6] Prediction at position 2: 0
For Length 7
Array: [9 1 3 0 6 2 4] Prediction at position 2: 1
Array: [9 2 2 7 0 0 6] Prediction at position 2: 2
Array: [6 1 6 0 0 8 0] Prediction at position 2: 1
For Length 8
Array: [2 9 0 9 4 9 9 5] Prediction at position 2: 9
Array: [7 7 7 8 6 8 5 6] Prediction at position 2: 7
Array: [2 2 5 3 6 3 3 2] Prediction at position 2: 2
For Length 9
Array: [8 3 2 4 3 7 3 7 2] Prediction at position 2: 3
Array: [9 2 1 6 4 9 5 6 2] Prediction at position 2: 2
Array: [2 1 0 4 4 6 8 0 1] Prediction at position 2: 1
For Length 10
Array: [8 5 5 3 0 1 7 5 9 7] Prediction at position 2: 5
Array: [2 5 1 8 2 2 1 1 9 0] Prediction at position 2: 5
Array: [7 5 4 4 7 0 3 2 1 7] Prediction at position 2: 5
```

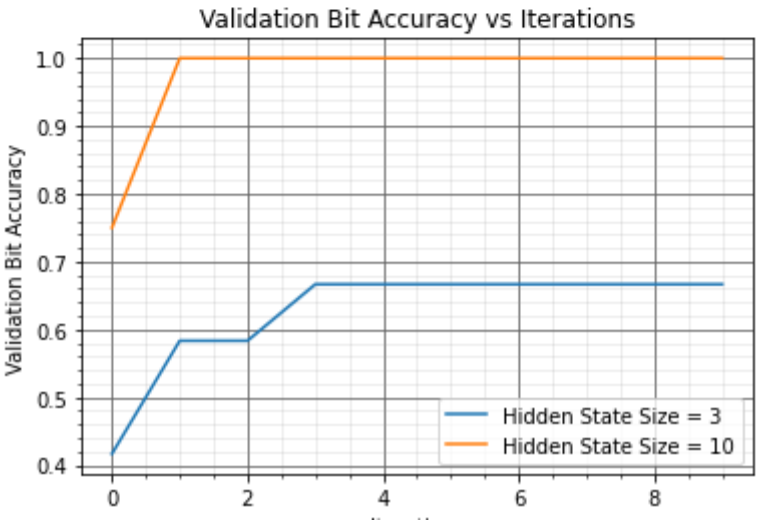
Observations

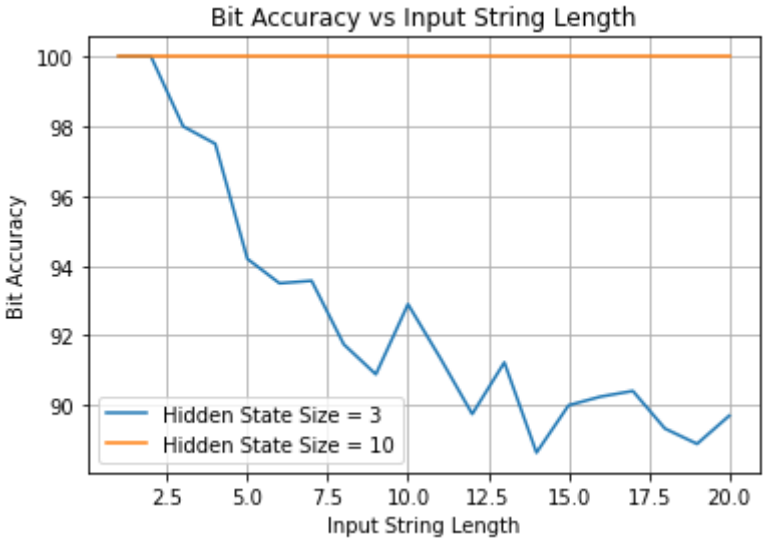
- We see that as the size of the hidden length increases, the capability to remember increases and therefore the accuracy also improves.
- Amongst RNNs, LSTMs and GRUs, the latter performs the best as the inputs are all small length sequences.
- Observing the predicted outputs, we can see that the trained models indeed remember the index very well.
- When trained for lesser number of epochs prediction wasn't right , it learns more with learning
- since the sequence is small its able to remember it and give the right output

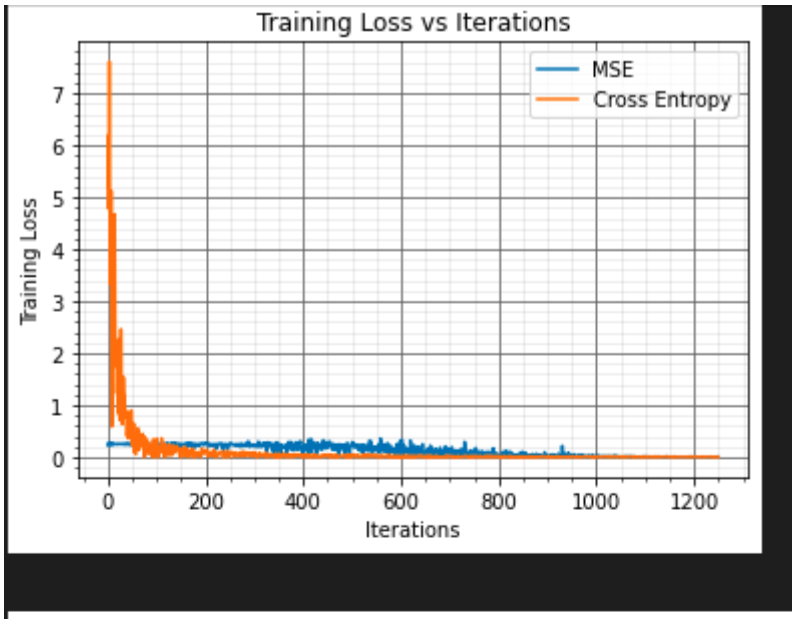
3 Adding two binary strings

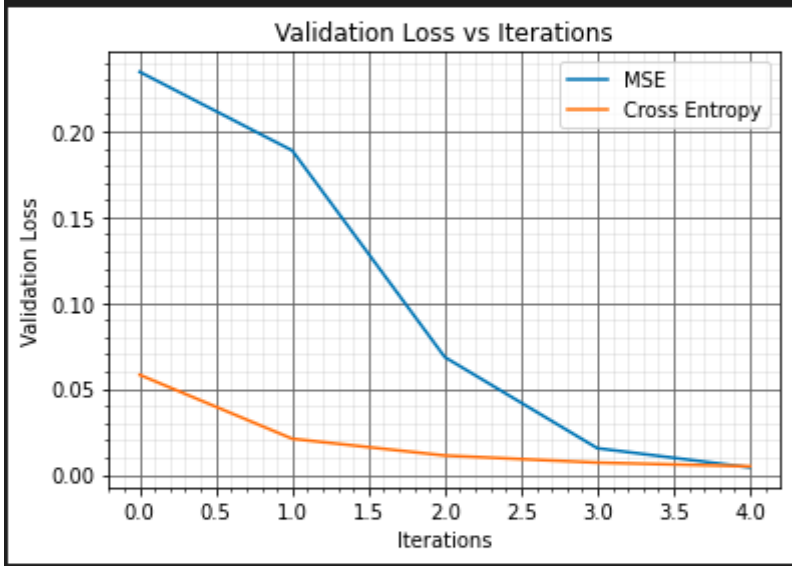

Training Loss vs Iterations
Hidden State Size = 3
Hidden State Size = 10
Training Loss
Iterations
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
500
1000
1500
2000
2500

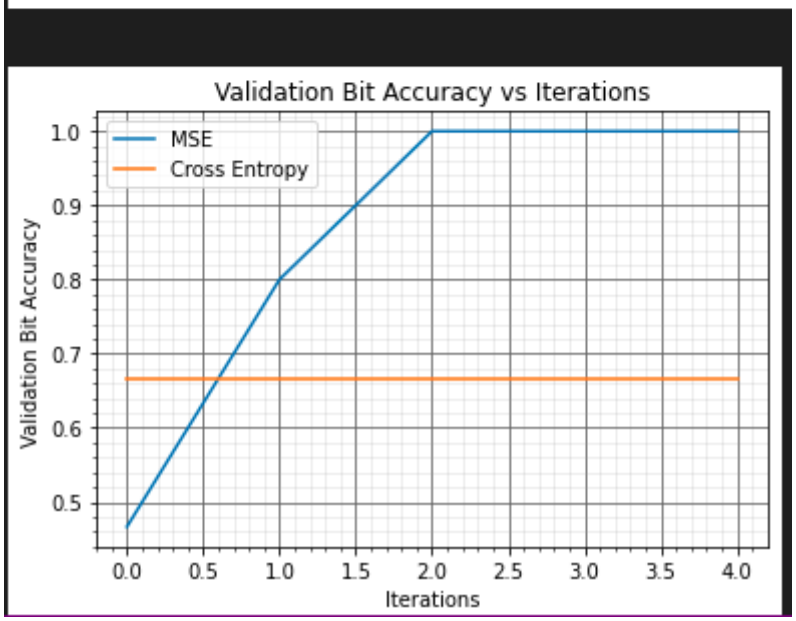

Validation Loss vs Iterations
Hidden State Size = 3
Hidden State Size = 10
Val Loss
Iterations
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8


Validation Bit Accuracy vs Iterations
Hidden State Size = 3
Hidden State Size = 10
Validation Bit Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8


Bit Accuracy vs Input String Length
Bit Accuracy
Input String Length
Hidden State Size = 3
Hidden State Size = 10
100
98
96
94
92
90
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0







Observations

- cross entropy loss performs marginally better than the MSE loss
- More hidden layers ,more to learn , so increase in accuracy