IŞIK UNIVERSITY
COMPUTER
SCIENCE AND
ENGINEERING

DENSE VS FOCUSED ON DQN FOR REINFORCEMENT LEARNING

Bachelor's Thesis

Ersin ÇEBİ

217SE2407D

Supervised by

Dr. F.Boray TEK

June 2020

DENSE VS FOCUSED ON DQN FOR REINFORCEMENT LEARNING

Abstract

An important objective of the area of AI is to produce fully autonomous agents that interact with the environment to learn optimal behavior. The development of adaptive and efficient AI algorithms has been a longstanding challenge. RL is a principal mathematical system for autonomous learning guided by experience. Although in the past RL had some success, previous approaches lacked scalability and were inherently limited to relatively low-dimensional issues. This is because RL algorithms have the same complexity problem as other algorithms: memory complexity, computational complexity, and sample complexity, for machine-learning algorithms. The objective of this system is to develop an application for reinforcement learning and compare the efficiency of dense and focus layers.

Focusing neurons can generate unique connection maps for a problem. The new model uses no other tool than the back-propagation algorithm to learn its focus parameters which control the receptive field locations and apertures. So neurons can learn and adapts itself.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Focused: Focus scales weights and changes the variance of propagated signals.

Mu: Represents the focus center and controls the receptive field locations.

Sigma: Acts as the aperture of the focus's size control.

Python: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

Keras: Kerasis a Python-friendly open-source library for numerical computation that makes machine learning faster and easier.

Colab: Google Colab is a free cloud service and now it supports free GPU! You can; improve your Python programming language coding skills. develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV.

DENSE: A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

Q-learning: A model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.

DQN: The DeepMind system used a deep convolutional neural network.

CHAPTER ONE
INTRODUCTION

There are supervised and unsupervised learning techniques, Reinforcement Learning is under unsupervised learning technique. In unsupervised learning, the training data is unlabeled. The system tries to learn without a teacher. So basically the agent observers the environment and learns how to reach the endpoint or the goal that defined.

When we think about the concept of learning, one of the first things that come to our mind is interaction-based learning. While babies are playing games, waving their arms, or looking left and right, there is no teacher telling them what to do in their heads.

Reinforcement Learning agents have clear goals, they can feel the characteristics of their environment and choose the actions that will be effective in their environment. In the basic structure, the agent shows an action according to the environment, it is called policy and expects a response from the environment. The resulting reactions are subject to a predefined reward system. In line with the award won, the agent is trained and understands how wrong or right he is doing. The agent should try various actions and gradually choose the ones that look best. The purpose of this system is to design a reinforcement learning application and compare the Dense and Focusing layer's efficiency.

The focus idea is coming from the paper (Tek, 2018)[1]. This thesis is about implementing and examining Focused Neuron Model on reinforcement learning.

1.1 PURPOSE OF THE SYSTEM

The aim of the system is to design an accurate and efficient reinforcement learning DQN model that provides easy to implement, more effective than Dense neural networks and time-efficient. The system allows users to use different optimizer, loss functions as they wish. Also, there is an option for a focused layer for users so users can compare two-layer efficiency.

1.2 SCOPE OF THE SYSTEM

The project is based on a new approach to reinforcement learning that can learn its receptive field and so its local connections in a topological structure. In Keras, you assemble layers to build models. A model is a graph of layers. The most common type of model is a stack of the layer of the sequential model. By using Keras's Dense model we will implement and develop a structure for the project.

1.3 OBJECTIVES AND SUCCESS CRITERIA OF THE PROJECT

Following objectives are we plan to achieve:

1. Development of the new model
2. Testing of a new model
3. Development of a simple user interface to demonstrate model dynamics
4. Application and comparison of the model in common tasks

Success criteria of the project are to get more accurate results than a
The dense model with less computation.

1. More Effective than Dense
2. Easy to implement
3. Time Efficiency

CHAPTER TWO
LITERATURE REVIEW

Artificial neural networks are scalable computing structures, that can be applied with high precision to a wide range of time series able to predict problems. [4] The artificial neuron simply activates its output when more than a certain number of its inputs are active.

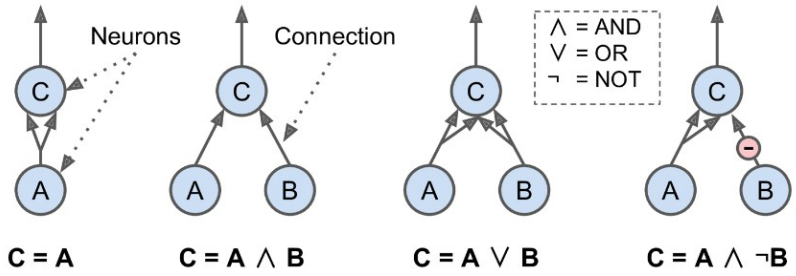

Figure 2-1. ANNs performing simple logical computations [3]

In the modern sense, the perceptron is an algorithm for learning a binary classifier called a threshold function: a function that maps its input x to an output value f(x). [5] A perceptron is a unit with weighted inputs that produces a binary output based on a threshold. An ANN node also sums weighted inputs, but instead of producing a binary output of 0 or 1 based on a threshold, it instead produces a graded value between 0 and 1 based on how close the input is to the desired category. The perceptron consists of 4 parts. One input layer, weights and bias, net sum, and activation function.

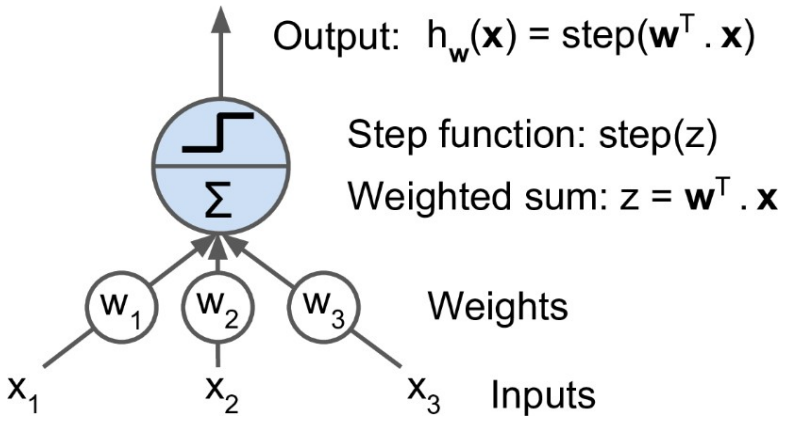

Figure 2-2. Linear threshold unit[3]

The operations of the Backpropagation neural networks can be divided into two steps: feedforward and backpropagation. [6] In the feedforward step, an input pattern is applied to the input layer and its effect propagates, layer by layer, through the network until the output is produced. The network's actual output value is then compared to the expected output, and an error signal is computed for each of the output nodes. Since all the hidden nodes have, to some degree, contributed to the errors evident in the output layer, the output error signals are transmitted backward from the output layer to each node in the hidden layer that immediately contributed to the output layer.

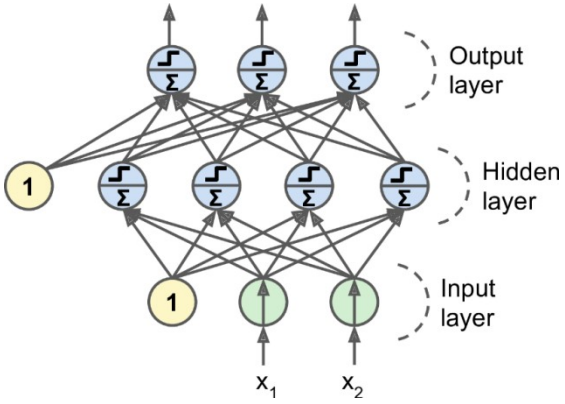

Figure 2-3. Multi-Layer Perceptron[3]

Once the error signal for each node has been determined, the errors are then used by the nodes to update the values for each connection weights until the network converges to a state that allows all the training patterns to be encoded. The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent [6]. The weights that minimize the error function is then considered to be a solution to the learning problem. The network behavior is analogous to a human that is shown a set of data and is asked to classify them into predefined classes. Like a human, it will come up with theories about how the samples fit into the classes.

Activation functions are mathematical calculations, which evaluate a neural network's results.[7] The function is applied to each neuron in the network and helps determine if it should be activated or not, based on whether the input of each neuron is relevant to the prediction of the model. Activation functions also help to normalize each neuron 's output to a range of between 1 and 0 or -1 to 1.

An additional feature of activation functions is that they must be computationally efficient, and for each data sample, they are measured through thousands or even millions of neurons.

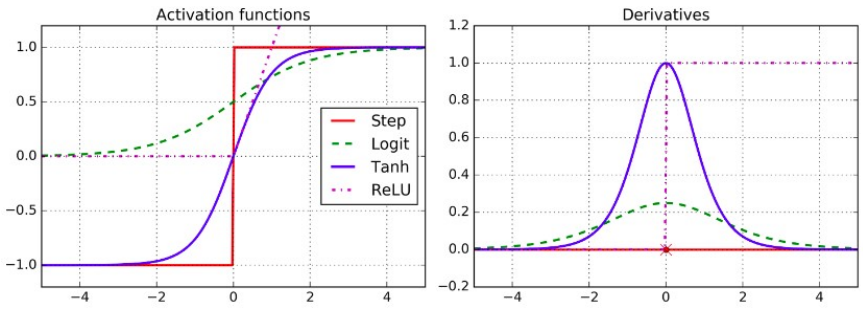

Figure 2-4. Activation functions and their derivatives[3]

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not [5]. Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1. If $\sum w.x > 0$, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.

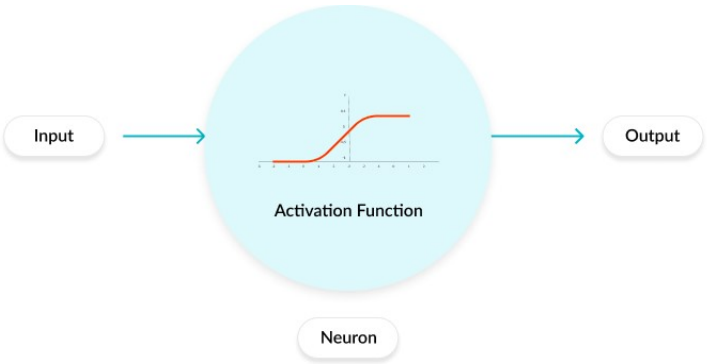

Figure 2-5. The role of activation function[8]

Q-LEARNING

There are plenty of ways of reinforcement learning[9]. One is q learning, this approach uses dynamic programming principles. Firstly we set random actions for states in the environment, this is called creating q-table. Q-table is a nested array that contains, actions on a specific state. Then we run our test on that table and get a Q value on every iteration. This q value calculated with an equation in Figure 2-6:

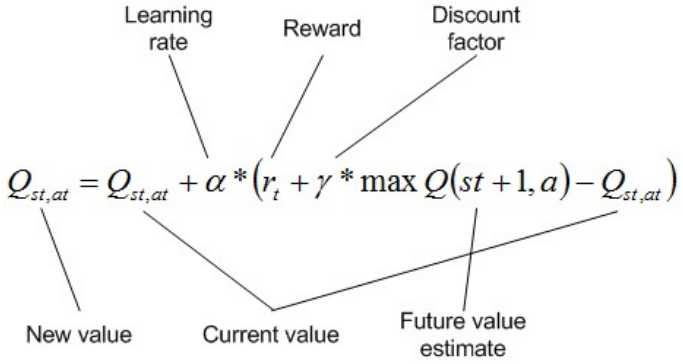

Figure 2-6: Deciding next Q-value on the table, according to the previous action that taken [10]

There is another way of q-learning, it's called Deep Q-learning [9]. In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an agent training application, the raw input may be a matrix of actions; the first representational layer may abstract the action and encode states; the second layer may compose and encode arrangements of states; the third layer may encode an action on a state, and the fourth layer may predict a proper outcome that the action of the current state.

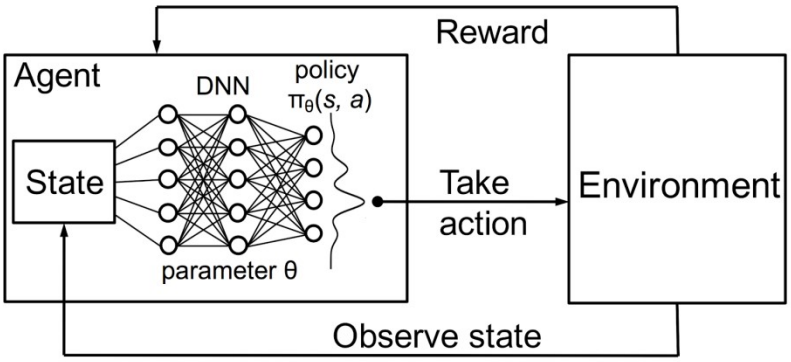

Figure 2-7: Reinforcement Learning using a neural network policy [11]

GYM ENVIRONMENT

The gym library gives us an agent and environments together. An agent is an artificial intelligence entity that has certain goals, must always remain watchful about things that can come in the way of these goals, and must, at the same time, pursue the things that help in attaining these goals. Because according to the actions of its environment the agent gets either prize or penalty.

According to the problem we give to the agent, the agent takes action. So, what decides this problem and action? The answer is the environment, the environment in the region available for the agent to navigate, and includes all the places the agent can go to, including the obstacles that the agent could crash in to. So the primary task of the agent is to explore the environment, understand how the actions it takes affect its rewards, be cognizant of the obstacles that can cause catastrophic crashes or failures, and then master the art of maximizing the goals and improving its performance over time.

Figure 2-8. shows the basic action taking the process of an agent according to its environment.

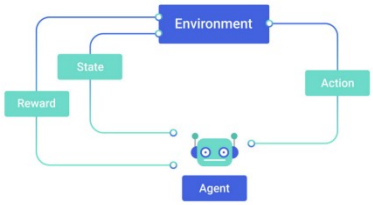

Figure 2-8: Agent decision diagram [12]

We said that the agent takes actions according to its environment, by the outcome of the previous action by meaning that the state(observation), reward, the agent does make these choices by the policy [9]. The policy is the algorithm that determines the action of the agent can use in the environment. we use DQN as a policy, that takes all observation as input and creates output and agent takes those outputs and decides the best action.

POLICY LEARNING

A policy defines the guidelines for an agent's behavior at a given state [9]. In mathematical terms, a policy is a mapping from a state of the agent to the action to be taken at that state. As a coder of the agent, we expect the best outcome from it, but the policy is not always deterministic, by mean that, it is possible that the policy takes only random actions, there is a chance policy does not observe the environment, it is called a stochastic policy. It is usually denoted as $\pi(a|s)$ – that is, it is a conditional probability distribution of taking an action a in a given state s. Policies can be deterministic, wherein the exact value of a is known at s, or can be stochastic where a is sampled from a distribution – typically this is a Gaussian distribution, but it can also be any other probability distribution.

The other way of policies genetic algorithms. In this type of policies we run for example 100 agents in an episode at the same time and choice the best ones, let say there is 20 succeeded, in the next episode you create 4 offspring each of the successful agents for the next generation.

Q-LEARNING WITH NEURAL NETWORKS(DQN)

Reinforcement learning is an unsupervised learning technique, that is why first we should create data to process, this data is unlabeled and we come up with this data according to the conditions of our environments [9]. An environment limits the agent’s actions according to the problem, this problem might be driving a car, a walking person, etc. We could create our own environment by using different libraries but there is OpenAI Gym. The gym gives us plenty of problems with the environment itself, these problems are solved in an environment, each environment has its own action space, observation space, and state variables.

In the agent training, we will use the Keras Sequential model with layer Dense and Focused [9]. While running tests users can switch between Dense and Focused layers. Focused layer it's a receptive field in the spatial domain inputs and by using the back-propagation algorithm to learn its focus parameters that control the receptive field locations and apertures, the action space of an environment can be too small. That is why we added a large size Dense layer before switch part of the model, in case of choosing Focused as a layer, so the Focused layer can work more efficient.

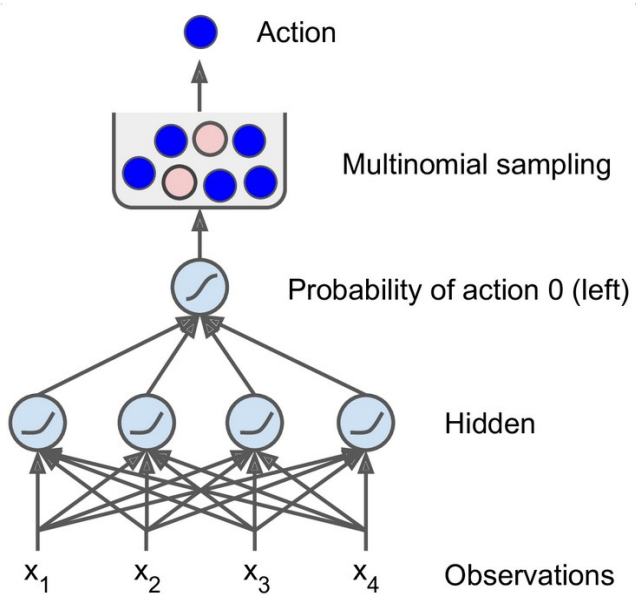

Figure 2-9: Neural network policy [9]

FIXED Q-VALUE

DQN has some variants, one is fixed q-value targeting. Deep Q Networks take as input the state of the environment and output a Q value for each possible action. The maximum Q value determines, which action the agent will perform. But the same weights apply to both the target and the predicted value. It is like making circles around the same path.

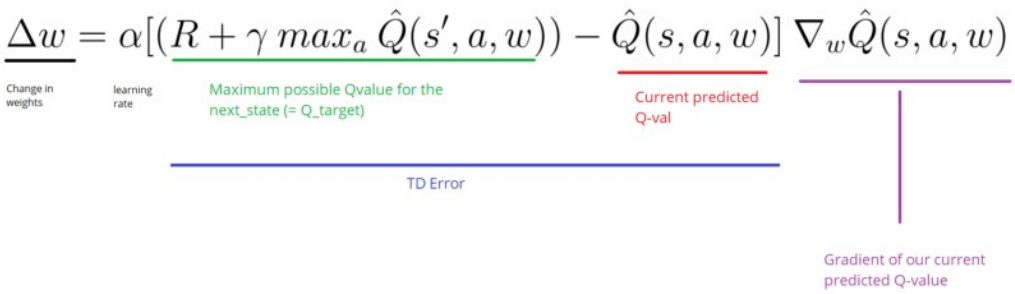

Figure 2-10: Calculating weight [13]

We move the output closer to the target, but we also move the target. So, we end up chasing the target and we get a highly oscillated training process. For overcoming this problem we use two models [9]. One is for the agent, the other is for the online working model. Instead of update the weight on each run, for example we calculate them on every 50 runs. This way we updated much less often the target model than the online model, as an outcome our Q-Value targets are more stable.

DOUBLE DQN

There is also a method called Double DQN [15]. In Double DQN the solution involves using two separate Q-value estimators, each of which is used to update the other. Using these independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator. We can thus avoid maximization bias by disentangling our updates from biased estimates. On the other hand, the Target network is responsible for the evaluation of that action.

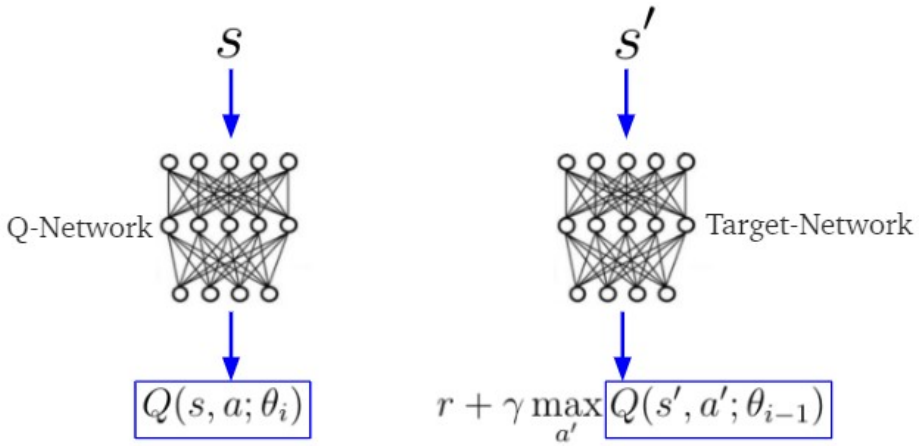

Figure 2-11: Double DQN [14]

Q-network selects the best action with the best Q-value of the next state [15]. Target Network calculates the estimated Q-value with action that selected. Update the Q-value of Deep Q Network based on the estimated Q-value from Target Network. Update the parameters of Target Network based on the parameters of Deep Q Network per several iterations. Update the parameters of Deep Q Network based on defined optimizer function.

CHAPTER THREE
PROPOSED METHOD

The Focused layer implementation allows people to test the datasets by trained Focused layer. It trained faster than the Dense layer and comparable accuracy rate to Dense. Focusing neurons can generate unique connection maps for a problem. The new model uses no other tool than the back-propagation algorithm to learn its focus parameters which control the receptive field locations and apertures. So neurons can learn and adapts itself.

In this project, we replace the dense layers in deep q learning with focused neurons. For each environment observation space taken for input size and action space taken as output size.

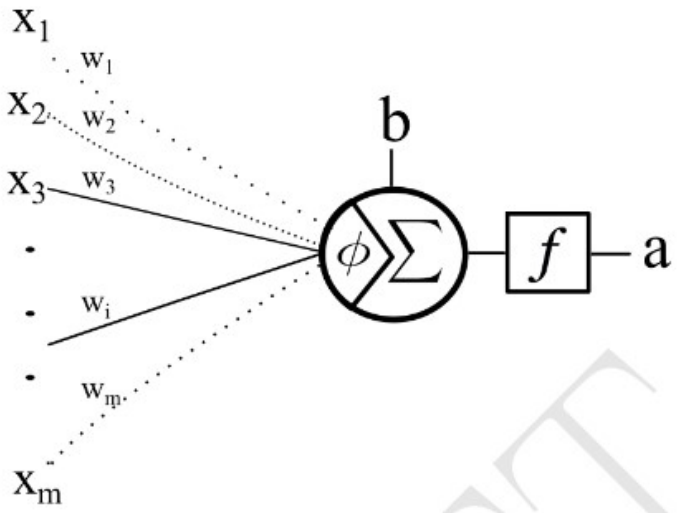

Figure 3-1: The focusing neuron model[1]

In Figure 3-1, the focus attachment φ allows the neuron to change its receptive field and adjust the aperture size[1].

In our tests, we use three different gym-environment to create train-test data. we run both Focused layer and Dense layers and take several graphs, such as loss and reward, we create score graphs for each Focused layer, Dense runs, also search for a trend. While training the environments on building model block environments creates 10K to 130K dimensional data and we run 600 iterations for each, not all agents show ascending trend.

One way is playing with the functions defined(loss, activation functions) and changing neuron count for each layer, but it might drive you to an endless road because it's like a BlackBox, you can not be sure the result and values you have given are right for each environment and agent. We will try to find out which parameters are best.

CHAPTER FOUR
IMPLEMENTATION DETAILS

In this project, we replace the dense layers in deep q learning with focused neurons. We will construct deep q-networks of focusing neurons for the following problems:

1) **CartPole:**

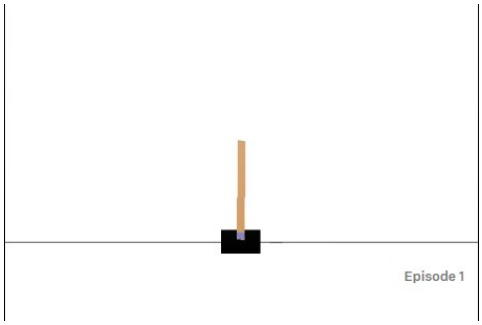

Figure 4-1: CartPole-v1[16]

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track[16]. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.

There are two actions taken, right and left moves, respect to these observation points is called Box() observation in the OpenAI Gym, and they are not discrete; Cart Position, Cart Velocity, Pole Angle, Pole Velocity At Tip

- Input(4)-Neuron(32)-Neuron(32)-Output layer(2)

2) **Acrobot:**

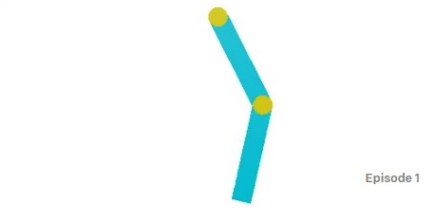

Figure 4-2: Acrobot-v1[17]

Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards[17]. The goal is to swing the end-effector at a height at least the length of one link above the base.

The observation space is a numpy array in this environment. State calculation, [cos(theta1) sin(theta1) cos(theta2) sin(theta2) thetaDot1 thetaDot2]

The action is either applying +1, 0 or -1 torque on the joint between the two pendulum links.

- Input(6)-Neuron(32)-Neuron(32)-Output layer(3)

3) MountainCar:

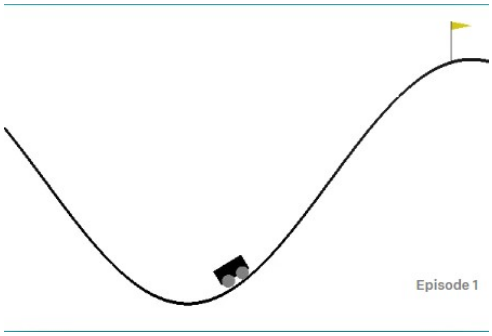

Figure 4-3: MountainCarContinuous-v0[18]

A car is on a one-dimensional track, positioned between two mountains. The goal is to drive up the mountain[18].

The observation space is a two dimensions that contains position and velocity.

The action space is push left, right, and no push. The action space is takes floating values between +1 and -1. It is not discrete so we need to discretize the action space to 10.

- Input(2)-Neuron(32)-Neuron(32)-Output layer(10)

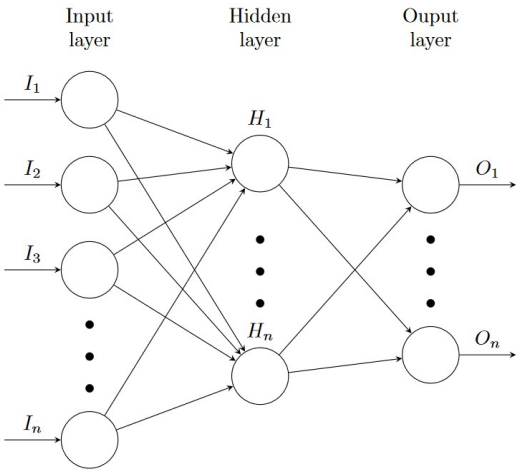

Figure 4-4: Neural network representation

For each environment observation space taken for input size and action space taken as output size.

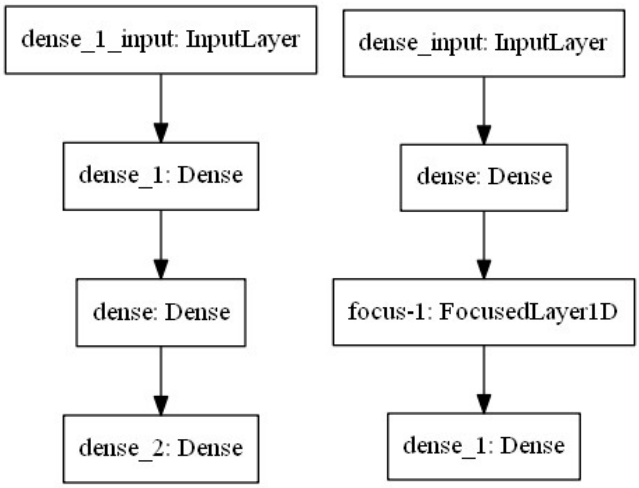

Figure 4-5: Neural network layers

CHAPTER FIVE
TESTS, AND EXPERIMENTS

In this part we will discuss the train results for each environment. As sigma value we use .25, .2, .1 and .05 to evaluate the results.

1) CartPole:

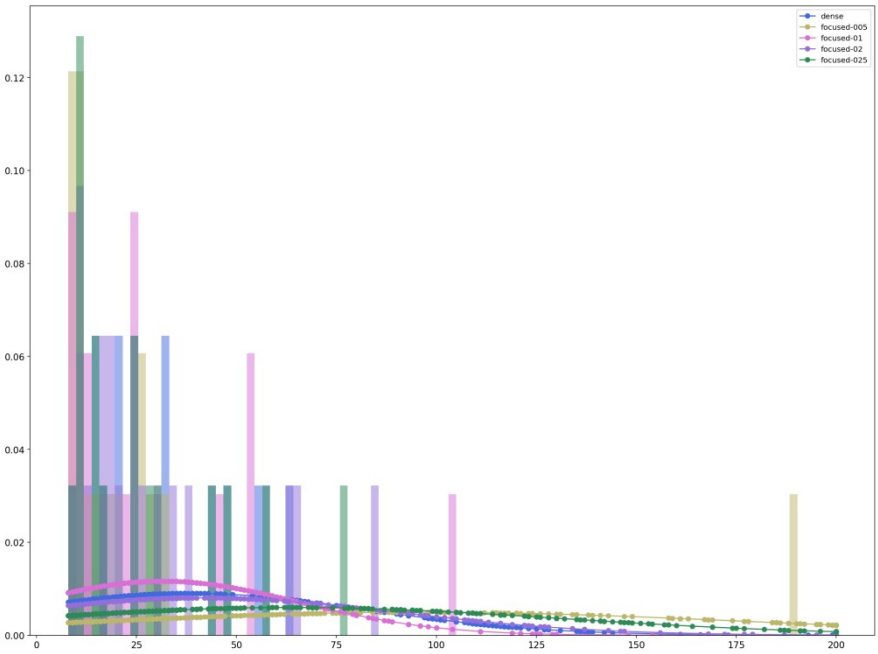

Figure 5-1: CartPole dense and focused score value distributions

In Figure 5-1 representation we see the score distributions of the CartPole problem. Also we measure the score curve area to decide which sigma value is better for solving this problem.

Neuron Type	Sigma	Areas	Mean	Standard Deviation
Dense	-	22990.0	38.490	44.217
Focused	0.05	57696.0	96.333	80.598
Focused	0.1	18919.0	31.705	34.317
Focused	0.2	24641.0	41.241	49.684
Focused	0.25	38196.0	63.833	66.866

Table 5-1: CartPole results curve areas

According Table 5-1, results we decided to sigma value .05 is better to solve this problem. Lets also see the score trend comparison with dense neuron of focused neuron with sigma .05.

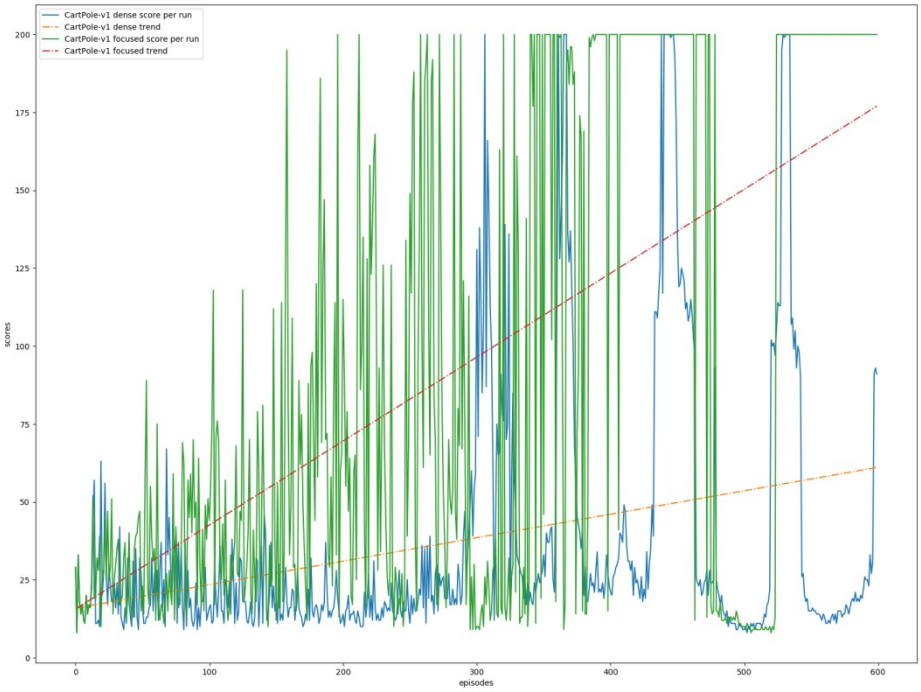

Figure 5-2: CartPole score trend of focused(sigma .05) and dense

In Figure 5-2 as we can see the focused neuron with sigma 0.05 has learned the problem more faster and works more efficient than dense neuron.

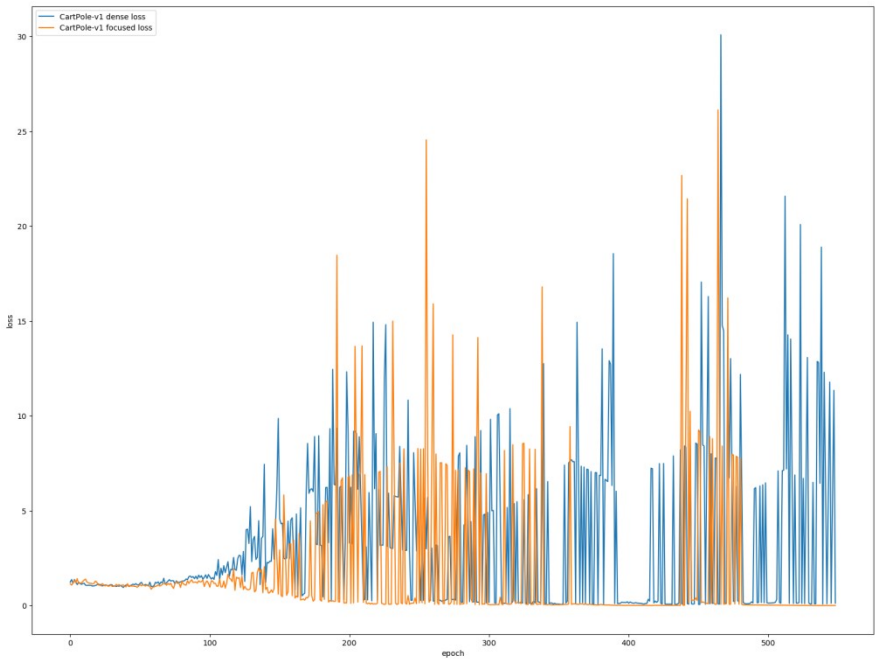

Figure 5-3: CartPole score trend of focused(sigma .05) and dense

In Figure 5-3, we compare the loss graphics of both focused and dense neurons the area of dense 1797.24 and area of focused: 921.12. In here more the area is less means the neuron has learned the problem more efficient and accurate.

We can observe the values are oscillating in the loss graphic in figure 5-3. It is because, in DQN during the q value calculation, we calculate our target q-value with the random samples we gathered before, afterwards we calculate the next q- value and compare those two q-values. That is why we see an oscillating graphic.

Lets also put our results into t-test to see whether our hypothesis is true or not. Our t-test results are follows:

- statistic: -15.399402902553792
- pvalue: 5.8536936390293696e-49

Here we can see the p-value is less than 0.05. Means that our hypothesis is true, and focused neuron type solved the problem more accurate.

Lets apply the same test to other enthronements.

2) MountainCar:

Neuron Type	**Sigma**	**Areas**	**Mean**	**Standard Deviation**
Dense	-	-8847.269	-14.765	2.989
Focused	0.05	-8969.718	-14.974	2.372
Focused	0.1	-7147.111	-11.930	5.456
Focused	0.2	-8194.177	-13.678	3.475
Focused	0.25	-9165.748	-15.294	2.794

Table 5-2: MountainCar results curve areas

In Table 5-2, there is a special case. In this environment penalty is higher for not reaching the goal. The car between two mountain, makes an oscillating movement to reach the goal, thus until the car reaches the goal point, it takes lots of penalty. So all scores are below zero. So we take a the value closest the zero as the best. In this case it is the focused with sigma 0.1.

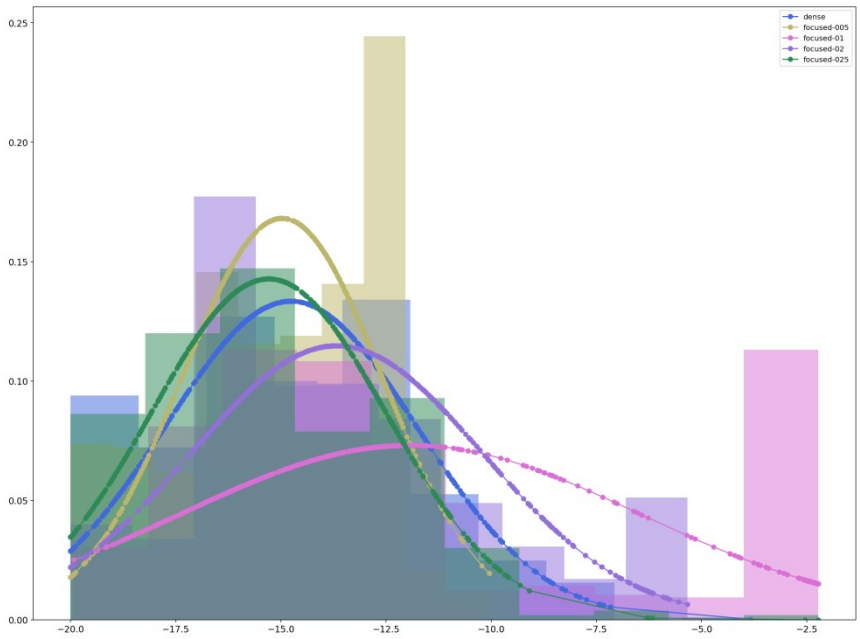

Figure 5-4: MountainCar dense and focused score value distributions

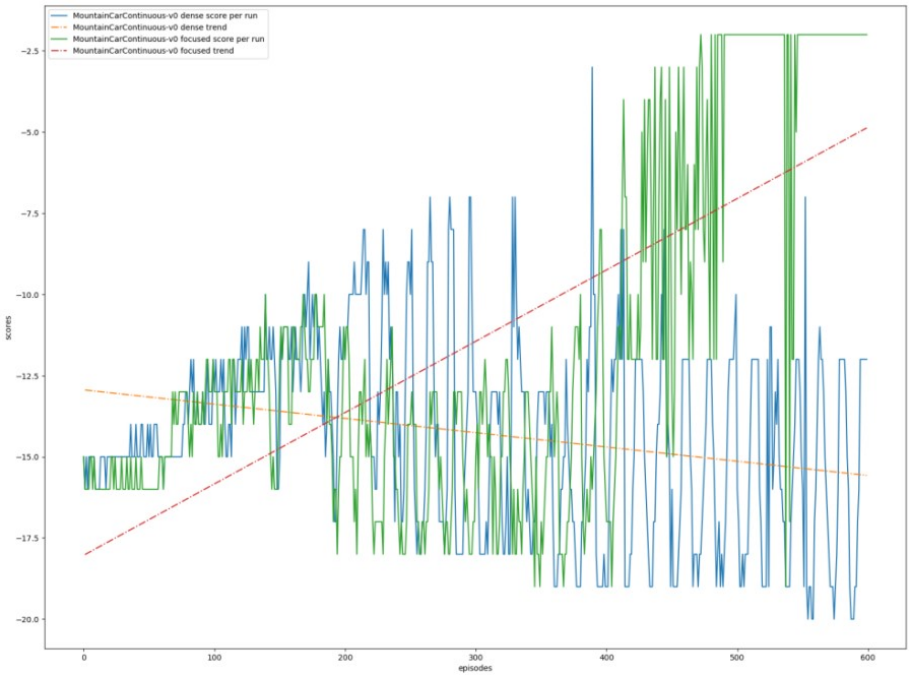

Figure 5-5: MountainCar score trend of focused(sigma .1) and dense

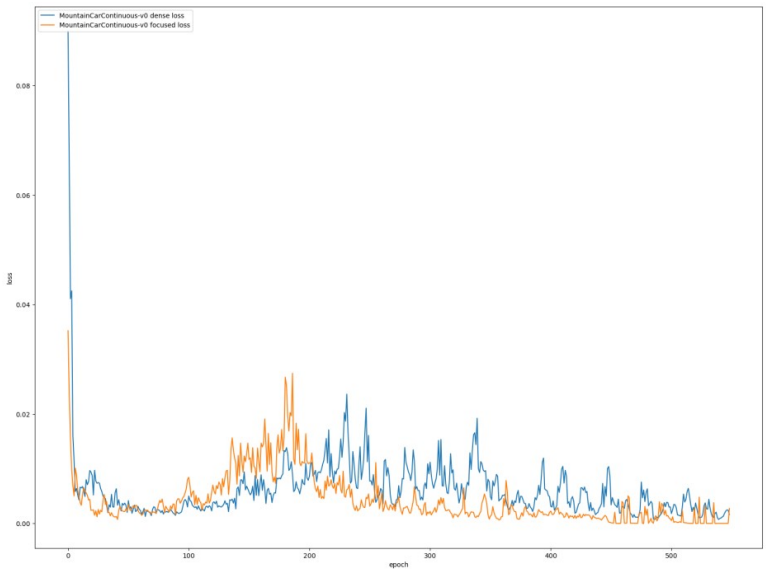

Figure 5-6: MountainCar score trend of focused(sigma .1) and dense

loss areas of focused(for sigma 0.1)

- Area of Dense: -8847.269245070718
- Area of Focused: -7147.111283395016

Again we can observe here loss of focused with sigma 0.1 is better than dense neuron type. We also apply t-test on the results:

- statistic: -11.151027489360999
- pvalue: 1.5226142272798123e-27

In the t-test results, we can see the p-value is less than 0.05. Means that our hypothesis is true, and focused neuron type solved the problem more accurate.

3) Acrobot:

Neuron Type	Sigma	Areas	Mean	Standard Deviation
Dense	-	-117293.5	-195.723	15.218
Focused	0.05	-113961.0	-190.171	23.654
Focused	0.1	-107789.0	-179.866	37.103
Focused	0.2	-102255.0	-170.643	43.652
Focused	0.25	-113158.0	-188.825	27.638

Table 5-3: Acrobot results curve areas

In the Table 5-3 as we can observe, the same special case is valid for this environment. Again we take a the value closest the zero as the best. In this case it is the focused with sigma 0.2.

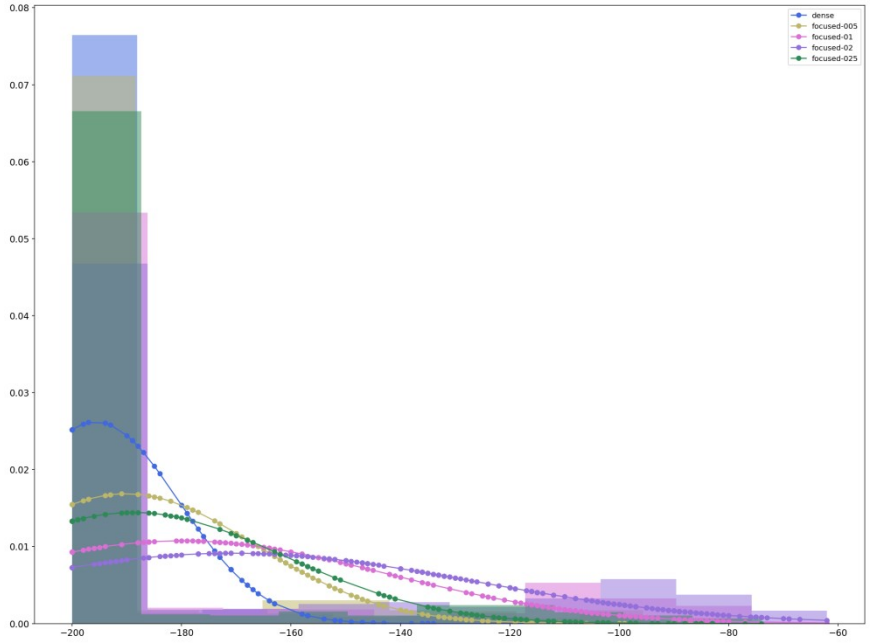

Figure 5-7: Acrobot dense and focused score value distributions

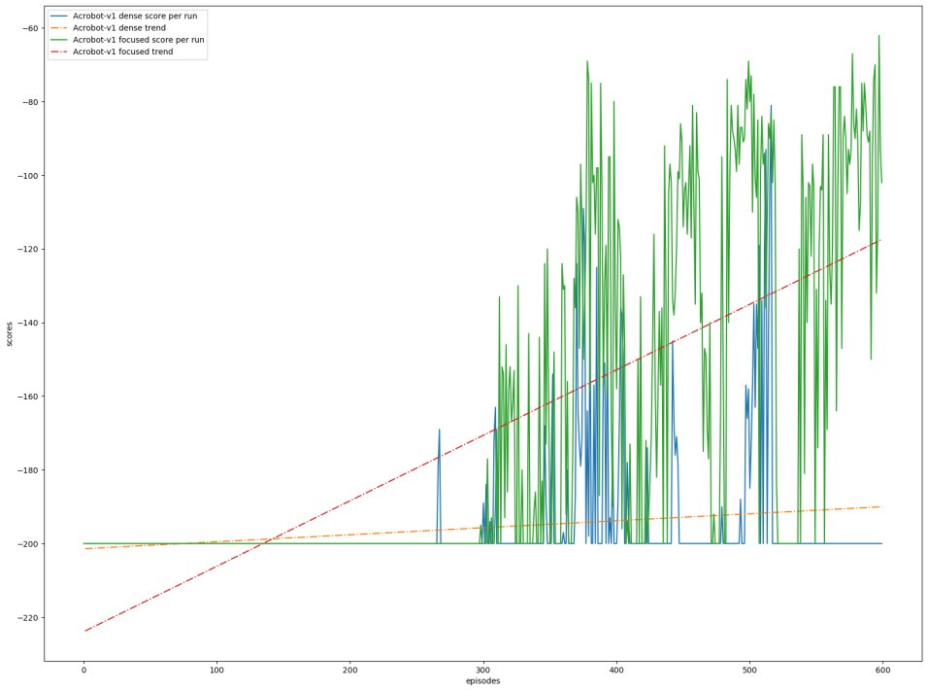

Figure 5-8: Acrobot score trend of focused(sigma .2) and dense

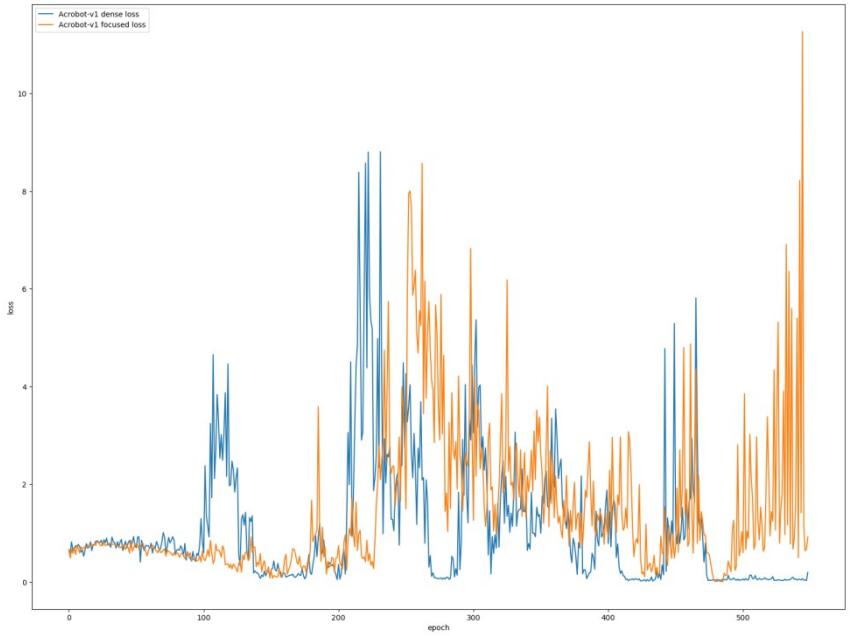

Figure 5-9: Acrobot score trend of focused(sigma .2) and dense

loss areas of focused(for sigma 0.2)

- Area of dense: 614.7205757629126
- Area of focused: 815.2219126047567

Again we can observe here loss of focused with sigma 0.2 is not better than dense neuron type. But for being sure lets apply t-test to see the results:

- statistic: -13.277667545492406
- pvalue: 1.2347315664584795e-37

In the t-test results, we can see the p-value is less than 0.05. Means that our hypothesis is true, and focused neuron type solved the problem more accurate.

After training phase, we have save the trained models and started the test phase. In test phase we run hundred episode and take record for each environment. Now lets examine the test results for each environment. On each environment we make comparisons dense test vs focused test. In this part we will see those comparisons and statistical results.

1) CartPole:

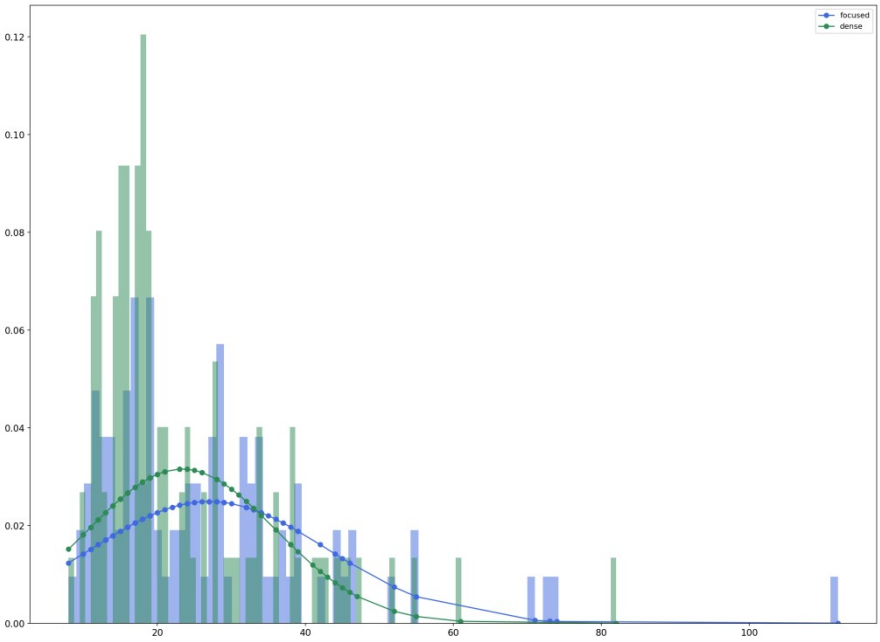

Figure 5-10: CartPole-v1 dense focused test distribution

In figure 5-10, dense test mean of score is 23.33 and standard deviation is 12.64, for focused test mean of score is 27.0 and standard deviation is 16.02. Lets see also t-test results for the training results.

- statistic: -1.7885389189084273
- pvalue: 0.07521799109053397

The p-value is over 0.05 according to t-test, so we cannot reject the null hypothesis.

2) MountainCar:

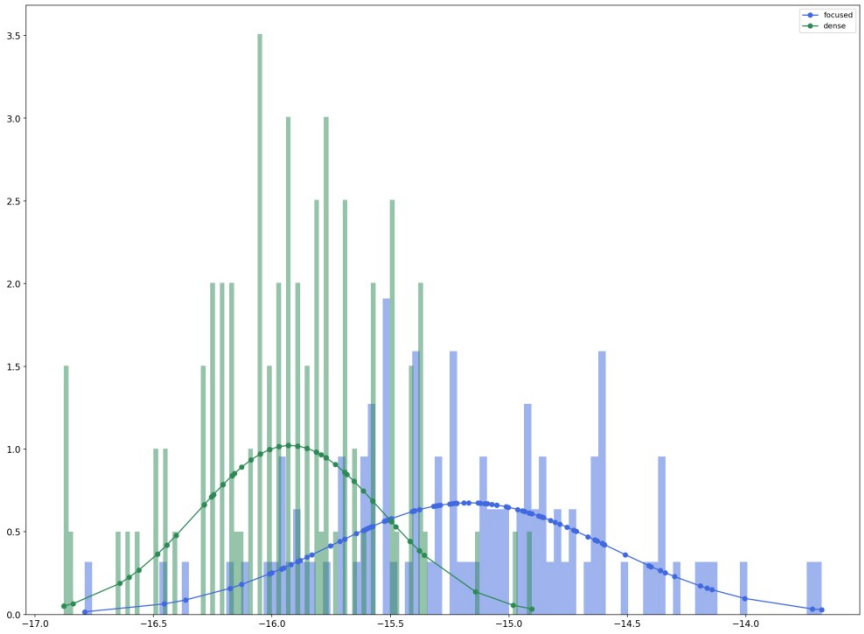

Figure 5-10: MountainCarContinuous-v0 dense focused test distribution

In figure 5-11, dense test mean of score is -15.92 and standard deviation is 0.40, for focused test mean of score is -15.17 and standard deviation is 0.59. Lets see also t-test results for the training results.

- statistic: -10.589982583360232
- pvalue: 4.728857463367546e-21

The p-value is under 0.05 according to t-test, so we can accept that focused test results are better than dense test results.

3) Acrobot:

In this environment training results were good, in test phase the agent could not show any improvements. If we examine the test data, dense test mean is -200.0, standard deviation is 0.0 and for focused test mean -200.0, standard deviation is also 0.0.

This might occur the random initialization of the environment. If we check the figure 5-8 score graphics we can see that the agent couldn't do anything until the 300th episode.

CHAPTER SIX
CONCLUSIONS AND FUTURE WORK

In this thesis, we cover the DQN model training and comparison for Dense and Focused neuron type. One of the main objectives is proving the focused neuron is working better than a dense layer.

The most important thing was tuning and deciding the right parameters, μ is the main coefficients of focusing structure. For making sure the focusing layer working right or not, a way to evaluate is making loss comparison. By taking loss values on every q-value calculation and try to estimate according to those values, the same evaluation can be run.

For being sure of the values on each μ, we did experiments for each setting we did experiments 5 times so that we could sure the results are stable or not. Then we recognize that we need to have different weight initializer because our layer model has different parameters we need to initialize weights differently. As we can see the final results on the 5th chapter. Our score, loss, and test results are better than the dense neuron. Another thing is time efficiency, Focused layer model is working faster than Dense Layer model.

Many different adaptations, tests, and experiments have been left for the future due to lack of time. Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity.

It would be very interesting to use an image as an observation space. The logic is here, the behavior of the focusing layer requires too many values. In our test, the environments have four observations and it is very limited and unnecessary to use the focusing layer as the input layer. Thus we use dense layers as the input layers. But if we use an image on a given time as an input, our observation will be larger and we can use the focusing layer as the input layer.

REFERENCES

1. Tek, F. Boray \An Adaptive Locally Connected Neuron Model: Focusing Neuron." ArXiv.org, 31 July 2019, https://arxiv.org/abs/1809.09533.
2. Online:https://github.com/btekgit/FocusingNeuron-Keras, date accessed: Feb 2020
3. Géron, Aurélien. “Neural Networks and Deep Learning.” Mar. 2018, https://learning.oreilly.com/library/view/neural-networks-and/9781492037354/.
4. McCulloch, Warren S., and Walter Pitts. ”A logical calculus of the ideas immanent in nervous activity.” The bulletin of mathematical biophysics 5.4 (1943): 115-133.
5. Rosenblatt, Frank. ”The perceptron: a probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.
6. VanderPlas, Jake. Python data science handbook: essential tools for working with data. ” O’Reilly Media, Inc.”, 2016.
7. Farhadi, Farnoush. *Learning Activation Functions in Deep Neural Networks*, ProQuest Dissertations Publishing, 2017.
8. Online:https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/, date accessed: May, 2020
9. Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2017.
10. Online:https://www.ibm.com/developerworks/ssa/library/cc-reinforcement-learning-train-software-agent/index.html, date accessed: May 2020
11. Online:https://ithelp.ithome.com.tw/articles/10228127, date accessed: May 2020
12. Online:https://miro.medium.com/max/2000/1*BgOahnFLqAbYl4_eIUBTLg.jpeg, date accessed: May 2020
13. Online:https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/, date accessed: May 2020
14. Online:https://devhunteryz.wordpress.com/2019/02/10/kendi-kendine-ogrenen-yz-ajanlari-bolum-ii-derin-q-ogrenme/amp/, date accessed: May 2020
15. Hasselt, Hado van, Guez, Arthur, and Silver, David “Deep Reinforcement Learning with Double Q-learning.” ArXiv.org, 22 Sep 2015, https://arxiv.org/abs/1509.06461
16. Online:https://gym.openai.com/envs/CartPole-v1/, date accessed: Jan 2020
17. Online:https://gym.openai.com/envs/Acrobot-v1/, date accessed: Jan 2020
18. Online:https://gym.openai.com/envs/MountainCarContinuous-v0/, date accessed: Jan 2020