BITS F464 - Machine Learning

Major Project Report

Prepared By

Mohanish Mhatre - 2017A7PS0938G

Ajinkya Deshpande - 2017A7PS0152G

Reshma Seby John - 2017A7PS1909G

April 2020

Abstract:

Reinforcement Learning is a method in Machine Learning in which an agent interacts with and takes actions in an environment so as to achieve maximum reward. In this project a reinforcement learning model was built to solve the task of balancing a pole on a cart, which is running on a flat-terrain, with a noisy set of parameters and varying environment. We analyzed various approaches in order to obtain the model which gave the best results. Our results showed that the Double Deep Q Learning model converged the fastest in a smaller action space.

Introduction:

The primary objective of our project was to solve the Cart-Pole problem: this consists of a pole which is placed on a cart in a fixed environment with classical physics. The pole needs to be balanced for a certain amount of time inside an acceptable region in which it is free to move using Machine Learning.

Problem Description:

To build a Reinforcement learning model that will be capable of balancing a pole on a cart which is running on a flat-terrain with a noisy set of parameters like action and sensors, in addition to having randomly varying values for gravity and friction at each step for the same episode. The cart-pole problem had to be solved with the following settings:

1. With random variation in gravity and friction.
2. With the previous setting's noise and noisy controls. This means that the cart's force in the desired direction may be less/more than expected.
3. With both the previous modifications and noisy sensors/sensor observations of the pole angle at any moment.

In order to achieve the above, the Reinforcement model built should arrive at a point in which the cart is able to balance the pole in as few steps as possible, with less exploration of possible moves. The performance metric given helped to evaluate the different models tried in different approaches.

Evaluation metric:
The evaluation of the Reinforcement learning model built will be based on how many trials/runs the agent takes to start achieving an average episode score of greater than 480 over the last 100 runs.

In this project we built and tested three types of Reinforcement models to train the agent, all of which contained Neural Networks.

Approaches:

To solve the task, different models based on various approaches in Reinforcement learning were tried. In each approach we aimed at improving the learning capability of the agent by introducing more complicated techniques.

First approach: NN with dense layers only.

A neural network is a machine learning model in which, like the neurons in our brain, converts the given input into an output. A dense layer is a hidden layer with a group of neurons/nodes that take the inputs and produces desired output. In a Neural Network with dense layers, several dense layers have been added one after another with the number of nodes wanted in each layer specified. Varying these values controls the capacity of the neural network. The layers are fully connected and the output of one layer becomes the input of the next.

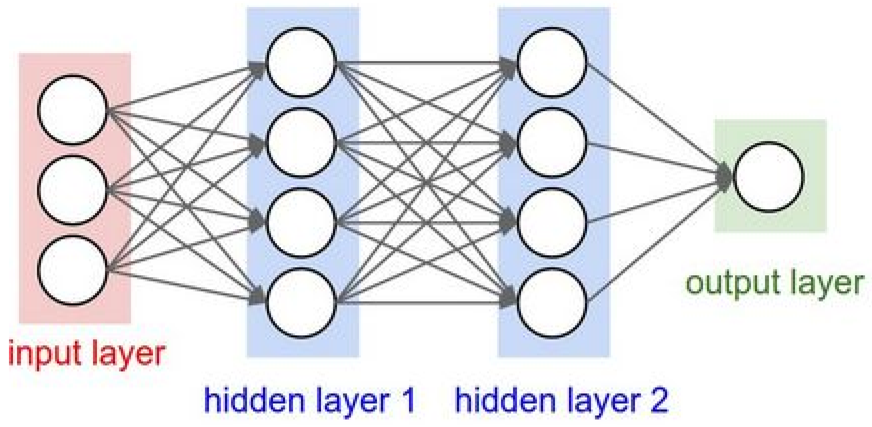


Figure 1: Diagram of Neural Network with Dense layers

The neural net we used is similar to the diagram. We have one input layer with 4 nodes that receives information and 5 hidden layers with number of nodes: 256, 512, 1024, 512, 256. This is then connected to the output layer that has 2 nodes.

Problem with this approach: The neural network tried to predict an action without accounting the reward or if the game was over.

Second approach: Deep Q Learning with Dense neural network:

In this approach, Q-Learning is combined with a (deep) neural network, which results in a technique called Deep Q-Learning (DQN).

The Q-value is a measure of the expected reward (it calculates the expected upcoming value) and is denoted as Q(s,a,θ) in DQN where s is the state the agent is in, a is the action it performs and θ represents the weights the network is parameterized by.

In Deep Q network algorithm, a neural network is used to predict the Q-values and approximate the reward based on the state. This helps to cope with larger numbers of state-action pairs and hence with larger scale tasks.

The input is a vector of states and the output of the neural network is a vector of Q-values for each doable action.

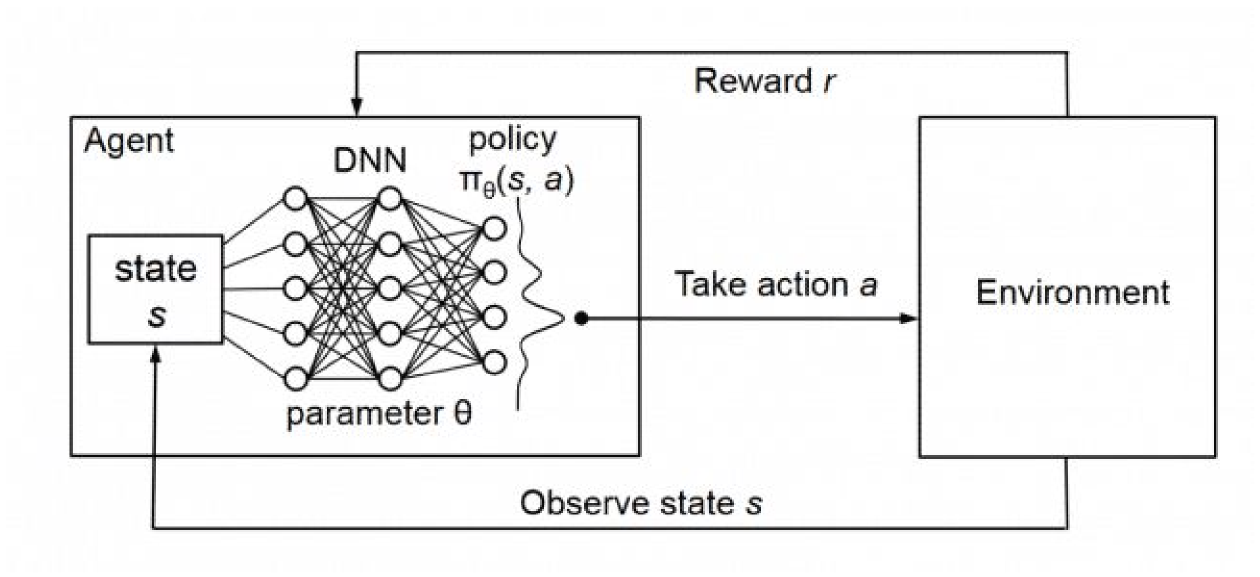


Fig2: Diagram showing the working of Deep Q Learning.

Experience Replay is used to do the training; the possible random actions that the agent tries and the consequent experiences are stored into memory. The data stored includes those necessary to perform learning including the states, state transitions, actions and rewards. The previous experiences are then used to re-train the model by the implementation of a replay function.

Using this approach, the learning speed increases and by every episode the network and prediction improves.

DQN uses a single Q network to generate the Q value to update but by doing so, the network is updating its parameters to more effectively predict what it itself outputs. This can lead to faster and ineffective updation of the network. Hence this approach was improved using Double Deep Q Learning with Dense neural network:

Double DQN is an enhanced version of the original DQN in which two Q networks are used to facilitate the action selection; target Q network and primary Q network. The target is being updated while the other one decides the next best action while remaining still.

Double DQN helps us reduce the overestimation of Q values and helps us train faster and have more stable learning.

While trying the different approaches, the number of neurons in the hidden layer was changed to see the improvement in the score.

We used a simple neural network with 2 hidden layers and 24 neurons each (which were found to be the best fit). We trained our model in a stochastic manner as it led to a faster convergence over minibatch which converged really slowly.

We choose the following hyperparameters after short trials to estimate their convergence.
Hyperparameters:
Hyperparameters are the parameters that need to be passed to a reinforcement learning agent.

- Gamma - also called the discount rate, is multiplied to future rewards to discount them. We have set gamma to 0.99.
- Epsilon - this is the exploration rate, which gives the rate in which an agent randomly decides its action. Epsilon has been set to 1 in order to make it 100% certain that the agent will start by exploring the environment.
- Epsilon_decay - this gives the rate by which we want to decrease the exploration as time goes by. We have set this to 0.99.
- Epsilon_min - gives the minimum amount of exploration we want the agent to do. This has been given as 0.01.

We trained the model to reach a score of average 490(over the last 100 episodes) within just 2400 episodes.

Nevertheless, we decided to dive deeper to improve the score rating and the training speed.

Third approach: Actor-Critic Model

In the Actor-Critic model the primary idea is to split into two models:

- Actor: controls how our agent behaves by updating its policy parameters and deciding which action to take (policy-based).
- Critic: evaluates the action taken by computing the value (Q-value) and tells the Actor how it should adjust (value-based).

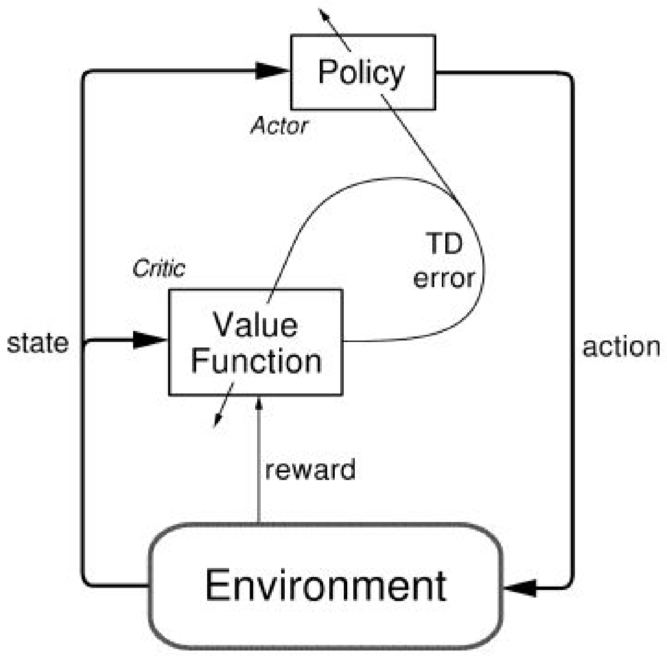


Fig3: Diagram of the Actor-Critic Model.

The concept of this model is that the Critic must learn about and criticise the policy currently taken by the actor. The Actor must then use this feedback to update its policy and get better. The training of the Actor and Critic networks are performed separately.

With time, the models get better with their role, i.e, the Actor produces better actions and the Critic shows improvement in evaluating those actions.

We tried implementing the Actor-Critic model using both Deep Q learning and Double Deep Q learning. After trying the Actor-Critic model approach we became aware that the Double Deep Q Learning with Dense neural network gave faster results in performing the tasks.

Training Logs:

Try 1: **NN with dense layers only(256,512,1024,512,256)**: Score - 10.9

Train Episodes: 2000

Try 2: **Deep Q Learning w/ Dense neural network(24, 24)**:

Train Episodes: 1000
Epsilon = 1
Min_epsilon = 0.01
Epsilon_decay = 0.99
Gamma = 0.995
Batch_size = 1
Training average Score - 110

Try 3: **Double Deep Q Learning w/ Dense neural network(24, 24) stochastic**:

Train Episodes: 2400
Epsilon = 1
Min_epsilon = 0.01
Epsilon_decay = 0.99
Gamma = 0.99
Batch_size = 1
Update_every = 1
Training average Score - 280

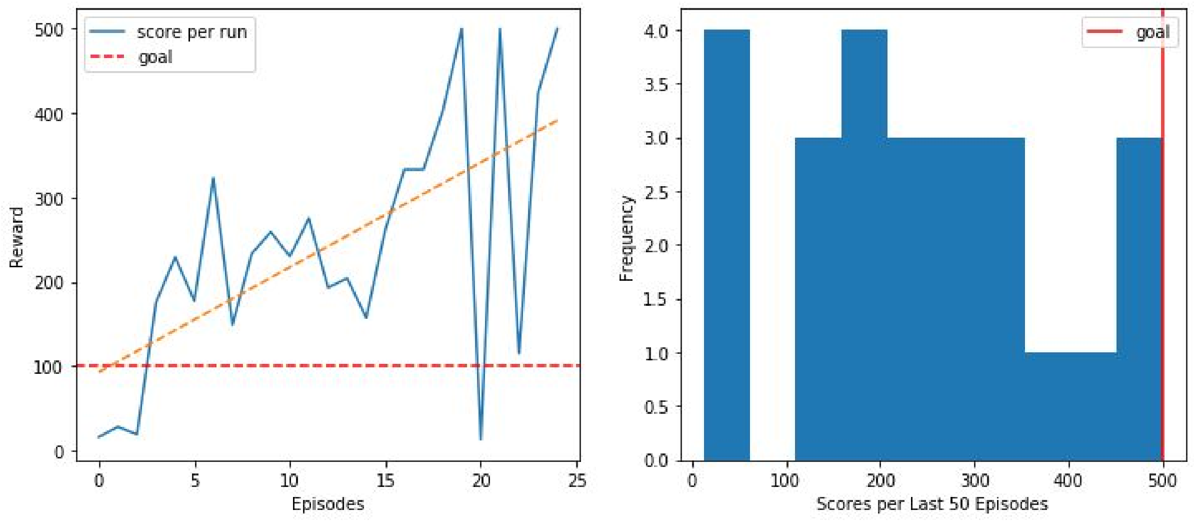


Figure 4: Graphs depicting the scores of episodes in Double Deep Q Learning w/ Dense neural network(24, 24)

Try 4: **Double Deep Q Learning w/ Dense neural network(32, 32) stochastic**:

Train Episodes: 7000
Epsilon = 1
Min_epsilon = 0.01
Epsilon_decay = 0.99
Gamma = 0.99
Batch_size = 1
Update_every = 1
Training average Score - 127

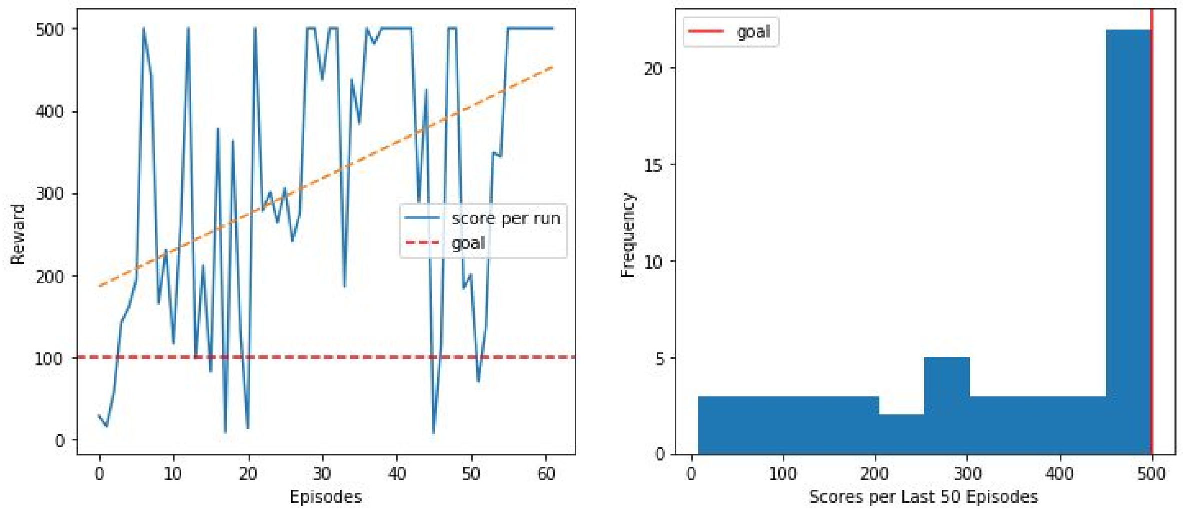
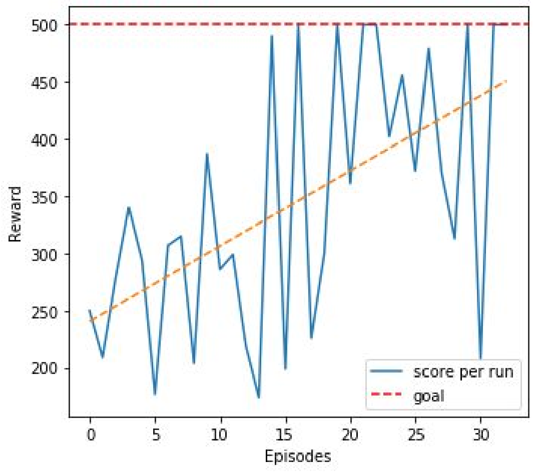


Figure 5: Graphs depicting the scores of episodes in Double Deep Q Learning w/ Dense neural network(32, 32)

Try 5: **Actor critic (Deep Q learning) w/ (24, 24), ((24, 24),(24, 24), 24):**

Train Episodes: 100000
Epsilon = 1
Min_epsilon = 0.01
Epsilon_decay = 0.99
Gamma = 0.99
Batch_size = 32
Training average Score = 340



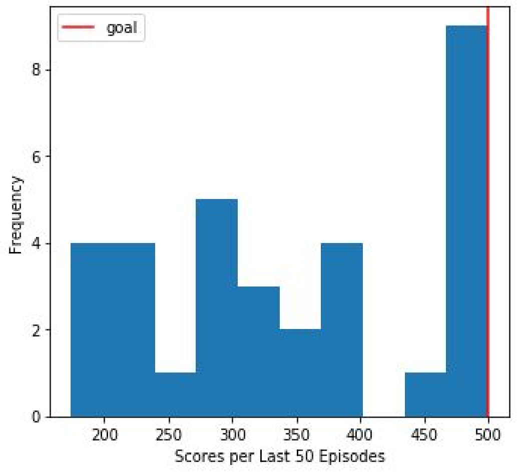


Figure 6: Graphs depicting the scores of episodes in Actor Critic (Deep Q Learning) w/ (24, 24), ((24, 24),(24, 24),24)

Try 6: **Actor critic (Double Deep Q learning) w/ (24, 24), ((24, 24),(24, 24), 24):**

Train Episodes: 100000
Epsilon = 1
Min_epsilon = 0.01
Epsilon_decay = 0.99
Gamma = 0.99
Batch_size = 32
Update_every = 20
Training average Score = 360

Test score over 100 episodes: 500

Conclusion:

1. Reinforcement learning involves training a model based on the reward it gets from each action.
2. The simple model of Deep Q learning converged before the complex actor-critic model. Thus, the simple Double Deep Q Learning model converges faster in a smaller action space than an Actor-Critic model.
3. The solution to the cart-pole problem is to keep moving slowly in a general direction as learnt by our models. Our models learnt to move slowly to the right while keeping the pole straight to balance it over a 100 episodes with 500 frames.

Contributions:

We worked together on the Deep Q learning part. The individual contributions were as follows:
Mohanish - Actor-critic model and documentation.
Ajinkya - Improving code quality and providing abstraction.
Reshma - Model research and documentation.

References:

Papers:

https://www.aaai.org/ocs/index.php/FSS/FSS15/paper/download/11673/11503

https://www.researchgate.net/publication/3206302_A_Cartpole_Experiment_Benchmark_for_Trainable_Controllers

https://arxiv.org/pdf/1602.01783.pdf

https://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf

Online articles:

https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288

https://medium.com/@ranasinghiitkgp/introduction-reinforcement-learning-with-epsilon-greedy-bandit-game-algorithm-73c22c1646b3

https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9

https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

https://keon.github.io/deep-q-learning/

https://dev.to/n1try/cartpole-with-a-deep-q-network

https://theaisummer.com/Actor_critics/

https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677 https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e

https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69

https://towardsdatascience.com/qrash-course-ii-from-q-learning-to-gradient-policy-actor-critic-in-12-minutes-8e8b47129c8c

https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b