# Reinforcement Learning Network and CNN on MNIST Dataset

Shivani Sharma
Faculty of Computer Science, Dalhousie University
6050 University Ave, Halifax, Nova Scotia, Canada, B3H 4R2
Sh477091@dal.ca

## Abstract

A reinforcement learning model was integrated into a pre-trained MNIST handwritten digit recognition model to create an environment, where a handwritten digit is given as input and decide whether to count up or down from that digit based on the state and the rewards assigned to that state. Adequate results for Q-value & policy were achieved with the corresponding integrated model when the MNIST layers were frozen. Eventually, the above-combined model was also implemented without any pre-training network of the MNIST recognition model. The results were not as expected when the MNIST model was not pre-trained beforehand.

## 1. Introduction

Reinforcement learning is the training of machine learning models to make a sequence of decisions. In reinforcement learning, artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward. Multi-Layer Perceptron (MLP) is a class of feed-forward Artificial Neural Networks (ANN). MLP typically performs well at speech recognition, image recognition and machine translation applications. Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

This Project utilized a reinforcement learning model to develop a policy based on the ten-state system. This ten-state system included a reward of 1 at state 0 and a reward of 2 at state 9. In the beginning, a one-hot encoding network is implemented to test the basic functionality of the reinforcement learning model. The existing functions of the reinforcement learning were modified to accept an MNIST hand-written digit as input. The MNIST recognition layers were originally frozen, and the reinforcement model was trained. The MNIST recognition layers were then unfrozen, and the model retrained to evaluate the performance. Finally, an attempt was made to train the entire model without any pre-training of the MNIST recognition layers.

## 2. MNIST Dataset

The MNIST [3] data-set consists of 70,000 images of handwritten digits, out of which 60,000 are used for training and 10,000 are used for testing purposes. Each instance is a 28x28 pixel gray-scale image.

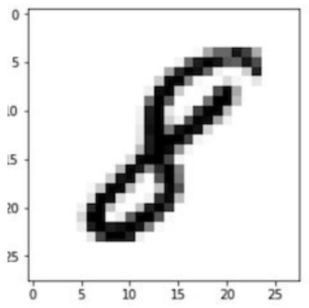

*Fig 1: MNIST Raw Dataset*

The MNIST model was trained using a categorical cross-entropy loss function and the Nadam optimizer. The training data was a one-hot representation of the value of the digit between 0 and 9. Thus, the model was trained to produce a one-hot representation of the handwritten digit, though due to the soft-max function, the values of the one-hot only achieved values near 1 or 0, so the argmax function was used to achieve a true one-hot representation based on the maximum value of the output array. In order to add the layers and dense the model, we are using a relu function. The output here is predicted using the reinforcement model and the y_test_data.

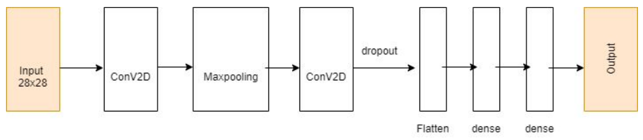

***Fig 2: MNIST Network Architecture***

## 3. Building the Network

The reinforcement learning model is designed to learn how to navigate a ten-state system with rewards at state 0 and state 9, as indicated in the table below. The rewards are given just for state 0 and state 9. This model uses the value of gamma as 0.8 and that of the inverse temperature of invT=1.0. After 50 trials, the value of invT is reduced to = 0.001. 400 trial iterations were used in total for the initial implementation. The model learned the Q values, the maximum value of which would indicate whether the model should count up or down from a given state, and convert to -1 (countdown) or +1 (count up). The instruction to go whether up or down is called a policy, and it was created using the argmax of Q. Q is found using the input image and then predicting the output for each step. This prediction Qs is then appended to Q to get the policy.

## 4. Baseline Model

The MNIST model and the reinforcement learning model were combined, which allowed the model to be called later without retraining. The model was later on loaded into the reinforcement learning model. The two models were then combined by setting the output of the MNIST model as the input of the reinforcement learning model. The agent starts at any random state other than 0, and 9. If the agent reaches the states 0 or 9, the session or trial is complete, and the environment is reset (re-initialized). The transition matrix for calculating analytical Q-values is as follows:
T[0,0]=1; T[1,1]=1; T[2,0]=1; T[3,4]=1; T[4,2]=1
T[5,7]=1; T[6,4]=1; T[7,9]=1; T[8,7]=1; T[9,11]=1
T[10,9]=1; T[11,13]=1;T[12,11]=1; T[13,15]=1; T[14,13]=1 T[15,17]=1; T[16,15]=1; T[17,19]=1;
T[19,19]=1

## A. Stage 1 - Using Pre Trained MNIST Recognizer

Stage 1 of this implementation was to test the performance of the above-created baseline model with both frozen and unfrozen layers. In the beginning, the layers are not frozen, so to freeze these layers we are using layer.trainable = False for these layers of the model. After training, these layers were frozen, and are appended with dense layers which predict Q-values. .Further, the MNIST layers were set as trainable, so both the MNIST model and reinforcement model could be trained during the experimental time. While exploring different ranges for the trials, it was quite evident that as we increase the number of trials, the more sensible results are yielded for the policy and raising the number of trials with invT=1.0 to 30.

The error values of the iterations are stored in a list, which will be further useful to compare the results of frozen and unfrozen layers. The model was retrained with the frozen layers, by tweaking the invT, and discount factor(gamma).

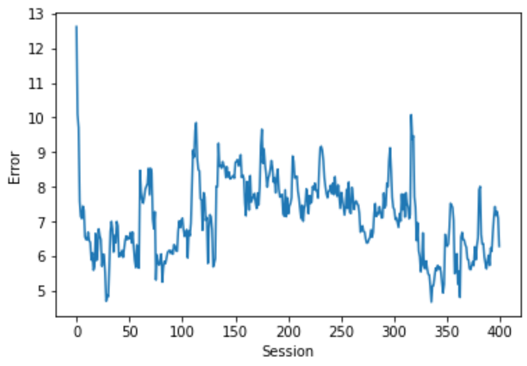

***Fig 3: invT=1.0***

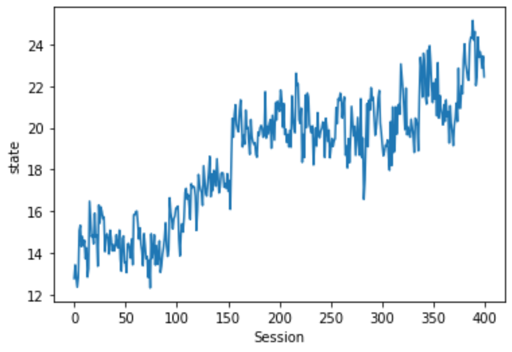

***Fig 4: invT = 3.0***

## B. Stage 2 - Deep Convolutional Q-Learning without Pretraining MNIST Recognizer

In this stage, the MNIST recognizer is not pre-trained and the model is expected to learn the policy from the ground up, just by taking the MNIST digit images as input. Challenges were encountered when trying to implement the combined MNIST recognition model and reinforcement learning model without any pre-training of these components. One of the primary challenges seemed to be a lack of exploration of the state space, which is presumed to be important to this training exercise, even if the model doesn't know for sure what the handwritten digits mean initially. As observed from the error rates, the model without pretraining did not converge as smoothly as the one with pretraining. This is because the model has to learn the features of the images as well as the objective Q-function at the same time.

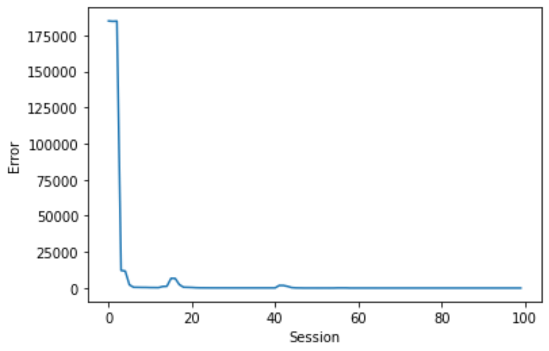

***Fig 5: invT = 1.0***

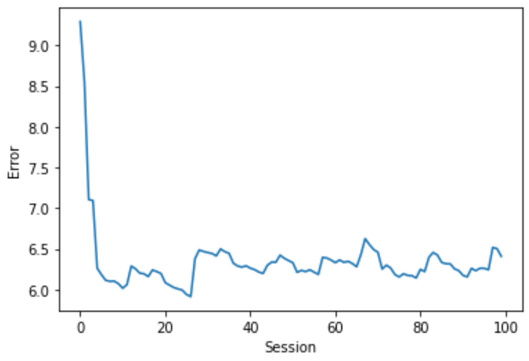

***Fig 6: invT = 0.001***

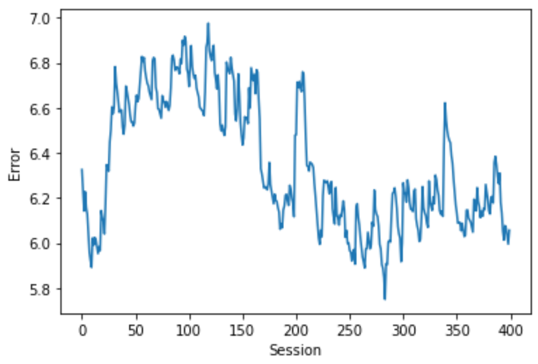

***Fig 7: invT = 10.0***

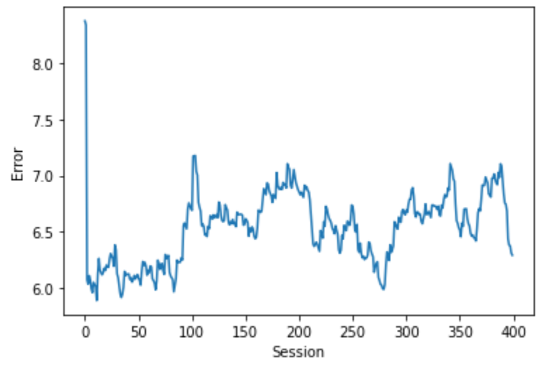

***Fig 8: invT = 30.0***

## 5. Q-values and Policy

| Target Q values at each state | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 |
| r = 1 | 1.0 | 0.41 | 0.8 | 0.52 | 0.64 | 0.65 | 0.51 | 0.81 | 0.40 | 1.0 | 0.32 | 1.28 | 0.26 | 1.6 | 0.20 | 2.0 | r = 2 |

**Policy**
[ 0 -1 -1 1 1 1 1 1 1 0]
In this table, -1 corresponds to counting down, and +1 corresponds to counting up. Thus, for any state greater than or equal to 3, the model should count up, and for any state less than or equal to 2, the model should count down.

## 6. Conclusion

A ten-state reinforcement learning system was implemented using Keras library functions. This was combined with an MNIST handwritten digit recognition system to produce a model that could accept handwritten MNIST digits as input. Also. it can be observed that the agent should be sufficiently allowed to explore the state-space to learn the Q function, where it is not pre-trained. Here in two different stages, we learned how an agent learns the Q-function and comes with a policy that gives more reward.

## 7. References

[1] Keras. (n.d.). ‘Keras Applications’ [webpage]. Available at: https://keras.io/api/applications/> [Accessed: 4th December 2020].

[2] Trappenberg, T. (2019). *Fundamentals Of Machine Learning*. Oxford: Oxford University Press. [Accessed: 10th December 2020]

[3] LeCun, Y., Cortes, C. & Burges, CJ. (2010). *Mnist handwritten digit database. AT&T Labs [Online].* Available at: http://yann.lecun.com/exdb/mnist [accessed 4th December 2020].