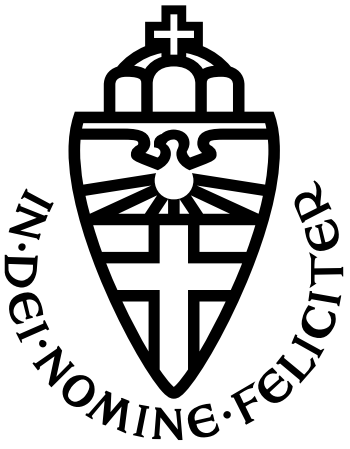
Radboud University and enliteAI



An Investigation of Model-Free Deep Reinforcement Learning

POLICY GRADIENT or Deep Q Networks? Which algorithm performs better on the cartpole environment in terms of epochs

Bsc Thesis Artificial Intelligence

Author:
Alexander Zehetmaier
s1013644

Supervisor:
dr. Luc Selen
Radboud University

Second reader:
dr. Patrick Blies
EnliteAI

July 2020

Contents

1 Introduction **2**

2 Methods **3**
2.1 Deep Reinforcement Learning 3
2.2 Deep Q Networks 5
2.3 Policy Gradient 6

3 Experiments **6**
3.1 Environment 6
3.2 TensorFlow 2 7
3.3 Hyperparameters 7
3.4 Hardware Specifications 7
3.5 Implementation 8
3.5.1 Differences 8
3.6 Results 10
3.6.1 Deep Q Networks 10
3.6.2 Policy Gradient 10
3.6.3 Comparison 11

4 Discussion **12**

A Tables **14**
A.1 Deep Reinforcement Learning Variables and Functions 14
A.2 Hyperparameter Values 14
A.3 Hardware Specifications 15

B Images **15**
B.1 Deep Learning Performance 15
B.2 Visualized Markov Decision Process 16
B.3 Reinforcement Learning Landscape 16
B.4 Deep Q Networks Best Trial 17
B.5 Deep Q Networks Worst Trial 17
B.6 Deep Q Networks Average Trial 18
B.7 Policy Gradient Best Trial 18
B.8 Policy Gradient Worst Trial 19
B.9 Policy Gradient Average Trial 19
B.10 Deep Q Networks vs Policy Gradient 20

1 Introduction

In the past, legged robots walked through a rule-based programming paradigm and as a result different terrains needed to be hard-coded [1]. Recent developments in the field of artificial intelligence and optimal control will probably cause a paradigm shift to more data-driven walking solutions. Instead of writing rules beforehand, sensors collect and 'perceive' the environment. In the best case, the robot uses this collected information and learns how to adjust its way of walking given the current environmental properties. One possible way is to get this environmental data through interaction with the environment. A specific machine learning algorithm applying this data collection technique is called reinforcement learning [2].

Deep reinforcement learning recently branched off from the field of reinforcement learning. Some types of deep reinforcement learning are very well suited for robotic locomotion. This especially holds for the domain of continuous action space reinforcement learning algorithms, since the motors driving the robotic legs behave continuously. Research already confirmed the proof of concept and managed to teach robots how to walk [3]. Nevertheless, reinforcement learning has a stochastic nature and even some minor initial changes can lead to a non-satisfactory result. Simply changing the seed of the random number generator which initializes the parameters of the algorithm could lead to a diverging training process. Modelling a probability distribution over the agents available actions, tries to approximate this stochastic nature.

In Model-free reinforcement learning no prior information about the environmental properties is provided to the agent. This leads to a higher variance of the probability distributions over the actions. Nevertheless, model-free reinforcement learning is very attractive for robotic locomotion, especially since no additional information about the environment needs to be gathered and injected into the already complex system. Model-free reinforcement learning can be divided into policy optimization and Q-learning. The main difference between the two model-free reinforcement learning algorithms is their different approach to maximizing the long term reward from the agent's interaction with the environment. Both domains of model-free algorithms have their pros and cons but it is not obvious which one to choose for real-world robotic problems.

The domain of deep reinforcement learning emerged relatively late. As the name already reveals, deep reinforcement learning combines the fields of deep learning and reinforcement learning. The first time deep reinforcement learning came to fame was in 2013, when DeepMind managed to win a series of Atari games [4]. Raw pixel inputs from the games were directly provided to their deep reinforcement learning algorithm. Just with the raw pixels as input, and the game scores as feedback, the algorithm managed by itself to figure out the different game mechanics. For three out of the six games, their deep reinforcement learning algorithm performed better than human experts in these games.

This thesis investigates the basic policy optimization algorithm called policy gradient and compares it to the basic Q-learning algorithm, deep Q networks. The hypothesis of this thesis is, that the policy gradient algorithm is more stable than the deep Q networks algorithm, but less sample efficient on the cartpole environment. This assumptions are made since deep Q networks is an off-policy reinforcement learning algorithm that allows to reuse historical data points for training [5]. The policy gradient algorithm is an on-policy reinforcement learning algorithm and does not allow the reuse of historical data for training[5]. Part of the thesis was to implement these algorithms from scratch. The deep Q networks algorithm belongs to the category of discrete action space deep reinforcement learning algorithms. Policy gradient has the advantage of being able to solve discrete and continuous action space environments. Both deep reinforcement learning algorithms build the foundation for more sophisticated continuous action space algorithms. This

category of algorithms is especially useful for the state of the art robotic locomotion.

2 Methods

2.1 Deep Reinforcement Learning

Deep reinforcement learning is a young domain in the field of artificial intelligence, although its constituent parts, neural networks and reinforcement learning, have been around for a long time. Geoffrey Hinton and David C. Plaut already published their paper about the prominent backpropagation algorithm for learning weights in multi-layered perceptrons in 1986 [6]. Reinforcement learning is even older and has its origins in trial-and-error learning of animal behavior [7]. Reinforcement learning is adopted from the concept of reinforcement in psychology. In psychology, reinforcement is associated with classical or operant conditioning.

Deep learning is mostly a reference to deep neural networks and had multiple breakthroughs in the last decade. Whilst other machine learning algorithms are converging to an asymptotic threshold in performance, deep neural networks manage to push this threshold further into the direction of a better performance. See Figure 1. This phenomenon is possible since deep neural networks unlike other machine learning algorithms, follow the universal approximation theorem. This means that deep neural networks manage to approximate real-valued continuous functions arbitrarily well [**?**]. More training data decodes more information that can be encoded following the implications of the universal approximation theorem. It follows, that more training data and more layers can improve the performance with the right requirements.

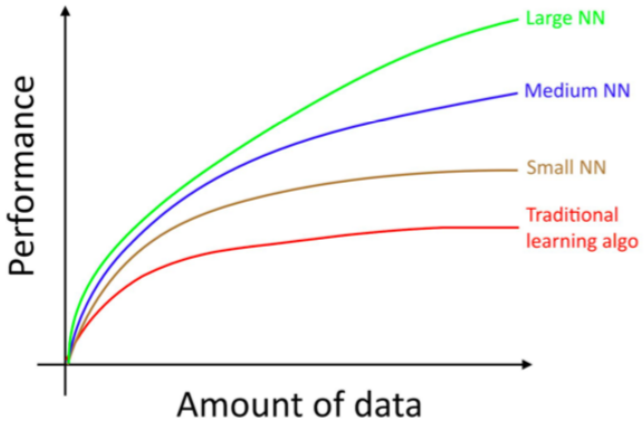


Figure 1: The advantage of deep learning compared to other machine learning algorithms [8].

Reinforcement learning is a machine learning paradigm where an agent interacts with its environment [2]. The environment is typically described with a Markov decision process, which consists of the following quad-tuple $< S, A, R, T >$.

- (S)tate
- (A)ction
- (R)eward
- (T)ransition Function

A Markov decision process is an extension of a Markov chain with an additional reward associated with each state in the environment. The complete agent-environment interaction loop is visually represented in Fig. 2.

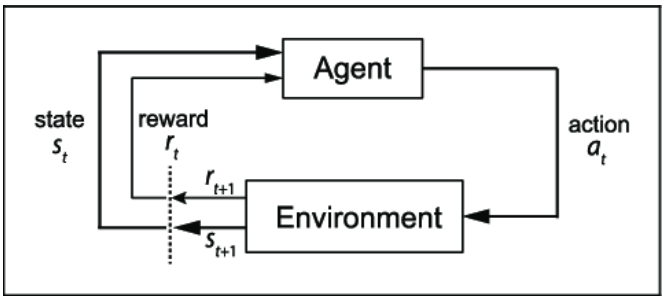


Figure 2: A visual representation of the agent-environment interaction loop [9].

This thesis falls into the domain of model-free reinforcement learning. Model-free reinforcement learning does not put any effort into predicting, understanding, or simulating the environment [5][2]. Model-free reinforcement learning tries to approximate behavior through interaction, without having any understanding of the environment. In contrast, model-based reinforcement learning tries to learn a model of the environment. This removes the dependency on the real environment and a trajectory can simply be generated by this model without interacting with the real world. Model-based reinforcement learning can highly increase the sample efficiency. See Fig. 3 for a possible split of domains in reinforcement learning.

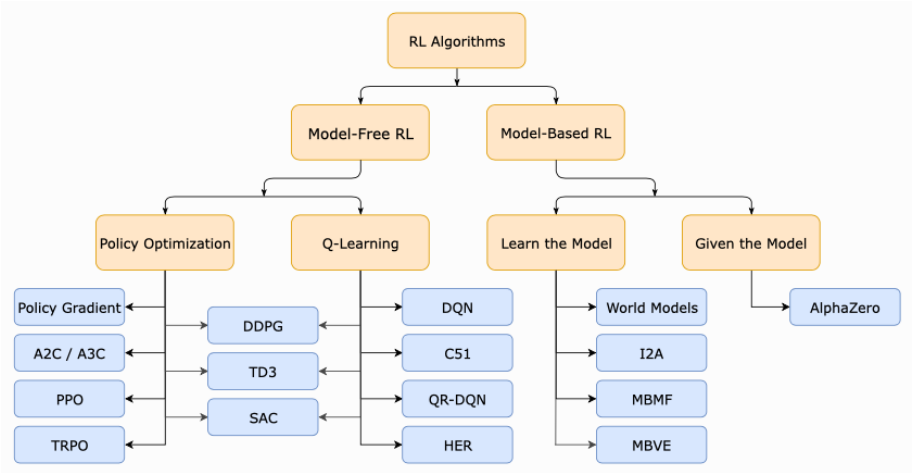


Figure 3: A tree describing one possible split of domains in the field of reinforcement learning [10].

As said above this thesis investigates model-free reinforcement algorithms only. More specifically the basic algorithms of policy optimization and Q-learning are compared. This split can again be seen in the reinforcement learning landscape from OpenAI in Fig. 3.

Combining deep learning and reinforcement learning comes with a specific problem. Whilst neural networks, like most machine learning algorithms, assume independent and identically distributed (i.i.d) data, reinforcement learning relies on dependent data

points [4]. The next state after a transition with the Markov decision process is always dependent on the current state. The Markov decision process uses the Markov property to mathematically describe this dependency: "The future is independent of the past, given the presence", more formally articulated. This property emerges since the current state has already some information about the past states encoded. This saves time and computational power since we do not have to recalculate back every single state the agent has been to before. This leads to highly correlated sequences of states, which is unsuited data for neural networks.

To overcome this problem an experience replay buffer is used: A batch of data points is collected through interaction with the environment. Once the experience replay buffer is full, independent and identical data points are randomly sampled and provided as input to the neural network. Now the i.i.d. data assumption holds and allows to describe the data-generating process with a probability distribution over a single example [11].

2.2 Deep Q Networks

The deep Q networks algorithm is based on the Q learning algorithm, where the Q table is replaced with a neural network [5]. The neural network gets the current observations as an input and outputs a suggested action. As explained in the section above, since neural networks and reinforcement learning presuppose contradicting assumptions a buffer saves experienced datapoints.

The variables and functions described in the table below are important for further investigation of the two deep reinforcement learning algorithms:

Variables & Functions	**Description**
State s	The current state the agent is in.
Action a	The action the agent takes
Reward r	The expected reward after taking the selected action
Next State s'	The expected state to be after applying the selected action
π	Policy representing the actions given the observations
γ	Discount factor to account less for later rewards
$\mathcal{L}$	Loss function which has to be minimized
$Q_{s,a}$	Function associating a value with every state action pair

Table 1: Important variables and functions for the deep Q networks algorithm.

A pseudo-code version of the deep Q networks algorithm was explained in the book hands-on deep reinforcement learning from Maxim Lapan [5]. An excerpt about the pseudo-code of deep Q networks from this book is shown below. For simplicity reasons, the buffer replay is excluded but an implemented version can be found in the code folder provided with this file.

1. Initialize Q(s, a) with some initial approximation
2. By interacting with the environment, obtain the tuple (s, a, r, s')
3. Calculate loss: $\mathcal{L} = (Q_{s,a} - r)^2$ if episode has ended
   or $\mathcal{L} = (Q_{s,a} - (r + \gamma \max_{a' \in A} Q_{s',a'}))^2$ otherwise

4. Update Q(s, a) using the stochastic gradient descent algorithm, by minimizing the loss with respect to the model parameters

5. Repeat from step 2 until converged

2.3 Policy Gradient

In contrast to the Q-learning algorithm above, the policy gradient algorithm optimizes the weights of a neural network directly. The goal is to maximize the neural network's output and consequently the discounted total reward. The variables described in the Table 1 are also important for the policy gradient algorithm:

The pseudo-code version of the policy gradient algorithm was explained in the book hands-on deep reinforcement learning from Maxim Lapan [5]. An excerpt about the pseudo-code of policy gradients from this book is shown below.

1. Initialize the network with random weights

2. Play N full episodes, saving their (s, a, r, s') transitions

3. For every step t of every episode k, calculate the discounted total reward for subsequent steps $R = \sum_{i=0} \gamma^i r_i$

4. Calculate the loss function for all transitions $\mathcal{L} = -\sum_{k,t} R \log\left(\pi\left(s_{k,t}, a_{k,t}\right)\right)$

5. Perform stochastic gradient descent update of weights minimizing the loss

6. Repeat from step 2 until converged

An important detail to recognize is the minus symbol in the loss function of the policy gradient algorithm [12]. It shows that we are maximizing the loss by applying stochastic gradient descent.

3 Experiments

The two model-free deep reinforcement learning algorithms were tested on the virtual cartpole-v1 environment from OpenAI [13]. The neural network architecture and hyperparameter values were shared across the two deep reinforcement learning algorithms as close as possible. Some variables and functions are existent in the deep Q networks algorithm but not in the policy gradient algorithm and vice versa.

3.1 Environment

The cartpole environment version 1 from OpenAI was used. It simulates the situation of a pole being attached to a cart through an un-actuated joint [13]. The cart can either move one step left with a provided value of -1 or one step to the right with a provided value of $+1$. The starting position is with the pole standing upright on the cart. One episode ends if the pole is more than 15 degrees away from upright, or the cart moves too far away from the center. The maximum reward possible in this environment is 500 unspecified units and is provided by the environment itself.

3.2 TensorFlow 2

TensorFlow 2 was used as a computational processing library for calculating the gradients and updating the weights in the neural networks. Keras, a sub-package of TensorFlow 2 was used for the construction of the neural network architectures. TensorFlow and Keras were chosen since there were already PyTorch implementations available and additional effort was needed to implement the two deep reinforcement learning algorithms. TensorFlow 2 was executed in eager mode, which resulted in more readable code and longer training times.

3.3 Hyperparameters

The process of choosing and comparing hyperparameters and their outputs in deep reinforcement learning algorithms is a scientific field of its own. They are often interrelated and dependent on each other. The hyperparameter values in this thesis were chosen based on existing literature and only modified marginally through experimentation [4]. Below is a table with the exact values chosen.

	Deep Q Networks	**Policy Gradient**
Optimizer	Adam [14]	Adam
Learning Rate	0.01	0.01
Neural Network Type	Feed Forward	Feed Forward
Number of Hidden Layers	2	2
Number of Neurons per Hidden Layer	16	16
Activation Function in Hidden Layers	tanh	tanh
Activation Function in Output Layer	linear	softmax
Epochs	500	100
Buffer Size	5000	5000
Minibatch Size	32	
Exploration Rate Gamma	0.95	
Minimum Exploration Rate	0.1	
Exploration Decay	0.994	

Table 2: The hyperparameters and their values for both deep reinforcement learning algorithms.

As can be seen above, the deep Q networks algorithm needs more hyperparameter values to be set than the policy gradient algorithm.

3.4 Hardware Specifications

The deep reinforcement learning algorithms were executed on my laptop, which leads to long training times. The specifications of this computer can be found in the table below.

Hardware	Specification
GPU	Intel HD Graphics 6000 1536 MB
CPU	1,8 GHz Intel Core i5
RAM	8 GB 1600 MHz DDR3
Hard Drive	256 GB SSD
Operating System	macOS Mojave

Table 3: Relevant hardware specification for training deep reinforcement learning algorithms.

The training time could have been significantly reduced with the use of an external **G**raphical **P**rocessing **U**nit. The other components of the computer are not great either for deep reinforcement learning purposes. Unfortunately, I did not have time and resources to set up a cloud environment with better fitting specifications.

3.5 Implementation

The complete implementation can be found in the code folder provided with this thesis. This paragraph only investigates the major implementational differences between the policy gradient algorithm and the deep Q networks algorithm.

3.5.1 Differences

One of the major differences between the two model-free deep reinforcement learning algorithms is their approach to optimizing the policy [5]. Deep Q networks optimize the value function according to the Bellman equations. Through optimization of the value function, the directly related policy of the agent becomes optimized as well. In contrast, the policy gradient algorithm directly optimizes the policy without the use of the value function and Bellman equations.

Since we can use a standard predefined loss from TensorFlow the computation of the loss is short and we simply have to call the .fit() function on the neural network.

```
def deep_Q_networks_compute_loss(state, action,
                                 reward, next_state):
    target = reward
    if not done:
        # here we can see the Bellman Equations
        target = reward + gamma * np.amax(
            neural_net.predict(next_state)[0]
        )
    target_function = neural_net.predict(state)
    target_function[0][action] = target

    # simply fitting the state to the target
    neural_net.fit(state, target_function,
                   epochs=1, verbose=0)
```

Unfortunately, the loss is not predefined as a standard loss in TensorFlow for the policy gradient algorithm. One possible implementation for optimizing the policy following the mathematical description from section 2.3 is shown in the function below.

```
def policy_gradient_compute_loss(state, action, weights):
```

```
    # one-hot-encode the action
    one_hot_encoded_action = tf.one_hot(
        np.array(action), n_acts,
        dtype=tf.float64
    )

    # create the loss function according to
    # the mathematical description
    with tf.GradientTape() as tape:
        predicted_action = logits_net(np.array(state))
        loss = -tf.reduce_mean(
            (tf.reduce_sum(
                one_hot_encoded_action *
                log_softmax(predicted_action), axis=1)) *
            np.array(weights)
        )
    gradients = tape.gradient(
        loss,
        logits_net.trainable_variables
    )

    # gradients and trainable vars need to be an iterable
    # project, since that is what the apply_gradient
    # requires
    grients_trainables_iterable = zip(
        gradients,
        logits_net.trainable_variables
    )
    optimizer.apply_gradients(grients_trainables_iterable)

    return loss
```

Another important difference between the two model-free deep reinforcement learning algorithms is their way of selecting an action. Once deep Q networks decided to exploit its knowledge about an action, it always takes the action with the highest expected reward. The policy gradient algorithm does not have any additional code about taking an action completely randomly. Instead, policy gradient directly samples from the policy, which is the probability distribution of actions, given the current observations.

```
def deep_Q_networks_get_action(state):
    if np.random.rand() <= exploration:
        return random.randrange(action_size)
    action_values = logits_net.predict(state)
    return np.argmax(action_values[0])

def policy_gradient_get_action(observation):
    logits = neural_net(observation)

    # constructing a probability distribution
    # with tensorflow_probability named tfp below
    policy = tfp.distributions.Categorical(logits)
```

```
        action = policy.sample()
        return action
```

For further implementation details, I suggest looking into the code files that are provided within the code folder of this thesis.

3.6 Results

Training deep reinforcement learning algorithms is computationally expensive [5]. Training took a long time since the hardware of the computer was not suited for deep reinforcement learning purposes. This is why the results in this section are limited to 10 training trials for each algorithm. A training trial consisted of 500 epochs in the case of the deep Q networks algorithm and 100 epochs in the case of the policy gradient algorithm. An epoch is defined as the agent's interaction with the environment from the beginning to the end. The end is either a winning or a losing state. In all plots, in this section, the y-axis represents the return and the x-axis represents the number of epochs. Since the maximum return possible in this environment is 500 units all y-axis display a maximum of 500.

3.6.1 Deep Q Networks

With the hardware specifications from Table 4 in section 3.4, one training trial took about one hour and 30 minutes.

The experiment managed to get a deep Q networks algorithm converging within the ten trials towards the 500 units of return on the environment within 500 epochs. A plot about this trial can be seen in Figure 4 on the left side. The worst of the ten trials using the deep Q networks algorithm never managed to exceed the 300 return mark. This plot is represented in the middle of Figure 4. Unluckily most of the ten trials looked more similar to the worst trial than to the best trial. This effect can also be seen in the plot on the right. This plot represents the average of all ten trials.

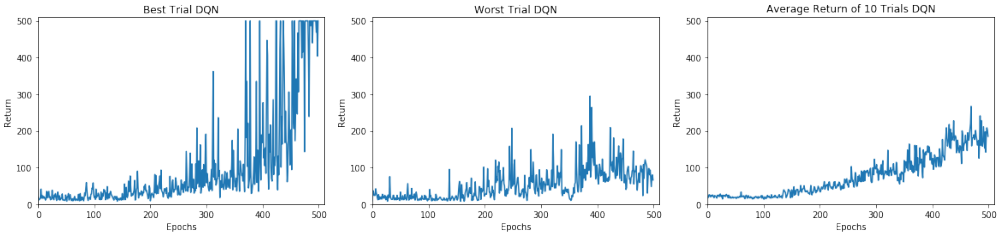


Figure 4: Out of ten training trials the left image represents the best, the middle image the worst and the right image the average return with the deep Q networks algorithm. The x-axis represents the amount of epochs whilst the y-axis represents the average accumulated reward or return.

3.6.2 Policy Gradient

With the hardware specifications from Table 4 in section 3.4, one training trial took about one hour and 30 minutes. For visualizing the absolute difference between the algorithms the x-axis shows a maximum of 500 epochs, even though 100 epochs were executed. Only 100 epochs per trial were executed since 500 epochs per trial would have trained for about five hours. Ten trials were desired, so we could average across the distinct trials. On the left of Figure 5, the best out of ten trials is represented for the policy gradient algorithm. In about 90 epochs the algorithm figured out how to balance

the pole on the cart perfectly. In the middle, the worst of the ten trials can be seen. Even the worst trial managed to peak to the maximum reward within 100 epochs. All the ten trials show a similar return curve. This effect is represented on the plot on the right of Figure 5, showing the average of all ten trials.

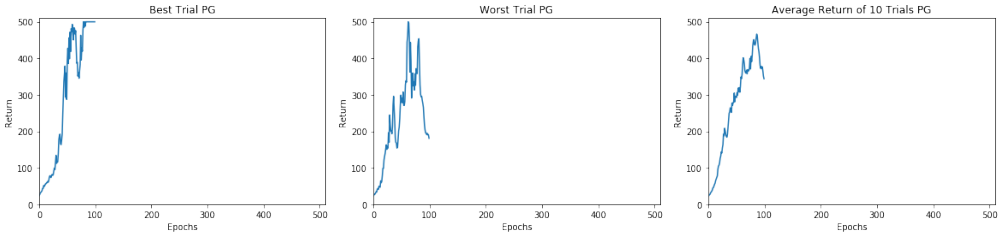


Figure 5: Out of ten training trials the left image represents the best, the middle image the worst and the right image the average return with the policy gradient algorithm. The x-axis represents the amount of epochs whilst the y-axis represents the average accumulated reward or return.

The dip after the algorithm already received the maximum return most likely occurs because the learning rate value of the neural network is too high for later epochs. Either reducing the learning rate value in general or dynamically reducing it over the numbers of epochs could downsize the dip.

3.6.3 Comparison

The policy gradient algorithm appears to be the absolute winner over the deep Q networks algorithm in terms of the number of epochs applied to the cartpole-v1 environment from OpenAI. The policy gradient's learning curve is steep, stable, and consistent.

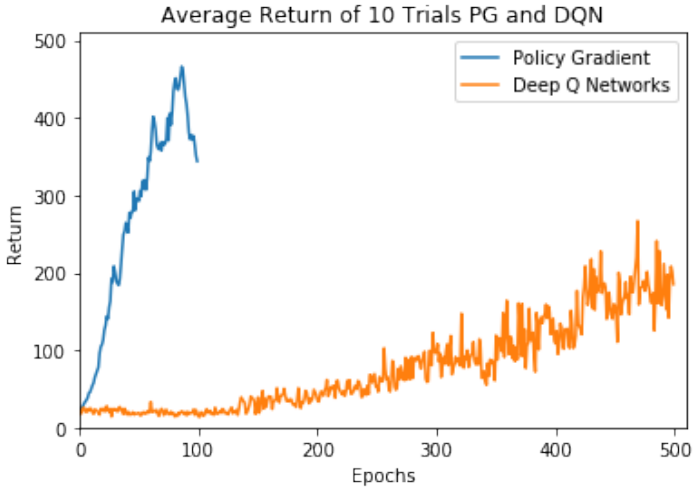


Figure 6: The advantage of the policy gradient algorithm in terms of epochs for the cartpole-v1 environment from OpenAI.

4 Discussion

This thesis showed that the policy gradient algorithm is better suited to solve the cartpole problem from OpenAI. Twenty independent training trials were executed on the cartpole environment. Ten of these twenty trials were executed with the implementation of policy gradient. The ten other trials used an implementation of the deep Q networks algorithm. The two algorithms were compared in terms of sample efficiency and the number of epochs. The plots showed clear results.

Stability and sample efficiency in terms of epochs makes the policy gradient algorithm shine for the cartpole problem. Nevertheless, the field of deep reinforcement learning is a highly stochastic field of machine learning. The success of training an agent to an appropriate result is dependent on many factors. Success for deep reinforcement learning is not guaranteed within a reasonable amount of time for many algorithms. The change of a random seed often leads to a completely different outcome.

Some resources show that deep Q networks have a better action sample efficiency than policy optimization algorithms [2][5]. The implementation of the deep Q networks algorithm in this thesis did not have a better sample efficiency. Somehow, the worst trial out of ten trials from the policy gradient algorithm was nearly one order of magnitude more sample efficient, than the best out of ten trails using the deep Q networks algorithm. A possible explanation for this result might be, that the shared hyperparameter values play against the favor of the deep Q networks algorithm.

A possible future direction could involve an investigation into more sophisticated model-free deep reinforcement learning algorithms and environments. Furthermore, an investigation into the bad sample efficiency of the deep Q networks algorithm would be interesting. A possible improvement could involve an advantage function, which describes how much better or worse an action is on average. It would also be interesting to study the family of deep reinforcement learning algorithms that share properties of both worlds, policy optimization, and Q-learning. The deep deterministic policy gradient algorithm falls into this family of deep reinforcement learning algorithms [12]. This algorithm can be seen as a deep Q learning algorithm for continuous action space agents.

Continuous action space algorithms are especially useful for state-of-the-art robotic locomotion, especially since this was the primary motivation for this thesis.

References

[1] G. Bledt, M.J. Powell, B. Katz, J. Di Carlo, P.M. Wensing, and S.Kim. MIT cheetah 3: Design and control of a robust, dynamic quadruped robot. *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[2] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction, 2nd Edition.* The MIT Press, 2018.

[3] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. *Google Brain, Berkeley Artificial Intelligence Research and University of California Berkeley*, 2019.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *DeepMind*, 2013.

[5] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more, 2nd Edition.* Packt, 2020.

[6] Geoffrey Hinton and David C. Plaut. Learning sets of filters using back-propagation. *Computer Science Department, Carnegie Mellon University. Pittsburgh, PA 15,713. C1.S.A.*, 1986.

[7] B.F. Skinner. Operant behavior. *American Psychologist.*, 1963.

[8] Andrew Ng. *Machine Learning Yearning.* 2018.

[9] Isaac Galatzer-Levy, Kelly V. Ruggles, and Zhe Chen. Data Science in the Research Domain Criteria Era: Relevance of Machine Learning to the Study of Stress Pathology, Recovery, and Resilience. 2018.

[10] Josh Achiam. Kinds of deep reinforcement learning. `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* 2018.

[12] Josh Achiam. Spinning up in deep rl. `https://spinningup.openai.com`.

[13] Cartpole-v1 description. `https://gym.openai.com/envs/CartPole-v1/`.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2014.

A Tables

A.1 Deep Reinforcement Learning Variables and Functions

Variables & Functions	Description
State s	The current state the agent is in.
Action a	The action the agent takes
Reward r	The expected reward after taking the selected action
Next State s'	The expected state to be after applying the selected action
π	Policy representing the actions given the observations
γ	Discount factor to account less for later rewards
$\mathcal{L}$	Loss function which has to be minimized
$Q_{s,a}$	Function associating a value with every state action pair

Table 4: Important variables and functions for the deep Q networks algorithm.

A.2 Hyperparameter Values

	Deep Q Networks	Policy Gradient
Optimizer	adam	adam
Learning Rate	0.01	0.01
Neural Network Type	Feed Forward	Feed Forward
Number of Hidden Layers	2	2
Number of Neurons per Hidden Layer	16	16
Activation Function in Hidden Layers	tanh	tanh
Activation Function in Output Layer	linear	softmax
Epochs	500	100
Buffer Size	5000	5000
Minibatch Size	32	
Exploration Rate Gamma	0.95	
Minimum Exploration Rate	0.1	
Exploratin Decay	0.994	

Important variables and functions for the deep Q networks algorithm.

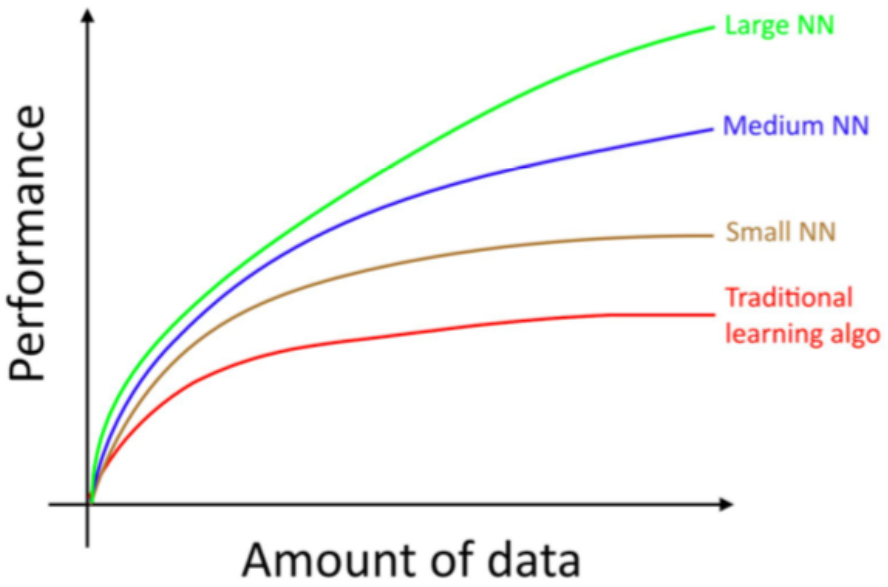
A.3 Hardware Specifications

Hardware	Specification
GPU	Intel HD Graphics 6000 1536 MB
CPU	1,8 GHz Intel Core i5
RAM	8 GB 1600 MHz DDR3
Hard Drive	256 GB SSD
Operating System	macOS Mojave

Relevant hardware specification for training deep reinforcement learning algorithms.

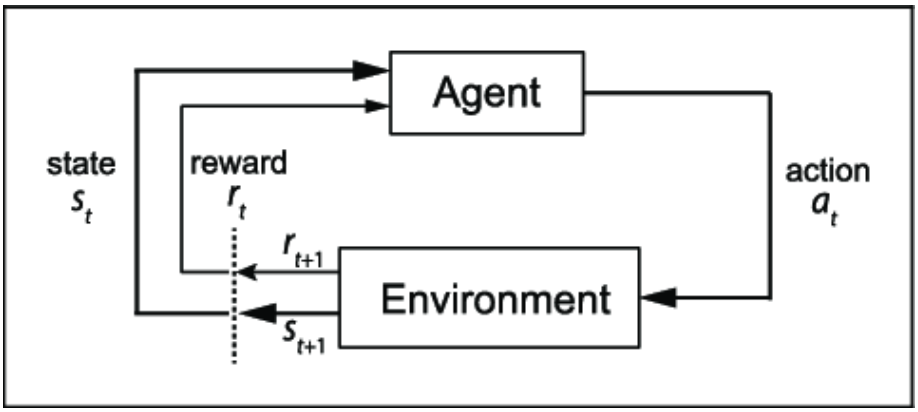
B Images

B.1 Deep Learning Performance



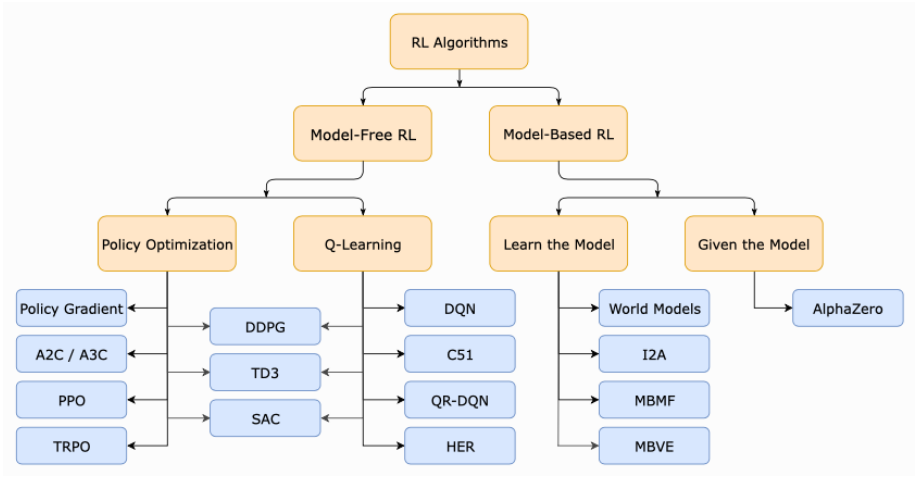
The advantage of deep learning compared to other machine learning algorithms [8].

B.2 Visualized Markov Decision Process



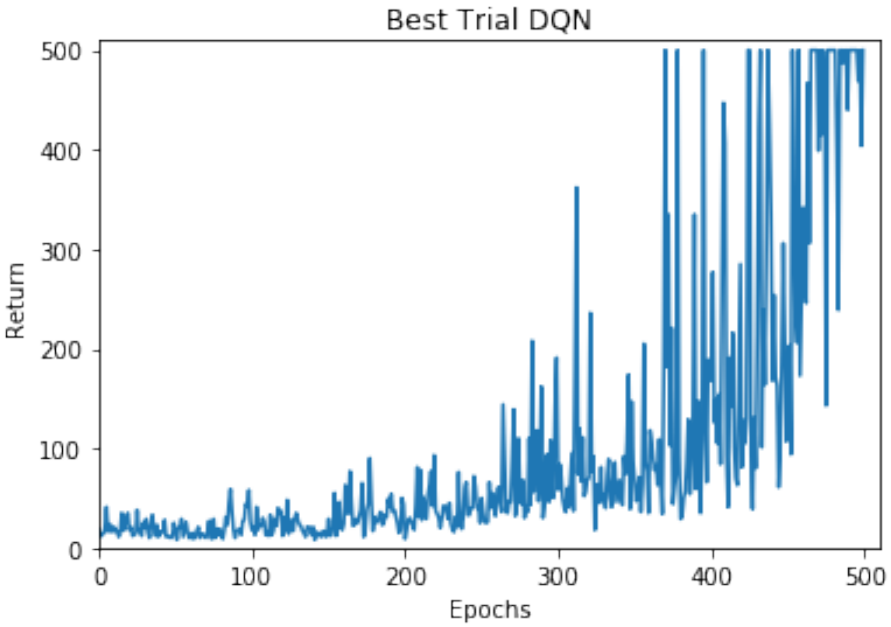
A visual representation of a Markov decision process in the context of reinforcement learning [9].

B.3 Reinforcement Learning Landscape

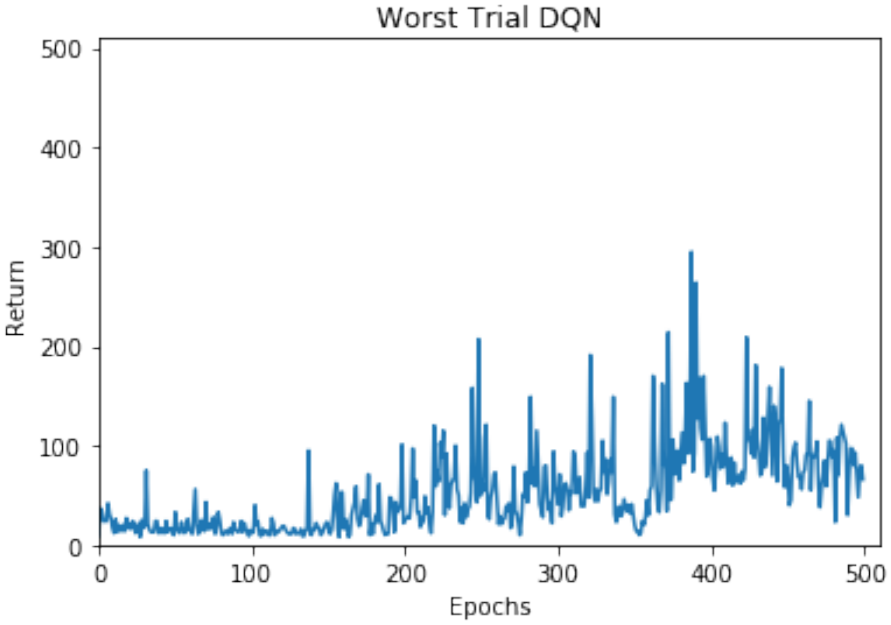


A tree describing one possible split of domains in the field of reinforcement learning. [10]

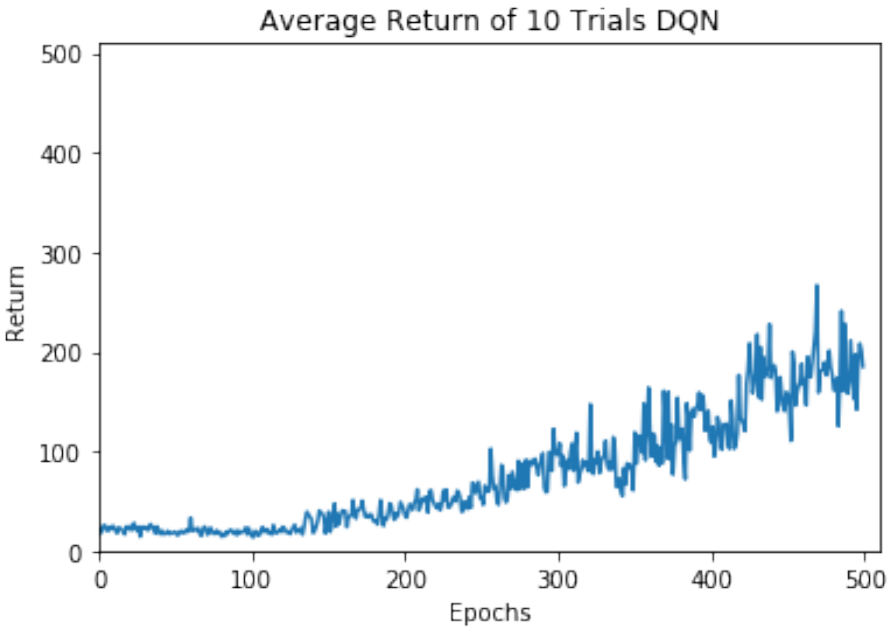
B.4 Deep Q Networks Best Trial



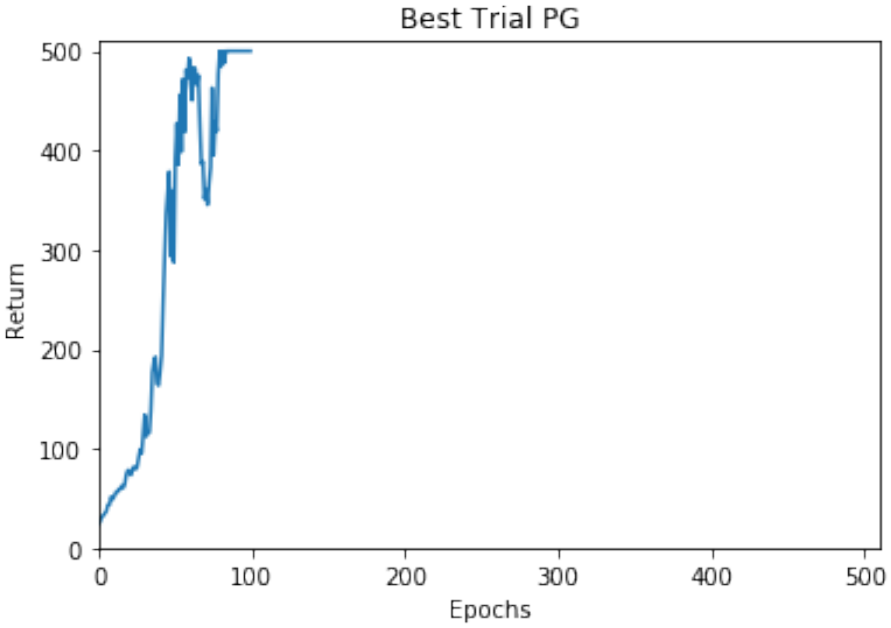
B.5 Deep Q Networks Worst Trial



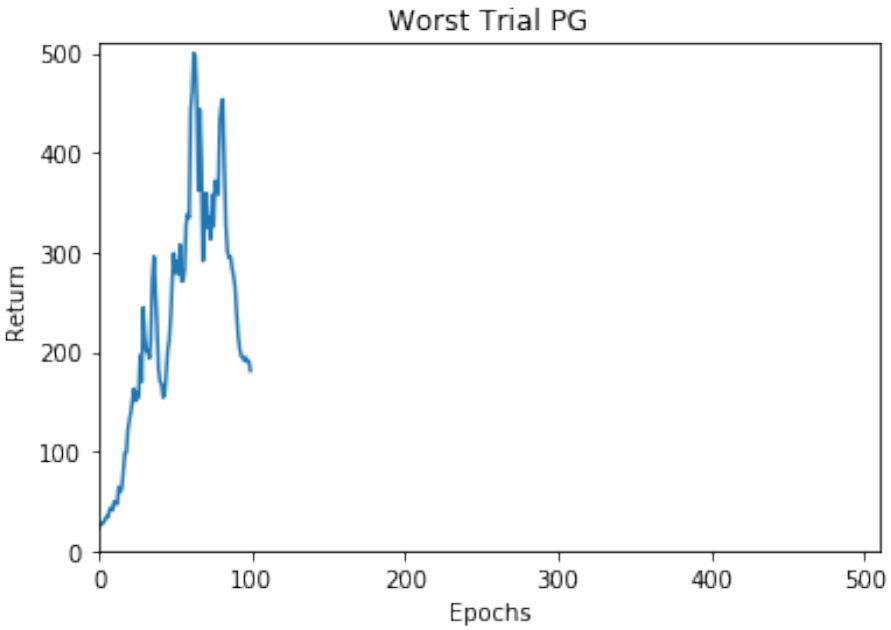
B.6 Deep Q Networks Average Trial



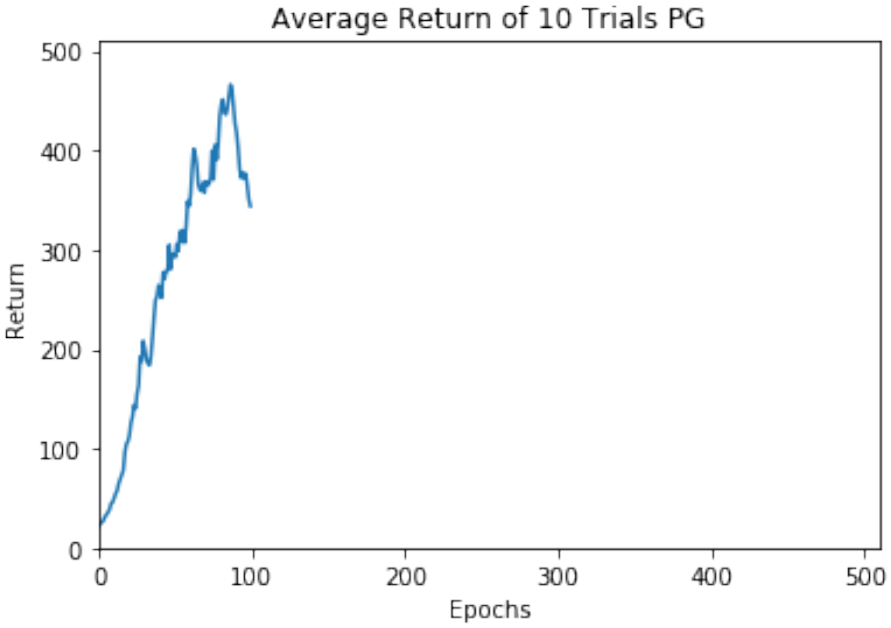
B.7 Policy Gradient Best Trial



B.8 Policy Gradient Worst Trial



B.9 Policy Gradient Average Trial



B.10 Deep Q Networks vs Policy Gradient