university of groningen / faculty of science and engineering

QVA-learning for playing the game of Snake

Bachelor's Project Thesis

Ilse Pubben, ilse.pubben@gmail.com,
Supervisors: Marco A. Wiering

Abstract: In this thesis we will introduce a new reinforcement learning algorithm, QVA-learning, which is a combination of QV-learning and advantage updating. We will test this algorithm on the game of snake and compare its performance with Q-learning and QV-learning. The state will be represented with vision grids of size 3×3, 5×5 and 7×7. We will also make use of an MLP as function approximator. We found that overall QVA-learning did not perform better than Q-learning or QV-learning. We also found that QVA-learning started learning earlier than the other algorithms when using a vision grid of size 7×7 (i.e., with more input nodes).

1 Introduction

Reinforcement learning algorithms are used to teach an agent certain behaviour by letting it interact with its environment (Sutton and Barto, 2018). One of the oldest and most well-known algorithms is Q-learning (Watkins and Dayan, 1992). Q-learning is an off-policy algorithm which tries to learn the state-action value function. Another more recent reinforcement learning algorithm is QV-learning (Wiering, 2005). QV-learning is an on-policy algorithm which learns the state-value function through temporal difference methods and uses this value function to learn the state-action value function. Another fairly old algorithm is advantage updating. Advantage updating learns the state-value function and the advantage function. The advantage function estimates the increase of the value function when a certain action is taken as opposed to the action that is considered the best (Harmon, III, and Klopf, 1995).

QV-learning has higher convergence rates than Q-learning (Wiering, 2005) and advantage updating learns faster than Q-learning (Baird, 1994). Therefore it would be interesting to see if combining these two algorithms will lead to faster and better results. This is exactly what we aim to do with the new reinforcement learning algorithm, QVA-learning, which we will introduce in this thesis.

QVA-learning aims to learn both the state-value function and the state-action value function just like QV-learning. On top of that it also aims to learn the advantage function based on the state and state-action value function.

In this thesis we want to answer the following research question: does QVA-learning begin earlier with learning and does it lead to better results than Q-learning and QV-learning? We will test this on the game of Snake using vision grids of several sizes as state representation. Vision grids have proven to be effective on the game of Tron, a similar grid-based game (Knegt, Drugan, and Wiering, 2018).

2 The game of snake

Snake is a well-known video game concept where the player has to maneuver an object on a bordered plane. As the object moves it leaves a trail behind. The objective of the game is to 'eat' items (apples in our case) by running into them, while avoiding hitting the borders of the plane and the trail the object is leaving behind. 'Eating' the items causes the trail to become longer. This concept originates from the arcade game 'Blockade' in 1976.

For the experiments performed in this thesis we implemented the concept as follows: the playing field is 12×12; the agent starts out with a head and a tail of length three; the goal is to 'eat' as many apples as possible, each time an apple is eaten the tail of the agent increases by one. The agent can choose between four actions: moving up, down, right and left. The starting location of the agent is always in the middle of the playing field. Figure 2.1 shows a snapshot of the game.

The playing field is relatively small, but we chose it to be this size to reduce training time. A small playing field increases the chance that the agent will stumble upon the apple while exploring, which

is necessary for the agent to start learning.

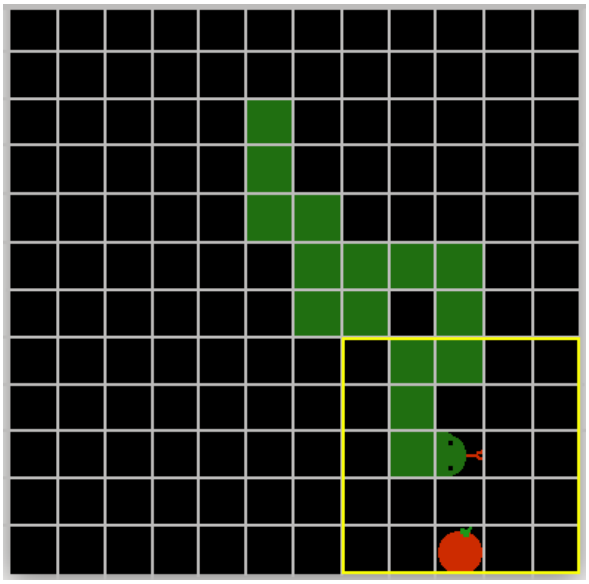


Figure 2.1: Snapshot from the game of snake. The yellow square represents what an agent with a 5×5 vision grid can see.

3 Preliminaries

In this section we will explain some of the main principles used in this thesis.

3.1 Reinforcement learning

Reinforcement learning is a machine learning paradigm which aims to teach an agent certain behaviour by letting it explore and interact with its environment. Such an environment is often explained in terms of a Markov decision process (MDP). For the experiments in this thesis we use a finite MDP which has the following components (Bellman, 1957):

- A finite set of states S, where $S_t \in S$ is the state at time t.
- A finite set of actions $\mathbb{A}$, where $A_t \in \mathbb{A}$ is the action taken at time t.
- A reward function $R(s, a, s')$, representing the reward achieved when taking action a in state s which leads to state s'. In the MDP we use for the game of snake the reward is 5 for scoring a point (e.g., 'eating' the apple), -10 for dying and -0.01 otherwise. Why the rewards are chosen as such will be explained in section 5.
- A transition function $T(s, a, s')$, which specifies the probability that we will end up in state s' when we are in state s and take action a.
- A discount factor $0 \leq \gamma \leq 1$, which discounts future rewards. Increasing γ increases the importance of future rewards.

The aim of the reinforcement learning agent is to learn the optimal policy π_*. A policy is a mapping of states to actions, $\pi : S \rightarrow A$. An optimal policy is the policy which leads to the maximum (discounted) cumulative reward G_t. G_t is calculated as follows:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} ...$$

We will now look at the value function, also called the state-value function (v_π). A value function is the expected($\mathbb{E}$) (discounted) cumulative reward when starting from state s and following a certain policy.

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

We will also look at the state-action value function (q_π), the expected (discounted) cumulative reward from state s when taking action a under policy π.

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

As mentioned before, we want the agent to learn the optimal policy π_*. An optimal policy is a policy which maximizes the state-value function V_* and the state-action value function Q_*.

$$\pi_*(s) = \arg\max_{a \in \mathbb{A}} q_*(s, a)$$

When the transition function is known V_* and Q_* can be computed using the Bellman optimality equation (Sutton and Barto, 2018):

$$V_*(s) = \max_{a \in \mathbb{A}(s)} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma v_*(s')]$$

$$Q_*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} q_*(s', a')]$$

However often the transition function is unknown and will have to be learned by the agent by interacting with the environment and gathering experience.

3.2 Algorithms

In this section we will explain the reinforcement learning algorithms that we compare in this thesis.

Q-learning: Q-learning was originally designed by Watkins (1989). It is an off-policy reinforcement learning algorithm, meaning that the policy used for generating the behaviour, the *behaviour policy*, differs from the policy that is improved, the *target policy* (Sutton and Barto, 2018). Q-learning aims to learn the action value function through the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a \in \mathbb{A}} Q(s',a) - Q(s,a)]$$

where $0 \leq \alpha \leq 1$ is the learning rate. With a lookup table Q-learning is guaranteed to converge to the optimal policy when the agent visits each state-action pair an infinite number of times. With a function approximator it might not converge but it will approach the optimal policy (Jaakkola, Jordan, and Singh, 1994).

QV-learning: QV-learning is an on-policy reinforcement learning algorithm designed by Wiering (2005). QV-learning aims to learn both the state-action value function and the state value function. The state-action value function is updated with the value function as opposed to temporal difference (TD) methods which are used by Q-learning.

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma V(S') - Q(S,A)]$$

The state value function itself is learned through TD methods.

$$V(S) \leftarrow V(S) + \beta[R + \gamma V(S') - V(S)]$$

where $0 \leq \beta \leq 1$ is the learning rate. Because QV-learning is an on-policy algorithm it is more likely to converge to the optimal policy than Q-learning when using function approximators (Wiering, 2005).

QVA-learning: QVA-learning aims to combine QV-learning and advantage updating. Advantage updating was shown to converge faster than Q-learning (Baird, 1994). The way QVA-learning is implemented is similar to QV-learning, but it uses an additional function, the advantage function $A(s,a)$. The advantage function is updated as follows:

$$A(s,a) \leftarrow A(s,a) + \alpha[Q(s,a) - V(s) - A(s,a)]$$

The actions are chosen based on the advantage model instead of the Q-model which is used by Q-learning and QV-learning. Algorithm 3.1 shows the full QVA-algorithm.

3.3 Multi-layer perceptron

As mentioned before, reinforcement learning aims to learn the state value function, state-action value function and/or advantage function, depending on the algorithm used. We could store these values in a lookup table. However, for the state-action value function (q(s,a)) alone this would lead to $|A| \times |S|$ entries and even more policies. For many problems, the state space is extremely large and therefore this is not practical. There is also no guarantee that we will encounter all state-action pairs while training. Therefore it is better to approximate these values with a function approximator. In our case this will be done with a multi-layer perceptron (MLP).

An MLP is a neural network with an input layer, at least one hidden layer and an output layer for which every node in a layer is connected to every node in the next layer. It attempts to match the actual output vector to the target output vector using backpropagation. Backpropagation is a learning procedure which tries to minimize the difference between the actual output vector and the target output vector by adjusting the weights of the connections between the nodes. Due to this the hidden nodes start to represent important features of the presented problem (Rumelhart, Hinton, and Williams, 1986).

The MLPs used for the experiments in this thesis will have one hidden layer with 100 nodes. The output layer of the MLP representing the state-value function will have one node, while the MLPs representing the state-action value function and advantage function will have four nodes, one for each action (figure 3.1). The number of nodes in the input layer will depend on the state representation which will be described in the next section. The hidden layer of the MLPs will use the sigmoid activation function:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

which returns a value between 0 and 1. The input and output layer both use a linear activation function.

Algorithm 3.1 QVA-learning

Require: Q learning rate $lr_Q \in (0, 1]$, V learning rate $lr_V \in (0.1]$, A learning rate $lr_A \in (0, 1]$, discount factor $\gamma \in [0, 1]$

```
for all episodes do
    S ← initial state
    for all steps of episode do
        Choose action a ∈ 𝔸 from S using policy derived from A (e.g. ϵ-greedy)
        Take action a, observe R, S'
        A(S,a) ← A(S,a) + lr_A[Q(S,a) − V(S) − A(S,a)]
        Q(S,a) ← Q(S,a) + lr_Q[R + γV(S') − Q(S,a)]
        V(S) ← V(S) + lr_V[R + γV(S') − V(S)]
        S ← S'
    end for
end for
```

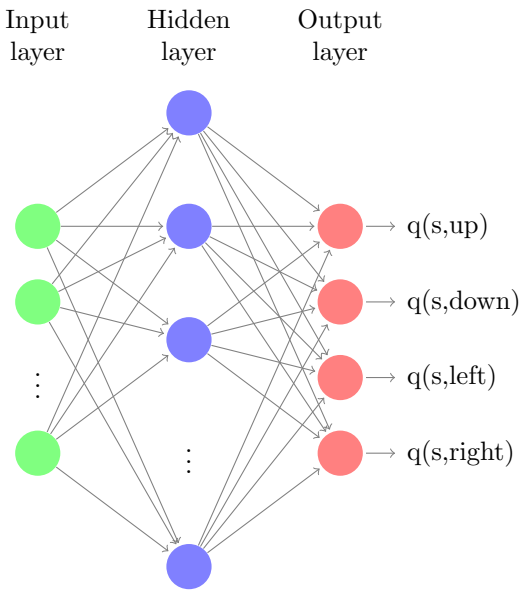


Figure 3.1: MLP to approximate q(s,a) with an output node for each action.

4 State representation

In this section we will describe the state representation that is used to teach the agent to play the game of Snake.

First we will look at the different elements of the game that are relevant to the agent. The agent needs to recognize the apple, the tail of the snake, the borders of the playing field and the head of the snake. For the game of snake we want the agent to avoid moving into a space with a tail or a border, thus we categorize the tail of the snake and the border of the playing field as the same thing, an obstacle. This leaves us with three different features: obstacle, apple, head. If we were to use a lookup table to represent the whole 12×12 field, there would be 144 positions for the head, 144 positions for the apple, and a maximum of 2^{144} possible states for all obstacles (of course not all of these obstacle configurations are possible). Therefore, we use an MLP to represent the value functions. If we were to supply the agent with the entire 12×12 playing field this would lead to $12 * 12 * 3 = 432$ input nodes, which would take a long time to train. To decrease the number of input nodes we will use vision grids, which have shown to increase learning speed and enhance performance by Knegt et al. (2018). A vision grid shows part of the environment from the perspective of the agent. It takes the shape of a grid with uneven dimensions around the location of the agent and shows only those things that fall within the grid. This will also allow us to get rid of the feature *snake head* as this will always be in the middle of our vision grid. This leaves us with only two different features, obstacle and apple. We will provide the agent with a separate vision grid for each feature, where 1 will indicate that the feature is present and 0 will indicate that it is not. Everything outside of the playing field will be considered as an obstacle. We will experiment with a vision grid of size 3×3, 5×5 and 7×7. To illustrate what such a vision grid could look like see figure 4.1 for the 5×5 vision grid of the snapshot displayed in figure 2.1. In addition to these vision grids we will also

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Vision grid obstacles Vision grid apple

Figure 4.1: 5×5 vision grid of situation shown in figure 2.1

provide the agent with the location of the apple relative to the agent and scaled between -2 and 2. For example, when the apple is in the same column and two spaces below the snake, as it is in figure 2.1, the agent will be given $locationX = 0/12 * 2 = 0$ and $locationY = -2/12 * 2 \approx -0.33$. The vision grids and the location will be combined in an array and given to the MLP. This will lead to $3*3*2+2 = 20$, $5 * 5 * 2 + 2 = 52$ or $7 * 7 * 2 + 2 = 100$ input nodes depending on the size of the vision grid used.

5 Experiments and results

In this section we will explain the experiments we performed and show their results.

To train the different algorithms the following parameter settings (Table 5.1) were kept the same throughout all experiments and were chosen based on preliminary tests. The agent was trained for 20000 epochs. We used a discount factor of 0.99. For exploration we used the ϵ-greedy exploration policy with ϵ starting at 0.05 and decreasing to 0 in 18000 epochs (the last 2000 epochs were trained without any exploration). Furthermore we used MLPs with one hidden layer containing 100 hidden nodes and the sigmoid activation function.

We will now describe the reward function we used for the game of snake (Table 5.2). The agent gets a reward of 5 for eating the apple, -10 for hitting a wall or its tail (ending the epoch) and a default reward of -0.01 for every other step. This default reward was chosen to ensure the agent does not get stuck in an endless loop.

This leaves us with one parameter that has not yet been set, the learning rate. The learning rate is especially challenging to tune for QVA-learning, because QVA-learning uses three different MLPs and consequently has three (different) learning rates. For the remainder of this section we will focus on tuning the learning rates. For each vision grid size we will do four experiments. First we made a distinction between training the agent with a constant learning rate versus learning rate annealing. Additionally we compared the performance when the learning rates are the same for all MLPs ($lrQ = lrV = lrA$) versus when the learning rates for the Q-MLP (rlQ), V-MLP (rlV) and A-MLP (rlA) have a ratio of $lrQ : lrV = 1 : \frac{1}{3}$ and $lrQ : lrA = 1 : 3$. In case of a constant learning rate $lrQ = 0.005$. In case of learning rate annealing lrQ linearly decreases from 0.005 to 0.0005 over the course of 20000 epochs.

Table 5.1: Parameters used in all experiments

Parameter	Value
Epochs	20000
Exploration policy	ϵ-greedy
Starting epsilon ϵ	0.05
Final epsilon ϵ	0
Final ϵ reached at epoch	18000
Discount factor γ	0.99
Hidden nodes	100
Hidden layer activation	sigmoid

Table 5.2: Reward function

	Reward
Eating apple	5
Dying	-10
Default	-0.01

In the experiments we measured the progress of the agent by the number of points it is able to score in an epoch. The agent scores points by eating the apple, each time the agent eats the apple it gains one point.

Each experiment consists of 50 simulations. A single simulation entails the training of the agent followed by an evaluation. In the training phase we take the mean points earned during each 1000 games and plot this every 100 games. After training the agent is evaluated by letting it play 100 games and looking at the mean points it is able to get.

5.1 Vision grid 3×3

In this section we will discuss the results of the experiments with a vision grid of 3×3.

The results of the experiment constant & equal learning rates can be found in table 5.3 Exp1. Q-learning performed significantly better than both QV-learning ($p = 0.0073$) and QVA-learning ($p = 0.0047$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.8477$).

The results of the experiment annealing & equal learning rates can be found in table 5.3 exp2. Q-learning performed significantly better than both QV-learning ($p = 0.0400$) and QVA-learning ($p = 0.0010$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.2223$).

The results of the experiment constant & different learning rates can be found in table 5.3 exp3. Q-learning performed significantly better than both QV-learning ($p = 0.0020$) and QVA-learning ($p = 0.0010$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.1972$).

The results of the experiment annealing & different learning rates can be found in table 5.3 exp4. Q-learning performed significantly better than both QV-learning ($p = 0.0054$) and QVA-learning ($p = 0.0010$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.3370$).

There was no significant difference in the performance of QVA-learing depending on the different learning rate settings ($p = 0.0648$) (figure 5.1).

Table 5.3: Evaluation of the models with vision grid 3×3. Mean over 40 runs with standard deviation(SD) shown.

Exp1: constant & equal learning rates:
$lrQ = 0.005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	18.33	1.74
QV-learning	17.15	1.76
QVA-learning	16.92	2.23

Exp2: annealing & equal learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	18.55	0.95
QV-learning	17.89	1.44
QVA-learning	17.40	1.19

Exp3: constant & different learning rates:
$lrQ = 0.005$, $lrV = lrQ/3$, $lrA = lrQ * 3$

Algorithm	mean	SD
Q-learning	18.30	1.97
QV-learning	14.21	7.30
QVA-learning	16.32	2.47

Exp4: annealing & different learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	18.58	0.99
QV-learning	15.43	6.28
QVA-learning	16.87	1.55

5.2 Vision grid 5×5

In this section we will discuss the results of the experiments with a vision grid of 5×5.

The results of the experiment constant & equal learning rates can be found in table 5.4 Exp1. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.2768$).

The results of the experiment annealing & equal learning rates can be found in table 5.4 exp2. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). QV-learning performed significantly better than QVA-learning ($p = 0.0010$).

The results of the experiment constant & different learning rates can be found in table 5.4 exp3. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). There was no significant difference between the performance of QV-learning and QVA-learning ($p = 0.5420$).

The results of the experiment annealing & different learning rates can be found in table 5.4 exp4. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). QV-learning performed significantly better than QVA-learning ($p = 0.0010$).

QVA-learning (figure 5.2) performed significantly

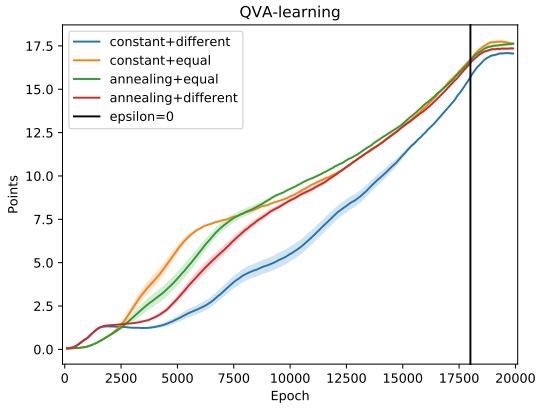


Figure 5.1: Learning curves of QVA-learning with a vision grid of 3×3 and different learning rate settings. Mean over 50 runs with standard error (SE) shown.

better with annealing & equal learning rates than with constant & different ($p = 0.0243$) or with constant & equal ($p = 0.0010$) learning rates. There was no significant difference in the performance of QVA-learning between annealing & equal and annealing & different learning rates ($p = 0.1278$). There was no significant difference in the performance of QVA-learning between annealing & different and constant & different ($p = 0.3321$). QVA-learning performed significantly better with annealing & different learning rates than with constant & equal learning rates ($p = 0.0010$). There was no significant difference in the performance of QVA-learning between constant & different and constant & equal learning rates ($p = 0.0747$).

5.3 Vision grid 7×7

In this section we will discuss the results of the experiments with a vision grid of 7×7.

The results of the experiment constant & equal learning rates can be found in table 5.5 exp1. There was no significant difference in performance between any of the algorithms ($p = 0.1774$).

The results of the experiment annealing & equal learning rates can be found in table 5.5 exp2. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). QV-learning performed significantly better than QVA-learning ($p = 0.0100$).

Table 5.4: Evaluation of the models with vision grid 5×5. Mean over 40 runs with standard deviation (SD) shown.

Exp1: constant & equal learning rates:
$lrQ = 0.005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	23.02	1.99
QV-learning	19.16	1.82
QVA-learning	18.43	2.42

Exp2: annealing & equal learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	23.45	0.85
QV-learning	22.29	1.33
QVA-learning	21.15	1.13

Exp3: constant & different learning rates:
$lrQ = 0.005$, $lrV = lrQ/3$, $lrA = lrQ * 3$

Algorithm	mean	SD
Q-learning	22.93	1.60
QV-learning	20.35	1.65
QVA-learning	19.82	2.71

Exp4: annealing & different learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	23.51	1.01
QV-learning	21.88	0.76
QVA-learning	20.60	1.10

The results of the experiment constant & different learning rates can be found in table 5.5 exp3. There was no significant difference in performance between any of the algorithms ($p = 0.5797$).

The results of the experiment annealing & different learning rates can be found in table 5.5 exp4. Q-learning performed significantly better than both QV-learning ($p = 0.0010$) and QVA-learning ($p = 0.0010$). QV-learning performed significantly better than QVA-learning ($p = 0.0010$).

QVA-learning (figure 5.3) performed significantly better with annealing & equal learning rates than with annealing & different ($p = 0.0010$) or constant & different ($p = 0.0010$) or constant & equal ($p = 0.0010$) learning rates. QVA-learning performed significantly better with annealing & different learning rates than with constant & differ-

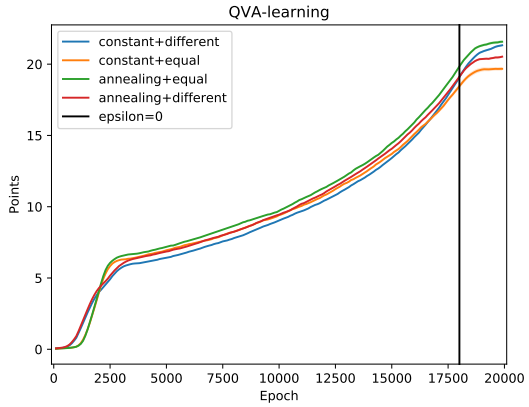


Figure 5.2: Learning curves of QVA-learning with a vision grid of 5x5 and different learning rate settings. Mean over 50 runs with standard error (SE) shown.

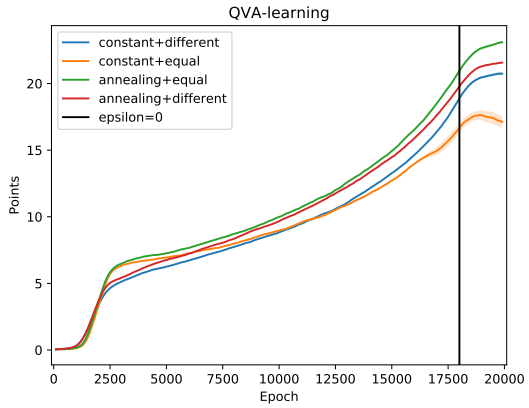


Figure 5.3: Learning curves of QVA-learning with a vision grid of 7×7 and different learning rate settings. Mean over 50 runs with standard error (SE) shown.

ent ($p = 0.0010$) or constant & equal ($p = 0.0010$) learning rates. QVA-learning performed better with constant & different learning rates than with constant & equal learning rates ($p = 0.0010$).

5.4 Learning curves

We will compare the learning curves of the experiment annealing & equal learning rates. We do this because QVA-learning seems to benefit from these settings and it would get too chaotic to look at all learning curves. On top of that the things we will discuss in this section are visible in the learning curves of all the experiments. Graphs of all the learning curves can be found in appendix A.

With a vision grid of 3×3 Q-learning started learning first, followed by QVA-learning. QV-learning was the slowest (figure 5.4). When the vision grid was increased to 5×5 the order in which the algorithms started learning remained the same (figure 5.5). However, the difference between the time when they started learning was a lot smaller. When the vision grid was further increased to 7×7 QVA-learning was the first algorithm to start learning, closely followed by QV-learning and Q-learning (figure 5.6).

Table 5.5: Evaluation of the models with vision grid 7×7. Mean over 40 runs with standard deviation (SD) shown.

Exp1: constant & equal learning rates:
$lrQ = 0.005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	16.60	10.85
QV-learning	14.32	5.30
QVA-learning	16.31	4.66

Exp2: annealing & equal learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	25.48	1.55
QV-learning	23.59	1.14
QVA-learning	22.88	1.01

Exp3: constant & different learning rates:
$lrQ = 0.005$, $lrV = lrQ/3$, $lrA = lrQ * 3$

Algorithm	mean	SD
Q-learning	18.58	9.65
QV-learning	18.81	6.78
QVA-learning	19.69	1.72

Exp4: annealing & different learning rates:
$lrQ = 0.005 \rightarrow 0.0005$, $lrV = lrQ$, $lrA = lrQ$

Algorithm	mean	SD
Q-learning	25.71	1.04
QV-learning	22.68	0.95
QVA-learning	21.46	0.83

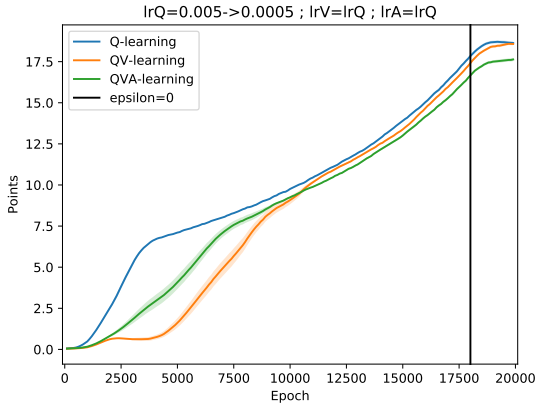


Figure 5.4: Learning curves of the algorithms with annealing & equal learning rates and a 3×3 vision grid. Mean over 50 runs with standard error (SE) shown.

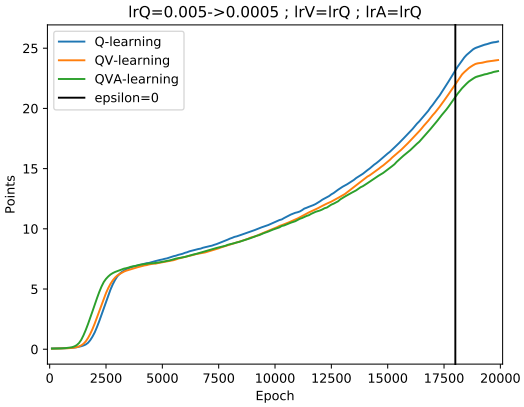


Figure 5.6: Learning curves of the algorithms with annealing & equal learning rates and a 7×7 vision grid. Mean over 50 runs with standard error (SE) shown.

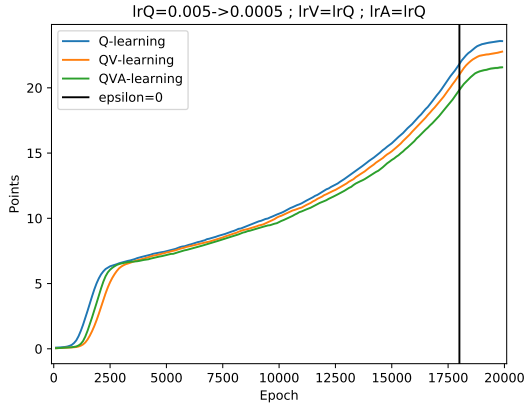


Figure 5.5: Learning curves of the algorithms with annealing & equal learning rates and a 5×5 vision grid. Mean over 50 runs with standard error (SE) shown.

6 Conclusion

In this paper we tested a new reinforcement learning algorithm, QVA-learning, on the game of Snake. We did this for three different state representations. Namely with a vision grid of 3×3, 5×5 and 7×7. For each vision grid size we performed four experiments which each handled their learning rates differently, constant & equal learning rates, annealing & equal learning rates, constant & different learning rates and annealing & different learning rates.

Q-learning performed the best in all but two of the experiments. It did not perform the best in the experiments with constant learning rates and a vision grid of 7×7. In these experiments none of the algorithms performed significantly better than any of the others. This is likely due to the high standard deviations of the data for these experiments. In most of the experiments there was no significant difference between the performance of QVA-learning and QV-learning. When there was a significant difference between those two algorithms QV-learning performed better.

QVA-learning seemed to benefit from annealing & equal learning rates. However it is unclear for now whether this is because QVA-learning actually benefits from equal learning rates between all models or that it benefits from starting with a learning rate of 0.005 as opposed to 0.015.

With a small vision grid (and consequently fewer input nodes) Q-learning starts learning earlier than QV-learning and QVA-learning. However as the vision grid increases QVA-learning becomes the first algorithm to start learning. This suggests that QVA-learning might be beneficial for problems with a larger input size.

7 Discussion

Although QVA-learning did not perform well in the experiments performed is this thesis it is important to test it further on different problems because of the 'no free lunch theorem'. The 'no free lunch theorem' states that a model that might work best on one problem, is not guaranteed to work best on another. Therefore it is important to try a model on multiple different problems before concluding whether or not it is a good model (Wolpert, 1996), (Wolpert and Macready, 1996).

In the experiments performed in this thesis QV-learning performed worse than Q-learning even though it has been shown to work better than Q-learning in several cases (Wiering, 2005), (Wiering and Van Hasselt, 2009). Seeing that QVA-learning is derived from QV-learning it would be interesting to see how QVA-learning performs in these situations.

One thing to note is that for a lot of the experiments the standard deviations were very high. These high standard deviations were caused by runs in which little to no learning occurred. QVA-learning did not have as many runs in which no learning occurred and consequently had lower standard deviations. This might indicate that QVA-learning is a more reliable algorithm.

As mentioned before, QVA-learning started learning earlier than Q-learning and QV-learning with a vision grid of 7×7. Therefore it would be interesting to see how QVA-learning performs on problems with more input nodes.

Lastly, due to the fact that QVA-learning keeps track of multiple value functions through multiple MLPs it has more hyperparameters to tune. In this thesis we only tested a limited number of parameter settings, spending more time on tuning these parameters could increase the performance of QVA-learning.

References

L. C. Baird. Reinforcement learning in continuous time: advantage updating. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2448–2453, 1994.

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

E. Mance Harmon, C. Baird Leemon III, and Harry A. Klopf. Advantage updating applied to a differential game. *NIPS*, pages 353–360, 1995.

Tommi Jaakkola, Michael Jordan, and Satinder Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

Stefan JL Knegt, Madalina M Drugan, and Marco A Wiering. Opponent modelling in the game of Tron using reinforcement learning. In *ICAART (2)*, pages 29–40, 2018.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Marco Wiering and Hado Van Hasselt. The qv family compared to other reinforcement learning algorithms. pages 101 – 108, 05 2009.

Marco A Wiering. QV (lambda)-learning: A new on-policy reinforcement learning algorithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, pages 17–18, 2005.

D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

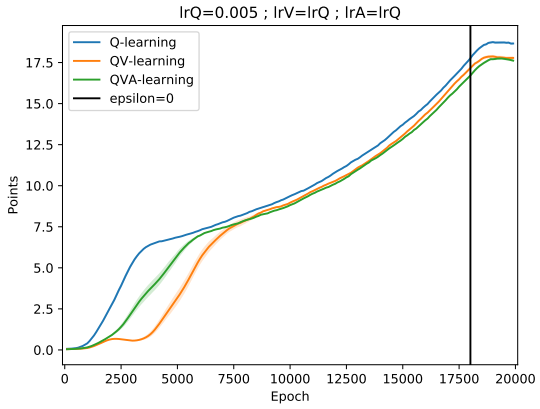
David Wolpert and William Macready. No free lunch theorems for search. 1996.

Appendices

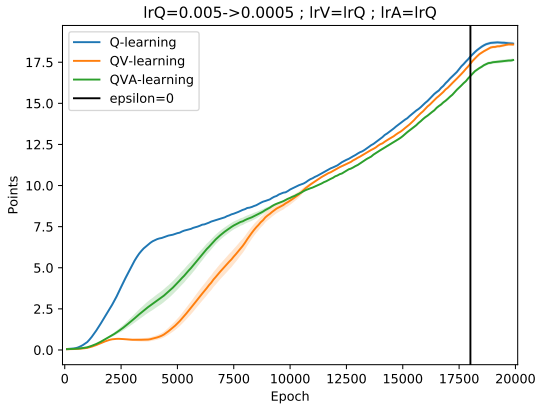
A Figures

Figure A.1: Learning curves of the algorithms with a vision grid of 3×3. Mean over 50 runs with standard error (SE) shown.

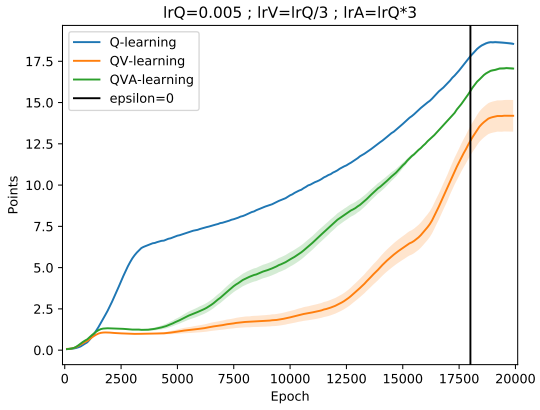
Exp1: constant & equal learning rates



Exp2: annealing & equal learning rates



Exp3: constant & different learning rates



Exp4: annealing & different learning rates

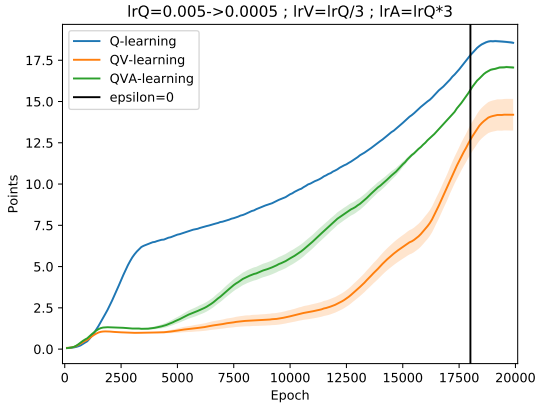
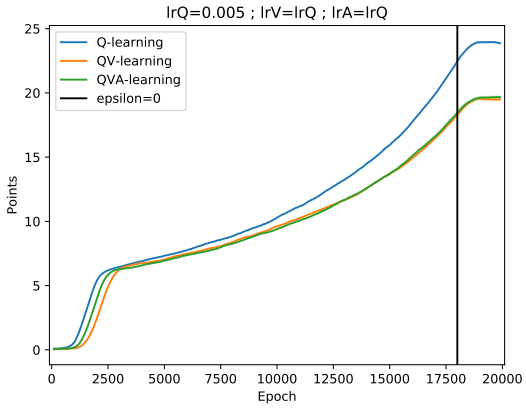
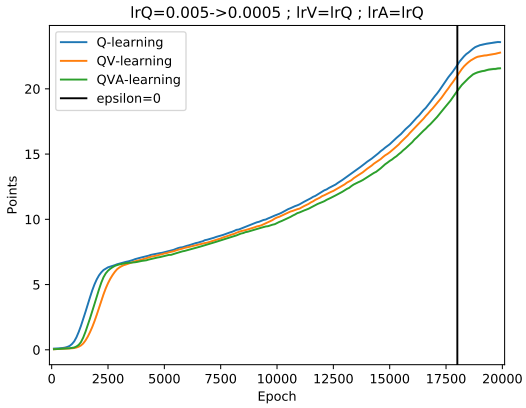


Figure A.2: Learning curves of the algorithms with a vision grid of 5×5. Mean over 50 runs with standard error (SE) shown.

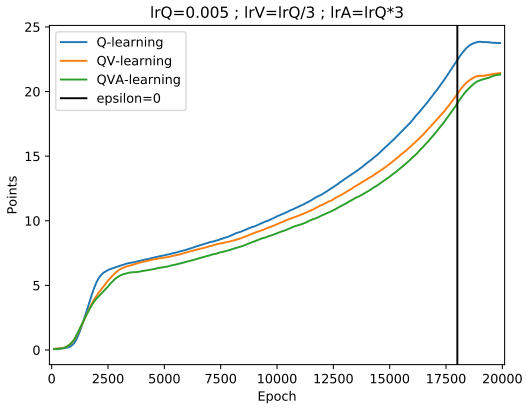
Exp1: constant & equal learning rates



Exp2: annealing & equal learning rates



Exp3: constant & different learning rates



Exp4: annealing & different learning rates

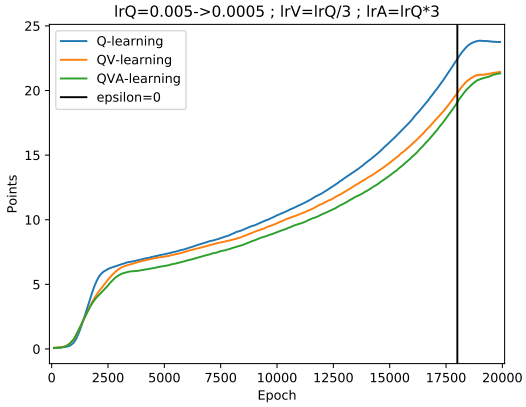
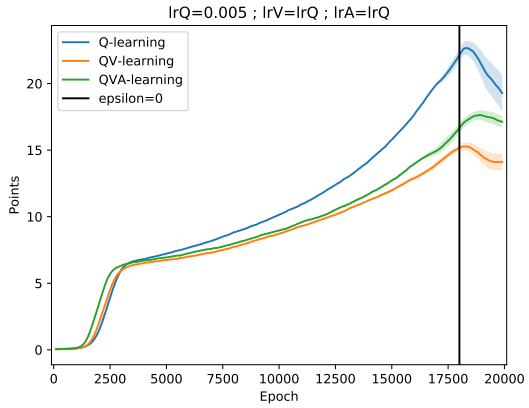
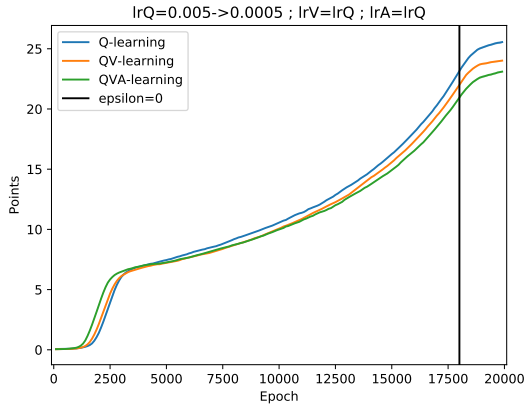


Figure A.3: Learning curves of the algorithms with a vision grid of 7×7. Mean over 50 runs with standard error (SE) shown.

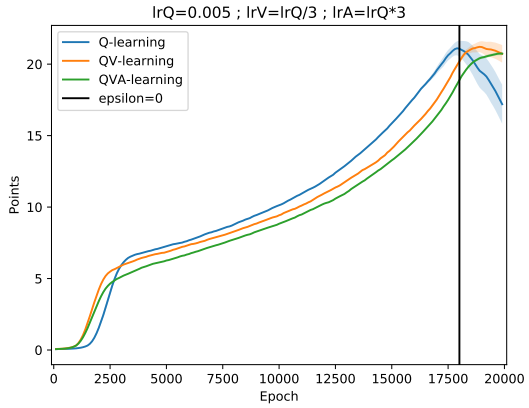
Exp1: constant & equal learning rates



Exp2: annealing & equal learning rates



Exp3: constant & different learning rates



Exp4: annealing & different learning rates