Prioritized Sweeping Neural DynaQ: Neural Networks over All?

Alexander Osiik
alexander.osiik@student.uni-luebeck.de
Seminar Cyber-Physical Systems (WS 2019/20)
Institute of Computer Engineering, University of Lübeck

January 10, 2020

Abstract

Reinforcement learning (RL) has gained a lot of popularity due to some spectacular successes. Q-learning is one of the most popular algorithms used for RL, where an agent builds an internal world model and action routine based on its interaction with the environment. However, the field of research becomes more and more complex and computationally demanding when trying to model naturally occurring psychological events and transfer them to artificial intelligence. Approaches including prioritized sweeping in dynamically growing neural networks were proposed, which are used to model and understand naturally occurring hippocampal replays, a memory consolidation process observed on rodents. This approach promises to improve the learning of simulated agents confronted a navigation task. Here we reproduced the performance results of Aubin et al. [2018] regarding Q-learning for the same, minimally modified environment, after a short introduction to the state of the art of RL. It is also briefly outlined whether the development of a complex neural network is worthwhile for such a simple task, as the importance of comparative metric *time* is often overlooked

1 Introduction

In recent years, reinforcement learning (RL) has gained a lot of popularity due some spectacular successes [Mnih et al., 2013] . Reinforcement Learning is learning what to do based on the environment and how to map the situations into actions. Q-learning is one of the most popular algorithms used for RL, where an agent builds an action policy based on its interaction with the environment, without knowing anything about the state-transition and reward model. However, there is a clear limitation, as the computation and storage of such world models becomes infeasible in real unrestricted environments.
To overcome this problem, neural networks are used. Learning in neural networks consists of finding and iteratively adjusting the right weights to approximate the needed value or policy function.
Over the past years, different architectures for neural networks have been proposed to expertise them for specific problems or to improve the general performance [Le, 2018]. Aubin et al. [2018] presented an approach including prioritized

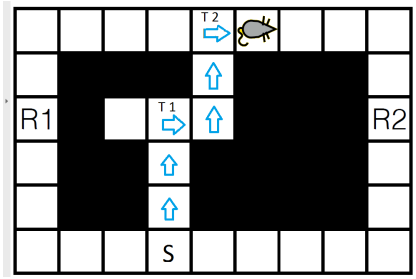


Figure 1: Discretized experimental setup. The agent has to make a decision at points T1 and T2, in which the decision at T2 leads to one of the rewarding sites. [Aubin et al., 2018]

sweeping in dynamically growing neural networks, used to model naturally occuring hippocampal replays, a phenomenon observed on rodents.
The content of this paper is divided into several parts. The introduction explains the biological background of Aubin et al. [2018] work. Then, the basic mathematical concepts of reinforcment learning are reviewed. Ultimately, the performance of a self-developed Q-learning model is presented and evaluated regarding the results of Aubin et al. [2018].

1.1 Biological background

The hippocampus is the main memory of our brain and the switch point between short-term and long-term memory. O'Keefe and Dostrovsky [1971] observed in several rat experiments that if there is a disturbance in this area, no new information can be stored in the brain. Regarding animals, the hippocampus is known to be important for navigation since the discovery of place cells, which signal the current allocentric location of the animal in the environment [Maguire et al., 2006]. Interaction within the environment leads to activation of these place cells. O'Keefe and Dostrovsky [1971] postulated that the hippocampus functions as a spatial map, where single hippocampal neurons increased their firing rate whenever a rat traversed a particular region of an environment, as concluded by Nakazawa et al. [2004]. They are engaged during the consolidation of spatial memories. However, it has been observed that these activations also occur during rest or sleep, at a significantly faster pace. This reactivation can be seen as a reprocessing of the experience gained during the day, something that is usually the case when dreaming [Caz et al., 2018].
Aubin et al. [2018] presented an approach to convert the reactivation of the hippocampus' place cells into a reinforcement learning model.

1.2 Experimental setup

Aubin et al. [2018] set up an experimental task, which was derived and slightly modified from Gupta et al. [2010]. The environment consisted of two successive T-mazes with lateral return corridors and rewarding food pellets on each side, see Figure 1. A rat was placed in the maze and trained to decide at position T2, intending to get the reward on the left or right-hand side based on the task pursued at the moment. The tasks were 1) always turn right, 2) always

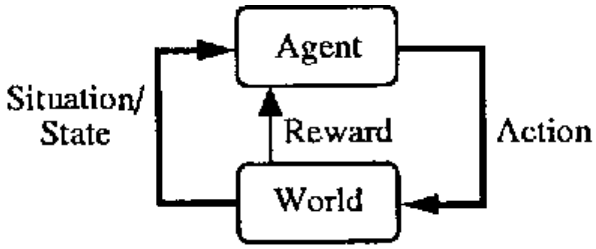


Figure 2: The Problem Formulation Used in Dyna. The agent's objective is to maximize the total reward it receives over time. [Sutton, 1991]

turn left, 3) alternate between left and right. At reward locations, the rat's hippocampal replays were analyzed. It has been shown that the rats reflected recently experienced trajectories, and also those that occurred a long time ago. To mathematically model and explain the hippocampal replay phenomenon, algorithms from the Dyna family of algorithms were used. Dyna is an integrated architecture for learning, planning and reacting, proposed by Sutton [1991], see Figure 2. The Dyna idea is a trivial approach that planning is equivalent to trying out multiple things in the own mind, under the condition that a certain internal model of the problem exists. Ultimately, this architecture was chosen because it is designed to make the best possible use of alternation between on-line and off-line learning phases [Sutton, 1991]. Aubin et al. [2018] concentrated on the Q-learning version of Dyna (Dyna-Q) extended by prioritized sweeping, by that optimizing the choice of reactivated cells.

2 Reinforcement Learning

To further understand the underlying methods it is important to know the general mathematical rules and machine learning approaches. In the following, the terminology as well as mathematical background of the research subject is briefly summarized and explained.

2.1 Markov Decision Problem

A Markov Decision Problem (MDP) is a model for problems, where an agent tries to interact with the environment in such way that the utmost reward is achieved. Specifically, the robot is moving (*transition*) through various *states*, having chosen a specific *action* in each state. Each *reward* is determined by the initial state, the action, and the following state. All transitions are not deterministic, but rather probabilistic, where each probability only depends on the current state and the current action. This memoryless property of stochastic processes is referred to as **Markov condition**. That way there has to be one initial state, but multiple end states are possible.

The main goal is to find a reward maximizing **policy**, by which the agent selects actions where the maximum reward can be expected. This policy is an optimal plan, or sequence of actions, from start to a goal [Klein and Abbeel, 2019].
A Markov decision process consists of

- Set of states $S : \{s_1, s_2, \ldots, s_n\}$
- Set of actions $A : \{a_1, a_2, \ldots, a_n\}$
- Transition function, which is the probability of going from state s to state s' via action a: $T : S \times A \times S$
- Reward function $R : S \times A \times S \to \mathbb{R}$

The main goal is to find an optimal policy Π and exploration factor γ, where

- Π is the policy, where an optimal action is assigned to each state
- $\gamma \in [0, 1]$ is the discount factor. This factor determines the degree of exploration for the agent. For $\gamma = 0$, the agent will stick to the policy, and exploit the current (possibly) optimal policy. For $\gamma = 1$, the agent will take into account the next states' reward, leading to exploration behavior. A value < 1 is a constraint, which limits the maximally obtainable reward, leading to a reduction of cycles.

The **V-values** and **Q-values** are certain grading schemes used to solve MDPs. The values are the total reward the agent can expect if it performs the optimal, or maximally benefitting, action a in state s, and continues to act optimally thereafter:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \tag{1}$$

As the values from equation 1 are hard do compute, a technique called **value iteration** is used. It is used do discretize the equation:

$$V^k(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^{k-1}(s')] \tag{2}$$

where k is the radius from the goal state to the agent. For example, regarding the Manhattan norm in a 2D plane, the amount of steps the agent has left until it reaches an end state.
After that, **policy extraction** is performed. It is the assignment of an action to each state, maximizing the expected total reward:

$$\Pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \tag{3}$$

2.2 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm which converges to optimal policy even if if the performed actions are suboptimal [Klein and Abbeel, 2019]. The letter **Q** stands for **quality** of a certain action in a given state, as

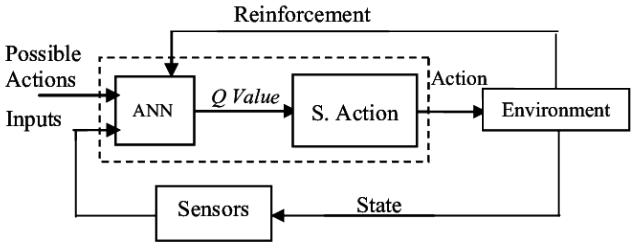


Figure 3: The structure of reinforcement learning based on an Artificial Neural Network [Hatem and Abdessemed, 2009]

the values represent the maximum discounted future reward when action a in state s is performed. Q-values are calculated similar to the value iteration in Equation 2:

$$Q^*(s,a) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{4}$$

The funcion $Q(s,a)$ can be estimated using Q-Learning, which is a form of **Temporal Difference Learning** (TD). TD means that an update is performed after every action based on a maximum reward estimate, and not only after receiving the reward.
Here, value $Q(s,a)$ is iteratively updated using the **Bellman Equation**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)] \tag{5}$$

The Q-values are represented in a **Q-table** where each cell represents the highest achievable reward performing the state-action pair ($s \in S, a \in A$). This makes Q-learning suitable for small environments only, as an increase in states or actions increases the amount of memory required to save and update that table $\mathcal{O}(n^2)$. The amount of time required to explore each state to create the required Q-table would be incredibly large.
To counteract the use of large Q-tables for look-up, neural networks are used in **Deep Q-learning** (DQN) to approximate the Q-value function, where the state represents the network's input and the Q-values of all actions are its output.
The challenge is to tune the parameters α, γ such that the best possible policy is found in minimal time.

2.2.1 Prioritized Sweeping

As mentioned in Section 2.2, the memory consumption of plain Q-Learning increases significantly with the amount of possible states and actions. Regarding large environments, the update of each value is extremely inefficient and time consuming.
This is the idea behind *prioritized sweeping*(PS) proposed by Moore and Atkeson [1993]. In PS, a queue is maintained of every state-action pair whose estimated value would have the most significant change. [Sutton and Barto, 2018]. The Q-values are then updated multiple times, without actually performing any action.

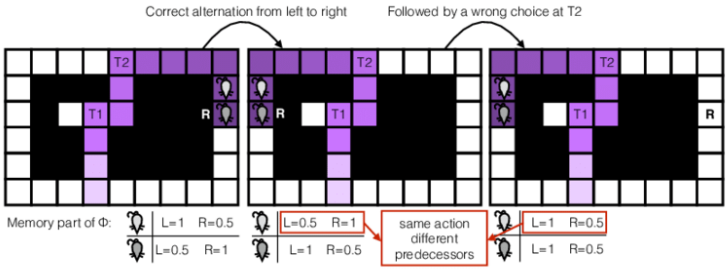


Figure 4: Example of multiple predecessors in the alternation task. At the last position on the third run (right), the location component of the predecessor state (white mouse) is identical but the memory component is different. [Aubin et al., 2018]

This is advantageous, since the execution of a movement usually takes more time than the calculations of the simulated actions, significantly shortening learning time.

3 GALMO

Aubin et al. [2018] encountered a problem when modelling the experimental environment, namely that under certain circumstances some states may have more than one predecessor, see Figure 4. This occurs especially during the third task, where reward sides alternate each lap.
To confront this issue, Aubin et al. [2018] created a growing type of algorithm called **GALMO**, an abbreviation for *growing algorithm to learn multiple outputs*. This algorithm allows the creation of multiple neural networks N_i so that multiple outputs can be generated for a single input. Each of the N_i networks is coupled with a gating network G_i deciding which network's output is considered for further calculations, see a schematic example on Figure 5. The algorithm should also track the statistics of the minimal training errors of each sample to detect outliers. It is assumed that an outlier is caused by an input that should predict multiple outputs. GALMO then creates a new network based on a copy of the most suitable network for this input and trains it once.
The second part of the algorithm works as neural network-based Dyna-Q with prioritized sweeping. The rat moves in the maze environment and stores the samples in a priority queue, where the priority is the absolute value of the reward prediction error. At point of reward, replays are simulated given a certain budget, representing the phenomenon of hippocampal replays that occur in real-life situations [O'Keefe and Dostrovsky, 1971]. As expected, the higher prioritized samples are replayed first. Their predecessors are then estimated and placed back into the queue until budget exhaustion.
The reason behind the implementation of a complex algorithm GALMO was to create a framework to analyze the role of hippocampal replays in the learning process, and whether there are patterns that can be observed in the artificial network which correspond to those in real neural networks.

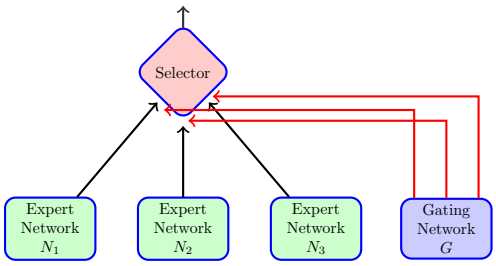


Figure 5: A system of expert and gating networks. Each expert as well as gating network is a feedforward network with same amount of inputs and outputs[Jacobs et al., 1991]. Derived scheme.

4 Project and Results

As part of practical work, a simplified T-maze configuration with a discrete environment similar to the work of Aubin et al. [2018] has been reproduced, see Figure 6. The programming language of choice was Python, as it offers many up-to-date machine learning libraries and allows a fast and understandable implementation. Helpful tutorials by Harrison [2019] were used for orientation and knowledge acquisition in the present metier.
Again, the agent can choose between four actions: moving North, South, East and West. However, an extension was implemented where the agent can decide to run into the boundary. Although this does not change the position of the agent, it will be detected and punished with a higher penalty than the one for "successful" movement.

The environment was modified in such way that the tasks mentioned above can be executed. For task 1) and 2), the food pellet (*reward*) was placed in a rigid position on the left or the right-hand side. For the successful completion of task 3 (reward sides alternate) the state space was extended by two components: the left side reward memory (L) and the right side reward memory (R), as proposed by Aubin et al. [2018]. They take a value 2 if the last reward was obtained on that side, 1 if the reward was obtained on that side the previous time, and 0 if there was no reward found on that side during the last 2 runs.
Additionally, the starting point of the agent was moved from point S to decision point T1, see Figure 1 for comparison. This step was necessary, otherwise

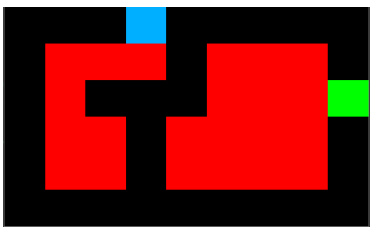
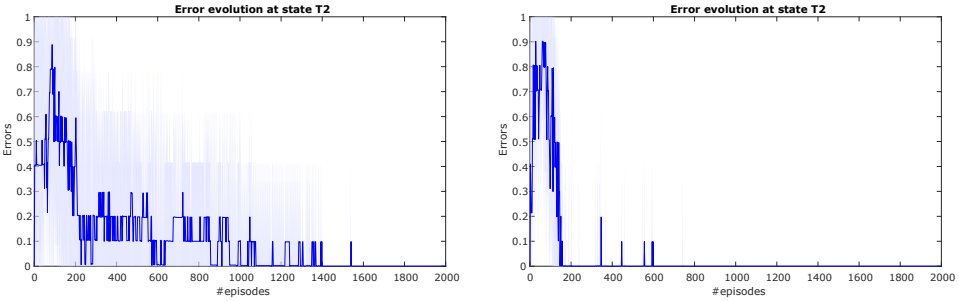
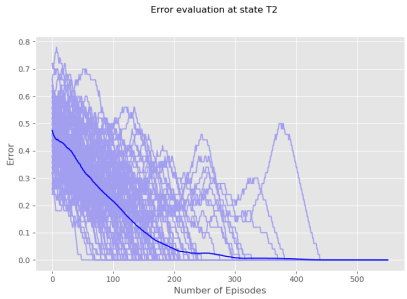


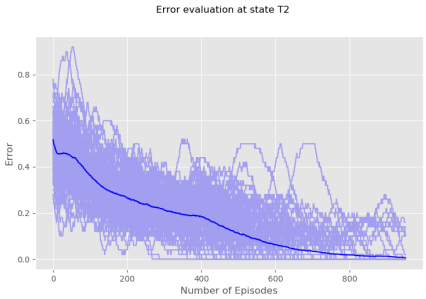
Figure 6: Graphical display of the created environment. Blue: agent's position; green: the food pellet/reward; red: maze walls.



(a) Learning without (left) and with (right) replays [Aubin et al., 2018].



(b) Performance during Task 1 in the project.



(c) Performance during Task 3.

Figure 7: **Learning dynamics of the own Q-learning implementation in comparison to GALMO performance**. Figures 7b,7c show the evolution of the proportion of decision errors at point T2 during Task 1 and Task 3, respectively.

the agent would have exploited the much shorter route around the maze walls, without ever passing through points T1 or T2.

A posteriori, the efficiency of the developed environment model with the corresponding Q-learning has been compared with the Dyna-Q model with GALMO from Aubin et al. [2018]. The present model is capable of learning every of the given tasks. The performance of task 1) and 2) were nearly identical since the distance to reward sides from starting position is identical left and right, see Figure 7b.
The resulting performance of the projects Q-learning implementation and the Neural Dyna-Q implementation were broadly reproduced. Both algorithms needed around 1000 episodes to converge, see proportion of errenous choices Figures 7a,7c. The projects Q-learning converged around 30% faster, but this is based on the fact that the path to rewarding pellets was reduced by about this percentage due to the changed initial placement of the agent.
Ultimately, this minimal improvement cannot keep up with the Neural DynaQ results of Aubin et al. [2018], since their implementation already converged after about 200 episodes, being five times more effective.

5 Conclusion

In this paper the biological background of reinforcement learning was described. The current mathematical methods and approaches were reviewed, followed by a brief explaination of why associated implementation problems occur.
A practical part was also created for this work, being Python implementation of Q-learning for a simple environment as well as its learning performance analysis. The results were very similar to the respective results of Aubin et al. [2018]; there were small differences due to own minimal changes of the task.
Concerning GALMO, Aubin et al. [2018] would like to test the dynamic algorithm in a larger and more complex environment to better evaluate its functionality, since the used test environment was quite limited. It has been shown that the corresponding Neural DynyQ with prioritized sweeping performs five times better than the plain Q-learning implementation.

Unfortunately, no information was given on the duration of the neural networks's learning phase, which would certainly be interesting considering a cost-benefit calculation. Especially since the project's Q-learning implementation for a simple environment requires only a few seconds until convergence.

References

L. Aubin, M. Khamassi, and B. Girard. Prioritized sweeping neural dynaq with multiple predecessors, and hippocampal replays. In *Living Machines 2018*, LNAI, page TBA, Paris, France, 2018. URL https://hal.archives-ouvertes.fr/hal-01709275.

Romain Caz, Mehdi Khamassi, Lise Aubin, and Benot Girard. Hippocampal replays under the scrutiny of reinforcement learning models. *Journal of Neurophysiology*, 120, 10 2018. doi: 10.1152/jn.00145.2018.

Anoopum S. Gupta, Matthijs A.A. van der Meer, David S. Touretzky, and A. David Redish. Hippocampal replay is not a simple function of experience. *Neuron*, 65(5):695 – 705, 2010. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2010.01.034. URL http://www.sciencedirect.com/science/article/pii/S0896627310000607.

Harrison. Reinforcement learning tutorial series, 2019. URL https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/.

Mezaache Hatem and Foudil Abdessemed. Simulation of the navigation of a mobile robot by the qlearning using artificial neuron networks. volume 547, 01 2009.

Robert Jacobs, Michael Jordan, Steven Nowlan, and Geoffrey Hinton. Adaptive mixture of local expert. *Neural Computation*, 3:78–88, 02 1991. doi: 10.1162/neco.1991.3.1.79.

Dan Klein and Pieter Abbeel. Uc berkeley cs188 intro to ai, 2019. URL http://ai.berkeley.edu/.

James Le. The 8 neural network architectures machine learning researchers need to learn, 2018. URL https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html.

Eleanor Maguire, Rory Nannery, and Hugo Spiers. Navigation around london by a taxi driver with bilateral hippocampal lesions. *Brain : a journal of neurology*, 129:2894–907, 12 2006. doi: 10.1093/brain/awl286.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. In *Machine Learning*, pages 103–130, 1993.

Kazu Nakazawa, Thomas Mchugh, Matthew Wilson, and Susumu Tonegawa. Nmda receptors, place cells and hippocampal spatial memory. *Nature reviews. Neuroscience*, 5:361–72, 06 2004. doi: 10.1038/nrn1385.

J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171 – 175, 1971. ISSN 0006-8993. doi: https://doi.org/10.1016/0006-8993(71)90358-1. URL http://www.sciencedirect.com/science/article/pii/0006899371903581.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

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