Reinforcement Learning

Data Mining Project

732A65

Rasmus Holm

December 16, 2017

Introduction

Reinforcement learning has made huge advances in the last couple of years, in particular with games such as when AlphaGo from Google DeepMind won against Lee Sedol in the game Go in 2016 [7]. It has also been possible to learn the computer to play Atari games [6] and this year at the largest eSports event in the world, the International 2017, OpenAI demonstrated that reinforcement learning can be used to defeat some of the best players in Dota 2 in 1 versus 1. [1]

Reinforcement is another large area in machine learning alongside supervised and unsupervised learning. The idea behind reinforcement learning is learning by interacting with an environment, which is how learning in nature happens very often. A child putting its hands on a hot stove will quickly feel a painful sensation and remember that was a terrible idea, and ultimately learn from it. Learning can also have positive effects such as when a mouse reaches the end of the labyrinth and finds a piece of cheese.

Reinforcement learning is a computational framework to this type of learning by having a so called *agent* interact with an *environment* and learn how to respond to different scenarios. The goal is to create learning methods that can learn an optimal *policy*, a mapping from the current sensory input to a decision in the form of an action, according to a behavior we want it to achieve. That might be from learning the computer to play tic-tac-toe to more complex behavior such as flying a helicopter where the agent do not posses full control of the environment.

In this report we are interested in investigating some of the fundamental building blocks of the reinforcement learning framework which will be presented in the next chapter. The questions we have posed are

1. Is the state representation important for learning?
2. Is it important for the reward function to reinforce behavior we want the agent to learn?
3. Is punishing bad behavior equivalent to reinforcing good behavior?
4. Is it better if the reward function combines reinforcement and punishment?

[1] https://blog.openai.com/dota-2/

Theory

In this chapter we will formalize the reinforcement learning problem mathematically and define the methods to solve our particular reinforcement learning problem.

Framework

To begin the definition of the reinforcement learning problem we first need to define a few fundamental terms that are required in the following formalization. The basic idea is to learn from interactions in order to achieve a goal. The learner is called the *agent* which interacts with the *environment* by performing *actions* in a sequential manner. The agent is continuously interacting with the environment and it responds based on those interactions that arise in the form of a representation of the *state* of the environment. Not only may the environment change its state, it also give rise to *rewards*, numerical values that the agent is trying to maximize. Figure 1 shows the interactions between the agent and the environment. A complete specification of an environment and how its rewards are determined defines a *task*, which is an instance of a reinforcement learning problem.

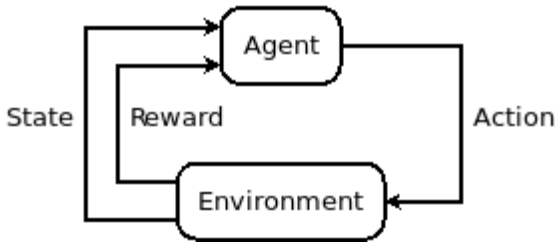

Figure 1: The agent-environment interaction loop.

This rather simple framework is actually abstract and flexible enough to be applied in many problems in many different ways. The actions could range from low level controls such as turning the wheels on the car to driving from point A to point B. Similarly, the state could represent current road surface condition to traffic flow in the nearby area. The reward will however always be a single numerical value since we require it to be comparable to be able to determine the best action, the action that will maximize the total amount of reward the agent receives. We will come back to what that means in the following section.

Markov Decision Process

We are going to limit our problem formalization to a special case with discrete time steps, finite state space, finite action space, fully observable state, and episodic tasks. It can be generalized to continuous time steps, infinite state space, infinite action space, partially observable state, and continuing tasks, but it is out of scope for this report. Bertsekas and Tsitsiklis give an in-depth coverage of reinforcement learning, also known as optimal control, in *Neuro-dynamic programming* [2]

Notation

Before continuing with the definition we need to present the most important notation that will be used from now on through the rest of the report.
$\mathcal{A}$: Action space
$\mathcal{S}$: State space
$A_t \in \mathcal{A}$: Action at time t
$S_t \in \mathcal{S}$: State at time t
$R_t \in \mathcal{R}$: Reward at time t
where $t \in \{0, 1, 2, \ldots, T\}$ since we have discrete time steps. Capital letters with a subscript such as A_t are regarded as random variables while a lowercase letter such as a is a realisation of a random variable.

Dynamics

We will now formalize the reinforcement learning problem through the *Markov Decision Process* (MDP), in particular the *finite* MDP, by sticking to the limitations mentioned above. This formalization assumes that the problem satisfies the *Markov property* which means that the one-step dynamics of the problem hold the following

$$Pr(S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t) = Pr(S_{t+1} = s', R_{t+1} = r | S_t, A_t),$$

which in words says that the future state-reward pair is independent of the past actions, states, and rewards given the current action-state pair. That is also equivalent to saying that the current state and action captures all the relevant information from the history. To simplify this formula we write it as

$$p(s', r|s, a) = Pr(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$

and it completely specifies the dynamics of the finite MDP, or equivalently the dynamics of the environment in which the agent is interacting. We can compute anything we might want to get from it such as the the *state-transition probabilities* and the *expected reward* for state-action pairs

$$p(s'|s, a) = \sum_r p(s', r|s, a),$$
$$r(s, a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a).$$

Policy

A *policy* π is a distribution over actions given states

$$\pi(a|s) = Pr(A_t = a | S_t = s).$$

It fully defines the behavior of an agent. It maps an observed state into an action and it could be stochastic in order to enforce *exploration* which we will come back to later. A policy could be as simple as a lookup table or a complicated search process which further makes this framework very flexible. We have used the former approach.

Reward

The *reward signal* is an essential part to learning that will guide the agent to the target behavior. This implies that it is crucial that during the construction of your reinforcement learning problem the underlying reward function that generates the rewards should reflect what you want accomplished. As a simple example, in the game of chess you would consider winning as good and losing as bad and so the reward should reflect that by for instance give +1 reward for winning and -1 reward for losing.

The sole goal of the agent is the maximize the total reward it receives of the long run so we define the return G_t as

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in [0, 1]$ is the *discount factor* that determines how much weight the agent should put on immediate rewards versus future rewards. By adding the discount factor $\gamma < 1$ to G_t we bound it so it does not become ∞. The goal of reinforcement learning is to maximize the return G_t but since it is a random variable what we actually do care about is the *state-value function*

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \middle| S_t = s \right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right],$$

and the *action-value function*

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \middle| S_t = s, A_t = a \right] = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right].$$

In words these functions expresses the expected return starting from state s and then following policy π, and the expected return starting from state s and performing action a and then following policy π respectively.

These recursive equations are known as the Bellman equations discovered by Richard Bellman in the 1950s [1]. What we are interested in is finding the Bellman optimality equations

$$\begin{aligned}
v_*(s) &= \max_\pi v_\pi(s) \\
&= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_*(s') \right], \\
\forall_s v_*(s) &\geq \forall_{s,\pi} v_\pi(s), \\
q_*(s, a) &= \max_\pi q_\pi(s, a) \\
&= \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right], \\
\forall_s q_*(s, a) &\geq \forall_{s,\pi} q_\pi(s, a),
\end{aligned}$$

and having those makes it trivial to find the optimal policy that maximizes the expected return which is the goal of the agent. It turns out that there always exists such a policy, possibly multiple such policies.

The optimal policy π_* is

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}}\, q_*(s,a) \\ 0, & \text{otherwise} \end{cases}$$
$$= \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right] \\ 0, & \text{otherwise,} \end{cases}$$

and we can clearly see that is it easier to acquire the optimal policy if we have access to the optimal action-value function. The problem is that there are, in general, no closed form solution to finding these optimal value functions and therein lies the difficulty of reinforcement learning. After an explanation of exploration and our problem formulation we will present a few iterative solution methods for approximating these optimal value functions, the action-value function in particular, that have been used in this report.

Exploration versus Exploitation

An important concept in reinforcement learning is the *exploration-exploitation* trade-off. It refers to the problem that the agent does not know how it should behave to maximize the expected return and thus have to interact with the environment to learn by trial-and-error. This means that it has to explore the environment by interactions and observe their consequences and the reward signals. This information does not contain what the "right" action was supposed to be but the agent can still construct a policy from it. Exploiting means that the agent is using its current policy to determine the best action, i.e., is greedy with respect to the policy. It can happen that the current policy is stuck in a local optima because the agent has not explored the search space thorough enough to build a good estimate of the performance of other actions.

A simple technique to enforcing exploration is by training using ϵ-greedy policy, see algorithm 1, that take the greedy action with probability $1 - \epsilon$ and a uniformly random action with probability ϵ. This is the approach taken in this report but there are more sophisticated techniques to balance these two acts.

Algorithm 1 ϵ-Greedy Policy

Require: State s, Action-value function q, Exploration rate $\epsilon \in [0, 1]$

function EPSILONGREEDYPOLICY(s, q, ϵ)
    sample $u \sim U(0, 1)$
    if $u < \epsilon$ **then**
        sample $i \sim U(\{1, 2, \ldots, |\mathcal{A}|\})$
        $a = \mathcal{A}_i$
    else
        $a = \underset{a}{\operatorname{argmax}}\, q(s, a)$
    return a

Snake

This report is meant to look at some of the fundamental building blocks of reinforcement learning and in order to do so we need a problem that we would like to solve. Snake is an old game[2] and has been around for a long time and there are many variations of it. In this report, the game is played out as simple as possible with only one player, i.e., the agent and no obstructing environment. The goal of the game is that you play as a snake by controlling the head and the aim is to eat as many apples as possible without colliding with either the snake's body or the edges of the game board. At each time unit the snake moves one unit in the direction it is headed and the difficulty is that each time you eat an apple your body becomes longer by 1 unit and thus larger portion of the game board is covered by it. In order to reduce the state space the game board is played on a 5x5 grid which can be seen in figure 2.

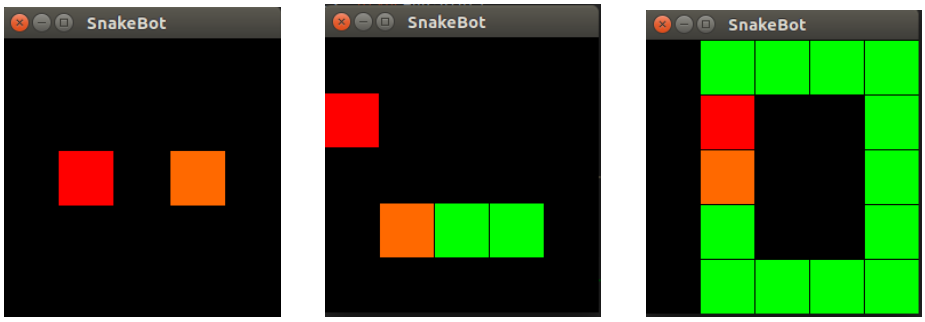

Figure 2: 5x5 board of Snake where the snake consists of the orange (head) and green parts (body) and the red part is the apple.

A term that was mentioned previously was *episodic task* which refers to problems that have a natural notion of final time step which Snake does, that is when the snake collides with something. We denote this state as the *terminal state* which will follow with a reset of the game in the same starting state independently of how it ended.

[2] https://en.wikipedia.org/wiki/Snake_(video_game)

MDP

With the previous formalization of the MDP we can now define Snake's MDP and its a rather simple definition. We assume that the actions are completely deterministic and the state transitions are also deterministic. Given our action space, $\mathcal{A} = \{\text{North}, \text{South}, \text{West}, \text{East}\}$, that means that if the snake moves north it will move north, not any other direction, and the next state will reflect that. Figure 2 shows three different states that the agent will be able to observe completely. There is no stochasticity in the environment except the positioning of the apple which will be randomly placed uniformly whenever eaten so there is only one food source at any single point in time. We have chosen that each eaten apple gives the player 100 game points and at the end of the game the total score is thus 100 $\times$ number of apples eaten.

Temporal Difference Learning

Temporal Difference (TD) learning is a learning methodology that learns directly from raw experience without a model of the environment's dynamics. It update new estimates partially on previous estimates, by bootstrapping, without waiting for the final outcome. The algorithms, or learning methods, that have been used in this paper are all based on TD learning and will be described below. Algorithm 2 shows the general outline of how it works and what will differ between the algorithms is line 9, the update rule of $Q(S, A)$. We have used a tabular representation of the Q-function, i.e., we map a state-action pair to an action-value using a hash table.

Algorithm 2 TD Learning Algorithm.

Require: Learning rate $\alpha \in [0, 1]$, Discount factor $\gamma \in [0, 1]$

```
1:  function TDLearning(α, γ)
2:      Initialize ∀_{a∈A, s∈S} Q(a, s) arbitrarily and Q(terminal state, ·) = 0
3:      for each episode do
4:          Initialize S
5:          Choose A from S using policy π (ε-greedy) derived from Q
6:          while S is non-terminal state do
7:              Take action A, observe state S′ and reward R
8:              Choose A′ from S′ using policy π (ε-greedy) derived from Q
9:              Update Q(S, A) based on (S, A, S′, A′, π, α, γ)
10:             S = S′
11:             A = A′
12:     return Q
```

Q-Learning

Q-learning is a widely used algorithm that was developed by Watkins [10] and variations of it is heavily used today combined with the emergence of neural networks [4], [9]. The update rule is defined as

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t). \right]$$

It is a so called *off-policy* method which is indicated by the max operation. That means it uses a greedy algorithm to estimate the return at the next step rather than following its current policy, therefore is off its own policy.

Sarsa

Sarsa on the other hand is an on-policy method for estimating the action-value function and its name refers to that we use the quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, and it has the update rule

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right].$$

The difference betwen Sarsa and Q-learning is that it does not use the max operator and is rather using its current policy to estimate the next action-value. There is also a natural extension to Sarsa called *Expected Sarsa* which takes the expected action-value estimate under the current policy. The update rule is defined as

$$\begin{aligned} Q(S_t, A_t) &= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E} \left[Q(S_{t+1}, A_{t+1}) | S_{t+1} \right] - Q(S_t, A_t) \right] \\ &= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \end{aligned}$$

Other Learning Methods

There are many other learning methods that are not used in this report that have interesting properties. Methods such as those based on *dynamic programming* which is an important optimization method, *monte carlo*, and also extensions to those as well as to the algorithms presented above. We highly recommend the book written by Sutton and Barto [8] for more information.

Method

In order the answer the research questions presented in the introduction, we have divided the experiments into two parts. One experiment in which we look at the correlation between the reward function and the goal of the game and another where we investigate the performance of different state representations.

Reward Experiment

Our hypothesis is that it is essential that the reward function reflect what the goal of the problem is and in our case that is to gather as many apples as possible, preferably in fewer time steps. To perform this experiment we have constructed 9 reward functions with all the combinations of the following characteristics

Reward at the end of the episode
$\pm$ score or 0
Reward after each time step
± 1 or 0

The reasoning for the construction of these particular reward functions is the answer the research questions.

Is it important for the reward function to reinforce behavior we want the agent to learn? If this is true we expect the reward functions that gives positive score rewards regardless of reward per time step to have the best performances.

Is punishing bad behavior equivalent to reinforcing good behavior? If this is true we expect the reward function that gives 0 rewards at the end of the episode and -1 reward each time step has the similar performance to the ones hypothesised as the best in the previous question.

Is it better if the reward function combines reinforcement and punishment? If this is true we expect the reward function that gives positive score rewards at the end of the episode and -1 reward at each time step has the best performance.

In order to do this experiment we also need to choose state representations and we choose to use two different. The board state with all additional information and the directional distance state with all additional information, as described below, since they contains the most amount of information.

State Experiment

In order to determine the importance of the the state representation we have decided upon 4 different representations. Additional, we have decided to test whether it is important for the state to contain the board dimensions to help the agent to infer deadly state-action pairs. And also if the current game score is important for the agent to be able to predict the future rewards more accurately.

Intuitively, it would be reasonable to assume that if the agent knows the dimensions of the board it could more easily infer state-action pairs that result in a terminal state and thus end the game with potentially less total reward.

By the same reasoning is it probably a good idea to give the agent access to its current score for it to improve the estimation of future rewards and thus making better decisions.

The 4 state representations are the following

- **Board**: Represents the complete board state as the game is designed.
- **SnakeFood**: Represents coordinates of the snake's head/body and the coordinate of the food source.
- **Directional**: Represents the direction the snake should travel to get to the food source without following the dynamics of the game. That means it might not be possible to change to that direction with a single action, e.g., the snake travels east but should travel west.
- **DirectionalDistance**: Represent the same direction as above but also contains the manhattan distance from the head of the snake to the food source.

and each state is split up into 4 different states considering with/without board dimensions and with/without score, which gives 16 states in total.

To perform this experiment we also require a reward function which was decided based on the reward experiment by picking the best performing one. It turned out to be the reward function that gave 0 reward each time step and positive score as reward at the end of the episode.

Hyperparameters

For all the experiments we have used the following hyperparameters based on trial-and-error experimentation

- ϵ-greedy policy with exploration rate $\epsilon = 0.15$
- Training for 1 million episodes
- Initialized the Q-function with 0s

Learning rate α	Discount factor γ
Q-Learning	
0.85	0.85
Sarsa	
0.15	0.95
Expected Sarsa	
0.15	0.95

Result

In this part of the report we will present the result that have been gathered from our two experiments on reward functions and state representations. It will be presented in the order aforementioned.

Reward Experiment

In this section we will present the results given by the reward experiment. All the following plots will show the same results for the different algorithms. Note that the left column contains result using the board state representation and the right column is result for using the directional distance state. All the results have been averaged over 3 different experiments.

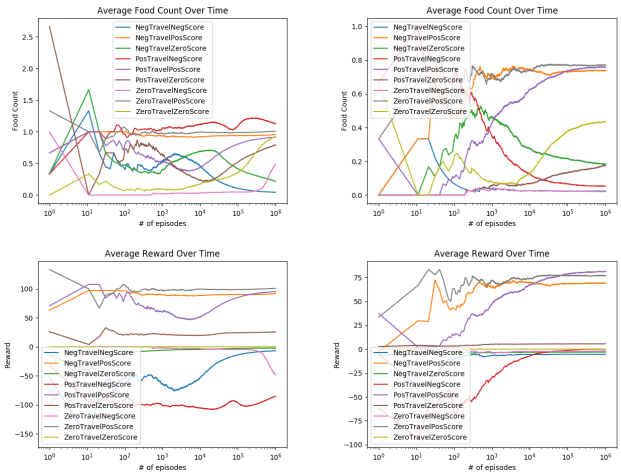

Figure 3: Results generated by Q-Learning.

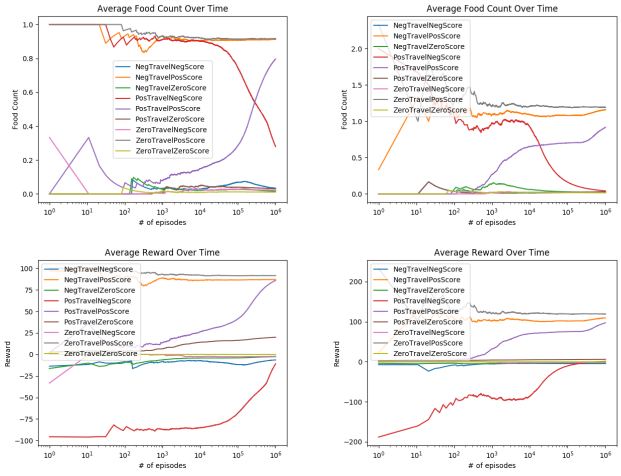

Figure 4: Results generated by Sarsa.

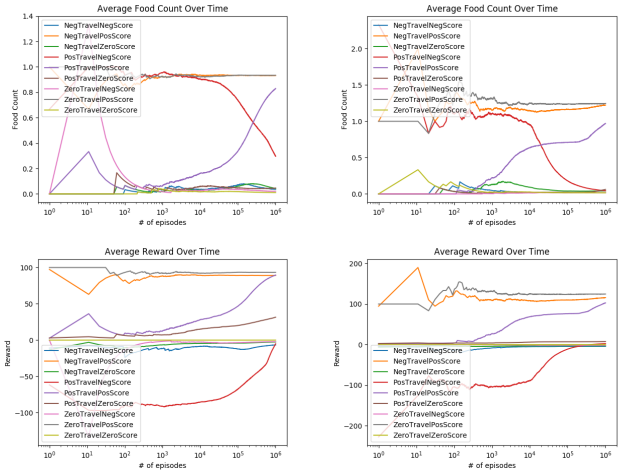

Figure 5: Results generated by Expected Sarsa.

The upper plots shows the average food the agent learns to eat on average which is the goal to maximize and the lower plots shows the average reward as a function of time for the different reward functions presented in the method.

What we mean by "average over time" is that we take the average of all data points up to (inclusive) the time we are interested in, i.e., a moving average with an infinite tail. In our case time refers to the number of episodes we have trained the model.

We also looked at the correlation between the curves in the column wise plots above shown in table 1.

	Avg. Correlation	Avg. Abs. Correlation
	Q-Learning	
Board State	0.536742184842	0.945093533818
Directional Distance State	0.49589001223	0.973537777401
	Sarsa	
Board State	0.395990504669	0.978606301141
Directional Distance State	0.521129553421	0.963340845075
	Expected Sarsa	
Board State	0.457350122159	0.898156494946
Directional Distance State	0.507751828694	0.968662042717

Table 1: Correlations between rewards and food counts.

State Experiment

In this section we present the results from the state experiment. For each algorithm we have looked at the 16 states presented in the method and they are presented in three plots. Since the goal is to look at the objective of the problem we plot the average food count each state gets over 1 million episodes of training. Same as before we have averaged the result over 3 experiments. Since there is an apparent pattern in the results each result has been divided into 3 plots to emphasize and demonstrate that.

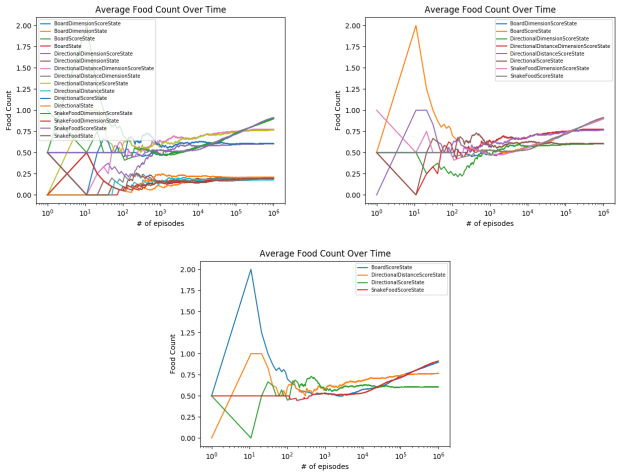

Figure 6: Results generated by Q-Learning.

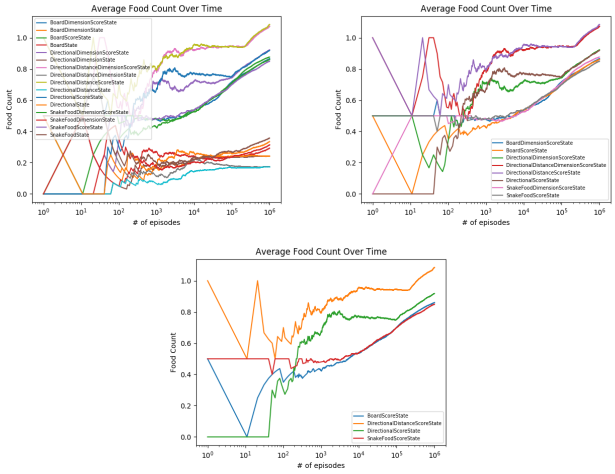

Figure 7: Results generated by Sarsa.

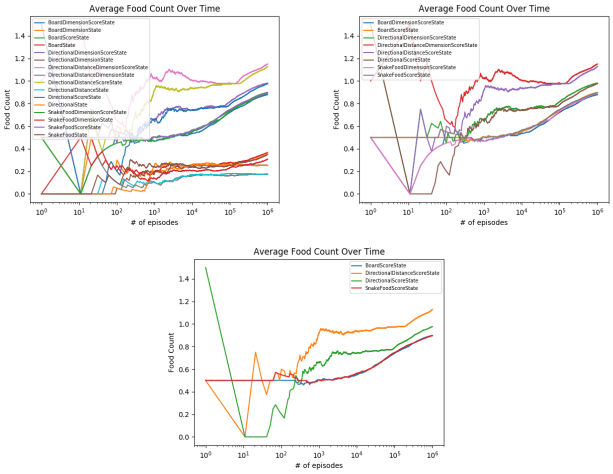

Figure 8: Results generated by Expected Sarsa.

The 3 plots for each algorithm shows the result for all the states (top-left), the states that contain score information (top-right), and those that contain score information but not board dimensions (bottom-center).

Discussion

In this part we discuss the results that we have gotten from our two experiments and point out what we have learned. We discuss them in the order they presented previously.

Reward Results

The first thing we noticed from the reward experiment is that Q-learning and the two Sarsa algorithms behave rather differently. Q-learning does not seem to be as susceptible to, what we would consider, bad reward functions that do not reflect the goal such as the *ZeroTravelZeroScore*-state. For the Sarsa variants these types of states seem to be hopeless for learning as was our hypothesis. Since Q-learning uses the max operator it is known to be overestimating the value function which might be one reason for not perform as badly. Methods for negating this problem have been developed such as double Q-learning by van Hasselt [3].

However, since the average food count is around 1 this means that the agent only eats a single apple which may not necessarily indicate that it has learnt anything and is just randomly moving around and gets lucky. By actually watching the agent play the game we have seen that it actually learns to navigate to the first food source but then commits suicide by colliding with the game border. On rare occasions when the next food gets located nearby it eats two of them before colliding with the game border.

The results suggest that it is actually important that the reward function to reinforce behavior we want the agent to learn. This is suggested by the fact that all the states that give a positive score reward at the end of each episode have better performance than the other states in almost all cases. This is further supported by the fact that the reward function and the actual food counts are highly correlated as shown in table 1 which indicate that is it important that the reward function reflect the goal that we want the agent to achieve. So the answer to our question **Is it important for the reward function to reinforce behavior we want the agent to learn?** is yes.

We asked the question **Is punishing bad behavior equivalent to reinforcing good behavior?** and the results suggests that this is not necessarily true. For that to be true we expected that the reward function *NegTravelZeroScore* to have similar performance to those rewards that give positive score reward at the end of an episode. This is not the case at all for this problem and by observing the agent's behavior given the former reward function we saw that it travels to the nearest border and commits suicide. One could argue that this reward function do not reflect truly bad behavior, but we defined bad behavior as randomly strolling around aimlessly as bad.

This brings us to our last question **Is it better if the reward function combines reinforcement and punishment?** and what we can observe is that *NegTravelPosScore* and *ZeroTravelPosScore* have very similar performance with the latter slighly better. This suggests that it is not necessary to combine reinforcement with punishment for best performance. Once again, one can argue that our definition of bad behavior is superficial and that this result may not hold when there is a clear definition of what bad behavior is. What is obvious is that good behavior has to be reflected in the reward function, but inconclusive

whether combining reinforcement and punishment give rise to best performance or not.

State Results

From the plots of all states we can clearly see a clustering and these clusters represent states with and without the score information. Those states that do contain score information have considerable better performance in terms of the average number of eaten apples. Looking closer at those states we can see that including board dimensions do not actually make much of a difference. This indicates that adding uninformative, in the sense that it is constant, information neither improve nor degrade performance. Overall it seems that the Sarsa methods have similar performance and that they are better than Q-learning.

As with the reward functions we can see that Q-learning and Sarsa methods do differ. The best performing states for Q-learning are *BoardScoreState* and *SnakeFoodScoreState* which basically stores the same information in different formats. *DirectionalDistanceScoreState* scores the best for the two Sarsa methods and this have probably to do with the fact that Sarsa is on-policy and Q-learning is off-policy. This state representation contain information about the next action that should be considered and thus an on-policy algorithm should be able to take advantage that information more so than an off-policy algorithm.

We can say, from the results, that the state representation is also very important for learning and the most beneficial representation depends on the characteristics of the learning algorithm. It seems that these methods are stable enough to not be disturbed by constant information such as the board dimensions and that such information can be considered essentially useless.

Conclusion

We have looked at two fundamental building blocks of reinforcement learning, the reward function and the state representation, to investigate their importance to learning. We used a simple variation of the game Snake as our problem. We have shown that the reward function and the state representation are important in order for the agent to learn the objective. The reward function should reflect the object and the state requires that it provides useful information. The characteristics of the learning algorithms do seem to play a part in which reward functions and state representations that are most useful which should be taking into consideration during the design phase of your reinforcement learning problem.

We used the algorithms as they were first discovered, but there have been many extensions to improve upon them. It would be interesting to see if the experiments give similar outcome with extensions such as double Q-learning [3], experience replay [5], and function approximations rather than a tabular representation of the Q-function.

References

[1] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[2] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[3] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[4] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[5] Long-H Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3/4):69–97, 1992.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[9] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[10] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.