Danmarks Tekniske Universitet

Katrine Bay, s183910
Gustav Gamst Larsen, s180820
Rasmus Ørtoft Aagaard, s164419

02465 - Introduktion til reinforcement learning og kontrol F20

Implementation and exploration of the $Q(\sigma, \lambda)$ method

8th May 2020
Danmarks Tekniske Universitet

Abstract

We wish to investigate the claims made by Markus Dumke [2] in which he suggests that his proposed algorithm $Q(\sigma, \lambda)$ can outperform other TD methods such as Sarsa and Expected Sarsa.This is done by testing the $Q(\sigma, \lambda)$ in a Stochastic Windy Gridworld environment to test its performance against the more classic TD methods, Sarsa and Expected Sarsa. The results showed that a $Q(\sigma, \lambda)$ is able to out-perform both Sarsa and Expected Sarsa with further tuning of the hyperparameters. In conclusion, this study finds potential in the $Q(\sigma, \lambda)$ method that supports Markus Dumke's claim.

Introduction

Problems concerning sequential decision making have shown to be well solved by techniques in the field of Reinforcement Learning - a discipline within machine learning. A well implemented branch of algorithms for solving this kind of problems is the temporal-difference (TD) learning algorithms, which is a blend between Monte Carlo (MC) methods and Dynamic Programming (DP) methods. The common and most used TD algorithms are Q-learning and Sarsa, and ideas about unifying them emerged in 2000 (Precup et al.) and the first concrete formulation of such a merged algorithm, the $Q(\sigma)$ algorithm, was presented by Sutton and Barto (2017). This algorithm along with the extention of Sarsa to Expected Sarsa lies as the foundation for the extension to an on-line multistep algorithm $Q(\sigma, \lambda)$ using eligibility traces presented in the end of 2017 by Markus Dumke[2]. This algorithm is suggested to outperform all other proposed algorithms within this field, and it will in this paper be implemented and investigated.

This paper will implement and investigate the three algorithms $Q(\sigma, \lambda)$, Sarsa and Expected Sarsa, and their performance will be tested in the Stochastic Windy Gridworld environment as seen on Figure 1 which was described by Sutton and Barto (1998) as a grid with a start state S and a goal state G wherein the middle pushes the agent upwards according to a number denoted below the grid. The environment is described in further detail in a later section.

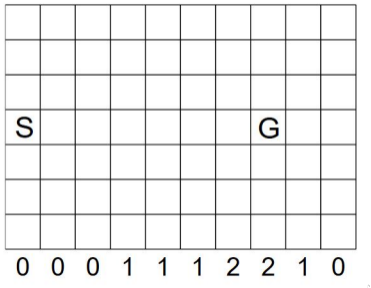

Figure 1: Stochastic Windy Gridworld

The hypothesis of this paper is that $Q(\sigma, \lambda)$ will outperform the Sarsa and Expected Sarsa algorithms.

Method & Analysis

The methods used to conduct the experiment investigated in this paper is in the sub-field of Machine Learning called Reinforcement Learning, where a software agent seeks to find the optimal policy for a sequential decision problem. In the presented experiment, the decision problem is concerning which way the agent should go through the Stochastic Windy Gridworld environment. What drives the agent is the rewards, it receives for each action it performs. This differ from the sub-field Supervised Learning, where the motivation is previously observed samples of correct or incorrect behaviour. The Reinforcement Learning algorithms implemented is the Sarsa, Expected Sarsa and $Q(\sigma, \lambda)$, which in this section will be explained in details with the main focus on the $Q(\sigma, \lambda)$ algorithm.

The RL environment used in the experiment is described in Sutton & Barto (1998) and all code is written in Python 3.7.

Temporal Difference Learning

Temporal Difference (TD) learning is a branch of algorithms created by a combination of both Monte Carlo (MC) methods and Dynamic Programming (DP) methods. With TD learning we choose to update estimates along every time step. The TD methods will then already learn after the next time step (t+1) and update its target at each time step. The algorithms of TD learning does not require a model of the environment and works based on samples, like MC methods, and TD algorithms use bootstrapping to make updates like algorithms of DP.

In this field of TD methods two subfamilies are often mentioned being off-policy and on-policy methods. The difference in the two methods is whether or not the behaviour policy, μ, and target policy, π, differ from each other. The most prominent algorithms of each branch is Q-learning as off-policy method and Sarsa as on-policy method. The algorithm Expected Sarsa is also considered an off-policy methods since the behaviour policy generating the behaviour are different from the target policy, which is being learned, while Sarsa is considered an on-policy since the behaviour and target policy are the same[1].

All of the used TD algorithms is value-based reinforcement learning methods, meaning they find the optimal action value function $q_* = \max_\pi q_\pi$. TD algorithms estimate q_π from sample transitions and updates the estimated values along the way using observed rewards and estimated values successor actions.

Each algorithm updates at each step by applying this:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\delta_t \tag{1}$$

What differentiates the algorithms of TD learning is the parameter δ_t, which is the TD error. This error is in general the difference between the estimate value and the later estimate of that value. The TD error is a time bound value, meaning the error at each time is the error in the estimate made at that time. Since the TD error depends on the following state and related reward, the error is not actually available until one time step later.[3]

On- and Off policy methods

A policy is used by the agent to decide what action to take at each state in the environment. That way the policy is a function mapping a given state s to the probabilities of selecting each available action a, which can be formulated as;

$$\pi(a|s) = \Pr\left\{A_t = a|S_t = s\right\} \tag{2}$$

All algorithms in the field of Reinforcement Learning must follow policies, making the algorithms either on- or off-policy methods. The difference in on- and off-policy methods lies in whether the policy creating the behaviour, *the behaviour policy* μ, and the policy that is being learned, *the target policy* π, are different or not. In the case, where the behaviour policy and the target policy are different, the method is considered an off-policy method, where as the on-policy methods have the same behaviour and target policy.

Q-learning

Q-learning (Watkins 1989; Watkins and Dayan 1992) is an off-policy TD method. The behaviour policy used to control the agent during learning is different from the target policy, whose values are being learned. It is adventurous in the scope, that the agent can employ an exploratory behaviour policy to ensure it gathers sufficiently diverse data while still learning how to behave once exploration is no longer necessary. The TD error of Q-learning is as follows;

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right) - Q\left(S_t, A_t\right) \tag{3}$$

As mentioned earlier, Q-learning is one of the fundamental algorithms and has been merged with Sarsa to create $Q(\sigma)$, which we later show the extension of as $Q(\sigma, \lambda)$.

Sarsa

The "State–Action–Reward–State–Action" algorithm (Sarsa) (Rummery and Niranjan (1994)) is an on-policy TD control method. It learns an action-value function rather than a state-value function, meaning that it is the transitioning from state-action to state-action which are being considered to learn a action-state value. Therefore, Sarsa is a Markov chain with a reward process. The backup diagram for the sarsa prediction model is pretty straight forward. As in all on-policy methods, we continually estimate q_μ for the behaviour policy μ, and at the same time change μ toward greediness with respect to q_μ. The ϵ-greedy policy as well as the ϵ-soft policy can be used. The update rule for Sarsa with the TD error presented in staples is as follows:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right] \tag{4}$$

This paper uses Sarsa as one extreme for the $Q(\sigma, \lambda)$ algorithm tested in this paper.

Expected Sarsa

Expected Sarsa looks a lot like Q-learning but has some differences that makes it perform significantly better in a great amount of scenarios. It generalises Q-Learning to arbitrary target policies and is an off-policy method. The update rule and the TD error is what really changes up the performance.

The TD error for Expected Sarsa:

$$\delta_t = R_{t+1} + \gamma \sum_{a'} \pi\left(a' | S_{t+1}\right) Q\left(S_{t+1}, a'\right) - Q\left(S_t, A_t\right) \tag{5}$$

The Q-learning algorithm can be seen as a special case of Expected Sarsa, where π is the greedy policy with respect to Q[2]. This algorithm moves deterministic in the same direction as Sarsa moves in expectation, which gives this method the name, Expected Sarsa. This method, given the same amount of experience, usually performs better than Sarsa, due to its random choice of action (A_{t+1}), which eliminates variance.

$Q(\sigma, \lambda)$

Prior the presentation of the $Q(\sigma, \lambda)$ algorithm one would find the unification of Sarsa and Expected Sarsa in the algorithm called $Q(\sigma)$ proposed in 2017 by Sutton and Barto. The TD target of this new algorithm is a weighted mean of the Sarsa and Expected Sarsa TD targets, where the parameter σ is the one responsible for the weighting. $Q(\sigma)$ with $\sigma = 1$ is equal to Sarsa and $Q(\sigma)$ with $\sigma = 0$ is equal to Expected Sarsa. With a dynamic σ-parameter i.e. intermediate values hereof, new algorithms are created, and these are suggested to achieve better performance (Asis et al. (2017)). [2]

The proposal of $Q(\sigma, \lambda)$ came in the scope of extending $Q(\sigma)$ by using eligibility traces to incorporate data of multiple time steps.

Algorithm 1 Q(σ, λ)

Initialize $Q(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

Repeat for each episode:

$E(s,a) \leftarrow 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

Initialize $S_0 \neq$ terminal

Choose A_0, e.g. ϵ-greedy from $Q(S_0, .)$

Loop for each step of episode:

Take action A_t, observe reward R_{t+1} and next state S_{t+1}

Choose next action A_{t+1}, e.g. ϵ-greedy from $Q(S_{t+1}, .)$

$\delta = R_{t+1} + \gamma \left(\sigma\, Q(S_{t+1}, A_{t+1}) + (1-\sigma) \sum_{a'} \pi(a'|S_{t+1})\, Q(S_{t+1}, a')\right) - Q(S_t, A_t)$

$E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$

$Q(s,a) \leftarrow Q(s,a) + \alpha\, \delta\, E(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

$E(s,a) \leftarrow \gamma\lambda E(s,a)(\sigma + (1-\sigma)\pi(A_{t+1}|S_{t+1})) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

$A_t \leftarrow A_{t+1}, S_t \leftarrow S_{t+1}$

If S_t is terminal: Break

Figure 2: Pseudocode for the tabular episodic $Q(\sigma, \lambda)$ method

The eligibility trace increases for each time a state-action pair is visited and is higher for the pairs recently visited. Following, the eligibility fades away over time if a certain pair is not visited. This is done with the trace decay parameter λ. The update rule for $Q(\sigma, \lambda)$ is

$$Q(s,a) \leftarrow Q(s,a) + \alpha \delta_t E_t(s,a) \tag{6}$$

where $E_t(s,a)$ represents the eligibility tracing and is computed as weighted average between the on-policy $Sarsa(\lambda)$ and off-policy $Q(\lambda)$ and is updated at each step:

$$E_{t+1}(s,a) = \begin{cases} \gamma\lambda E_t(s,a)\left(\sigma + (1-\sigma)\pi\left(A_{t+1}|S_{t+1}\right)\right) + 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a)\left(\sigma + (1-\sigma)\pi\left(A_{t+1}|S_{t+1}\right)\right), & \text{otherwise} \end{cases} \tag{7}$$

Stochastic Windy Grid World environment

The environment, which is depicted on Figure 1, and the problem to be solved is the stochastic Windy Grid world environment implemented as given from GitHub user ibrahim-elshar[1].

[1] `https://github.com/ibrahim-elshar/gym-windy-gridworlds`

This environment is a standard gridworld problem with start and goal states, but with the slight chance of additional wind running upwards in the middle part of the grid. The actions available for the agent is moving one square either up, down, left or right, but when the agent is moving into the windy area of the grid, the resultant state will be shifted upwards according to the strength of the wind, which ranges [0,2] for each column. The wind acts in a stochastic manner, meaning the effect of moving in the windy area can vary by 1 from the mean values of each column. The probabilities are set uniformly causing three variations. The agent can either move one cell above a third of the time, one cell below a third of the time and move truly according to the set strength of the wind a third of the time. The task of moving from the start state to the goal state is an undiscounted episodic task, where each action costs the rewards of -1 until the goal state is reached.[3]

Experimental procedure

The experiment of this paper is conducted by implementing the three different TD learning algorithms Sarsa, Expected Sarsa and $Q(\sigma, \lambda)$ as agents to solve the Stochastic Windy Gridworld environment. Several variations of $Q(\sigma, \lambda)$ was tested due to the parameters and strategies which can be tuned. Each agent is trained for 200 episodes with parameters as written on Figure 3.

Since the $Q(\sigma, \lambda)$ operates somewhere between Sarsa and Expected Sarsa, we can set $\sigma = 1$ to test for the one-step target of Sarsa, and $\sigma = 0$ to test for the one-step target of Expected Sarsa. In practice we ended up separating the agents entirely to avoid confusion. The paper will also test a $\sigma = 0.5$ to test for something in between the two one-step methods and at last, a dynamic σ that changes value along with different values for the λ value to inspect how eligibility traces have an effect on performance.

Results and discussion

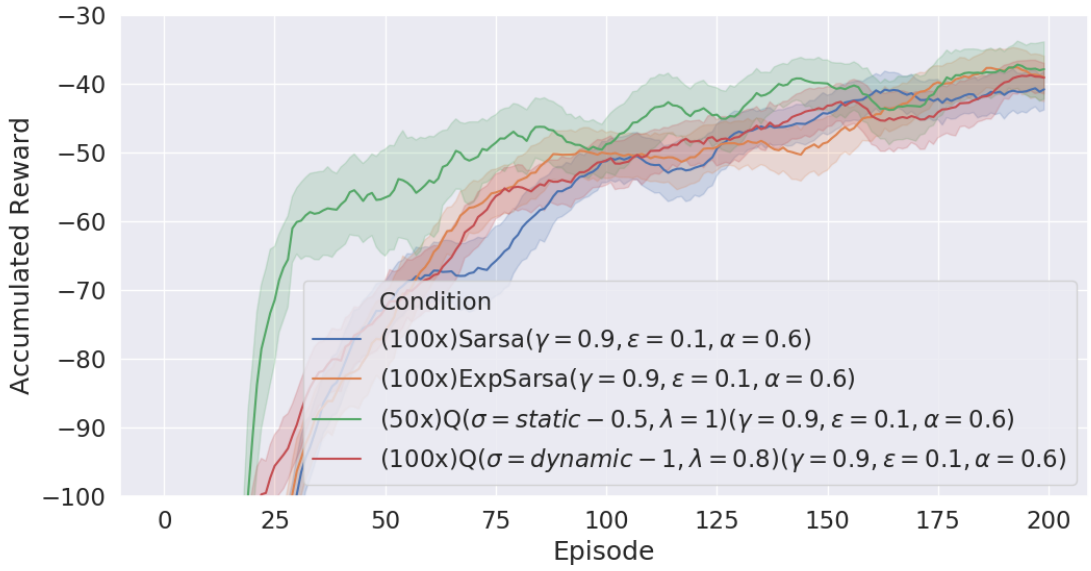

Figure 3: The performance of the $Q(\sigma, \lambda)$ given different λ-values along with different σ-strategies. Dynamic will decay the initial σ-value at a rate of 0.9 while constant refers to σ being the same value. These are compared to both Sarsa and Expected Sarsa in the stochastic windy gridworld environment.

The primary difference in performance in our four plotted agents is the number of episodes that $Q(\sigma, \lambda)$ with constant $\sigma = 0.5$ i.e. using an equal amount of Sarsa and Expected Sarsa to update Q and a full eligibility trace of $\lambda = 1$. It is noteworthy that the dynamic variant of $Q(\sigma, \lambda)$ performance so closely resembles Expected Sarsa. This is likely due to Expected Sarsa being equivalent to $Q(0)$ and convergence of σ as our update strategy.

When looking at Figure 3 of the four agents performance we see that certain parameters and strategies cause $Q(\sigma, \lambda)$ to out-perform the other methods. Convergence rate being the primary takeaway from this example. This goes to show that the method has a significant span in performance and *can* likely obtain excellent performance when given the right parameters. This tells us that parameter tuning and experimenting with strategies for choosing σ has great importance to this particular method. Unfortunately this step came too late in the project's process to implement and test properly.

For future research on the method, the algorithms could be tested in other environments e.g. of continuous character, and different ways to get a dynamic sigma could be explored.

To summarise and conclude on the conducted study, the hypothesis of $Q(\sigma, \lambda)$ outperforming its extremes is supported by the found results.

References

[1] Kristopher De Asis et al. "Multi-step Reinforcement Learning: A Unifying Algorithm". In: *CoRR* abs/1703.01327 (2017). arXiv: 1703.01327. URL: http://arxiv.org/abs/1703.01327.

[2] Markus Dumke. "Double Q(σ) and Q(σ, λ): Unifying Reinforcement Learning Control Algorithms". In: *CoRR* abs/1711.01569 (2017). arXiv: 1711.01569. URL: http://arxiv.org/abs/1711.01569.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

Appendix

Contribution table

	Introduction	Methods	Discussion x Results	Implementation
s183910	80 %	40 %	20 %	10 %
s180820	10 %	40 %	20 %	25 %
s164419	10 %	20 %	40 %	65 %

Link to repository

`https://gitlab.gbar.dtu.dk/s164419/02465-AgentZoo`