Machine Learning Engineer Nanodegree
Capstone Project

Markus Buchholz
05/08/2017

DQN algorithm in Reinforcement Learning control system
– Project –

Project Overview

This project discuses reinforcement learning approach to control CartPole. Research focuses on implementation of deep neural network to control game (CartPole) and hyper parameters tuning in order to achieve in shortest possible way the defined goal (reward of 195 over 100 consecutive trials).

In 2013 DeepMind company published its breakthrough paper: *Playing Atari with Deep Reinforcement Learning* [1], [2]. The paper demonstrates achievements in implementation of pioneering new algorithm called **Deep Q Network (DQN)** used to learn the computer to play Atari 2600 video games. The result of presented work (control approach), where the pixel screens were observed (environment state), reward given and control action for the game taken was outstanding. The main concept of DQN is to replaces the Q-table in ordinary Q-Learning algorithm by deep neural network to approximate the action reward based on the state.

Here, reinforcement learning can be characterized as an algorithmic attitude where intelligent agent (DQN) based on dynamic interactions (actions) with nondeterministic environment (CartPole game) , which is not known before, learn how to interact in order to maximize the reward (achieve the goal – pole does not fall over).

The reason for this project is not to compete with OpenAI (CartPole) players (to achieve game solution in lowest number of episodes) but rather to investigate the different scenarios, system parameters (project setup) in order to design system optimal and build the fundamental knowledge for future reinforcement learning projects.

Artificial intelligence has become the most modern and powerful technology in solving human being challenges and to control robots. It many cases advantages of using this approach is highly relevant with proper algorithm choice in order to solve mathematical problems, secure precise control and less computing time. Deep Learning Nanoprogram I am a student at and previous SmartCab project where the Reinforcement Learning and Q algorithm was investigated inspired me to look closer these technologies and study deeper presented approach. It seems that Deep Neural Networks and Reinforcement Learning will play predominant (as a fundament) role in Artificial Intelligence.

Problem Statement

The problem, which is going to be solved, is based on relatively simple environment in OpenAI gym (a game simulator). CartPole is a cart (moving on frictionless track) with the attached by an un-actuated joint pole. The cart moves (left – right) preventing from the pole falling over. The game agent (control systems) is rewarded +1 for every time step that the pole remains upright (pole does not fall over).
In this research there is a necessity to design automatic (intelligent) system to control (balance/prevent from falling over) the pole for the defined time (the problem is solved if agent receives average 195.0 score over 100 consecutive trials).

In this project the number of states (position of cart and pole) is continuous, therefore the simple tabular representation (Q – table in Q algorithm) will not work. In order to approximate arbitrary the continuous function, the neural networks had to be applied (the cartpole intelligent control system was based on DQN). The approximate Q function where $\overline{w}$ is a vector of weights of neural network, were optimized during the training process. Finally, the output from the iterative weight optimization process – trained neural network, was used to compute (predict) the action for the cartpole [3][5][6].

In this project, the goal was to secure the minimum score of the game. When the reinforcement learning [6][7][8] is applied to solve the problem with finite states of environment so the Q – learning algorithm based on discrete Q table can be successfully used. In this project case, the state space was approximated using following formula:

$$Q(s,a) \approx Q(s,a,\overline{w})$$

Project was clearly divided by two sections. One, when the DQN was learned (network weights and system hyperparameters were optimized) and the second (testing) where the agent played the game (system is autonomous; no parameters are adjusted). Here the final control system performance could be verified.

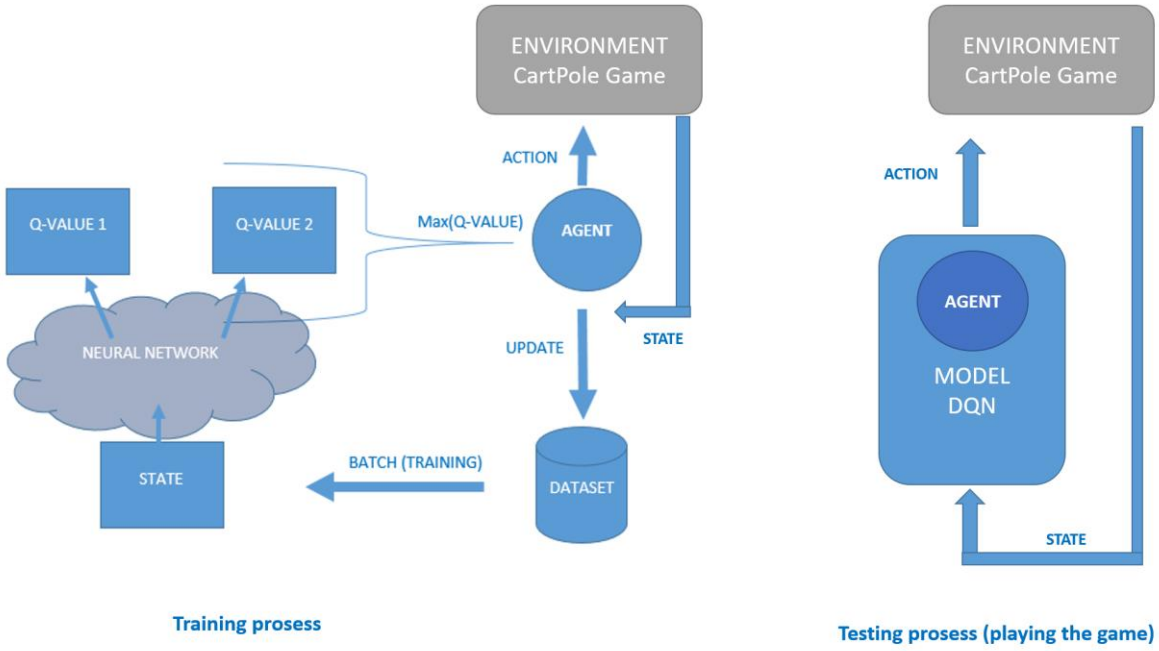


Figure 1. Architecture of project setup (development approach)

Metrics

In this project it was decided to follow the OpenAI specification and choose as a main metrics score of 195.0 over 100 consecutive trials which solve the cartpole game problem.
Beside the main metrics, it was decided to applied the internal metrics (only for development reason) which was used during the solution optimization process. Saved model of neural network was used to play a game (20 times). Number of game with the score over 195 was counted and compered during the solution development approach.

Pseudocode for applied in this project main metrics can be specified as follows:

```
"""
pseudo code explaining how the metrics for
the cartpole game is computed

"""
memory = [100] # size of memory (consecutive episodes)
mean_memory = [] # keeping episode value when the game was solved


for i in range (number_of_games):
    #work as a LILO - Last Input Last Output
    memory.pop(0) #remove value from memory
    memory.append(score_of_the_game[i]) #add to the memory score of the game

    if np.mean(memory)>=195: # check on the faly if the mean value of whole
                             # memory fulfills conditions
        mean_memory.append(i) # add to mean memory the number of trial for which
                              # the conditions of the game are fulfilled

print ("game is solved after episode: ", mean_buffer[0]) # print number of
                                                         #episode after which
                                                         #the game is solved
```

II. Analysis
Data Exploration

In this project, the dataset was not provided as an existed data base. The dataset was collected during the training process (playing the game).
Generally, during the game, while the agent is trying to balance the CartPole by giving the action (agent_action - agent by giving the command 0 or 1 moves the cart to left or right) the environment sends to agent a “tensor” including following (below) information:

next_state, reward, done, _ = env.step (agent_action)

1. **next_state** – consisting of information about the position if the cart on the track; angle of the pole to vertical line; linear velocity of the cart and angle velocity of the pole).
2. **reward** – Every time (frame) the agent "balanced" the pole (less than 15 degrees) the reward +1. In this project the target is a score of 195.
3. **done** - is a Boolean value telling whether the game ended or not.
4. _ - info (not relevant in this project)

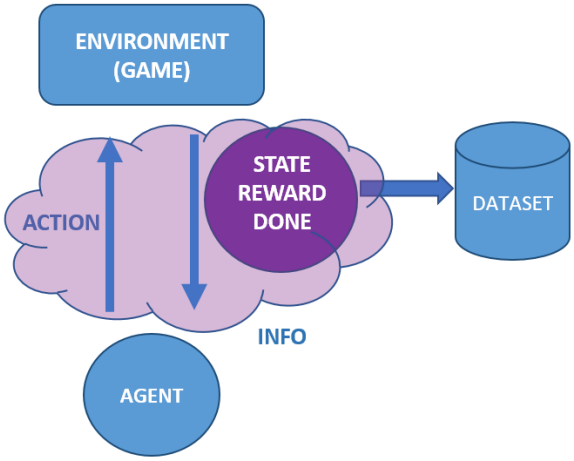


Figure 2. Process of dataset collection

In this project the set of states and actions (taken be the agent), together with rules for transitioning from one state to another, make up a **Markov decision process** (transition and reward depending only on actual action and state).

In order to train the neural network (to control the cartpole) the dataset has to be provided (dataset collected). When the agent play the game (send the actions), the state from environment (data) is collected (each step of the game is collected). Random sample of these information (memory experiences) as a batch of previous states are taken to learn the network (adjust the weights).
In this project, the size of the memory was limited (as in real systems) therefore data batch is sampled on newer experiences (not all data which were registered during the game is accessible). The data outside the size of memory is permanently erased.

During the training process (when the dataset is first collected and after used – as a batch to learn the neural network) the Q – learning algorithm (strategy) is applied.

Descriptive Statistics

Descriptive statistic for randomly collected date (500 states) can be summarizes on below tables and curves.

	position	pole angle	cart velocity	pole angle velocity
count	500.000000	500.000000	500.000000	500.000000
mean	0.010971	0.088129	0.021465	0.008644
std	0.102566	0.673481	0.100467	0.911166
min	-0.263658	-1.713984	-0.246709	-2.868278
25%	-0.041216	-0.378201	-0.034200	-0.600453
50%	-0.008681	0.018947	0.024838	0.027567
75%	0.043628	0.433649	0.086568	0.615820
max	0.510729	2.278438	0.242640	2.249161

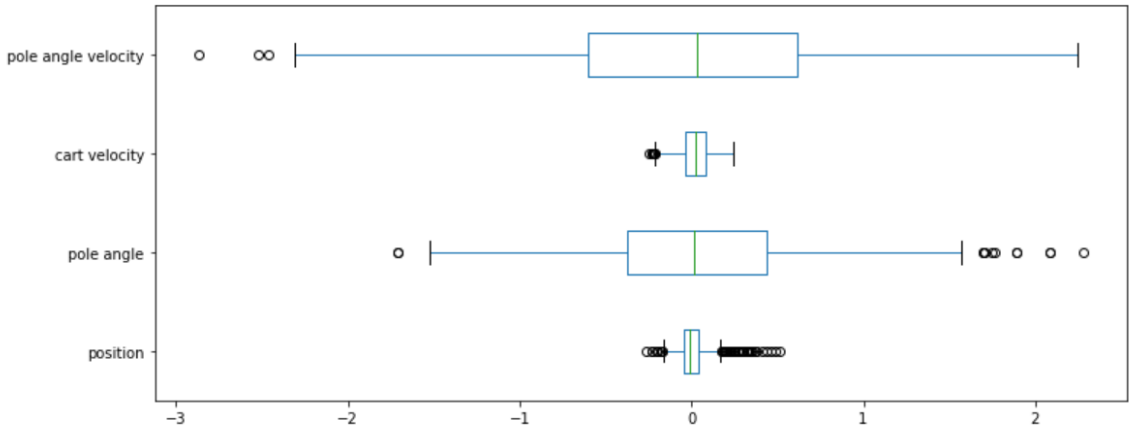
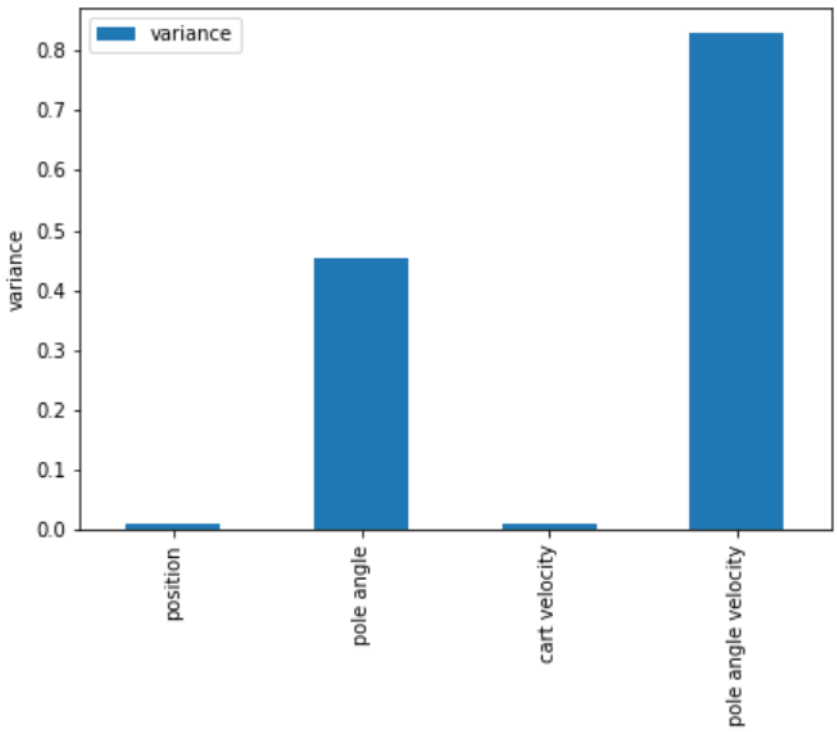
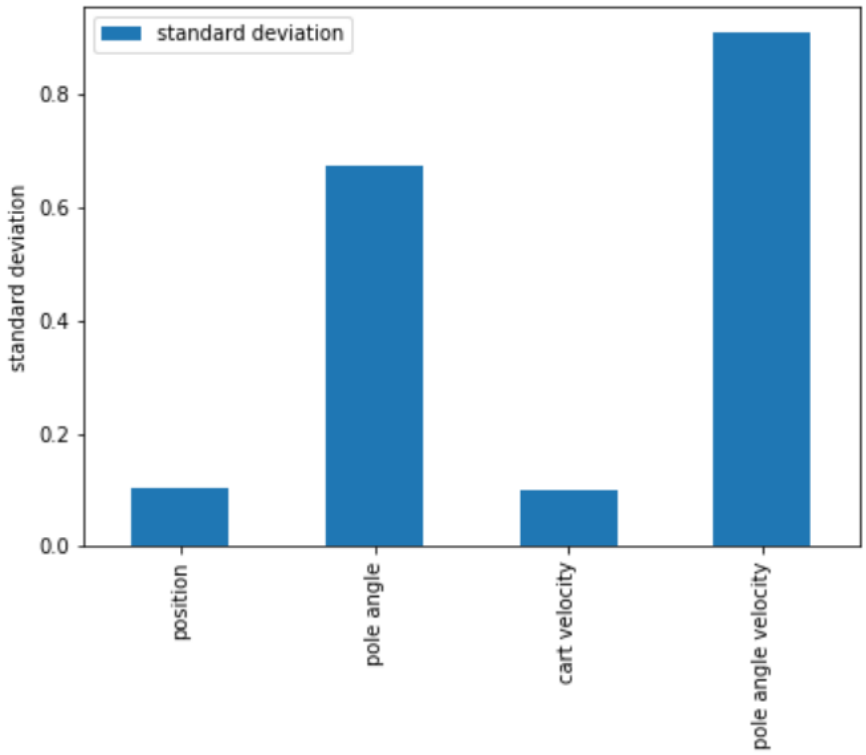


Figure 3. Summary statistics for data set




```
variance for  position = 0.0105198830326
variance for  pole angle = 0.453576963939
variance for  cart velocity = 0.0100936778591
variance for  pole angle velocity = 0.830224329814
```

Figure 4. Variance for data set




```
standard deviation for  position = 0.102566481039
standard deviation for  pole angle = 0.673481227607
standard deviation for  cart velocity = 0.100467297461
standard deviation for  pole angle velocity = 0.911166466577
```

Figure 5. Standard deviation for data set

	position	pole angle	cart velocity	pole angle velocity
position	1.000000	0.710455	-0.688438	-0.659299
pole angle	0.710455	1.000000	-0.498933	-0.947702
cart velocity	-0.688438	-0.498933	1.000000	0.677811
pole angle velocity	-0.659299	-0.947702	0.677811	1.000000

Figure 6. Correlation Matrix of values for data set

	position	pole angle	cart velocity	pole angle velocity
position	0.010520	0.049076	-0.007094	-0.061615
pole angle	0.049076	0.453577	-0.033759	-0.581561
cart velocity	-0.007094	-0.033759	0.010094	0.062048
pole angle velocity	-0.061615	-0.581561	0.062048	0.830224

Figure 7. Covariance Matrix of values for data set

Exploratory Visualization

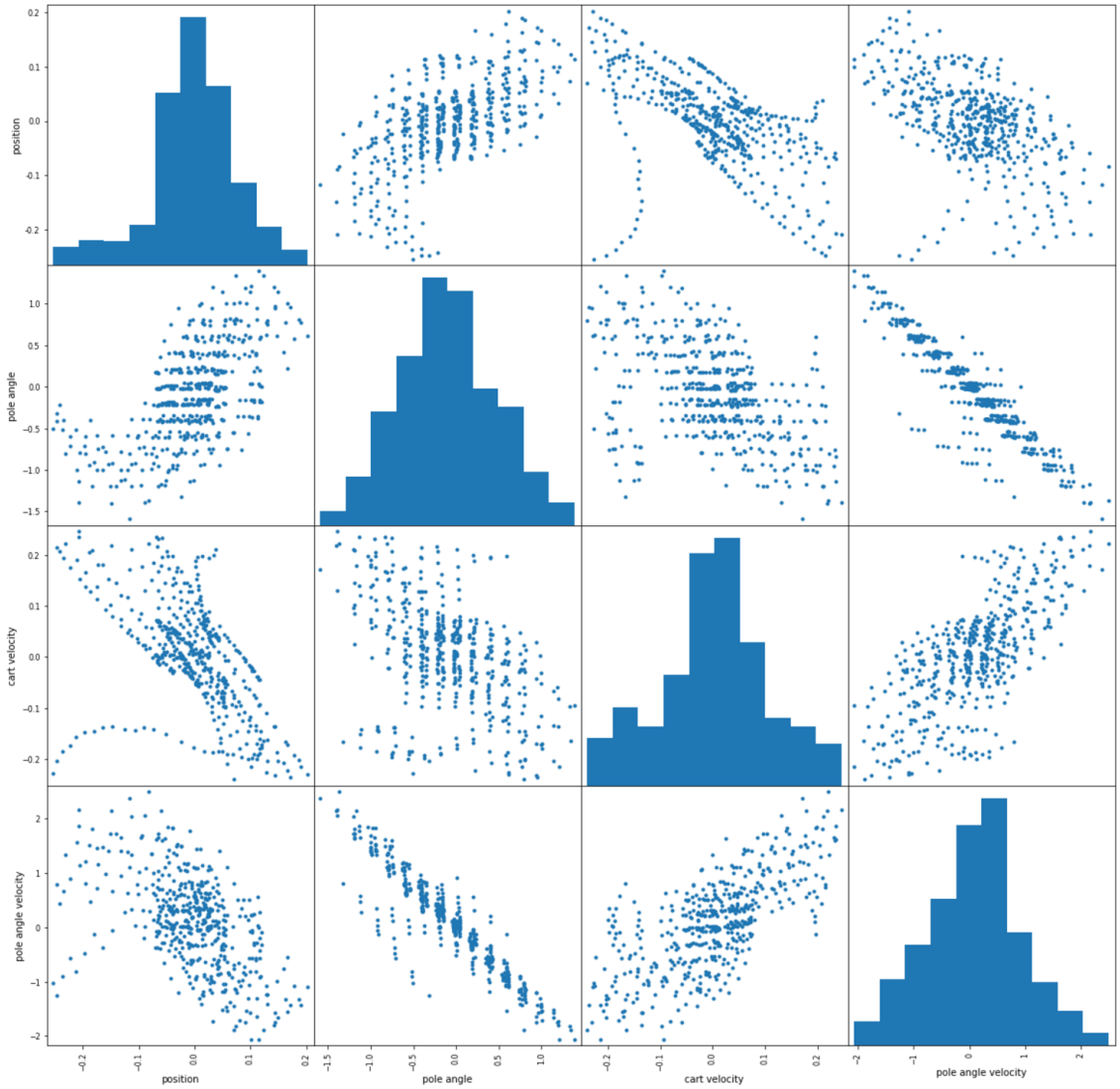


Figure 8. Project dataset (pairs of environment state variables)

Dataset collected during the learning process is based on random states (when the Q – algorithm is not used). The state consists of four variables: **position** if the cart on the track; **angle of the pole** to vertical line; **linear velocity** of the cart and **angle velocity** of the pole). Five hundred states of the game were take to build and depict below curves. The curves show the pair relations between cartpole variables. Each variable has a normal (gaussian) distribution. The collected randomly dataset (taken only for visualizing the dataset) indicates expected behavior of the game.

Each game starts while all the variable are null. Generally (as Gaussian curve indicates), more actions (and the states are collected) are taken while cart stays in the center of the track. The curves indicate direct rule and behavior (physics) of the game. Statistically for rather small distances from origin position the linear velocity of the cart and angular speed of the pole are small. Pairs: pole angle vs pole angle velocity; position vs cart velocity indicates clearly this phenomena. When the pole is falling over (angular speed increase) the external force pushes the cart in opposite direction (agent send proper action) so the linear velocity of the car increases also. This relation can be seen directly on curves (pole angle velocity vs cart velocity and opposite).

Algorithms and Techniques

Reinforcement Learning is concerned with learning control policies for agents interacting with unknown environments. Such an environment is often formalized as it was mentioned above with a Markov Decision Process (MDP) which is completely described by a 4 – tuple (S, A, S', R). At each timestep t an agent interacting with the MDP observes a state S, and chooses an action A which determines the R and next state S'. As it was indicated previously, due to continuous number of states of the game, each agent action is taken based on approximated (by neural networks DQN) state:

agent action = argmax (Q_neural_network).

In the project the action of the cartpole is predicted (during the training and after when the agent plays the game) by trained model of convolutional neural network, which is learned by used of DQN algorithm. Reinforcement learning algorithm Q – learning applied in this project is based on the Bellman equation, where the target of the game state can be defined as follows:

$$Q(s,a) = r + \gamma \cdot maxQ(s',a')$$

Where **s** is the environment state, **a** is an agent action, **s'** is the next state from state s and action a.
r represents the maximum discounted future reward when the action a is performed (in state s). γ – is a discount factor.
In order to minimalize the distance between the predicted and target state the loss function has to be defined (below). Further, given function will be used to update the weights of neural network.

$$loss = \left(r + \gamma \cdot maxQ(s',a') - Q(s,a)\right)^2$$

In order to stabilized the learning process, the neural network were updated by applying (as inputs to neural network) the random samples of memory. Batch size (sample size) was optimized during following research. Random updated allowed reducing the variance. Beside that the learning could be perform based on multiple samples not just from the recent transition.

Deep Neural Network used in this project was built by use of TensorFlow. Solutions presented on OpenAI website are mainly based on Keras. Here the best Udacity practices were deployed (solution uses concept presented in [9]). Final project deep neural network architecture can be presented as follows:

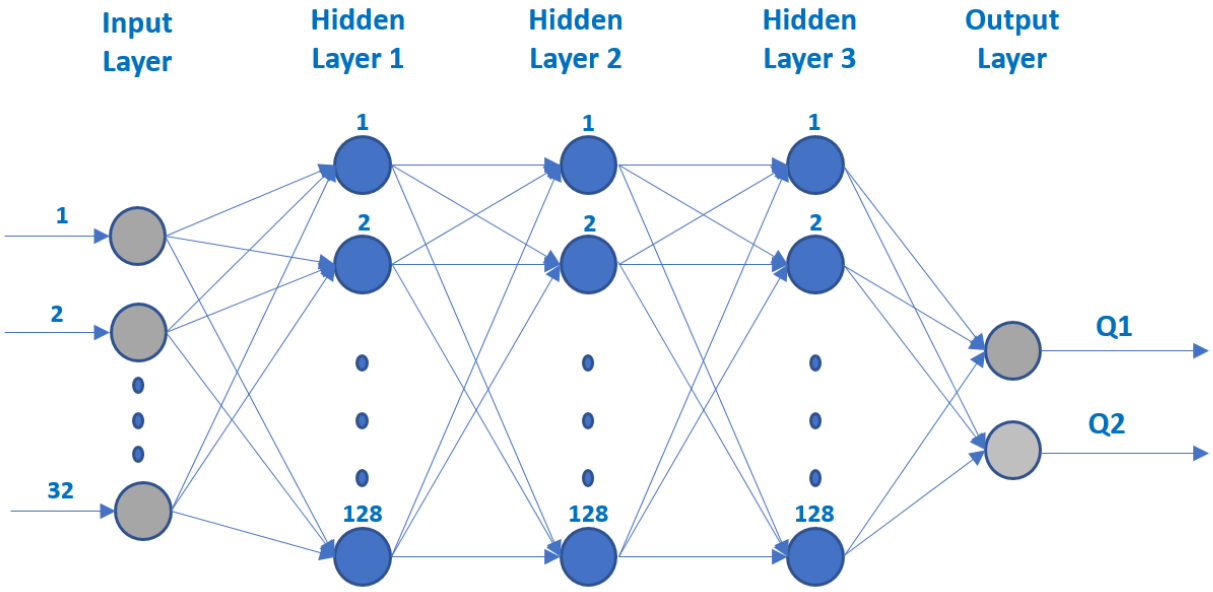


Figure 9. Architecture of Neural Network used to control CartPole

Input layer dependind directly on batch size applied in this project. The size of hidden layers 256 were chosen durinning the optimization project (iterational) approach. There was verified that three hidden layers secure best results. Size of output layer is determined by number of cart movement: left or right.

Benchmark

Benchmark for this project will be based on one of the solutions given on OpenAI and evaluation the number of episode used before solving the game. It was decided to compare provided in this project solution against with (benchmark) presented by David Sanwald:

https://gym.openai.com/evaluations/eval_GFtDBmuyRjCzcAkBibwYWQ

III. Methodology

Data Preprocessing

Dataset used in this project was collected during the training of applied neural network therefor no previous data was used and thus no data exploration was applied. Collected dataset is used for training reason as it is. As it was mentioned before in order to secure low variance the data provided as an input is random sampled (batch).

Implementation

The solution for the given problem was approach in two main steps, performed one by one in international preprocess. At the beginning, base on given hyperparameters and defined neural network architecture the training was performed. Here preliminary performance of solution could be verified (how many number of episodes had to played in order to guaranty that the project main metrics would be fulfilled: score of 195 in 100 consecutive trials). Subsequently, the train model of neural networks was verified during the test phase (neural network weight were not updated) and verification of number of score over 195 was taken into account. Based on both result (training and testing) the decision regarding the setup of hyperparameters and neural networks configuration (architecture) correct decision was taken. Choice of hyperparameters allowed achieving the project goal (solve to game). The final solution was afterward approximated by setup the other parameters which potentially would influence on final results (discussion about the hyperparameters choice is included in this project report). Train model was saved and then used in testing phase. Here the 20 games were played and game with score over 195 was registered. As it was mention above the project uses tensorflow to build DQN. DQN learns to estimate the Q-Values (or long-term discounted returns) of selecting each possible action from the current game state. DQN uses Q – learning algorithm (reinforcement learning algorithm).
It is worth mentioning that in order to succeed the project it was necessary to approach the final solution by use of iterative process which can be presented as follows:

1. Elaboration of convolution network architecture (the architecture to be optimized during the iterative approach design process)
2. Approach the value of hyperparameters both neural network and reinforcement learning process (environment and algorithm).
3. Perform the training:
    a. Initialize the memory and neural network weights
    b. For *number_of_episodes* do:
        i. With probability ε (decaying exploration factor) chose the random action – in order to explore environment otherwise choose the action computing the $a_t = \arg maxQ(s,a)$
        ii. Agent of the game to execute the action a_t, receives (Q_decision) the reward and carpole goes to state s_{t+1}
        iii. Above transition will be stored in defined previously memory
        iv. Sample the memory to build the batch and perform the neural network training
        v. Current output from neural network is used to compute a new target (Bellman equation) or the game is over with the current reward
        vi. Compute the loss and propagate backward to update the weights

vii. Verify the main project metrics (195.0 over 100 consecutive trials which solve the cartpole game problem

4. Save the solution (architecture and trained weights).
5. Test solution. Check if cartpole fulfills the project requirements (keep the pole upright for 195 frames – play 20 games).

Schematically the training process and Q- learning algorithm performance can be depicted on following figure.

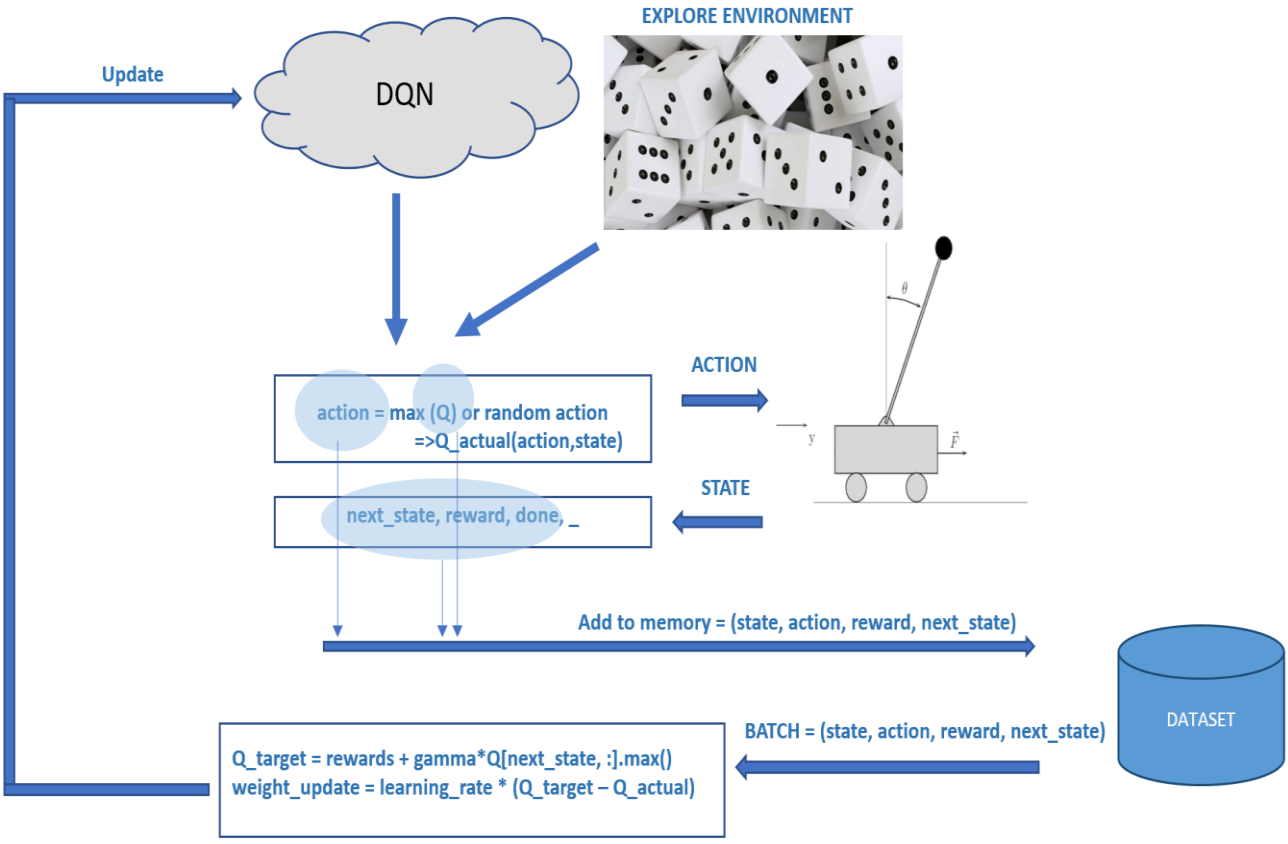


Figure 10. Architecture of system design

Refinement

During the solution approach the neural network architecture (hidden size, number of layers) and system parameters (hyperparameters as: learning rate, gamma factor, batch size) had to be optimized.

Initial setup of parameters taken into consideration were given in [9] and can be reported as follows:

- Learning rate = 0.0001
- Number of hidden layers = 2
- Hidden size = 64
- Batch size = 20
- Gamma factor = 0.99

Based on given initial parameters and neural network setup the training and testing were performed. Test result are depicted on figure 11. Base on preliminary setup solution (number of episodes was reduced to 500) of the game (according to definition of OpenAI) can not be achieved. The maximal score during the training is 160. Test phase confirms that autonomous control system based on DQN can play the game only with maximal score 150 (which not solve the given problem). Average score of 20 game is around 100.

Note: All figures where the results were presented contains two curves one (grey) with actual results and (blue) curve where the score is averaged over 10 observations.

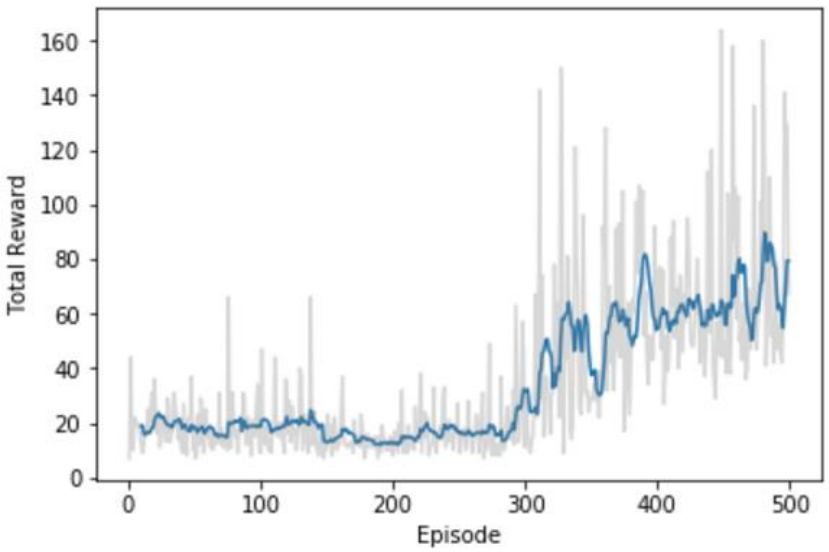


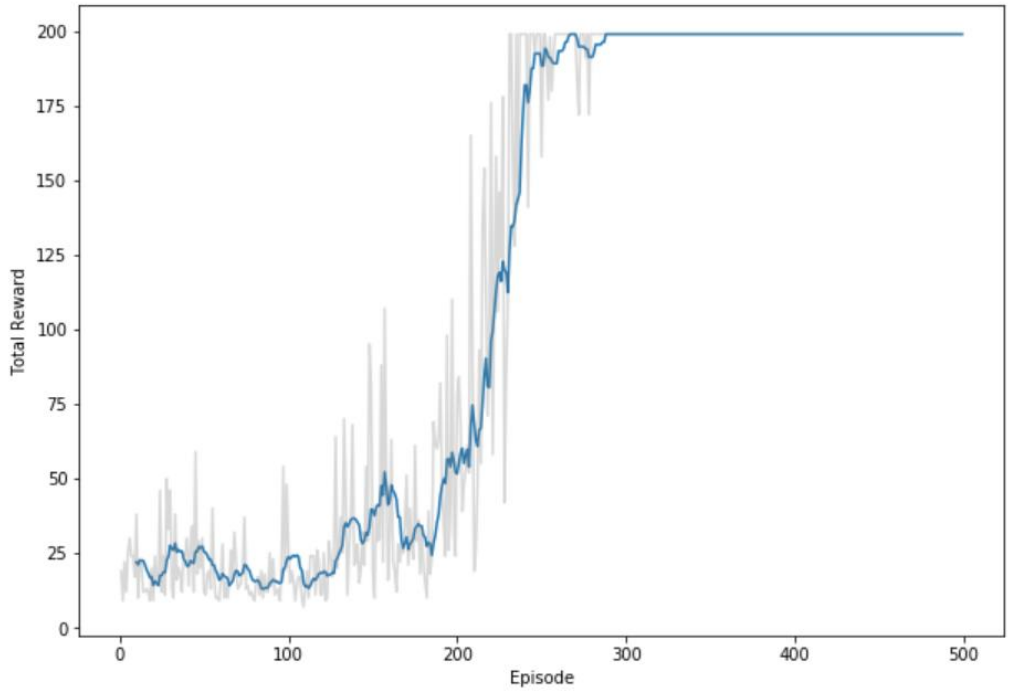
Figure 11. Training performance: Initial setup

Development process performed in this research approach final architecture and hyperparameters. The final architecture od neural network was presented on figure 12. Final setup of parameters can be summarized as follows:

- Learning rate = 0.0001
- Number of hidden layers = 3
- Hidden size = 128

- Batch size = 32
- Gamma factor = 0.99

Final architecture and system setup sufficiently improves intelligent control system performance (DQN). Figure 7 depicts the training performance. The game is solved after 332 episodes. Average score of game testing is 199.




```
: average_reward (rews)

game is solved after episode:  332
```

Figure 12. Training performance: Final setup

Figures 9 and 10 depict project approach. Choice of hyperparameters and neural network architecture was elaborated during the iteration process described in detail in Implementation section. Intermediate solutions (which depict hyperparameters setup have been discussed in Results section).

IV. Results

Model Evaluation and Validation

Solution approach was started based on initial setup of the system. The architecture of applied neural network (number of layers, size of hidden) was chosen to first investigation phase. Main project metrics was used to measure performance of a model (score of 195.0 over 100 consecutive episodes). Preliminary setup occurred to be fail. The game could not be solved after training period (500 episodes). Beside that during the testing phase the average score was approximate 100. Test results are depicted on figures 11. At the beginning it was decided to change the only the number of layers from 2 to 3. Size of hidden remained the same 64. In addition, it was decided to change the batch size to 32. This configuration could solve the game after 368 episode. The curve indicates that during training the convergence was "lost" and score was lowered to around 110. Convergence (score over 195) was achieved again after 40 episodes. During the testing system played the game with average score over 199.

In order to secure stability in score results the hidden size was doubled. In this case the solution was achieved after 332 episode and the curves stayed flat (converged) to the end of training periode. In this case as previously, system test achieved the average score over 199. It was thought that enlarging the size of hidden could accelerate the process when the solution is achieved but enlarging the hidden size to 256 the convergence while training was lost again. Finally, the architecture with 2 layers and 512 hidden was verified. In this case the solution is achieved after 420 episode.

Based on first phase of presented research investigation it was decided to use architecture with 3 layers and 128 hidden size (all other parameters were tuned on chosen final architecture). Below figures depict the test result of neural network architecture investigation. Figure presenting the score in testing period was depicted only one (the same average result were achieved for all presented architectures, details in text).

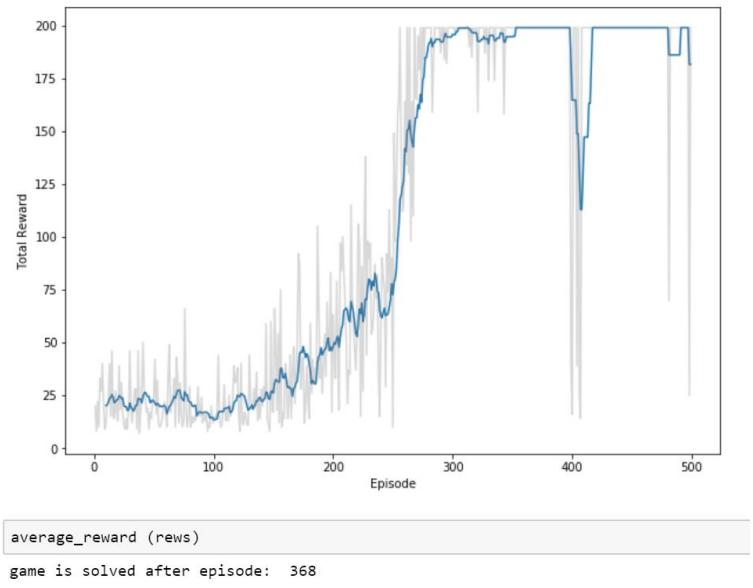
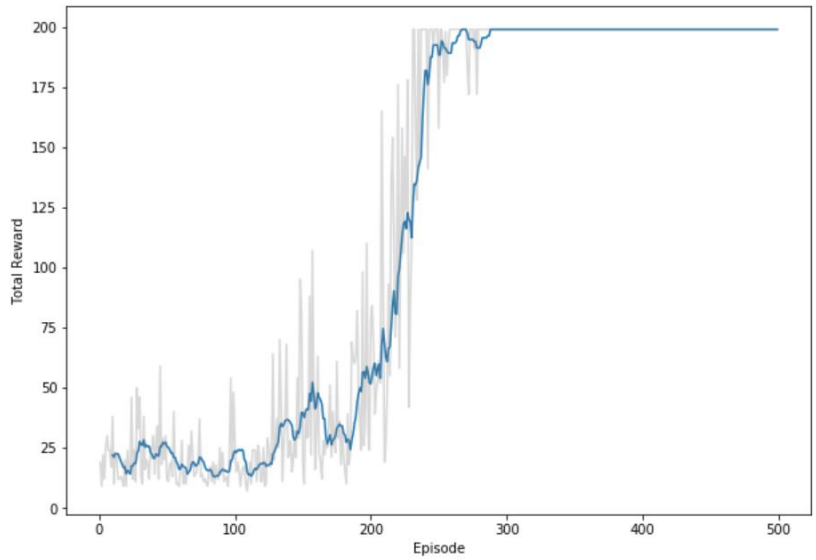


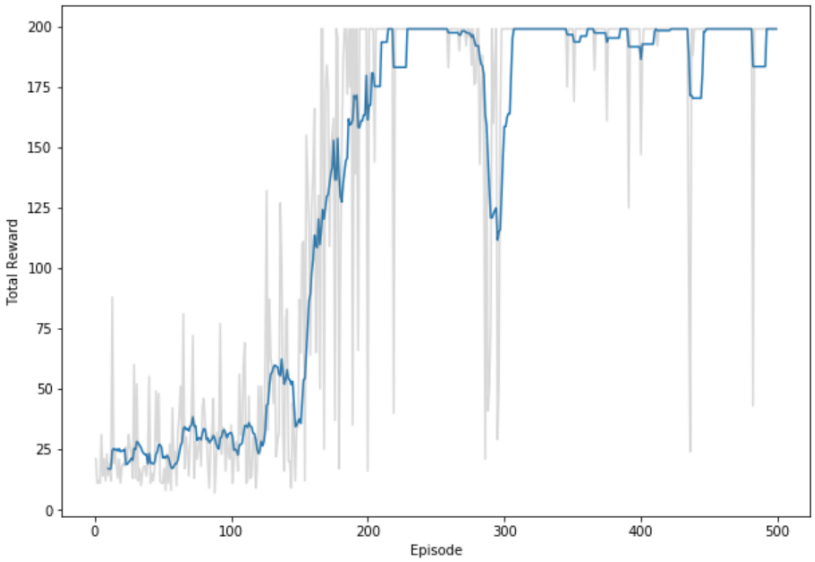
Figure 13. Training performance: 3 layers, 64 hidden size




```
: average_reward (rews)
```

```
game is solved after episode:  332
```

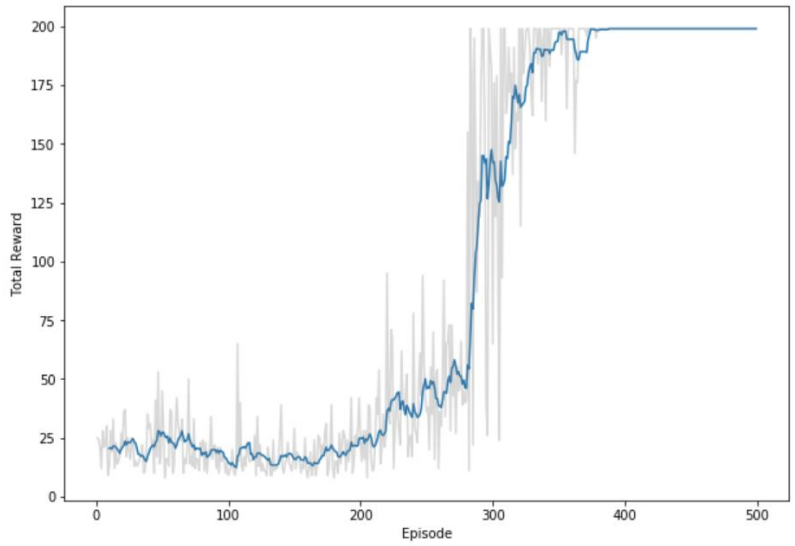
Figure 14. Training performance: 3 layers, 128 hidden size




```
: average_reward (rews)
```

```
game is solved after episode:  394
```

Figure 15. Training performance: 3 layers, 256 hidden size




```
: average_reward (rews)
game is solved after episode:  420
```

Figure 16. Training performance: 2 layers, 512 hidden size

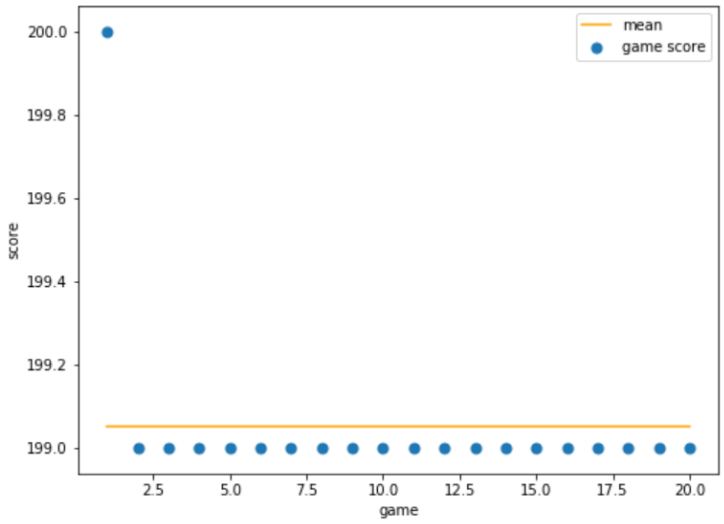


Figure 17. Testing performance

Choice of neural network architecture (3 layers, 128 hidden size) allowed starting next phase of solution approach where learning rate, gamma factor and batch size were investigated. It was decided to check influence of these hyperparameters on main metrics by increasing and decreasing the parameter value (one by one).

Increasing the learning rate by factor 10 to learning _rate = 0.001 influenced that the game was solved faster. It took only 264 episodes to solve the game. Unfortunately, the train and test performance indicate that the solution is not stable (during the training the convergence is "lost" several times). Playing the game (during the test) secure only average score equals 155. On the other hand, decreasing base learning rate by factor 10 to learning _rate = 0.00001 completely does not secure for final success (finding the solution). Solution in this case in not achievable (average score during the training is around 20).

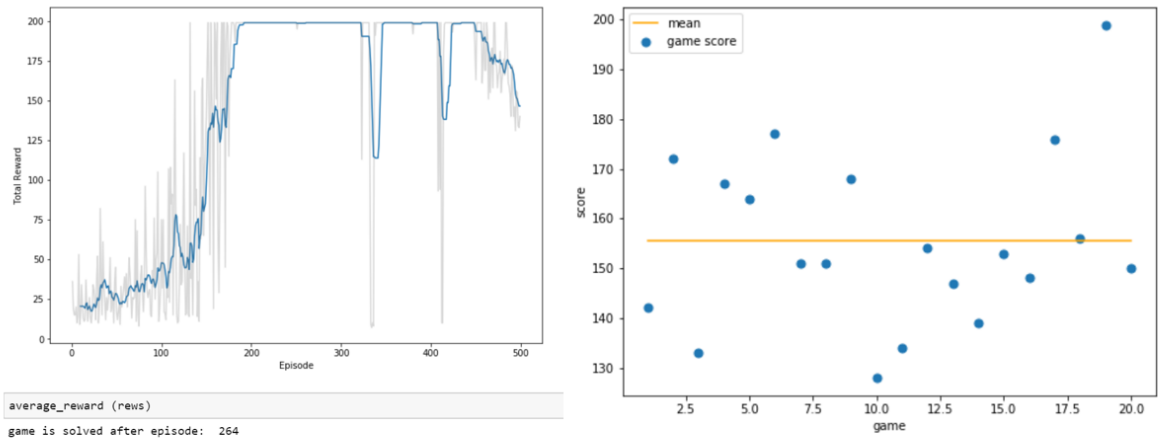


Figure 18. Training and testing performance learning _rate = 0.001

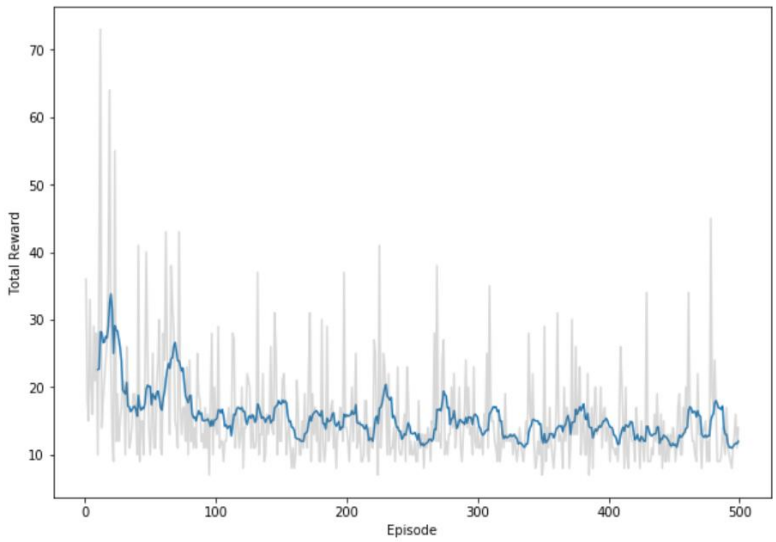


Figure 19. Training and performance learning _rate = 0.00001

Decreasing the batch size by factor 2 to batch_size = 16 negatively influences on system (training) performance. The solution during 500 episodes is not achievable. The batch size is too small to secure correct solution. Increasing batch size to 64 influences that the solution can be achieved after 337 episode but the training process in not stable (score convergence is lost after 450 episode and regained at approximately at 480 episode). Beside that, the testing phase depicts some local decrease in score result.

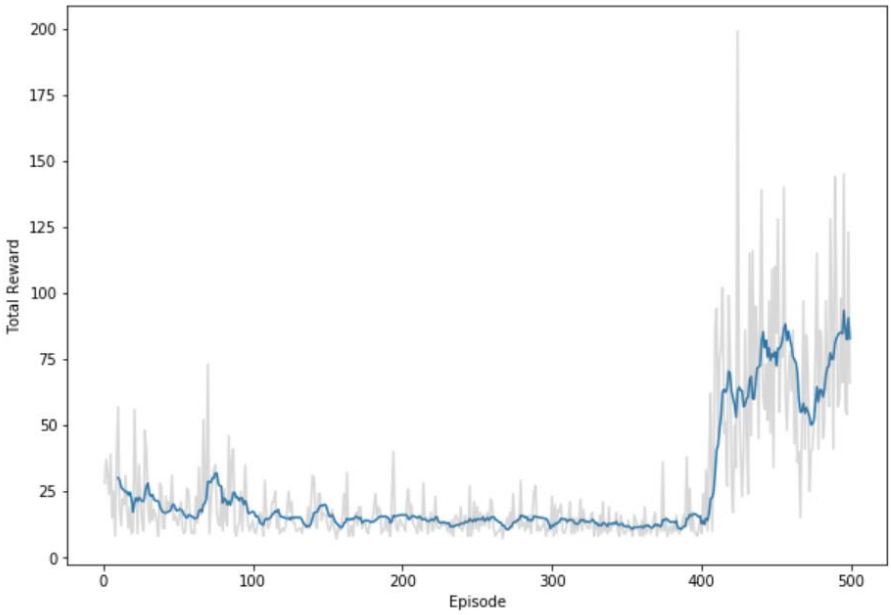


Figure 20. Training and performance batch_size = 16

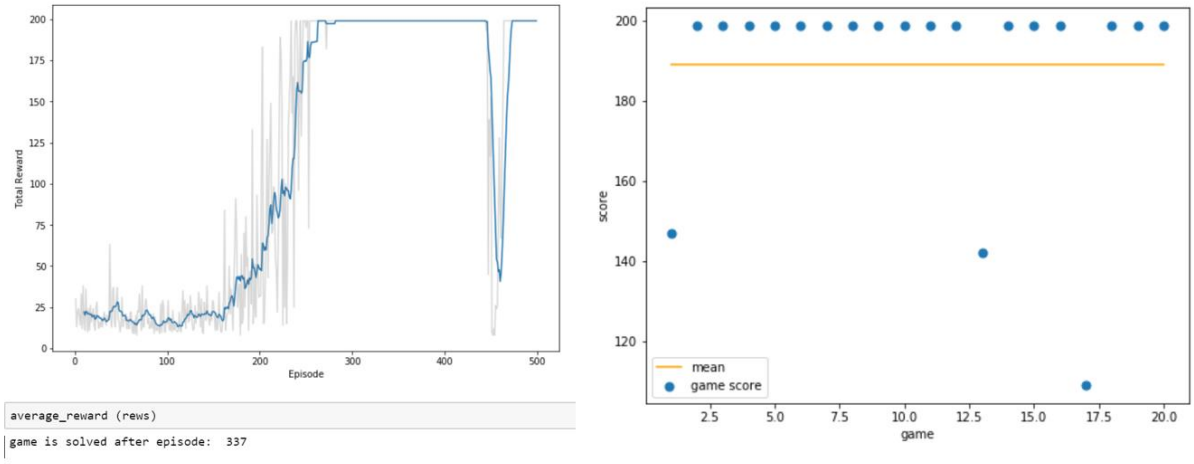


Figure 21. Training and testing performance batch_size = 64

Gamma factor, which regulates importance of short and long-term rewards plays significant role in reinforcement learning process. Here decreasing gamma factor to 0.95 (from 0.99) influenced that solution could be achieved after 346 episode but the score curve was not stabilized (like for gamma equals 0.99). Moreover, the test score also varies and the final average after20 games is 184. Adjusting the gamma factor to 1 make to solution unsolvable after 500 episodes.

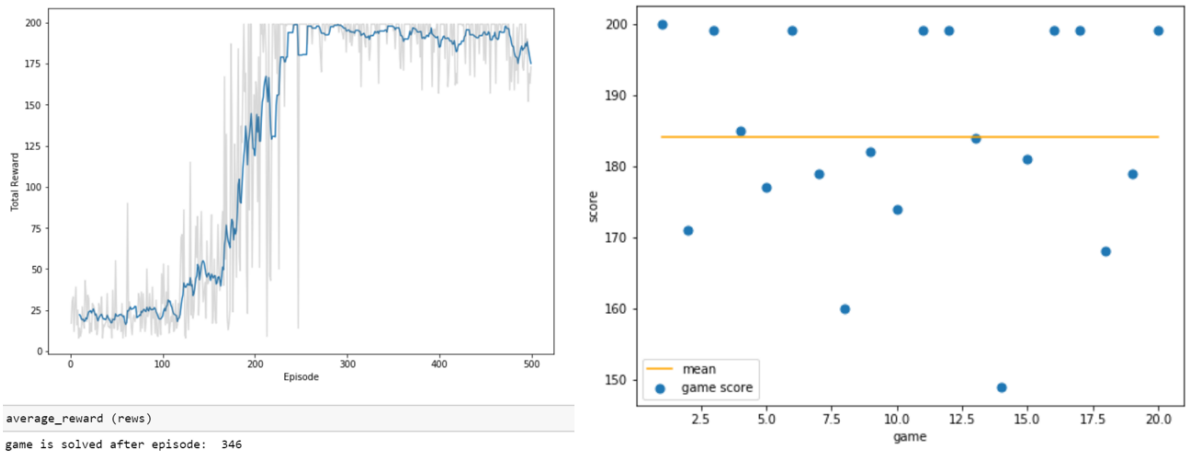


Figure 22. Training and testing performance gamma = 0.95

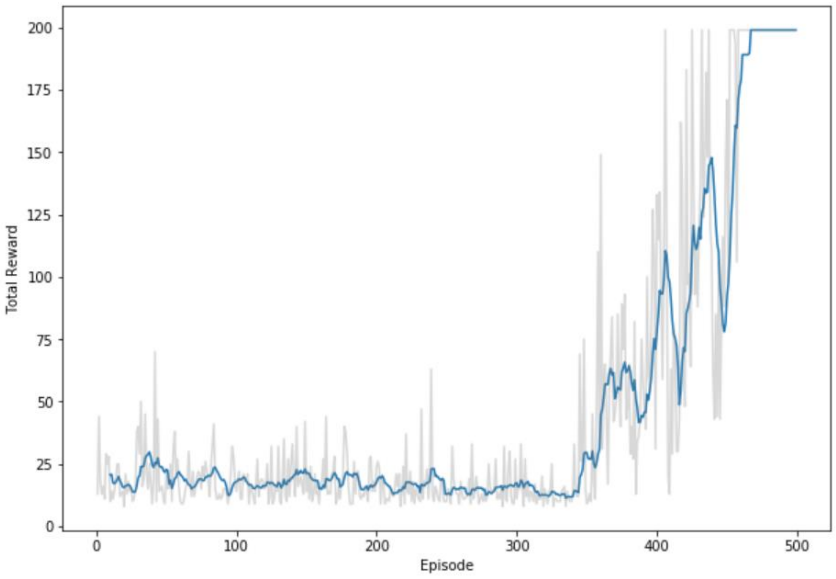


Figure 23. Training performance gamma = 1

Presented research results indicate that the best performers is achievable for the architecture and hyperparameters setup as follows:

- Learning rate = 0.0001
- Number of hidden layers = 3
- Hidden size = 128
- Batch size = 32
- Gamma factor = 0.99

Justification

Benchmark model uses Keras and provides high level implementation of the Q-networks. Solution is achieved after 51. The score is much better than achieved in discussed project. This phenomena could be explained by the fact of implementation the different kind of technique for neural network (tensorflow vs Keras). It seems that applying the Keras plays critical role for final result. Analyzing the OpenAI society it can be concluded that Keras as a high level neural network API is preferred. However, analyzing the score performance of training (for benchmark model) it can be seen that beside the curve in converged, small local drops (deviation/score decrease) can be seen. In addition, no testing is provided.

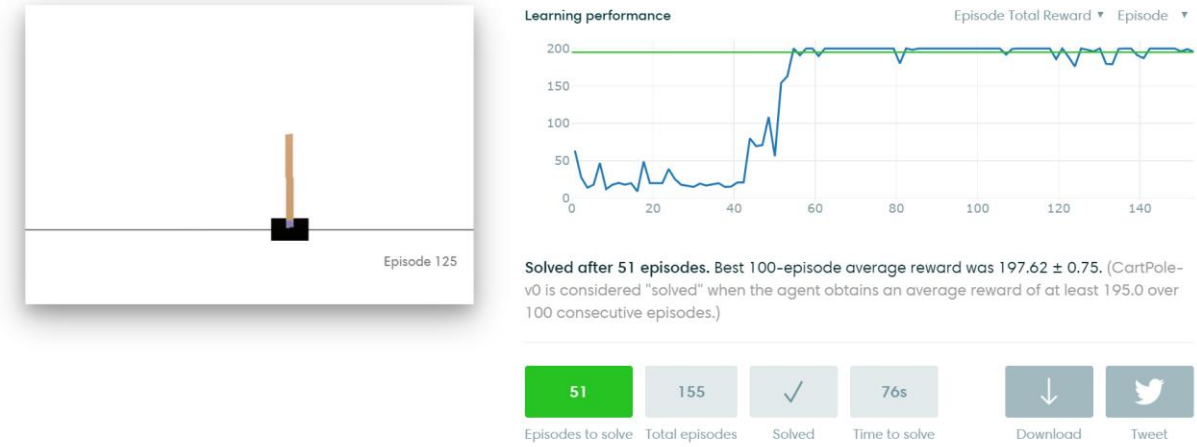


Figure 24. Benchmark performance

https://gym.openai.com/evaluations/eval_GFtDBmuyRjCzcAkBibwYWQ

The final solution approached in this project was discussed previously but it can be summarized as follows.:

- applying Q- learning algorithm based on deep neural networks (DQN)
- architecture of neural network: 3 layers, 128 hidden size, 32 inputs (batch size), 2 outputs (left, right)
- main adjusted hyperparameters: Learning rate = 0.0001, Gamma factor = 0.99, batch size = 32

Results

Final model (developed during this project) is robust enough to solve the given problem. During the training phase any local deviation is seen and the score is converged. Beside training takes longer time then benchmark model (332 vs 51) the trained model is trusted enough to play the 20 games (testing phase of the project) with the all scores over 195 (average 199.05). It is clear indication that the intelligent agent is reasonable and align with project expectation (solve the game). Moreover, the autonomous agent can play the random games with average score above the limit. However, it can be expected that future improvements for the presented solution will be focused on training phase. The learning process (when the game is solved) should be accelerated so the agent will find proper solution much faster. This parameter can be critical parameter for future systems where artificial intelligence based on reinforcement learning will be utilized.

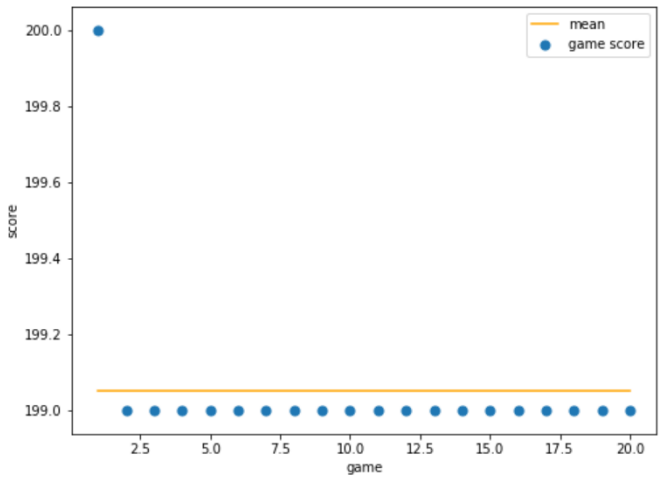


Figure 25. Testing performance after playing 20 games

V. Conclusion

Approached in this project solution was partial discussed in section Justification. The architecture of the system DQN was presented on figure 20. Architecture and system setup secure that train agent can play the game with average score (over 20 games) equals 199.05(figure 25).
Solution of the game (definition according to OpenAI) can be achieved after 332 episode.

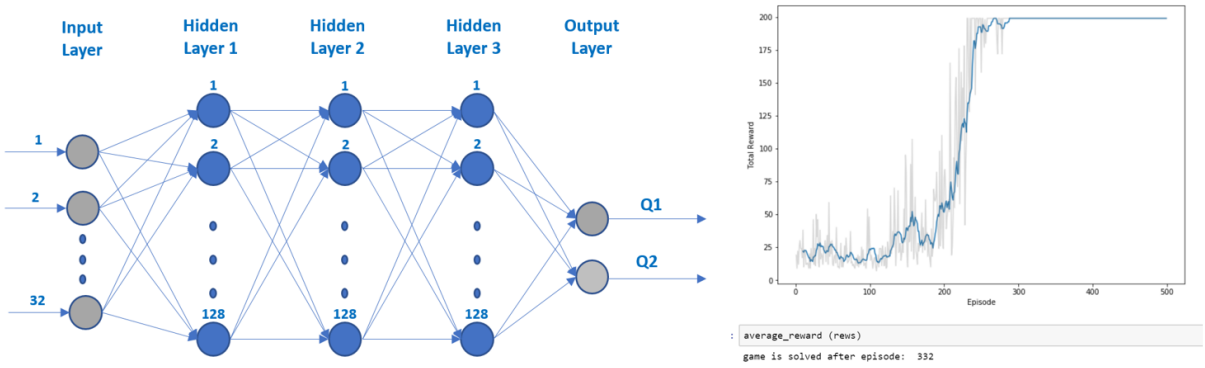


Figure 26. Final architecture and training results

Reflection

Provided by Udacity skeleton of the project together with supportive questions play significant role in successive solution approach and report submittal.
The Cartpole environment is quite simple environment but the solution based on Deep Reinforcement Learning has yielded proficient to control complex tasks. In project case the Q – learning algorithm (presented during Nanodegree program) was implemented. Together with deep network (build by used of tensorflow) proved to be sufficiently good solution. It can be assumed that other approach of this project can be very competitive to the solution presented on OpenAI. Reinforcement learning and deep neural network are fundamental for modern Artificial Intelligent systems and solutions.

This project was approached in proper defined development system architecture where, first the hyperparameters and setup of neural network was elaborated and intermediate solution was verified (testing phase where 20 games were played). Applied iterative approach could speed up the whole development process. The main metrics was correctly chosen and helped to benchmark the approach solution with other external to this project solution.

The project highlights (indicates) main hyperparameters, which have to be investigated while reinforcement learning will be applied.

The most interesting and exciting thing in this project was deeper understanding the conjunction reinforcement learning especial Q – learning with deep neural network. In addition, familiarization of DeepMind achievements encouraged me to be more familiar with this kind of knowledge. My future research fill be focused on better understanding the wakens of this discussed approach and exploration/innovative work towards potential improvements.

The most difficult part of the project was connected with implementation of deep neural network replacing Q – table in ordinary Q – learning algorithm Deppe learning solution changed the architecture of the final solution. Difficulties can be rewarded by elaboration of modern, fundamental for future AI solution.

The choice of Capstone subject fully fits my expectations. Experience from subject studying can be concluded that modern AI will be based on agents learning the environment base on exploration and applying proper strategy (reinforcement learning). It seems to be very applicable to apply investigated attitude (DQN) where continues state has to be approximated (where ordinary Q -table is not relevant/applicable).

Improvement

Performance of presented solution can be improved by implementation new techniques or replacement of applied in project solutions. It is recommended to consider in future work following modifications:

1. In order to elaborate more generative solution (applicable for wider range of autonomous control applications, where the reinforcement learning base on neural network is deployed) it will be necessary to investigate more OpenAI environments (applied transfer learning) and capture more experience in parameter tuning and architecture choice.
2. The performance of the presented solution can be improved by hyperparameter tuning. Correct setup of parameters is critical for final system quality (in this case metrics).
3. Applied Q – learning algorithm can be replaced by Sarsa algorithm, where actualization of Q function was developed by sacrificing the choice of future action which maximize the Q value. In Sarsa algorithm for setting new Q the action which is going to be performed is taken to consideration (not the future action which maximize the Q value).
4. Neural network can be implemented by use of Keras. I addition other network architectures can be investigated.
5. The Game performance could be also increased by implementation heuristics in choice of actions (not only randomly as a alternative to Q – algorithm but also based on elaborated heuristics).

References:

[1]. DeepMind technologies, Playing Atari with Deep Reinforcement Learning,
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[2]. Tambet Matiiasen, Demystifying Deep Reinforcement Learning,
https://www.intelnervana.com/demystifying-deep-reinforcement-learning/

[3]. David Silver, Deep Reinforcement Learning
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf

[4] Benchmark by David Sanwald
https://gym.openai.com/evaluations/eval_GFtDBmuyRjCzcAkBibwYWQ

[5] Defeating the Deadly Triad
https://davidsanwald.github.io/2016/12/11/Double-DQN-interfacing-OpenAi-Gym.html

[6] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
http://www.deeplearningbook.org.

[7] Matthew Hausknecht and Peter Stone
Deep Recurrent Q-Learning for Partially Observable MDPs

[8] Zachary C. Lipton Jianfeng Gao, Lihong Li, Jianshu Chen, Li Deng
Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear

[9] Udacity reinforcement learning
https://github.com/udacity/deep-learning/blob/master/reinforcement/Q-learning-cart.ipynb