# Experiment Setup

To reach the goals set out in the project some design decisions we have to make some design decisions. First, we have to define the different games that will be tested with the different deep reinforcement learning methods, the programming languages and packages that will be used. From this, we can derive our implementation of the different reinforcement learning methods, and run the experiments on the different games.

In the beginning phase of the project, we concluded that emulating a Doom environment, which was the original scope of the project, was too complex to get working. Because of this, the project needed to be simplified. This resulted in focusing on two simple environments distributed by OpenAI gym and applying the different reinforcement learning on these environments. We hypothesized that the results of adding and/or removing layers and neurons will become more apparent in simpler environments. To confirm this hypothesis the following games were used.

## CartPole

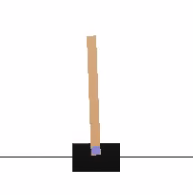


Figure 1. Cartpole

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every time step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. This in turn leads to the cart going off-screen. This problem defines "solving" as getting an average reward of 195.0 over 100 consecutive episodes [1].

* + 1. **Architecture of the CartPole Experiment**

The CartPole experiment will be performed as follows. The structure of the neural network to create the features and perform actions based on the visual input will look like this in the most basic form.

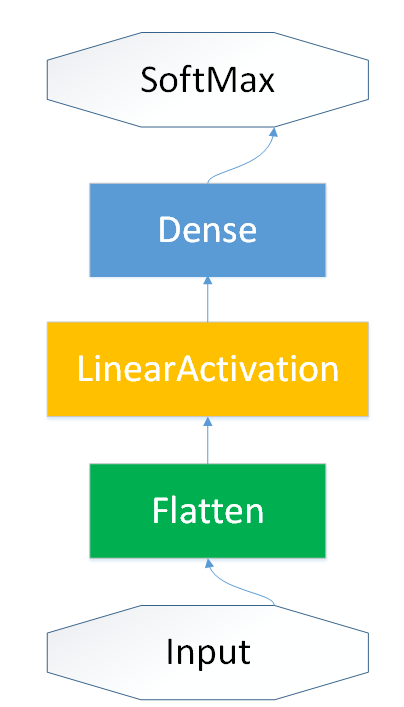


Figure 2. Neural Network architecture Basic Form CartPole Experiment

The visual input of the CartPole environment will first be flattened and will go through a Dense layer of 16 neurons with a Linear activation function. This will then result in a SoftMax function and this will be imported into either the CEM agent[3] or a dueling DQN agent [4].

The construction of the CEM agent will be as followed. It will incorporate the neural network model and actions from both our creation and OpenAI gym. It will have a warm up session of 2000 steps. This is done to make training run as smooth as possible. The training run will be 100.000 steps; this will make it possible for each architecture to converge to the ideal score. Lowering this amount to 10.000 or even 1.000 will greatly hinder the score and will give a biased view of the architectures.

The construction of the Dueling DQN agent is as followed. It will also incorporate the model and actions based on our input. However, in comparison to the CEM agent, the dueling method will only have 1.000-step warmup. This is done because the dueling architecture will try to increase its score based on the dueling network. With this, the focus will be more on increasing the score during the training time than during the warmup. The average score will be calculated between the two dueling agents. A possible alternative would have been using the Max score. But because of the experiment the exceeded score value will be 200. Therefore, this will give a biased representation of the architecture. Because of this, the average will be calculated. The training time for the DQN agent will also be 100.000.

For the experiment the following actions will be performed.

* + 1. **Execution of the CartPole Experiment**

The first section of the experiment will focus on three different activation functions: sigmoid[5], linear and relu[6]. These will be tested on both the CEM and DQN agent from layer one until layer ten. By slightly increasing the increment of the layers both the time, mean score and standard deviation can be controlled as to achieve the ideal setup. The scores will be based on rerunning each specific architecture five times and grabbing ten random episodes from each run. This will result in an unbiased score that will be compared with the other combinations in that specific architecture and agent.

The next part consists of selecting the best three architectures per agent and increasing the neurons of these layers by doubling the amount per increment, this goes from 16 until 256. Sixteen forms a good lower bound for testing out the different layers while 256 is a great stopping point. This will prevent bloated training times for intensive architectures with for example, ten layers.

The last part of this experiment will grab the overall best scoring method per agent and adds a dropout layer to the neural network while having only 16 neurons, as to prevent a bloated training time. By adding a dropout layer, a possible overfitting can be prevented.

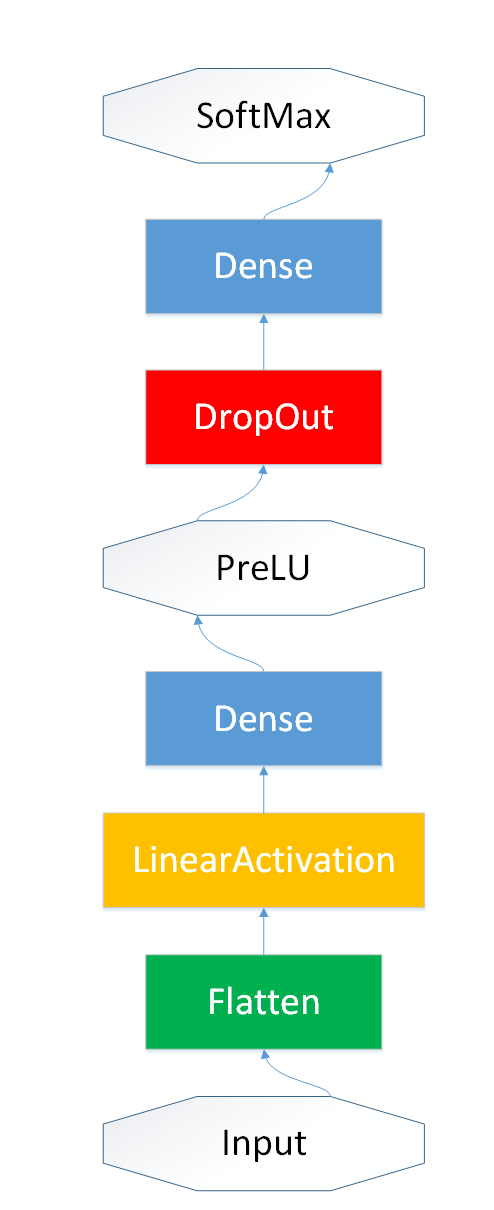


Figure 3. Neural Network architecture Last architecture CartPole Experiment

Afterwards the results will be compared with each other to see which setup works the best and how this differs with the agents and with the next experiment.

## Pendulum

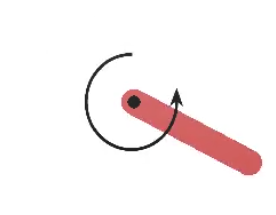


Figure 4. Inverted Pendulum

This environment is based on the inverted pendulum swing up problem. The pendulum starts in a random position, and the goal is to swing it up so it stays upright. This problem is currently in an unsolved environment, which means it does not have a specified reward threshold at which the problem is considered solved. This means that an N amount of episodes can be used to achieve the highest possible score [2].

* + 1. **Architecture of the CartPole Experiment**

The Pendulum experiment will be performed in a slightly different way than the CartPole experiment. The neural network will still be used to create the features and perform actions based on the visual input. However, the difference between the CartPole experiment are the last activation function layer and the fact that a second neural network will be used. In the CartPole experiment the SoftMax activation function was used, but in the Pendulum experiment a Linear activation function was used. This method was picked as to give a clearer representation of the architectures with a different activation function than SoftMax.

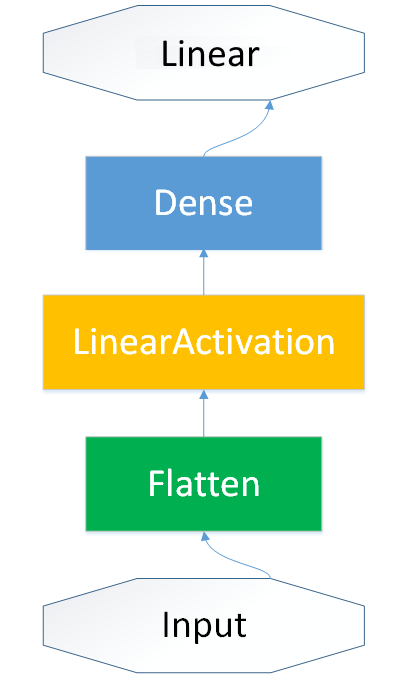


Figure 5. Neural Network architecture Basic Form Pendulum Experiment

The second neural network will not be changed in the experiment, because if merges the action input and observation into one model. The reason why this is not changed because the focus of this experiment was on focusing on the action input from the different architectures and how they perform and not the influence they can have on the observation space. The structure consists of a flatten layer and three Dense layers with a relu activation function with 32 neurons. The last layer consists of a 1-neuron layer that results in a linear activation function. Afterwards this forms the secondary input for the architecture.

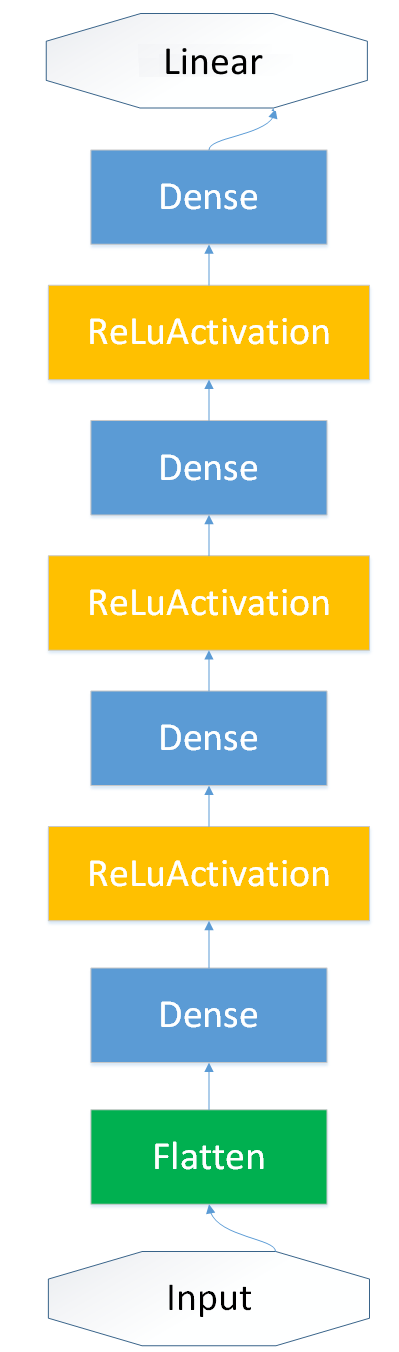


Figure 6. Neural Network architecture Action and Observation Merger Pendulum Experiment

The agent for this experiment will be the DDPG[[1]](#footnote-1) agent [7]. Both the CEM and DQN agent were unable to incorporate the OrnsteinUhlenbeck process into their process. Because of this a different agent needed to be used.

The construction of the DDP agent is as follows. It will incorporate the two neural networks and the necessary actions. Afterwards a warmup will be performed for both the basic form neural network as the secondary input network. This is done so that assembling the two different neural networks in the agent will run smoother. The amount of training steps will be 5.000, to prevent a bloated training time. However, it must be said that increasing the training time will increase the score by a wide margin. Nevertheless, for this experiment all the training will be done with 5.000 steps.

For the experiment the following actions will be performed.

* + 1. **Execution of the CartPole Experiment**

The first section of the experiment will focus on three different activation functions: sigmoid[5], linear and relu[6]. These will be tested on just the DDPG agent from layer one until layer ten. By slightly increasing the increment of the layers both the time, mean score and standard deviation can be controlled as to achieve the ideal setup. The scores will be based on rerunning each specific architecture five times and grabbing ten random episodes from each run. This will result in an unbiased score that will be compared with the other combinations in that specific architecture and agent.

The last part consists of selecting the best three architectures per agent and increasing the neurons of these layers by doubling the amount per increment, this goes from 16 until 256. Sixteen forms a good lower bound for testing out the different layers while 256 is a great stopping point. This will prevent bloated training times for intensive architectures with for example, ten layers. There will be no dropout section in this experiment. Because of the fact that this experiment currently has no solution and there is no max score like in the CartPole experiment. So, removing possible overfitting will be trivial in this situation because of this.

Afterwards the results will be compared with each other to see which setup works the best and how this differs with the previous experiment.

## Implementation

These games have been implemented using the Python language. The implementation is heavily built on the different environments written by OpenAI Gym. All these environments are built in such a way to they can run without a human player or input. Whenever the game needs a decision or action from the "player", it will propagate the input through the neural network. Afterwards an agent will be created with a LSTM and the characteristics of that specific deep reinforcement learning method. This will result in an animation that shows the N best episodes of that run.

OpenAI Gym shared some examples of different methods in their documentations. These will be used as the benchmark for our own variation of their examples to control the reward system by adding or removing layers and neurons from their neural network.

# Experiment Results

In this section, the results of the experiments will be reported as to show the effects of the different deep reinforcement learning methods on the two different environments that were mentioned in the experiment setup chapter. The reinforcement methods that will be used in this experiment will be different depending on the chosen game. Because of the fact that one method will be completely unfit for one game, but will be perfect for another game. The resulting methods will be compared per game and overall as to check which method performs the best overall with the ideal amount of layers. The reason why these methods are used is their easy-implementation and changeability of activation methods. This makes comparing and improving the results very easy. For this experiment, the training time will be expressed in seconds.

## CartPole Experiment

The first part of the experiment focuses on the effects of the activation functions and the amount of layers. This resulted in the following mean scores for the CEM agent. The maximum score of this experiment will be 200, so when a mean score of 200 is reached it can be seen as a ‘perfect’ architecture in this experiment.

* + 1. **CEM Layer Test**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Activation Function | | | | | | |
| Mean Score | ReLu | Training Time | Sigmoid | Training Time | Linear | Training Time |
| 1 | 129,6 | 20,226 | 98,9 | 19,587 | 123,4 | 17,231 |
| 2 | 176,6 | 19,652 | 73,2 | 22,047 | 133,4 | 18,534 |
| 3 | **164,4** | 19,899 | 81,6 | 23,578 | 164,1 | 19,344 |
| 4 | 142,6 | 22,914 | 9,5 | 24,732 | 182,4 | 21,031 |
| 5 | 109,3 | 23,432 | 9,5 | 25,232 | 189,3 | 22,983 |
| 6 | 77,9 | 25,327 | 9,1 | 25,823 | 162,6 | 23,653 |
| 7 | 94,3 | 24,973 | 9,4 | 26,321 | 189,9 | 23,843 |
| 8 | 61,3 | 26,483 | 9,3 | 27,164 | 175,5 | 24,721 |
| 9 | 169,9 | 28,293 | 9,2 | 28,432 | 158 | 25,926 |
| 10 | 62,2 | 28,765 | 9 | 29,568 | 169,8 | 28,322 |
| Average | 118,81 | 23,996 | 31,87 | 25,248 | 164,84 | 22,559 |

Table 1. CEM Agent Activation Function Mean Score and training time Results

The benchmark, score according to the example script, in this section is a three layered ReLu activation function. Based on the table it is clear that the linear activation function scores the best on average, while Sigmoid scores the worst on both average and specific layers. ReLu does score well on some specific layers but actually loses it on some layers to the linear function.

When it comes to training time, the scores are somewhat similar. Sigmoid is marginally slower when compared to Relu and Linear but the difference is trivial when compared to the mean scores. The training time for Relu and Linear are interchangeable at some points.

The next step is focusing on the reliability of a specific architecture the standard deviation was used to calculate the validity of the layers.

|  |  |  |  |
| --- | --- | --- | --- |
| Standard deviation | ReLu | Sigmoid | Linear |
| 1 | 26,492 | 75,180 | 63,574 |
| 2 | 40,426 | 11,007 | 60,432 |
| 3 | 22,005 | 17,794 | 21,342 |
| 4 | 53,894 | 0,500 | 12,666 |
| 5 | 46,673 | 0,500 | 18,401 |
| 6 | 30,700 | 0,700 | 40,212 |
| 7 | 56,736 | 0,490 | 16,224 |
| 8 | 20,519 | 0,458 | 25,738 |
| 9 | 38,263 | 0,539 | 38,387 |
| 10 | 33,528 | 0,632 | 30,127 |
| Average | 36,924 | 10,780 | 32,710 |

Table 2. CEM Agent Activation Function Standard Deviation results

The most interesting result from this figure is that both Relu and Linear had quite high standard deviations through the different layers. While sigmoid actually dropped to an almost zero after layer 3. This showed that even though Relu and Linear scored better, they are much less reliable to get the same result repeatedly when compared to the sigmoid function.

* + 1. **CEM Neuron and Dropout Test**

For the next part, the three best mean scores of the CEM agent will be compared with each other. This is solely based on scores and standard deviation. As to control the reliability and performance of these models by adding neurons to each layer. Even though, linear has a better score overall. For this section, the best Relu architecture will be used as prevent a positive bias to the linear activation function.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Relu | 2 | 176,6 | 181,3 | 191,8 | 100,8 | 170,2 | 164,14 |
| Linear | 5 | 189,3 | 197,5 | 198,7 | 188,1 | 122,5 | 171,95 |
| Linear | 7 | 189,9 | 168,2 | 175,8 | 130,7 | 166,2 | 166,16 |

Table 3. CEM Mean Scores purely based on neurons

The results had big margins when only based on the neurons. For example, 100.8 for Relu with 128 neurons while linear with five layers got a 188.1 mean score. However, the most interesting part is the small margin in the average scores. Only a 7-point difference between the three different architectures. This resulted in the need to calculate the standard deviation for both these models as to validate their performance better.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Relu | 2 | 40,426 | 18,900 | 8,875 | 41,073 | 45,766 | 31,008 |
| Linear | 5 | 18,401 | 5,123 | 3,288 | 15,162 | 44,709 | 17,337 |
| Linear | 7 | 16,224 | 38,856 | 26,045 | 69,618 | 29,832 | 36,115 |

Table 4. CEM Standard deviation Scores purely based on neurons

The results of the standard deviation are somewhat similar to the basic layers, where both ReLu and linear shared a high standard deviation on some layers. However, the average standard deviation of the Linear 5 layers is the lowest and will be tested for overfitting by adding a dropout layer. The Dropout layer will be added to a 16 neuron 5 layer Linear variant.

|  |  |  |
| --- | --- | --- |
|  | Linear 5 Layers No DropOut | Linear 5 Layer DropOut |
| Mean Score | 189,3 | 135,6 |
| Standard Deviation | 18,401 | 58,315 |

Table 5. CEM Agent Checking for overfitting

Adding the dropout layer shows that the architecture does overfit for both the mean score and the reliability of the model. This will be taken into account when comparing this score with the DQN agent.

* + 1. **DQN Layer Test**

The DQN layer test will be performed the same way as the CEM agent test. The scores are as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Activation Function | | | | | | |
| Mean Score | ReLu | Training Time | Sigmoid | Training Time | Linear | Training Time |
| 1 | 199,7 | 123,325 | 198 | 133,514 | 8,6 | 135,678 |
| 2 | 195,8 | 133,459 | 137,8 | 149,383 | 11,3 | 141,637 |
| 3 | **190,6** | 142,453 | 147,3 | 162,883 | 9,2 | 149,099 |
| 4 | 170,3 | 151,863 | 47,3 | 173,522 | 9 | 156,827 |
| 5 | 158,9 | 159,732 | 108,5 | 195,743 | 9,3 | 169,139 |
| 6 | 180,9 | 168,384 | 9,3 | 208,412 | 9,4 | 176,995 |
| 7 | 193,4 | 183,432 | 112,9 | 227,002 | 8,9 | 186,488 |
| 8 | 188,5 | 201,282 | 50,2 | 256,843 | 9,6 | 189,775 |
| 9 | 160,4 | 199,637 | 9 | 275,118 | 9,6 | 199,013 |
| 10 | 199,2 | 206,954 | 9,4 | 293,954 | 9,5 | 206,912 |
| Average | 183,87 | 167,052 | 82,97 | 207,637 | 9,44 | 171,563 |

Table 6. DQN Agent Activation Function Mean Score and training time Results

The benchmark, score according to the example script, in this section is a three layered ReLu activation function. Based on the table it is clear that the ReLu activation function scores the best on average, while Linear in this example scores the worst on both average and specific layers. The roles of Sigmoid and Linear are switched when compared to the CEM agent.

When it comes to training time, it takes ten times longer than the CEM agent. However, this could mean a more reliable architecture when the standard deviation is calculated.

|  |  |  |  |
| --- | --- | --- | --- |
| Standard deviation | ReLu | Sigmoid | Linear |
| 1 | 0,322 | 4,000 | 0,490 |
| 2 | 11,948 | 24,951 | 2,002 |
| 3 | 17,957 | 5,255 | 0,872 |
| 4 | 23,786 | 76,351 | 0,894 |
| 5 | 32,745 | 35,109 | 0,640 |
| 6 | 23,590 | 0,458 | 0,663 |
| 7 | 9,436 | 20,181 | 0,700 |
| 8 | 14,596 | 62,985 | 1,020 |
| 9 | 20,742 | 0,894 | 0,800 |
| 10 | 0,458 | 0,490 | 0,922 |
| Average | 15,558 | 23,067 | 0,900 |

Table 7. DQN Agent activation function Standard Deviation Results

The DQN shows a more consistent standard deviation score for both Relu and Linear. For the former it barely exceeded the 30 mark, while the linear activation function barely exceeds a standard deviation of one. However, the sigmoid activation function shows some peculiar behavior. It can reach a low standard deviation but at some layers, it got a very high standard deviation.

* + 1. **DQN Neuron and Dropout Test**

For the next part, the three best mean scores of the DQN agent will be compared with each other with the same method as the CEM agent. The chosen networks for this are 1-layer and 10-layer Relu and a 1-layer sigmoid. As to not create a bias towards Relu.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Relu | 1 | 199,7 | 197 | 196,5 | 199 | 199,5 | 193,08 |
| Relu | 10 | 199,2 | 187,5 | 132,2 | 90,3 | 197,2 | 161,28 |
| Sigmoid | 1 | 198 | 199,5 | 198,2 | 195,6 | 179,7 | 194,20 |

Table 8. DQN Mean Scores purely based on neurons

Overall, the scores of these architectures are higher than the ones from the CEM agent test. Both on specific layers as well as the average scores. The only one that does not follow this trend is Relu 10-layers. This architecture shows a decline at 128 neurons while the other activation functions are behaving properly. A possible reason for this could be that the worst episodes were printed for five runs in a row. The small margin between both the 1-layer architectures resulted in a need to calculate the standard deviation.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Relu | 0,322 | 4,583 | 5,500 | 3,000 | 1,500 | 0,322 | 2,981 |
| Relu | 0,458 | 14,794 | 6,867 | 84,821 | 5,930 | 0,458 | 22,574 |
| Sigmoid | 4,000 | 0,806 | 2,786 | 5,426 | 44,066 | 4,000 | 11,417 |

Table 9. CEM Standard deviation Scores purely based on neurons

The standard deviation of these results are much lower when compared to the CEM agent results. This does show that the DQN agents are much more reliable in handling the architecture and maintaining the same results than the CEM agent. The last step is to control the Relu 1-layer for possible overfitting.

|  |  |  |
| --- | --- | --- |
|  | Relu 1 Layer No DropOut | Relu 1 Layer DropOut |
| Mean Score | 199,7 | 166,8 |
| Standard Deviation | 0,322 | 9,185 |

Table 10. DQN Agent Checking for overfitting

Just like with the CEM agent, the best scoring DQN agent architecture also overfits. However, the standard deviation is lower compared to CEM for both dropout as non-dropout, while the score is 30 points higher than the dropout variant. This does show that DQN will take longer in training but creates much more reliable scores than a CEM agent.

## Pendulum Experiment

The pendulum experiment will be performed in the same structure as the CartPole experiment, but in this situation the environment did not have a max score per episode and a solution in the current time. This does mean that in this environment there is no ‘perfect’ architecture. For this experiment, a DPPG agent will be used. This does mean that the comparisons between this experiment and the CartPole experiment will solely be based on the effects that neurons and layers bring forth.

* + 1. **DPPG Layer Test**

The DPPG layer test will be performed the same way as the previous two layer tests.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Activation Function | | | | | | |
| Mean Score | ReLu | Training time | Sigmoid | Training time | Linear | Training Time |
| 1 | -1530,2 | 86,128 | -1480,1 | 85,776 | -1445,4 | 85,793 |
| 2 | -1487,9 | 86,171 | -1453,3 | 86,129 | -1480,5 | 85,843 |
| 3 | **-1442,9** | 85,989 | -1480,3 | 86,024 | -1394,7 | 86,132 |
| 4 | -1484,8 | 85,853 | -1418,3 | 86,039 | -1474,1 | 86,187 |
| 5 | -1479,1 | 85,986 | -1520 | 86,051 | -1451,9 | 86,342 |
| 6 | -1421,3 | 86,432 | -1380 | 86,347 | -1431,7 | 86,398 |
| 7 | -1455,9 | 86,862 | -1447,1 | 86,515 | -1484,4 | 86,432 |
| 8 | -1436,9 | 86,342 | -1461 | 86,472 | -1397,2 | 86,445 |
| 9 | -1483,7 | 87,203 | -1342,5 | 86,746 | -1468,4 | 86,663 |
| 10 | -1466,2 | 87,109 | -1528,6 | 86,763 | -1458,3 | 86,701 |
| Average | -1469,9 | 86,408 | -1451,12 | 86,286 | -1448,7 | 862,963 |

Table 11. DPPG Agent Activation Function Mean Score and training time Results

The difference in training time in this experiment is negligible. Neither activation functions excel in training time. The same can also be said between the scores. On average, the mean scores are marginal. However, some great performances can be spotted for example a 9-layer sigmoid function. A possible reason for this could be that an episode reached a positive result. This lowered the mean score tremendously for this specific architecture.

This reason also becomes clear when the standard deviation was calculated.

|  |  |  |  |
| --- | --- | --- | --- |
| Standard deviation | ReLu | Sigmoid | Linear |
| 1 | 77,477 | 64,208 | 48,069 |
| 2 | 44,071 | 51,122 | 24,324 |
| 3 | 77,059 | 49,602 | 434,337 |
| 4 | 54,146 | 74,252 | 27,424 |
| 5 | 44,581 | 32,333 | 33,047 |
| 6 | 97,961 | 47,571 | 69,247 |
| 7 | 73,934 | 57,736 | 166,173 |
| 8 | 69,685 | 61,368 | 64,179 |
| 9 | 72,479 | 149,038 | 33,696 |
| 10 | 42,362 | 164,151 | 81,865 |
| Average | 65,367 | 75,138 | 98,326 |

Table 12. DPPG Agent activation function Standard Deviation Results

The standard deviations are in this experiment very shocking. Especially the linear 3-layer structure. This shows that these runs had a couple of almost positive results. However, it also shows that overall the architecture in and of itself is not reliable in its performance.

* + 1. **DPPG Neuron Test**

In this section, the neuron test will be performed on the three best layers of this experiment. For this experiment, even with the high standard deviation score. The linear 3-layer structure will still be used as to check if the mean score was a fluke.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Sigmoid | 6 | -1380 | -1460,2 | -1466,8 | -1494,4 | -1521,2 | -1464,52 |
| Linear | 3 | -1394,7 | -1395,6 | -1491,8 | -1404 | -1331,7 | -1403,56 |
| Linear | 8 | -1388,3 | -1469,3 | -1413,8 | -1319,6 | -1454,4 | -1409,08 |

Table 13. DPPG Mean Scores purely based on neurons

Both linear architectures scored admirably well. This does show that the linear architecture has a higher chance of getting an almost positive result or at least a very low negative score than the sigmoid activation function. This is clear in the average code, which shows that both linear structures have a very wide difference between the sigmoid structures.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Layers | 16 Neurons | 32 Neurons | 64 Neurons | 128 Neurons | 256 Neurons | Average |
| Sigmoid | 6 | 57,571 | 75,898 | 49,735 | 77,426 | 91,292 | 70,385 |
| Linear | 3 | 434,337 | 155,503 | 41,607 | 106,299 | 445,791 | 236,707 |
| Linear | 8 | 64,179 | 44,576 | 50,829 | 441,789 | 76,761 | 135,627 |

Table 14. DPPG Standard Deviation Scores purely based on neurons

The standard deviation again prove the assumption that because of one almost positive result, the mean score results in a -1300 score. However all architectures show a relatively low standard deviation at 64 neurons. This does correspond with the somewhat mediocre mean scores.

References

[1]

Barto, A., Anderson, C., & Sutton, R. (2015). *CartPole version 0*. Retrieved May 23, 2017, from OpenAI Gym: https://gym.openai.com/envs/CartPole-v0

[2] OpenAI Gym. (2015). *Pendulum Version 0*. Retrieved May 23, 2017, from OpenAI Gym: https://gym.openai.com/envs/Pendulum-v0

[3] Kistemaker, S. (2008). *Cross-Entropy Method for Reinforcement Learning.* Amsterdam: University of Amsterdam. Retrieved June 25, 2017, from https://esc.fnwi.uva.nl/thesis/centraal/files/f2110275396.pdf

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonogluo, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* Toronto: University of Toronto. Retrieved June 25, 2017, from https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[5] Gibbs, M. (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks, 11*(6), 1458-1464. Retrieved June 25, 2017, from http://ieeexplore.ieee.org/document/883477/

[6] Nair, V., & Hinton, G. (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines.* Computer Science. Toronto: University of Toronto. Retrieved June 25, 2017, from http://machinelearning.wustl.edu/mlpapers/paper\_files/icml2010\_NairH10.pdf

[7] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, D. (2015). *Deterministic Policy Gradient Algorithms.* London: Deepmind Technologies. Retrieved June 25, 2017, from http://proceedings.mlr.press/v32/silver14.pdf

# Bibliography

Barto, A., Anderson, C., & Sutton, R. (2015). *CartPole version 0*. Retrieved May 23, 2017, from OpenAI Gym: https://gym.openai.com/envs/CartPole-v0

Gibbs, M. (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks, 11*(6), 1458-1464. Retrieved June 25, 2017, from http://ieeexplore.ieee.org/document/883477/

Kistemaker, S. (2008). *Cross-Entropy Method for Reinforcement Learning.* Amsterdam: University of Amsterdam. Retrieved June 25, 2017, from https://esc.fnwi.uva.nl/thesis/centraal/files/f2110275396.pdf

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonogluo, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* Toronto: University of Toronto. Retrieved June 25, 2017, from https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Nair, V., & Hinton, G. (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines.* Computer Science. Toronto: University of Toronto. Retrieved June 25, 2017, from http://machinelearning.wustl.edu/mlpapers/paper\_files/icml2010\_NairH10.pdf

OpenAI Gym. (2015). *Pendulum Version 0*. Retrieved May 23, 2017, from OpenAI Gym: https://gym.openai.com/envs/Pendulum-v0

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, D. (2015). *Deterministic Policy Gradient Algorithms.* London: Deepmind Technologies. Retrieved June 25, 2017, from http://proceedings.mlr.press/v32/silver14.pdf

1. Deep Deterministic Policy Gradient [↑](#footnote-ref-1)