Recurrent Neural Networks

EE6132 - Lab 4

Anirudh R

December 17, 2021

Contents

1 Introduction

In this assignment, we look to analyse Recurrent Neural Networks(RNN). We will be using the MNIST dataset, a Number Sequence dataset(self-generated) and a Binary Addition dataset(self-generated) to train various RNN models. We will use the PyTorch library to build, train and test the models. We will also try out various kinds of RNNs such as the vanilla RNN, LSTM(Long Short-Term Memory) and GRU(Gated Recurrent Unit). We will also be making training plots and testing the models to compare their training convergence and performance.

The following table shows some basic information about the models we will consider.

Parameter	Value
Dataset	MNIST/Number Sequence/Binary Addition
Loss Function	Mean Squared Error/Cross Entropy Loss
Optimizer	Adam
Train Mini-Batch Size	50(Num Seq), 100(Other 2)
RNN Activation	Tanh
Output Layer Activation	Softmax/Sigmoid

2 Concepts

2.1 Datasets

2.1.1 MNIST Dataset

We make use of the **MNIST**(Modified National Institute of Standards and Technology) Dataset here. This is a dataset of hand-written digits and the corresponding labels. This contains **50,000** training data points, **10,000** validation data points and **10,000** test data points. The digits have been size-normalized and centered in a fixed-size image (28×28 grayscale image).

2.1.2 Number Sequence Dataset

This dataset was generated for this assignment. The dataset generation script can be found in the code [4]. This dataset contains a sequence of numbers in one-hot representation as input to the RNN. The corresponding output will be the number that was given in the input at a particular position (Use `position` parameter in the generation script. Note that `position` is 0-indexed. So the first element of the sequence corresponds to `position = 0`). `elements_in_batch` and `num_batches` can be used to control the number of sequences per batch and number of batches overall. Note that every sequence generated will be of length between 3 and 10 (both included) and every batch will have sequences of only 1 length.

Here, we have set the following values for the parameters:

- `position = 1`
- `elements_in_batch = 50`
- `num_batches = 1000`

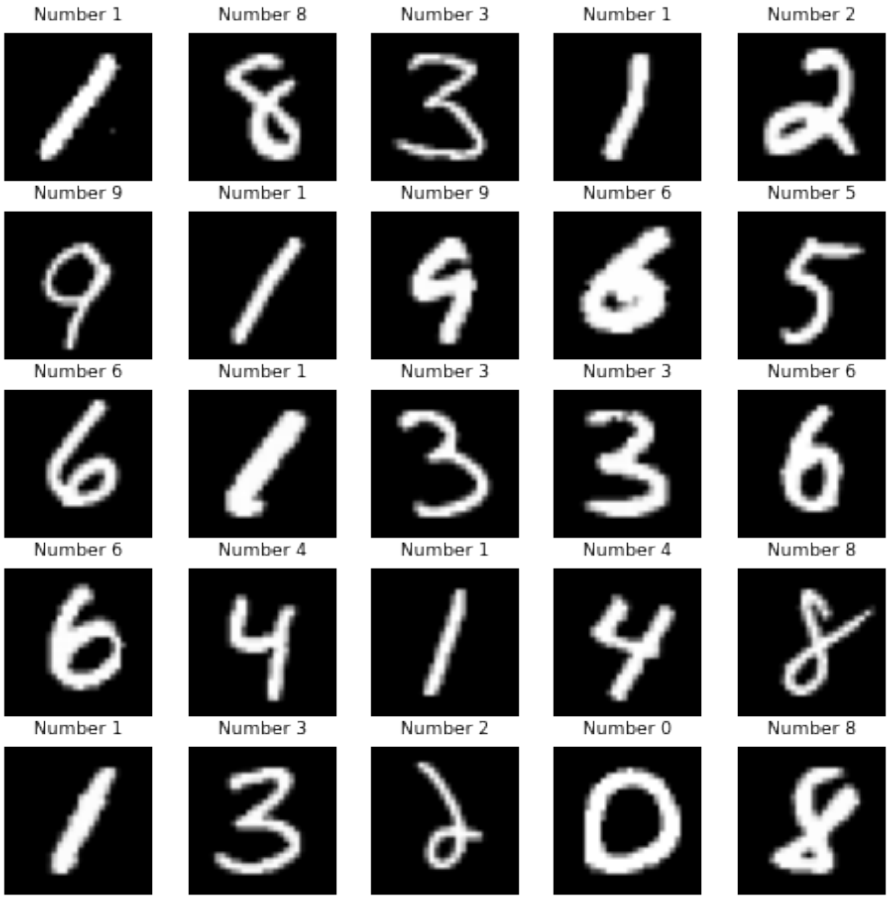

Figure 1: Samples from the MNIST Dataset

- Train: 80% of data; Test: 20% of data

Therefore, the models we train on this dataset will be trained only for this specific value of `position`. Some examples of data points:

- Input - 1,2,4,8,0,2 ; Output - 2
- Input - 1,6,0,1 ; Output - 6
- Input - 7,0,5,3,9,9,4,5 ; Output - 0

2.1.3 Binary Addition Dataset

Using this dataset, we wish to train the RNN to do binary addition. Similar to Number Sequence dataset, parameters can be set to decide number of elements in a batch and number of batches. In order to perform binary addition, we need 2 binary numbers. Here, for each data point, we generate 2 binary numbers as input. The 2 LSB bits from the 2 binary numbers will be the first element of the input sequence. The next most LSB of the 2 binary numbers will be the second element of the input sequence and so on. Here, the RNN models will give a binary digit as output at each time step which will be nothing but the predicted binary sum. Here, we have set the following values for the parameters:

- `length = 5`

- `elements_in_batch = 100`
- `num_batches = 500`
- Train: 80% of data; Test: 20% of data

The `length` parameter denotes the length of input binary numbers that we will be generating. So, here, our dataset only contains input binary numbers of length 5. We don't train our models on binary numbers of other lengths. Some examples of data points: (**Note:** All inputs are padded to be of length 5+1=6 to accommodate the extra bit that comes in the output as carry during binary addition)

- Input1: 001101, Input2: 010000 ; Output: 011101
- Input1: 001101, Input2: 011111 ; Output: 101100
- Input1: 001101, Input2: 000001 ; Output: 001110

The 2 inputs are fed and the outputs are obtained from right to left (LSB to MSB).

2.2 Recurrent Neural Networks

Recurrent Neural Networks are a class of Neural Networks that are a bit different compared to what we have seen so far. They are designed to handle sequences. They are the go-to models for sequence-to-sequence learning tasks. Some examples of sequence-to-sequence learning tasks are image captioning, machine translation, sentiment analysis of text sentences etc.

RNNs generally are used as cells that are unrolled through time to take input sequences of variable length. This allows for flexibility in what the length of the input sequence can be. This is very different from what we saw in something like MLP where the input had to have a specific size.

Some types of RNNs that we will use in this assignment:

- **Vanilla RNN**: This is the most basic RNN model. It contains a "hidden state" that acts as a summary of all past inputs. But, this suffers from vanishing gradient issues. So, if we want to "remember" something that was part of the input sequence many time steps ago, we shouldn't use this.
- **Long Short-Term Memory (LSTM)**: This is a much more complex RNN model with *gates*. Gates act as an explicit way for the network to decide what information to keep, attenuate and discard moving forward. This also has something called a "cell state" apart from the "hidden state". This cell state pretty much flows unimpeded through time. Therefore, this acts as a **Gradient Highway** and helps counter the vanishing gradient problem.
- **Gated Recurrent Units (GRU)**: GRUs are very similar to LSTMs. They are just a bit leaner (fewer parameters to learn). They don't have a separate "cell state" like LSTMs. Both states in LSTMs are sort of combined here into a single "hidden state". This hidden state acts as the gradient highway that helps counter the vanishing gradient problem. This is leaner and faster compared to LSTM.

LSTMs are better when we need long range dependency and when we have large datasets. GRUs are preferred when we want speed and fewer parameters to learn.

Now, let's see how the tasks we have taken up here map to RNNs...

- **MNIST Classification**: Here, the rows of input images are given as a sequence input to the RNN models and the model is expected to predict the class of the input image after we have given all rows.
- **Remembering the number at a particular index in a given sequence**: Here, the input sequence is a sequence of numbers between 0-9 in one-hot representation. The length of the sequences are allowed to be variable. The RNN predicts the number that came in the input sequence at a particular position after the whole input sequence has been passed just like in MNIST.
- **Adding two Binary Numbers**: Here, the input binary numbers are fixed to have a length of 5. At each time step, 1 bit from each binary number is given as input to the RNN. Unlike the other 2 cases above, here, the RNN gives a binary digit as output at each time step.

3 Experiments & Observations

3.1 MNIST digit classification using RNN

We look to classify the MNIST dataset using RNNs here. We consider 3 varieties of RNNs, namely, Vanilla RNN, GRU and bi-directional LSTM. We run the training for all the models for 5 epochs. The learning rate used by Adam is 1e-3. Regularization parameter was set to be 1e-3 for the regularized models we will see. To tune the hyper parameters, a grid search over layers(1,2) and hidden state size(64,128,256) was done (Values of the hyper parameters were chosen based on the references in 5). The convergence plots are shown for the best model obtained. The regularized models are also only built for the best case hyper parameters.

3.1.1 Vanilla RNN

From the hyper parameter search, **2** layers and hidden layer size of **128** was found to be the best in terms of prediction accuracy on the test set. Training plots 2, 3 and 4 were obtained.

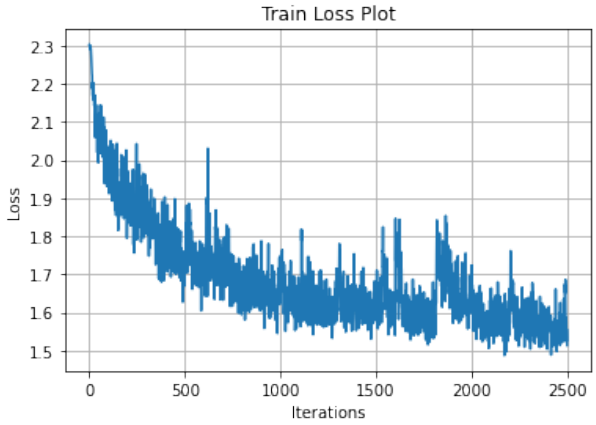

Figure 2: Unregularized Model - Vanilla RNN

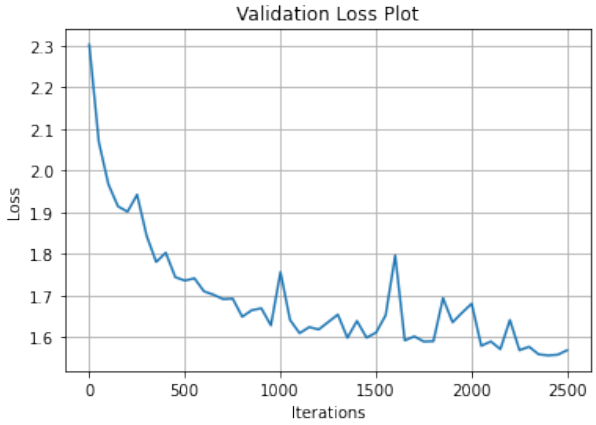

Figure 3: Unregularized Model - Vanilla RNN

The prediction accuracy on the test set is **89.9%**. Figure 5 shows some test examples with the true labels and the labels predicted by this network.

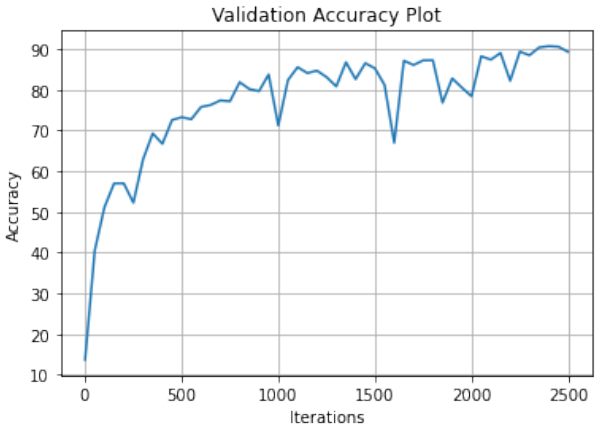

Figure 4: Unregularized Model - Vanilla RNN

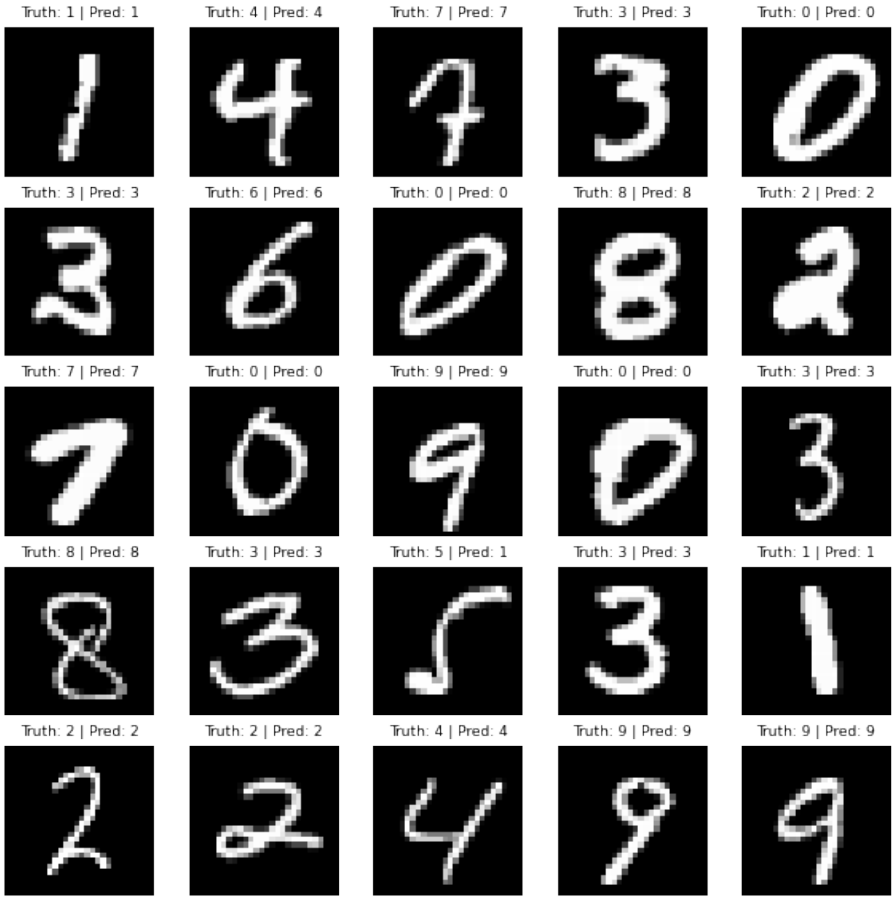

Figure 5: Test Samples for Unregularized Model - Vanilla RNN

The Regularized Network Training plots 6, 7 and 8 were obtained for the regularized network. The prediction accuracy on the test set for the regularized network is **89.0%**.

Comparison Now, we compare the convergence of the unregularized and regularized network. Figures 9, 10 and 11 show the convergence curves of both models together for comparison. [NOTE: The labels for the comparison plots are interchanged by mistake.]

- The unregularized network has better test prediction accuracy at the end. Consid-

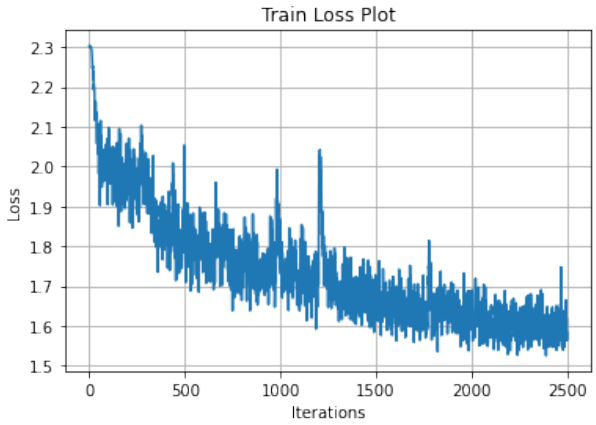

Figure 6: Regularized Model - Vanilla RNN

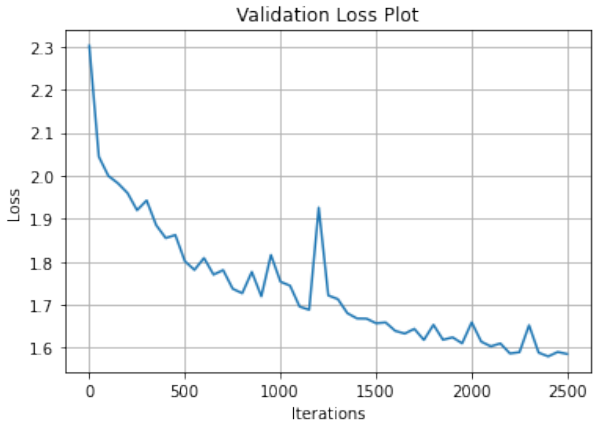

Figure 7: Regularized Model - Vanilla RNN

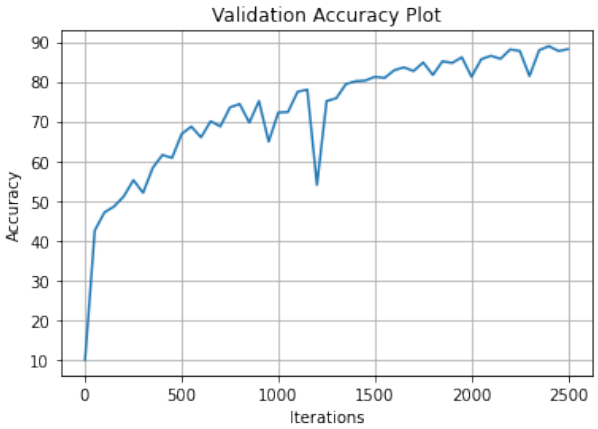

Figure 8: Regularized Model - Vanilla RNN

ering that we have a huge dataset to train the network, the chances of over-fitting are minimal. This means that adding a regularization term to the loss only affects the performance. So, it is understandable that the unregularized model is performing better.

- We also see that the the prediction accuracy increases more quickly and the losses

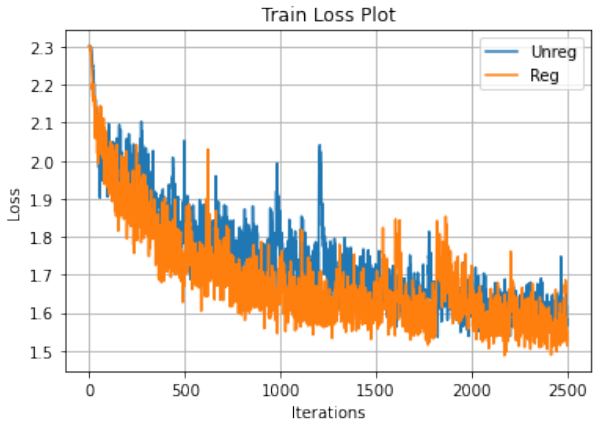

Figure 9: Comparison Plot - Vanilla RNN

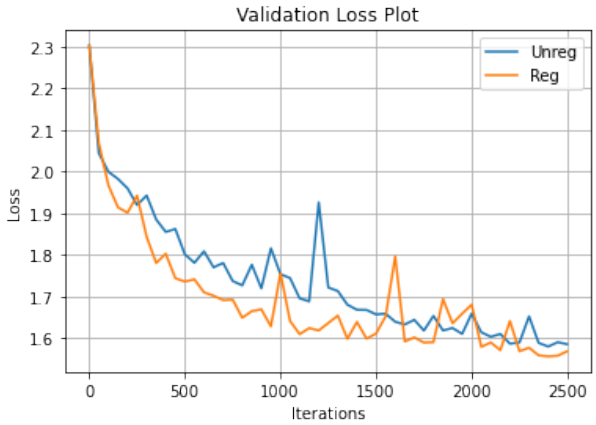

Figure 10: Comparison Plot - Vanilla RNN

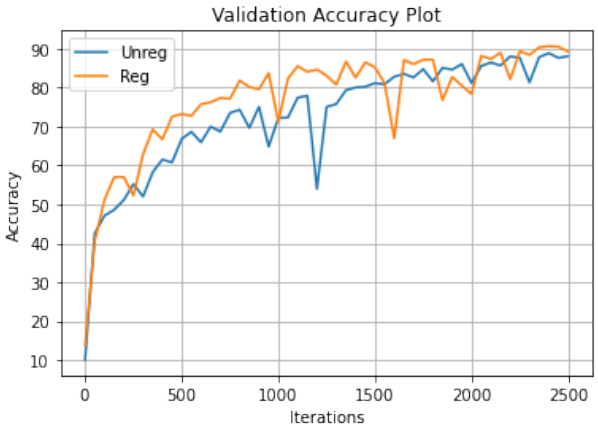

Figure 11: Comparison Plot - Vanilla RNN

drop more rapidly for the unregularized case during the initial iterations. But, eventually the regularized model sort of catches up to the unregularized model in terms of loss and accuracy.

3.1.2 GRU

From the hyper parameter search, **2** layers and hidden layer size of **256** was found to be the best in terms of prediction accuracy on the test set. Training plots 12, 13, 14 were obtained for the GRU.

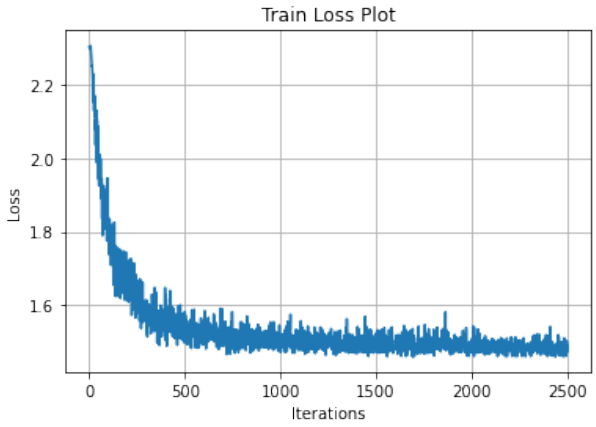

Figure 12: Unregularized Model - GRU

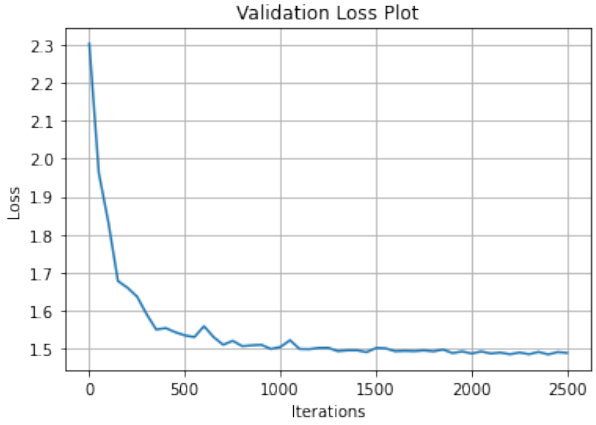

Figure 13: Unregularized Model - GRU

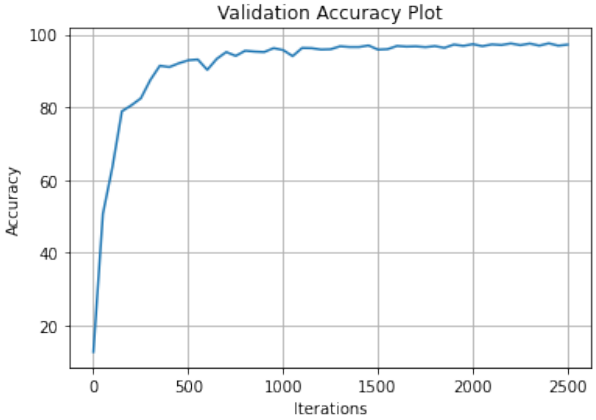

Figure 14: Unregularized Model - GRU

The prediction accuracy on the test set is **97.6%**. Figure 15 shows some test examples with the true labels and the labels predicted by this network.

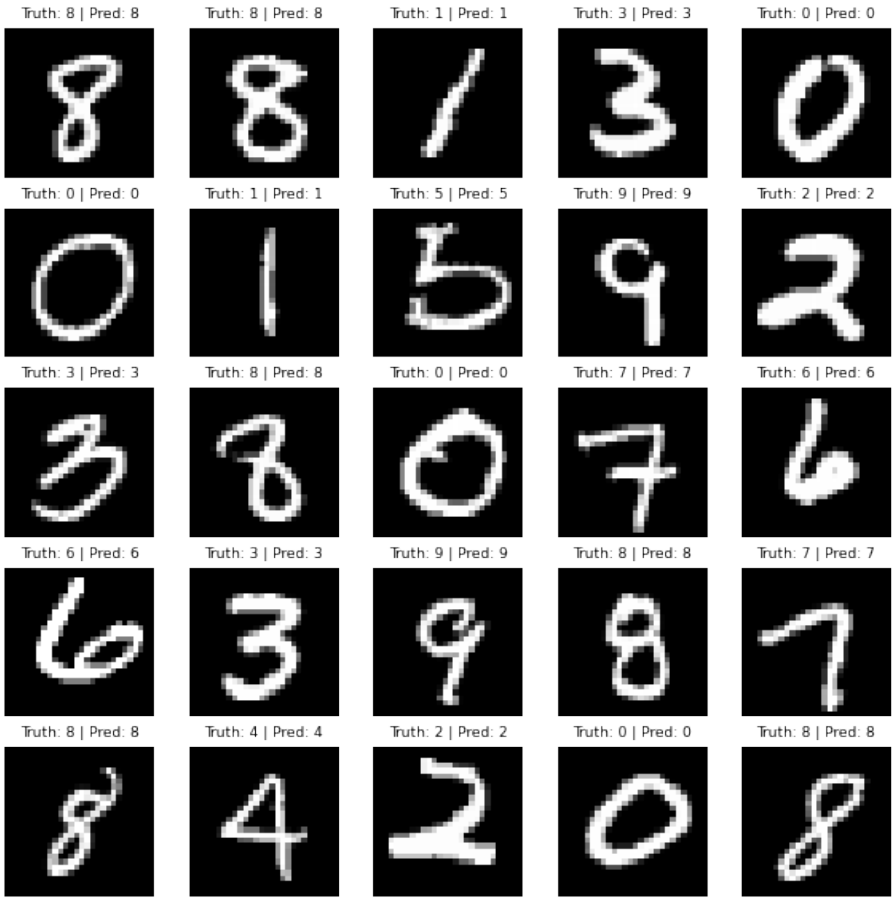

Figure 15: Test Samples for Unregularized Model - GRU

The Regularized Network Training plots 16, 17 and 18 were obtained for the regularized network. The prediction accuracy on the test set for the regularized network is **94.5%**.

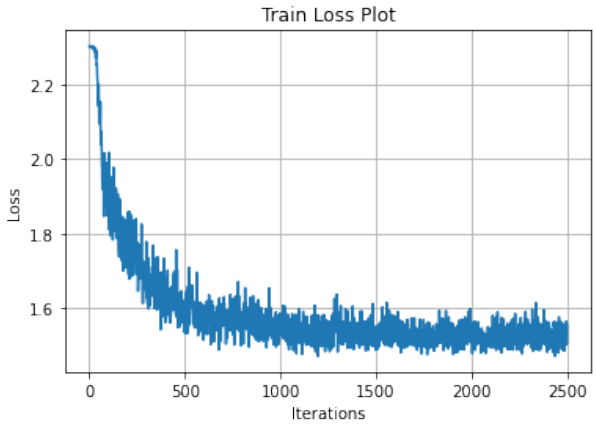

Figure 16: Regularized Model - GRU

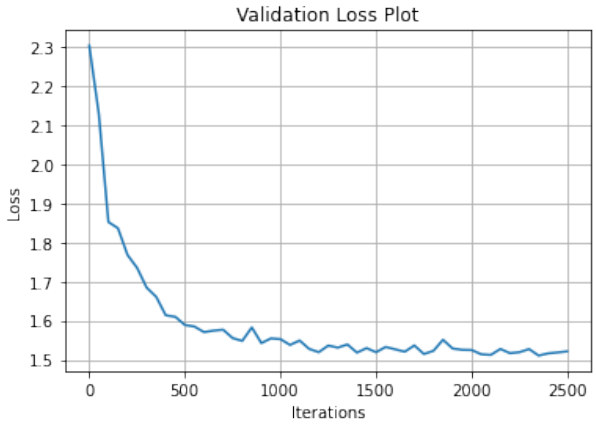

Figure 17: Regularized Model - GRU

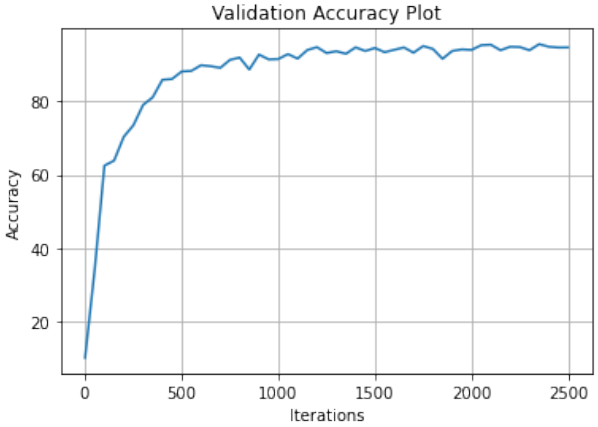

Figure 18: Regularized Model - GRU

Comparison Now, we compare the convergence of the unregularized and regularized networks. Figures 19, 20 and 21 show the convergence curves of both models together. [NOTE: The labels for the comparison plots are interchanged by mistake.]

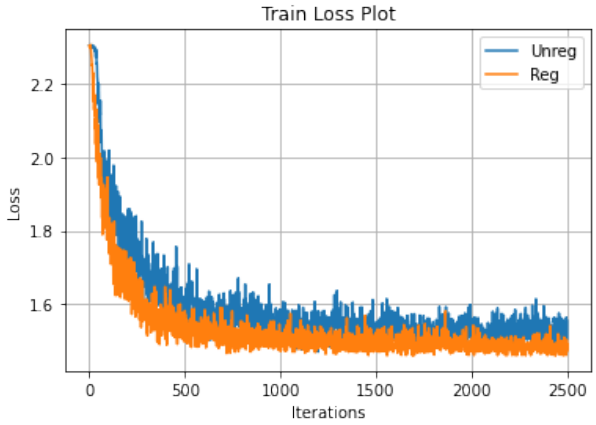

Figure 19: Comparison Plot - GRU

- We observe very clearly that the loss drops much more rapidly compared to the Vanilla RNN. The final test prediction accuracy achieved is also much better for

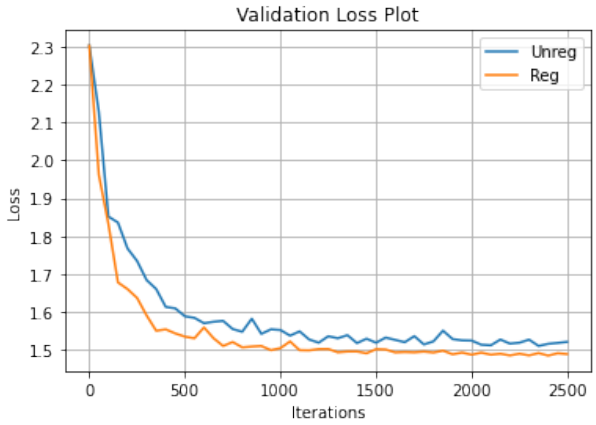

Figure 20: Comparison Plot - GRU

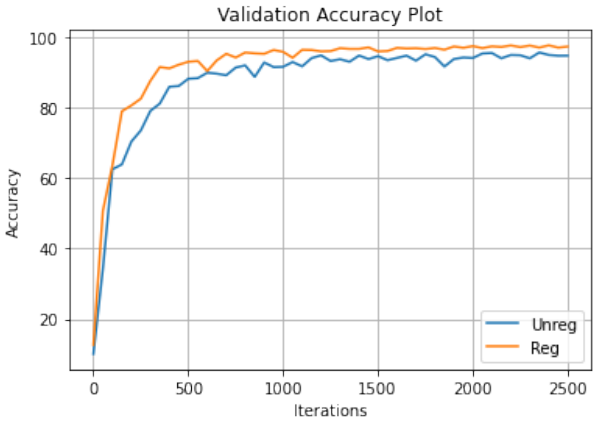

Figure 21: Comparison Plot - GRU

GRU. Considering that GRUs are a more complex model, it is understandable that they are able to perform better.

- The unregularized network has better test prediction accuracy at the end. Considering that we have a huge dataset to train the network, the chances of over-fitting are minimal. This means that adding a regularization term to the loss only affects the performance. So, it is understandable that the unregularized model is performing better.

3.1.3 Bi-Directional LSTM

From the hyper parameter search, **2** layers and hidden layer size of **128** was found to be the best in terms of prediction accuracy on the test set. Training plots 22, 23 and 24 were obtained. The prediction accuracy on the test set is **96.4%**. Figure 25 shows some test examples with the true labels and the labels predicted by this network.

The Regularized Network Training plots 26, 27 and 28 were obtained for the regularized network. The prediction accuracy on the test set for the regularized network is **93.8%**.

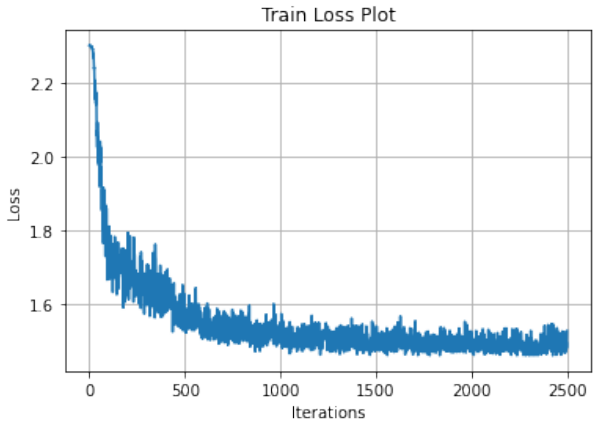

Figure 22: Unregularized Model - LSTM

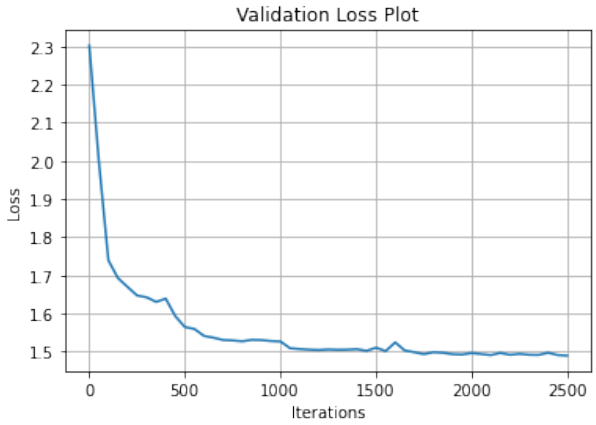

Figure 23: Unregularized Model - LSTM

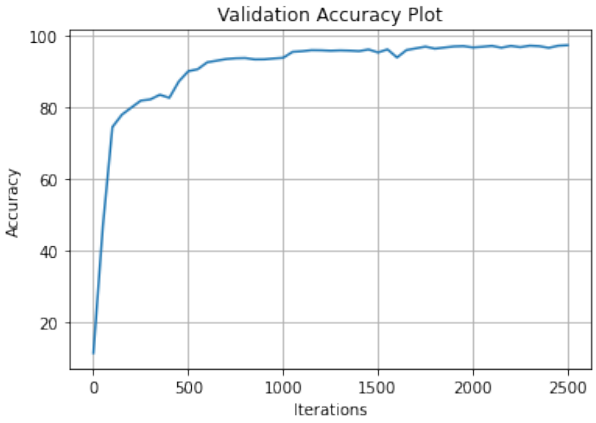

Figure 24: Unregularized Model - LSTM

Comparison Now, we compare the convergence of the unregularized and regularized network. Plots 29, 30 and 31 show the convergence curves of both models together.

- The unregularized network has better test prediction accuracy at the end. But, considering that the plots show the accuracies settling on pretty much the same value, even this difference in accuracy may just be 1 odd observation. If we compared the prediction accuracies at some other iteration, the difference may not be

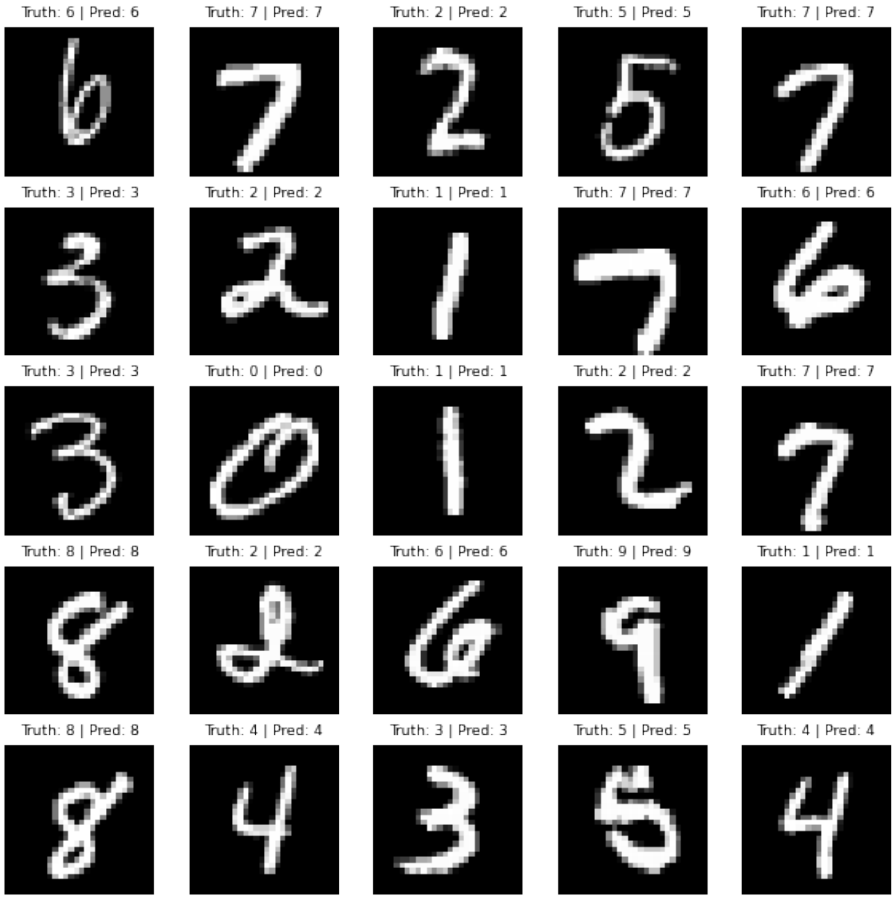

Figure 25: Test Samples for Unregularized Model - LSTM

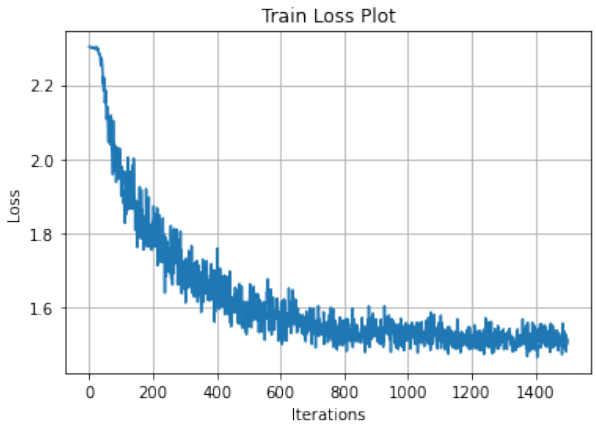

Figure 26: Regularized Model - LSTM

as pronounced.

- It is also interesting to note that the loss drops much more rapidly for the unregularized models in the initial iterations but at the end, with convergence, the difference between the loss of the 2 models is not as pronounced. Similarly, prediction accuracy also shoots up much more for the unregularized model initially, but, on convergence, both models perform pretty much the same.

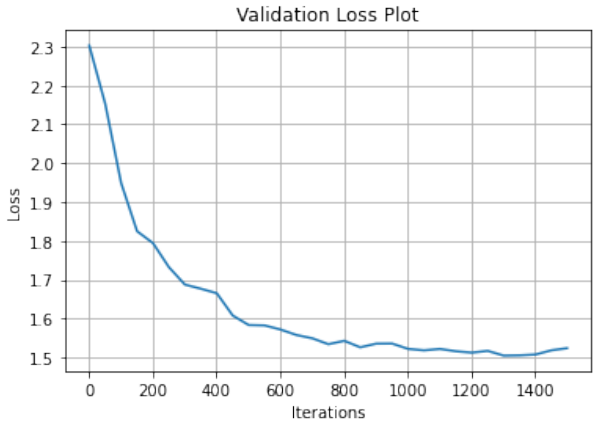

Figure 27: Regularized Model - LSTM

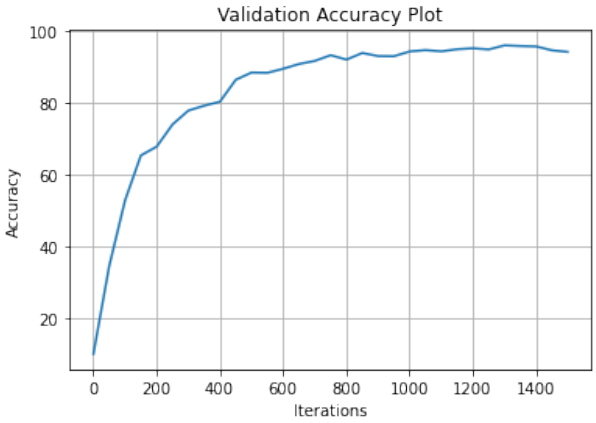

Figure 28: Regularized Model - LSTM

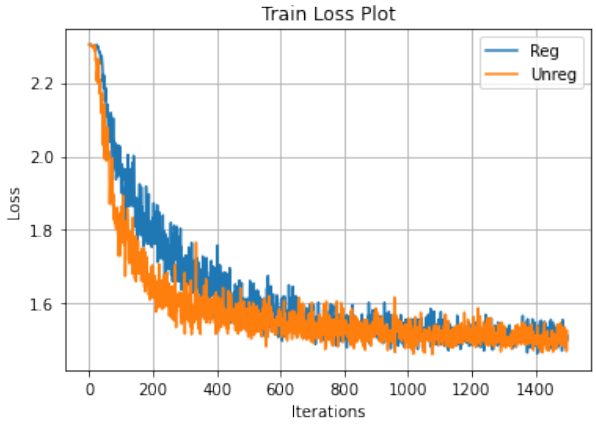

Figure 29: Comparison Plot - LSTM

Observations Overall, we see that the unregularized networks have performed better than their regularized counterparts. This is perhaps due to the abundant data, lack of overfitting & subsequent lack of need for regularization. We see that the unregularized GRU model has the best performance out of the 3 RNNs that we considered.

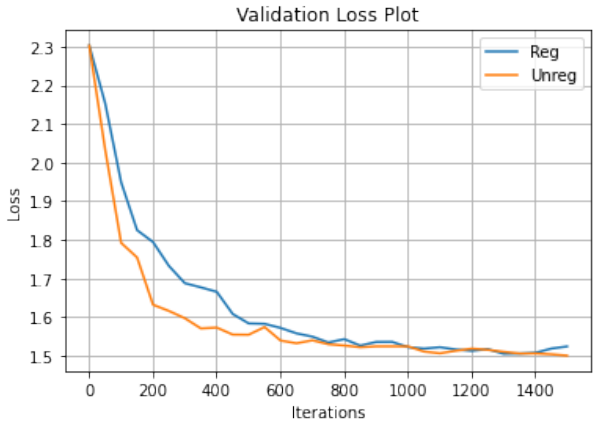

Figure 30: Comparison Plot - LSTM

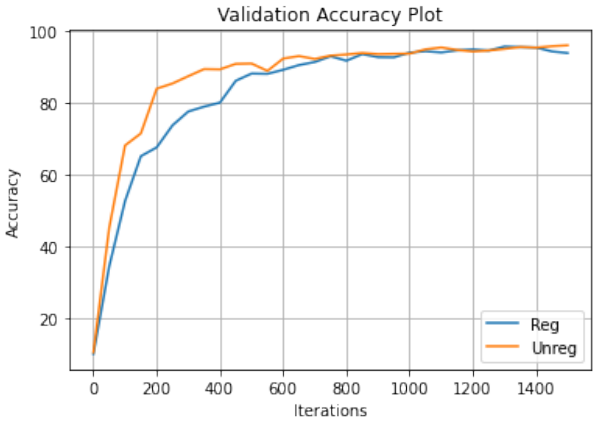

Figure 31: Comparison Plot - LSTM

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

We use bi-directional LSTMs as the model and vary the hidden state sizes. We run the training for all the models for 3 epochs. The learning rate used by Adam is 5e-3. Check 2.1.2 for more details about the dataset.

3.2.1 Hidden State Size 2

The prediction accuracy achieved for the test set is **47.0%**.

3.2.2 Hidden State Size 5

The prediction accuracy achieved for the test set is **95.8%**.

3.2.3 Hidden State Size 10

The prediction accuracy achieved for the test set is **100.0%**.

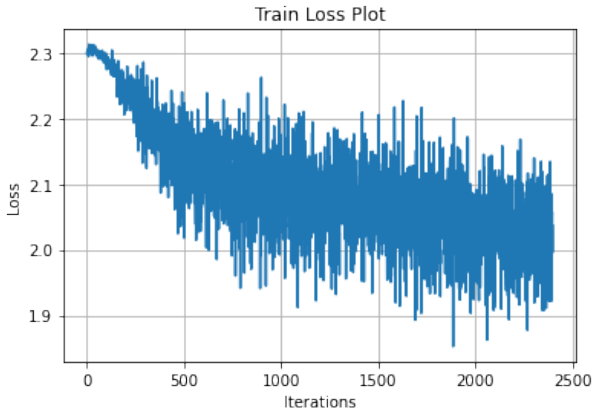

Figure 32: Training Plot: Hidden State Size 2

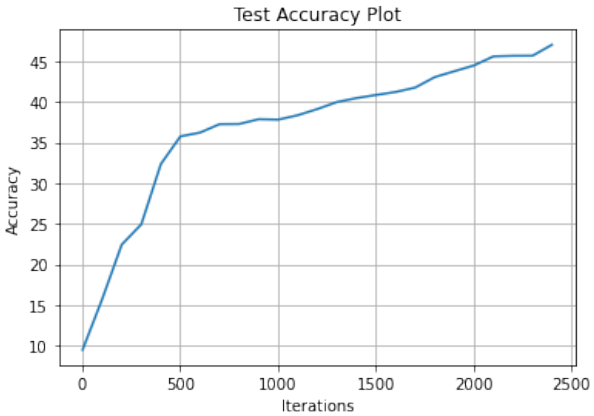

Figure 33: Training Plot: Hidden State Size 2

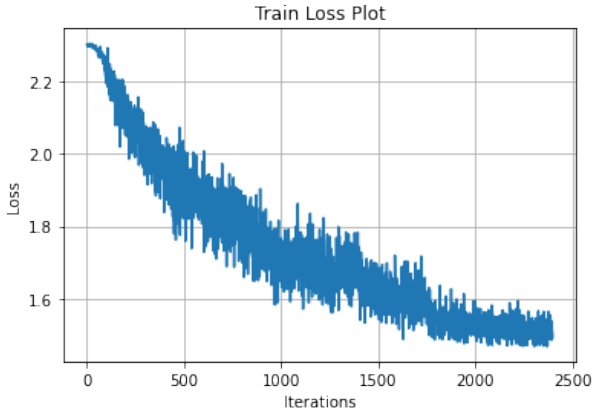

Figure 34: Training Plot: Hidden State Size 5

3.2.4 Testing the Best Model

It is clear from the prediction accuracy obtained that the **hidden state size 10** performs the best. So, we will see some test examples using this model.

- Length 3:
  - 3, 7, 5 **Truth**: 7, **Pred**: 7

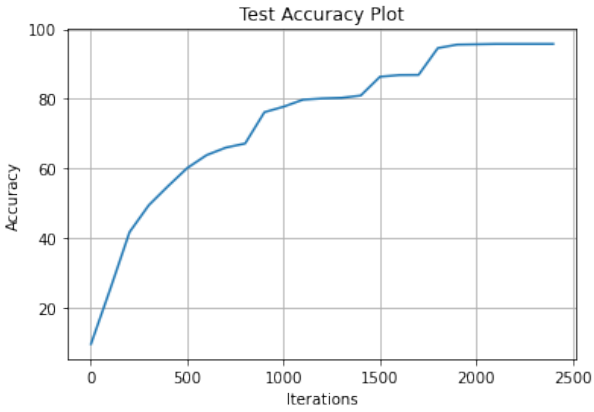

Figure 35: Training Plot: Hidden State Size 5

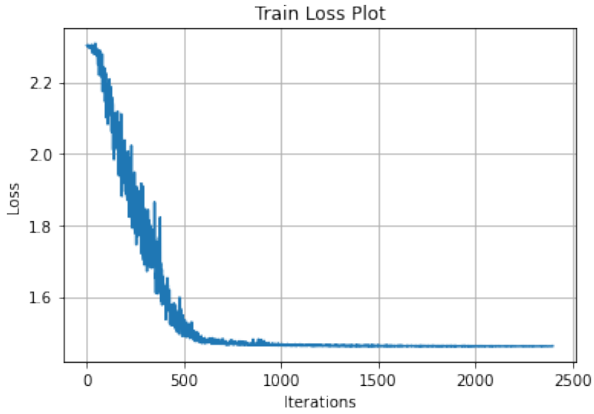

Figure 36: Training Plot: Hidden State Size 10

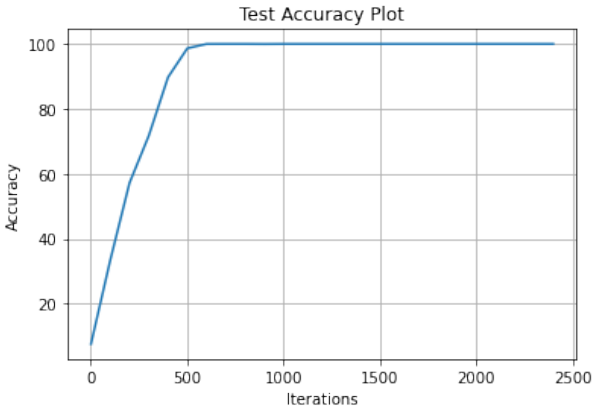

Figure 37: Training Plot: Hidden State Size 10

  - 2, 4, 6 **Truth**: 4, **Pred**: 4

- Length 4:
  - 8, 8, 7, 9 **Truth**: 8, **Pred**: 8
  - 8, 8, 0, 7 **Truth**: 8, **Pred**: 8

- Length 5:

- 7, 4, 7, 6, 8 **Truth**: 4, **Pred**: 4
- 6, 9, 2, 2, 8 **Truth**: 9, **Pred**: 9

- Length 6:
  - 0, 7, 2, 1, 8, 2 **Truth**: 7, **Pred**: 7
  - 8, 3, 0, 3, 8, 2 **Truth**: 3, **Pred**: 3

- Length 7:
  - 5, 1, 8, 8, 9, 7, 3 **Truth**: 1, **Pred**: 1
  - 6, 1, 1, 2, 0, 3, 6 **Truth**: 1, **Pred**: 1

- Length 8:
  - 4, 1, 5, 8, 2, 9, 5, 3 **Truth**: 1, **Pred**: 1
  - 5, 2, 5, 0, 4, 3, 8, 5 **Truth**: 2, **Pred**: 2

- Length 9:
  - 2, 5, 8, 2, 1, 2, 9, 6, 9 **Truth**: 5, **Pred**: 5
  - 0, 0, 8, 8, 8, 4, 9, 7, 1 **Truth**: 0, **Pred**: 0

- Length 10:
  - 2, 4, 2, 1, 4, 3, 1, 7, 9, 1 **Truth**: 4, **Pred**: 4
  - 1, 5, 6, 5, 2, 1, 1, 9, 7, 8 **Truth**: 5, **Pred**: 5

As expected from the test data performance, the model performs to perfection on these new sequences as well.
The low number of parameters and the sufficiently big dataset has ensured that the models don't overfit. The lack of overfitting is reflected by the matching performance on the test and training set. To look at the training and test loss values, refer to section 4.

3.3 Adding two binary strings

The batch size is taken to be 100. We use Adam for optimization with a learning rate of 5e-3. We train the models for 5 epochs. Number of layers is taken to be 1 as this is a fairly simple task. We train 3 models for hidden state sizes of 2, 5 and 10. We test out both Mean Squared Error loss (MSELoss) and Binary Cross Entropy loss (BCELoss). We use binary numbers of length 5 in our dataset. Check 2.1.3 for more details about the dataset.

We also need to note that unlike the previous 2 tasks where the RNN model only gave output in the last time step after the entire input sequence had been given, here, the RNN model gives a single bit output at every time step.

3.3.1 Hidden State Size 2

BCELoss The test set bit accuracy achieved with **BCELoss** is **80.9%**.

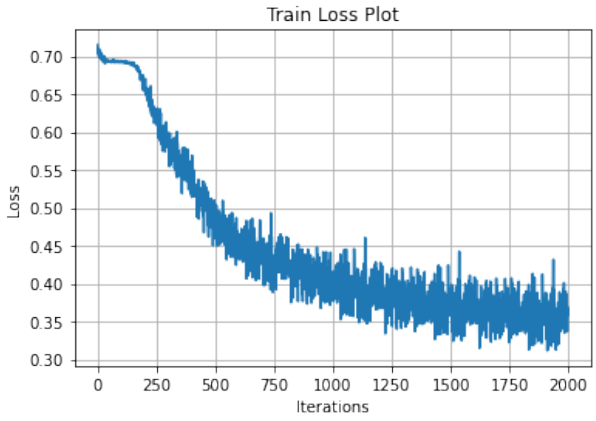

Figure 38: Hidden State Size 2 - BCELoss

Figure 39: Hidden State Size 2 - BCELoss

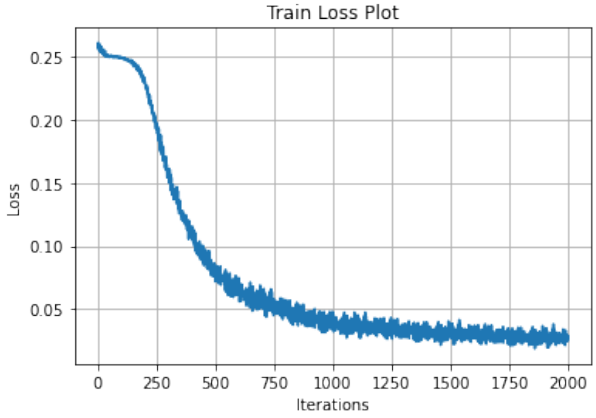

Figure 40: Hidden State Size 2 - MSELoss

MSELoss The test set bit accuracy achieved with **MSELoss** is **100.0%**.

3.3.2 Hidden State Size 5

BCELoss The test set bit accuracy achieved with **BCELoss** is **98.9%**.

MSELoss The test set bit accuracy achieved with **MSELoss** is **100.0%**.

Figure 41: Hidden State Size 2 - MSELoss

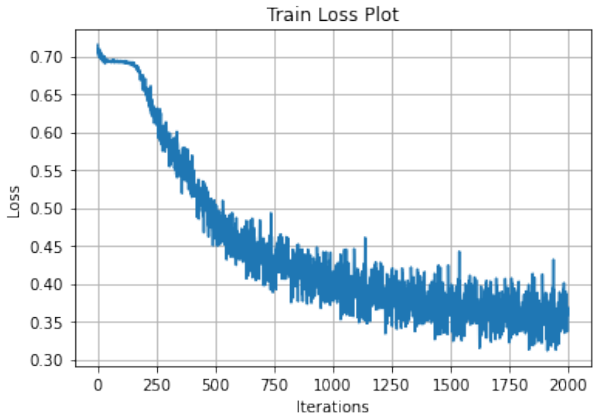

Figure 42: Hidden State Size 5 - BCELoss

Figure 43: Hidden State Size 5 - BCELoss

3.3.3 Hidden State Size 10

BCELoss The test set bit accuracy achieved with **BCELoss** is **100.0%**.

MSELoss The test set bit accuracy achieved with **MSELoss** is **100.0%**.

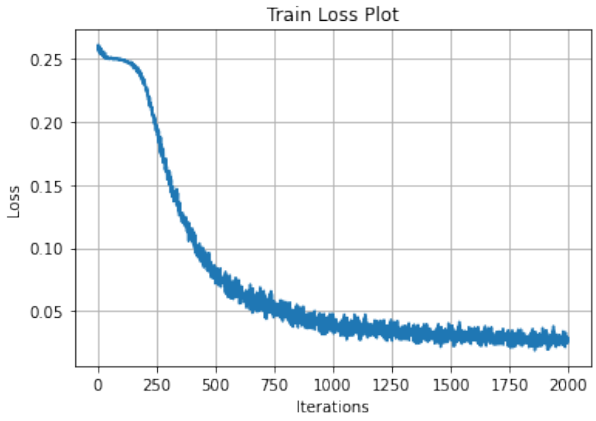

Figure 44: Hidden State Size 5 - MSELoss

Figure 45: Hidden State Size 5 - MSELoss

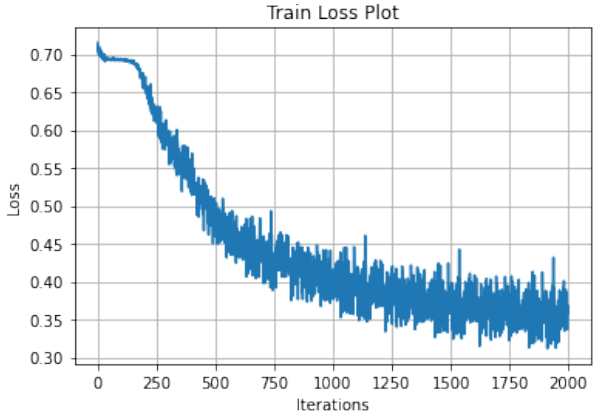

Figure 46: Hidden State Size 10 - BCELoss

3.3.4 Comparison

- Models of all 3 hidden state sizes have reached 100% test performance with MSELoss. But, this isn't the case with BCELoss. With BCELoss, model with higher hidden state size seem to perform better. Varying hidden state size for MSELoss models didn't have any effect as all of the models have perfect performance.

Figure 47: Hidden State Size 10 - BCELoss

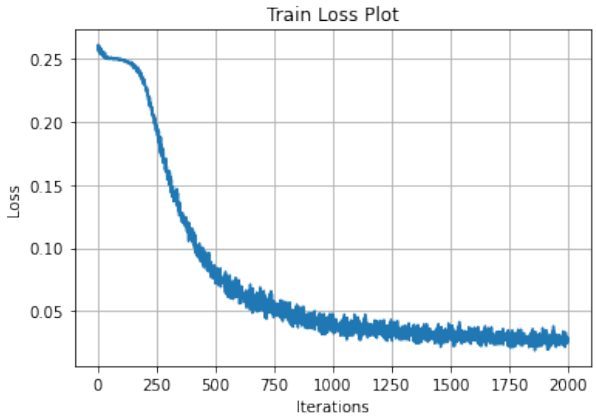

Figure 48: Hidden State Size 10 - MSELoss

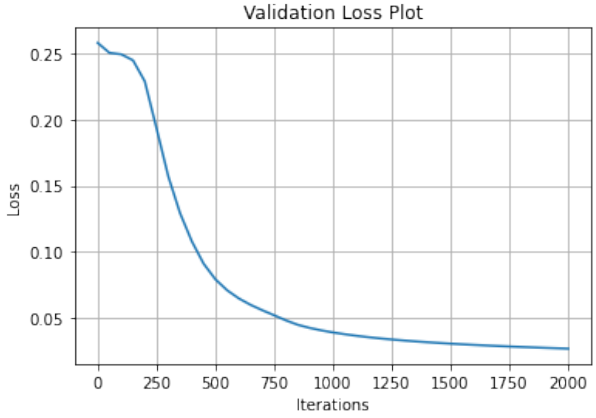

Figure 49: Hidden State Size 10 - MSELoss

- Models with hidden state size 10 and either of the losses have the highest test performance possible.
- For all 3 hidden state sizes, models trained with MSELoss have better performance compared to models trained with BCELoss.

3.3.5 Bit Accuracy Performance

As we train the models only on the binary inputs of length 5, we would like to see how well this model generalizes for input binary numbers of different lengths. So, we push binary numbers of different lengths and calculate how much % of output bits the model gets correctly. We then plot this bit accuracy against the length of the binary inputs given.

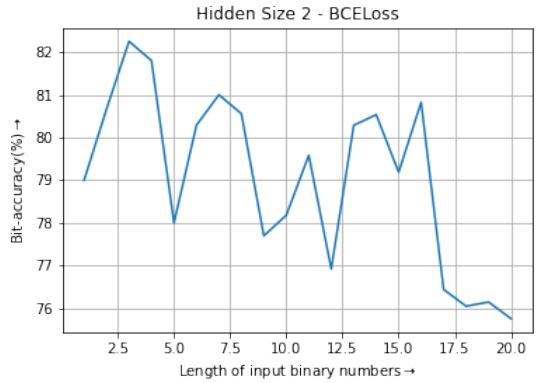

Figure 50: Bit accuracy vs Length Plot

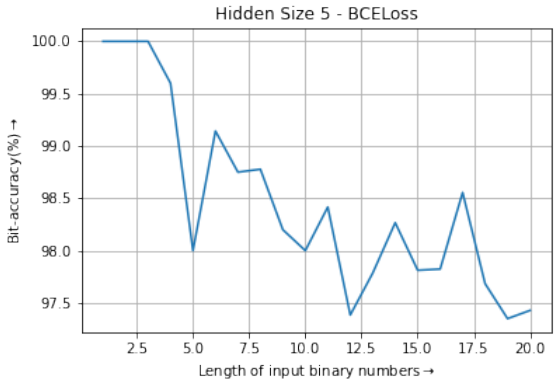

Figure 51: Bit accuracy vs Length Plot

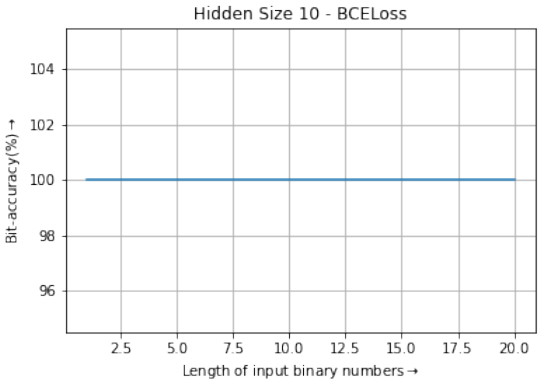

Figure 52: Bit accuracy vs Length Plot

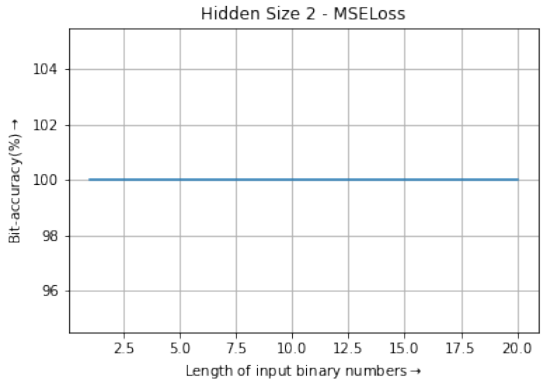

Figure 53: Bit accuracy vs Length Plot

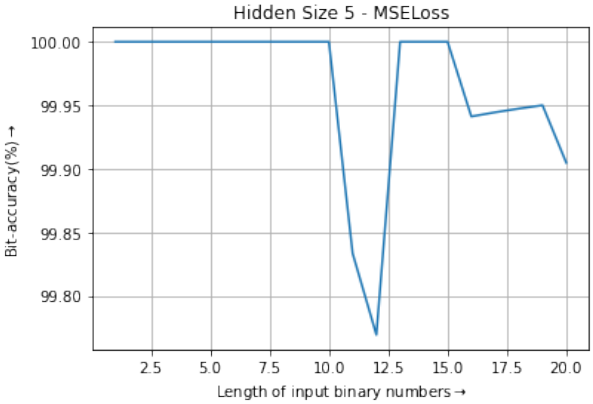

Figure 54: Bit accuracy vs Length Plot

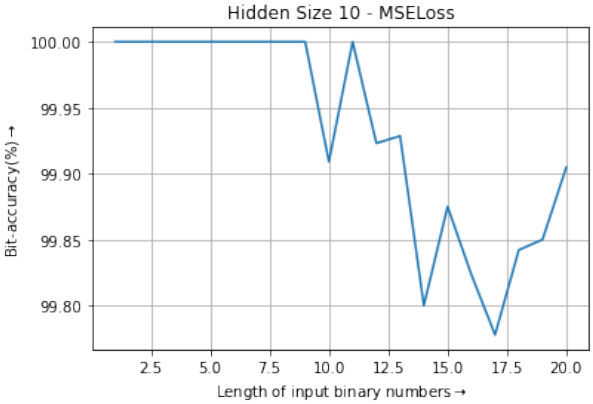

Figure 55: Bit accuracy vs Length Plot

The BCELoss models with hidden sizes 2, 5 seem to be a bit under-powered for what we want here (we have under-fitting). This has led to lesser performance. The best BCELoss model seems to be complex enough to capture binary addition and is also impervious to differences in input length. It seems to have understood the crux of binary addition which leads to its good performance even on inputs it wasn't trained on.

The MSELoss hidden size 2 model seems to be impervious to input size too and has perfect performance. This also seems to have understood the crux of binary addition. The other 2 MSELoss models perform perfectly for what they were trained on but don't

perform the same way with inputs of other lengths. This shows that these models have got trained to do addition for the specific length they were trained on rather than fully grasping the crux of addition. These 2 models have over-fitted to the dataset and have not fully understood the crux of binary addition yet due to overfitting (this is shown by the decreased performance for lengths other than 5). These 2 models were a bit overpowered for the data they needed to model.

Overall, BCELoss required bigger hidden state size to model binary addition well in comparison to MSELoss.

4 Code

The implementation of all the experiments used in this report can be found in the following colab notebook: Link

5 References

[1] T.-H. Chen, "MNIST Using Recurrent Neural Network," Machine Learning Notes, Jan. 13, 2018. Link

[2] M. R. Rezaei, "Lstm For_mnist Classification." Link

[3] S. User, "Vanilla_RNN_for_Classification." Link

[4] Advanced Deep Learning with TensorFlow 2 and Keras (Updated for 2nd Edition). Packt, 2021. Link

[5] "neural networks - How to select number of hidden layers and number of memory cells in an LSTM?," Artificial Intelligence Stack Exchange. Link

6 Conclusion

We have thus built various RNN models and analyzed them by applying them to 3 different datasets.