Implementation and Evaluation of Model Free RL Network with MNIST Data

Sumith Sai Rachakonda

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

***Abstract*-** The idea behind reinforcement learning is introduced in mid 1950s by Richard Bellman an American Mathematician. Reinforcement Learning (RL) is a technique in machine learning which uses trial and error method with feedback from its own actions and experiences by enabling an agent to learn interactively. This paper presents different implementations of a reinforcement learner which can count either up or down by taking an MNIST image and spits out a function approximator that can tell us which state to move next. Neural networks provide the flexibility to implement a reinforcement learner with rewards and punishments for every action. However, there exists many implementations. This paper contains an implementation and evaluation of a simple neural network enabled for reinforcement learning with two 10 Q-states and 2 actions on MNIST handwritten digits dataset. The goal is to analyze the policies generated with different implementations explained in this paper and evaluate the performance levels of these techniques and implementations. As a result, this simple RL network approximate the next best state for an agent.

I. INTRODUCTION

This paper uses the MNIST dataset for training and testing various RL models. The MNIST dataset consists of 60 000 samples of handwritten digits in the training set and 10 000 in the testing set. This paper compares the performance of a multi-layer perceptron (MLP) with RL layers appended at the end. We know that we can use MLP for classification problems. We have great modules which implements advanced RL networks but in this paper a network is implemented using basic MLP model with Keras, Python and Numpy to understand what exactly is going on while implementing an RL network. This is a minimalistic implementation without much complexity. In general, the models we use are more complex and accurate. The main idea behind reinforcement learning is to let the algorithm to find the suitable action model by maximizing cumulative rewards and minimize the punishments of the agent. The next sections of the paper explain three different types of implementations for a reinforcement learner which are based on book written by Thomas [1].

The Q-learning and State Action Reward State Action (SARSA) are the two RL algorithms that are mostly used. They differ in terms of exploration versus exploitation strategies. Q-learning uses the value that is derived from another policies action, whereas SARSA learns the value on current policy action. Q-learning is off-policy approach and SARSA is on-policy approach. These algorithms are basic yet simple and lacks some abilities when compared to advanced algorithms such as DQNs and DDPG. Here, we are implementing models like DQN model.

Before going forward, we need to know what a one hot representation for a number or state is because this paper uses one hot vector to represent the state values. Let us say, we can represent a number 1 in one hot vector as [0,1,0,0,0,0,0,0,0,0]. A one hot representation is a combination of ones and zeros. One represents the active state and zero represents the inactive state. In the above example we have all zeros except for index 1 (considering index starts from zero) because it says that index is active which means the number it is representing, similarly each position or index represents a

number with rest of the positions set to zero. This also applies to represent a state value for an agent. We also need to know about terms like environment, state, reward, policy, and value. The environment is where the agent operates. The State us the current situation or location of the agent. The reward is the feedback of the action taken by agent. The policy is a method we use to map the state of actions for an agent. The value is the future reward that can be received by an agent if it takes an action. We use a function tau which gives the next state, and a function rho which gives the reward, and a function to check whether the current or next state is terminal state. We have requirements such as reward for state 0 is one and for state 9 is two, we have a maze with 10 states to move, first and last states are terminal states. The reinforcement learning rule can be seen in the below image.

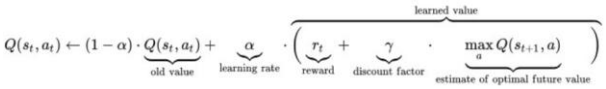

Figure 1 Reinforcement Learning Rule [2]

This rule can be used to update the Q-values for each trail. We use the old state and the next state Q values to update the present state Q-values.

II. APPROACH

2.1 Model-1

We are using a pre trained MLP model for classification of MNIST images to one hot representations. MLP is a basic or common model for supervised learning. These networks use backpropagation with fully connected layers to learn and optimize weights for each individual neuron. In simple terms a neuron will process the data with the help of predefined calculations. Each layer can have multiple neurons. The input layer will have neurons equal to the number of input features. The hidden layers can have different neurons. The output layer will have the same number of neurons we are going to predict. For example, if we want to predict the sales of a product then the output neuron will be one because we are only predicting the number. If we want to predict whether a dog or cat or bird in an image, then we need 3 neurons one for each type in the output layer. The data enters from input layer into the algorithm, all the calculations that is training will happen till output layer and finally the output will be given from the output layer.

This model first predicts the one hot value of a number from MNIST database. The MNIST database images are 28x28 sized images. All the 60,000 training images are used for training purpose and 20,000 images are used for validation purpose. With basic implementation, we have 98% accuracy to predict the given image. Later, to this model, we are appending an RL model so that we can use the output of the pretrained plus new network in combination to predict the next state or count either up or down for a given image.

This MLP model consists of one input layer, two hidden layers, one output layer. The input layer consists of 784 neurons because we have 784-pixel inputs. The first hidden layer has 392 neurons, and second hidden layer has 196 neurons. The output layer has 10 neurons because we have to predict one hot value of a number which needs ten positions to represent numbers from 0 to 9.

2.1.1 Implementation of Model-1

1. Importing packages required such as matplotlib to visualize the data and numpy for data manipulation and mnist for loading images, and keras for model initialization.
2. Loading data from mnist module and normalizing them by dividing train images with 255 so they are in 0 to 1 range which helps the model to learn easily. Flatten the images to feed to the network.
3. Converting the labels from number to its one hot representation.
4. Initialize a sequential model with one input layer (784 neurons), two hidden layers (392,196 neurons respectively), and output layer with 10 neurons. Set relu as activation function for hidden layers and softmax for output layer which can be used to convert the vector to one hot representation.
5. Compile the model with categorical cross entropy loss and adam optimizer and accuracy metrics.

6. Fit the model with all the training data and use test data to validate the model
7. Now we have a trained network that can classify MNIST images, we have to freeze all the layers in order to add the RL layers at the end. We can freeze the layers by setting trainable attribute for each layer to false.
8. Add two dense layers with 10 and 2 neurons respectively using relu and linear activations, respectively.
9. Again, compile the model with mean square error loss and RMSprop optimizer.
10. Define function tau to return next state by checking the current state, if the current state is not a terminal state then we have to roll the state to the given action value (1 or -1), if it is a terminal state then return the state.
11. Define a function rho which returns the reward for a state, the agent should get a reward of 1 if it is in state 0 and reward of 2 if it is in state 9, any other states gives 0.
12. Define a function to check terminal state, check if the state vector is in state 0 or 9, return true if it is, otherwise return false.
13. Set discount factor (gamma) to 0.5 and inverse temperature to 1.
14. Train model using DQN approach.
    Iterate through desired trails:
    a. Consider an image in dataset.
    b. Iterate through 10 next time cycles:
        i. Predict the Q-values for the given image using the combined model.
        ii. Get the fourth layer output and convert it to one hot vector, because this layer will give us the predicted number of given images.
        iii. Check if terminal state, if it is, break the iteration.
        iv. Check if trail greater than 30 and inverse temperature greater than 0.1, then reduce the temperature by 0.001
        v. Find the maximum Q-value's index. (we have to move this direction)
        vi. Check if a random number is less than inverse temperature, if it is, then shift the index.
        vii. Calculate the action by multiplying index with 2 and reducing the result with 1.
        viii. Find the next state using tau function by passing current state and action value.
        ix. If next state is terminal state then calculate reward.
        x. Else calculate the Q-value for next state.
        xi. Use this reward/Q-value to update the existing Q-value for index.
        xii. Fit the model with the updated predictions/Q-values.
        xiii. Use this next state as the initial state for the next iteration.
15. Using this trained model to generate a policy and save Q-values for each time cycle.
    a. Initialize a vector (size 10) with all zeros called policy, and an empty list Q to save all Q-values for each state.
    b. Get the image of state we want to start in maze.
16. Iterate through 10 steps:
    a. Predict the Q-values for an image and append it to Q list.
    b. Find the index of direction that model suggests by taking argmax of prediction.
    c. Calculate the policy for that iteration by multiplying the index with two and reducing with one from result.
    d. Get the 4^{th} layer output in order to get the current state of the image and convert to one hot representation.
    e. Roll to the next state by 1.
    f. Get the next state image

g. Manually set first and last state of policy to zero.

17. We have the calculated policy for a given image and its state.

2.1.2 Result for Model-1 implementation

For this model, we can calculate the performance by using a simple approach called absolute difference. We can calculate the ideal or optimal policy. The optimal policy for this 10 state problem would be [0,-1,-1,1,1,1,1,1,1,0]. 0 indicates no movement, -1 indicates left movement, 1 indicates right movement. We have to set alpha to 1 so we can it looks similar to the model-based case. However, we are not able to obtain an optimal policy because lack of finetuning of the model. We are able to get a good policy though. The model is sometimes unstable because of its stochastic nature of each state.

2.2 Model-2

It is similar to the first model, but here we are conducting experiment by using all layers, that is we are not freezing any layers.

This model behaves differently when compared to the previous model because, the weights in all layers are getting updated including the MNIST classification layers, so the model's accuracy for predicting the right state and next state is differing in each cycle with updated weights. This causes problem of misclassification when extracting the state from 4^{th} layer. As the model gets trained on the Q-values, the weights in the whole network are backpropagated and updated. So, we are not able to achieve optimal policy and also not so performant when compared to Model-1. The absolute difference between ideal and actual Q-values is higher than Model-1 in each iteration.

2.3 Model-3

It is also like the above models, but we are not using a pre-trained model, instead we are directly constructing a model with both classification and RL layers, then using this model to train on images and Q-values.

In this model we are using a helper model to predict the state of a given image and using this state value to calculate the reward and next state in the algorithm.

This model's performance is very low, and it is biased to a single state in each experiment. It is depending on the state image feed to the network. It is more biased towards the last state where we get high reward irrespective of the current state. The mean absolute difference of predicted and optimal Q-values is quite high because of its bias towards a single state.

When tried without using the helper model, and by extracting the 4^{th} layer output, its performance is way too low when compared to the model that uses helper model. This is because of the misclassification of image due to lack of training on MIST data. The mean absolute difference of this implementation is very high.

III. CONCLUSION

In this paper we have investigated different implementations of RL network using DQN approach by combining classification MLP model with RL layers. Three implementations have performed distinct in each experiment because of their stochastic nature. We can say that these implementations are not stable and not reliable. Future work can include in optimization for optimal policy and using DDPG approach.

IV. REFERENCES

[1] Trappenberg, Thomas P. Fundamentals of Machine Learning. Oxford, United Kingdom, Oxford University Press, 2020. (accessed Dec. 10, 2020).

[2] https://www.facebook.com/kdnuggets, "5 Things You Need to Know about Reinforcement Learning - KDnuggets," KDnuggets, 2018. https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html (accessed Dec. 10, 2020).