AMSI VACATION RESEARCH SCHOLARSHIPS 2019–20

EXPLORE THE MATHEMATICAL SCIENCES THIS SUMMER

Can Machines Learn like Fish:

An application of simulated multi-timescale Nexting

David Adams

Supervised by Enrico Valdinoci

University of Western Australia

Vacation Research Scholarships are funded jointly by the Department of Education and Training and the Australian Mathematical Sciences Institute.

Contents

Abstract

While advances in neural networks have brought much attention to artificial intelligence and machine learning not many people are aware of the foundations of learning and how they are still very applicable in their relative simplicity. The concept of learning from experience isn't very foreign to most people. Every job application will ask about previous experience and people will seek to continually learn from it. In a mathematical sense learning can be represented as a series of interactions with an environment and how an agent changes its behaviour in response. One might consider looking at fish a rather peculiar activity, but in being sufficiently different to humans and rats, whose learning processes are comparatively well understood, one might be able to generalise about the nature of learning. This project looks at the concept of Nexting, making lots of quick predictions about the immediate future, as a way to understand how an understanding of an environment develops within the context of reinforcement learning. In particular how a virtual fish learning to navigate a maze and how it compares to real life experiments.

Statement of Authorship

Many of the ideas that this report is based off of were developed by Richard Sutton, who arguably created the field of reinforcement learning and to not look at his ideas would be foolish. In particular his work in the paper *Multi-Timescale Nexting in a reinforcement learning robot* (Modayil et al. 2011) was very inspirational for this report. The particular application of an active learning agent, to the best of my knowledge, has not been explored before as well as the extensions and automatic parameter tuning. The simulation environment used for this project was Unity, a popular game making and animation software suite, all assets used were either open source with no licences or custom made and the same goes for the various control scripts.

1 Introduction

When looking at the current state of engineering, it is clear that biology is having an ever greater impact. Areas such as structural design, molecular machines and multi-use materials have either been heavily influenced or completely derived from nature.(Vincent 2006). One larger example that comes to mind is the Japanese bullet trains that were designed to be like a woodpecker's beak to reduce the impact of a sonic boom and other noise dampening structures based on owl's feathers (Mckeag 2012). More recently, the cognitive structures of animals have been used as inspiration for neural networks, a powerful tool for computational approximation and other complex tasks.

Despite the insights that have been gained from animals in the environment many of them are under threat from climate related pressures. In particular ocean fairing fish such as D.*arunaus* and others have been shown to effectively lose most protection from predators in the presence of high-tend concentrations of CO_2 (Munday P. 2014). In such conditions fish spend almost no time in shelter and take very little time to emerge from it. Currently the exact mechanisms and how increased concentrations of CO_2 affects different members of a population are unclear. It is also unknown how or even weather fish will be able to learn to adapt to such changes.

To understand what factors affect behaviour, especially in relation to predators in an environment, robots have been incorporated experiments for better consistency and repeatably. Examples of such experiments include preference testing (Kopman et al. 2013), behavioural consistency and modeling of sensitive periods. One thing that all of these studies have in common is they are in a static highly controlled environment, robots are strictly tied to a single model and all are done as live experiments with minimal capacity for digital simulation.

In terms of mathematical understanding of fish behaviour and in particular the learning of new behaviours, in general, reinforcement learning has been proposed as a possible model of understanding the concept of 'learning from experience'. Temporal difference learning is a form of reinforcement learning, which was greatly enhanced by Richard Sutton with the TD(λ) (Sutton 1988) algorithm and has been closely linked with the firing of dopamine receptors (Schultz et al. 1997). A small issue when trying to relate temporal difference learning to large scale realistic scenarios is its reliance on a single discount factor, or in other words only learns on one timescale. As learning happens on multiple timescales due to co-operation between the amygdala and striatum of a vertebrate's brain, with fast learning coming from action-dependant mechanisms of the amygdala and slower ones from dopamine dependant mechanisms in the striatum (Neftci et al. 2019), standard temporal difference learning does not seem a good choice of model for biological simulation.

The goal of this project is to produce a virtual environment to simulate fish behaviour. In particular the simulation chosen was a simple spatial task of learning to navigate a maze and the multi-timescale Nexting algorithm developed as a means of modeling learning at different rates. Such a model free approach was chosen in preference to a neural network as whilst still being applicable to dynamic environments, is not as computationally expensive to train or run potentially allowing it to be run on a per robot/individual level. Put simply the question is not can a machine learn like fish but, can we make our machine learn in the same way as a fish would.

1.1 A note to the Reader

This project assumes no prior knowledge of reinforcement learning and covers most concepts from the ground up. For a comprehensive look at the subject from the person who invented it is recommend to read *Reinforcement Learning: An Introduction* by Richard Sutton. For the purposes of only outlining the background necessary for understanding the end result of the project some key areas of reinforcement learning have been omitted such as dynamic programming, Monte Carlo methods and Q-learning and as such references to them are kept to a minimum. Other popular algorithms such as SARSA (Richard Sutton 2015) have also been ommited, and while considered in the implementation of this project were ultimately deemed not applicable to the end goal of this project which is real world experiments and being approachable.

2 Learning in the natural world

To simulate biological learning one must have an understanding or, more importantly, a point of comparison for models that are produced. For the particular application of navigating a maze a number of studies, going as far back as Edward Tolman in 1948, have developed the concepts of latent learning and cognitive maps in rodents. It was theorised that rodents had a natural predisposition to solving spatial problems, likely due to their underground burrows, and that there was an evolutionary basis for its development (Tolman 1948). More recent studies have looked at investigating weather the same principles apply to phylogenetically distinct species from humans and rats. One such example by Tyrone Lucon-Xiccato and Angelo Bisazza investigated the spatial learning capabilities of *Poecilia reticulata* otherwise known as the river guppy. Specifically they looked at the ability to solve complex spatial problems. They concluded that the guppies were indeed able to learn to solve the maze with about 80% correct responses by the fifth day of learning and that their performance was directly comparable with both humans and rodents (Tyrone et al. 2017)

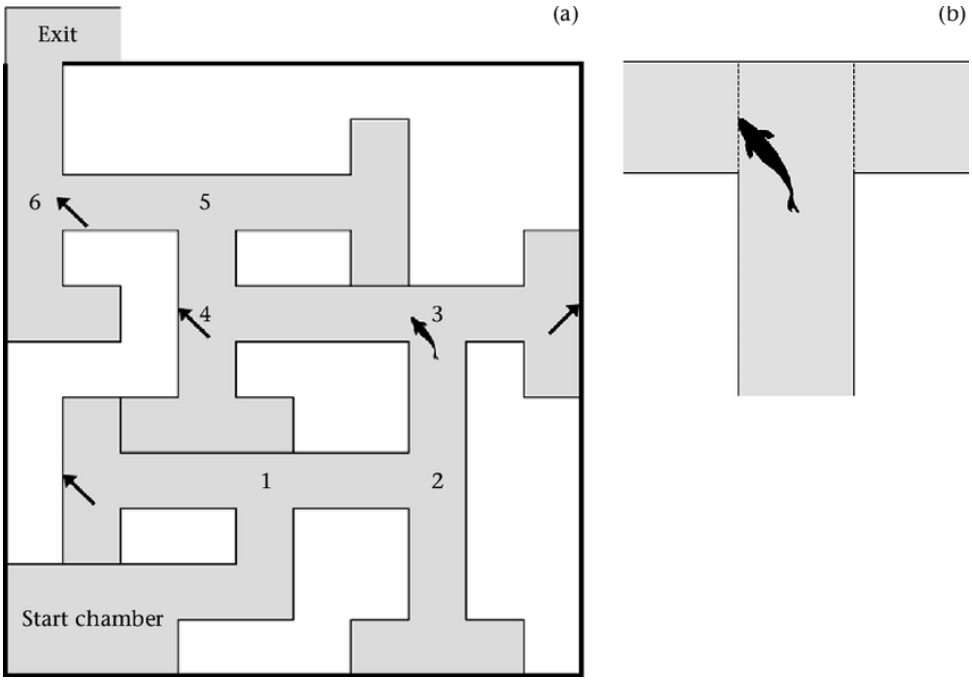

Figure 1: (a) Aerial view of the maze used in the experiments and (b) detail of a T-junction. Arrows indicate the position of the colour cues used for half of the subjects in experiment 1. Dashed lines indicated lines used to score the lift-right choice of the fish (Tyrone et al. 2017)

In 1 there is a clear downwards trend in the overall number of errors starting from around 12 to as low as 5. It should be noted that there were a number of other tests involving colour cues and wild versus domesticated guppies. The results of these other tests were simply that colour cues did not have much of an impact, which supports the idea of an internal representation of the environment, and regardless of the condition of the animal they were still able to learn effectively (Tyrone et al. 2017). These results will provide the basis for comparison of a simulated model.

3 Reinforcement Learning

Ultimately reinforcement learning is just a problem class with classes of solution methods associated with it. It aims to get an agent what to do, perform certain actions in certain situations, to maximise a numerical reward which is supplied by an environment external to the agent. The agent is not told what actions to take specifically, instead having to discern what actions are best given its environment.

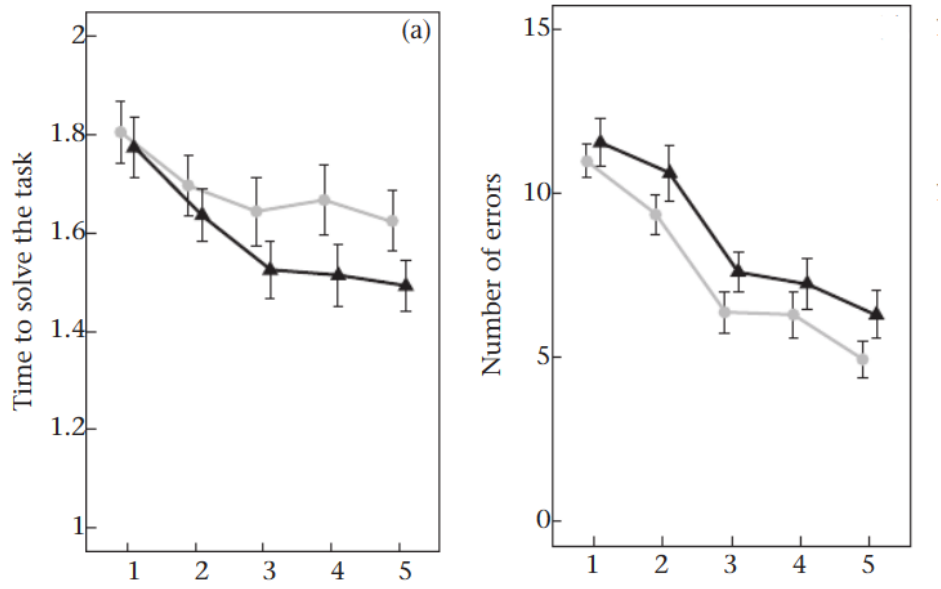

Figure 2: Learning progress of guppies over time. Black: males; grey: females. From (Tyrone et al. 2017)

3.1 Markov Decision Processes

The overall problem that reinforcement learning seeks to solve is that of a finite Markov decision process or MDP, which is a discrete time stochastic control process involving and agent making a decision in some state s at time t moving to a new state s' giving the agent reward $R_a(s, s')$. To be finite the process must have both a finite state space and action space.

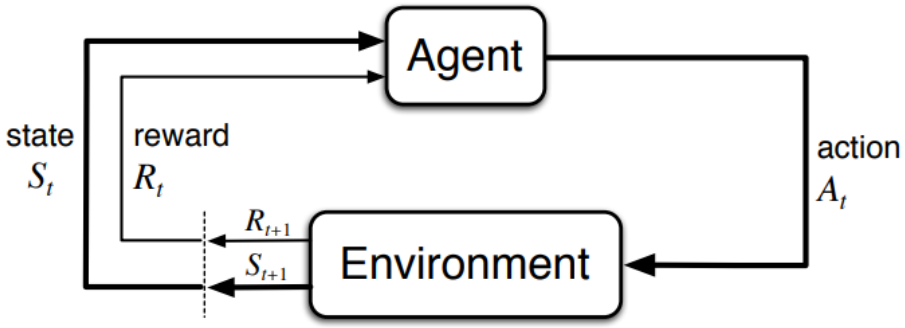

Figure 3: Environment-agent interaction model (Richard Sutton 2015)

a finite MDP is defined by both the set of actions and dynamics of the environment such that given any state s and action a the probability of each possible state and rewards s', r is given by:

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

from this description it is possible to to calculate the expected rewards of an action in a state and state-transition probabilities.

3.1.1 Rewards

A reward is a signal that comes form the environment providing some kind of information on the agents performance. Such feed back is given at each time step and it can be said that the goal of an agent is to get the 'most' reward possible. One can define a simple return function:

$$G_t = R_{t+1} + r_{t+2} + \cdots + R_F$$

which is simply the sum of rewards in a given reward sequence. A small problem with this representation is in the cases of infinite action sequences such when environment interactions cannot be broken down into discrete items. In such cases the return could be infinite. For example in the actual experiment of teaching a fish to navigate a maze, in the early stages, there were issues of the fish going around in circles infinitely. One possible solution to this issue of infinite returns is discounting the rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

This quantity can be considered as the agents way of balancing future rewards against those in the immediate future.

3.1.2 Policy

A policy is simply a of states to actions. For the maze navigation example if the fish can see that the next state is an exit it would move towards that direction. Here the policy would be that for the state that is close to the exit always move forward.

$$\pi : S \to A$$

of course a policy should be sufficient to deal with all possible states the agent encounters and can be most easily implemented as a lookup stable.

3.2 Value-functions

A natural extension of the concept of reward is value, or more precisely how valuable is a state. This is because in reality calculating the possible return of a large number of possible reward sequences and future states is computationally infeasible. Value functions provide an estimate of the 'goodness' of a

state. Going back to the example of the fish the value of a state very close to the end would be high. It would make sense to only really consider a value function in the context of a policy as if a given policy told the fish to move away from the goal when close it would not be very valuable as the state would not end up leading to the reward. Informally one can define the value of a given state under a given policy for a MDP:

$$v_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] = \mathbb{E}_\pi[G_t | S_t = s]$$

which is simply the expected return given the agent is in state s and is acting under policy π

3.3 Bellman equations

Considering that the value of any state under a policy can be calculated to find an optimal optimal policy should just be a trivial iteration over all possible policies and seeing which is best. This begs the question of what an optimal policy is, other than maximising the expected return of an agent in an environment. The most universally accepted definition for optimally in terms of a policy comes from Richard Bellman who states:

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman 2003).

A Bellman equation is simply writing the value of a decision problem (a MDP in this case) in terms of some initial choice and how the rest of the problem progresses as a results. This essentially breaks the problem up into smaller sub problems according to Bellman's idea of an optimal policy. It is possible to derive a Bellman equation, using the law of iterated expectation, from the definition of

the value function in the context of a MDP:

$$
\begin{aligned}
v_\pi &= \mathbb{E}[G_t|S_t = s] \\
&= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{(t+1)+k+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1}]|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_r p(r|s,a)r + \gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)v_\pi(s') \\
&= \sum_a \pi(a|s) \sum_r \sum_{s'} p(s',r|s,a)r + \gamma \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)v_\pi(s') \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)r + \gamma \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)v_\pi(s') \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)[r + \gamma v_\pi(s')].
\end{aligned}
$$

Logically the value of a state is simply the expected end return given we are in that starting state, but when expanded is the sum over all possible actions a in our original state multiplied by the return from going to future states. From this equation it is possible to derive and expression for the optimal policy $v_*(s)$:

$$v_*(s) = \max_\pi v_\pi(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

4 Temporal Difference Methods

Like Mote Carlo methods Temporal difference or TD methods use experience or putting the agent into an actual environment and seeing how it performs to inform how the agent acts. One way of thinking about TD learning is that it starts with some intital guess of the value of states, then updates those guesses based on the reward signals observed by the experience of the agent.

4.1 TD(0)

This is the most simple TD method where the estimate of a state, denoted $V(S)$, is updated at each time step to reflect the agents experience:

$$V(S_t) \leftarrow V(S_t) + a[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

essentially the value for each state is updated by the reward it gets in that state plus the 'temporal difference' between the state it just got to and the state it left, all of which is scaled by a learning rate a and a discount parameter γ. Notice that in this model there is a distinct lack of a mention of the policy or any kind of state transition probability. The entire learning process is only reliant on the reward that an agent gets and the current estimations of the value of states.

4.2 TD(1)

While not used in practice very often TD(1) is a more efficient version of TD(0) in that it is still very important as it demonstrates how simply applying eligibility traces. It has nearly an identical update for the value estimate of a state yet it is much more efficient at learning because unlike TD(0) it know what it has done in the past and can make better decisions during an update because of it.

4.2.1 Eligibility Traces

To take the more mechanistic view, an eligibility trace is a kind of memory of what the agent has done so far. For a state s at time t it is defined to be the random variable $E_t(s) \in \mathbb{R}^+$ and at each time step decays by a discount factor γ (Richard Sutton 2015).

$$E_t(s) = \gamma E_{t-1}(s)$$

When a state is visited its eligibility trace is increased by 1.

$$E_t(S_t) = \gamma E_{t-1}(S_t) + 1$$

This way an agent only updates the states that are relevant to its recent actions, focusing in on the states that either caused a bad turn of events or gave some good reward while not worrying about other states at all.

The TD(1) algorithm is essentially identical forego the eligibility trace on the TD update:

$$V(S_t) \leftarrow V(S_t) + a[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]E_t(s)$$

It is simply updating the value of the state based on the reward it receives, its temporal difference in relation to the next state and finally checked against the eligibility trace of that state to check weather it is actually important to consider.

4.3 TD(λ)

One of the most popular reinforcement learning algorithms TD(λ) (Sutton 1988) can be viewed as a balance between the purely temporal difference based TD(0) and the Mote Carlo like TD(1) method, both of which exist as special cases of TD(λ) when the titular parameter λ is set to zero or one. In the simplified case the update function is identical to that of TD(1) but the eligibility trace differs slightly:

$$E_t(S_t) = \gamma\lambda E_{t-1}(S_t) + 1$$

Here the eligibility trace is being scaled by a second parameter lambda and often referred to as the trace-decay parameter and essentially has the effect of smoothing out the decay in eligibility of a state from the more aggressive exponential nature of TD(1). Because states are not so harshly disregarded TD(λ) gives a better global picture to far away states.

5 Nexting

The concept of predicting is something that most people are familiar with. Everyone wants to guess who will win a sports game or the outcome of some event. But what most people might not be so aware of is the predictions that are innate, that don't require focus or conscious attention.

psychologists have observed that people and animals make lots of short-term predictions about their surroundings such as rats predicting what letter they will be shown (Rescorla 1980). One example for humans is in music when a transition occurs from the IV to the V chord of a key center, tension is created and most listeners will predict the release of that tension and might even be able to hear the I chord in their heads, and will be suitably confused or upset if the song should go to the III chord. In the physical world when running a person does not have to constantly look down at their feet to make sure they will hit the ground, their brain has an understanding of gravity and predicts that the foot will land on the ground in the right spot. This term has been coined nexting (Gilbert 2006).

Nexting constitutes a basic awareness and more importantly an understanding of an environment. So far the temporal difference learning methods that have been covered could be considered to constitute a single time scale version of Nexting (some might even want to call it classical conditioning).

But to consider longer term rewards and to look at different kinds of input, a different approach is required.

5.1 Muti-timescale Nexting

If a singular learned value function can be considered Nexting on a single timescale, than learning multiple value functions that look at different timescales could be considered Nexting on multiple timescales. As such each value function should seek to approximate a reward sequence at a given timescale

$$v_t^i \approx G_t^i = \sum_{k=0}^{\infty} (y^i)^k r_{t+k+1}^i$$

Here a specific value function should seek to approximate the discounted sum of returns for a particular timescale denoted by i and r is the reward to come from the environment at a particular time on a certain timescale.

Multi-timescale Nexting has been proposed before (Modayil et al. 2011) but not in the context of an actively learning agent under a policy.

As previously mentioned, it is infeasible to calculate the actual return so linear approximation must be used to calculate specific values. The state at time t can be represented as a feature vector $\phi_t \in \mathbb{R}^n$ and the value functions are calculated as inner products of ϕ_t with some corresponding weight vectors θ_t^i

$$v_t^i = \phi_t^\intercal \theta_t^i = \sum_j \phi_t(j)\theta_t^i(j)$$

here, $\phi_t^\intercal$ denotes the transpose of ϕ_t and $\phi_t(j)$ is the jth component of of that vector.

It is then possible to learn the weight vectors through TD(λ)

$$\theta_{t+1}^i = \theta_i^t + \alpha(r_{t+1}^i + \gamma^i \phi_{t+1}^\intercal \theta_t^i - \phi_t^\intercal \theta_t^i) e_t^i$$

with a corresponding eligibility trace $e_t^i \in \mathbb{R}^n$ which is initially set to zero and updated on each step with:

$$e_t^i = \gamma^i \lambda e_{t-1}^i + \phi_t$$

6 Experimental setup

To apply Multi-Timescale Nexting Unity 3d was chosen as the platform to develop the simulation environment. The technical specifications for the machine that the experiments were undertaken on were as follows:

Figure 4: Simulated environment created in unity

Unity Version	2019.3.0f6 Personal
OS	Windows 10 Pro
CPU	Intel i5-4300U @ 1.9Ghz
GPU	Inel HD Graphics
RAM	4GB
.NET Version	3.1

During the development of the environment a number of various methods were examined. The first was a simple 3D maze construction with a token based reward structure. This method was abandoned as it required manual placement of rewards which was time consuming. The second was a very simple computer vision based approach which was unfortunately too computationally expensive on the machine due to Unity's implementation of ray casting. The final approach taken was to directly attempt to emulate the maze from the original experiment (Tyrone et al. 2017).

While Unity's coordinate system provides adequate granularity in its coordinate system to simulate continuous movement, for the proposes of not blowing out the state representation on limited hardware resources a coordinate grid was chosen as the representation of state. When implemented, state is a ten by ten array which, although it could have been implemented as a dynamic array to emulate an eligibility trace and help with separating the agent from the environment, proved effective for producing the desired behaviour.

The value function and reward function were also implemented as two dimensional arrays. During the initialisation phase of the simulations the reward of each state was set to -1 excepting the goal state, S^* which was set to a reward of positive 100 and value was set to zero.

$$R(S) = \begin{cases} 100, & S = S^* \\ -1, & otherwise \end{cases}$$

Movement was implemented as a simple search of states directly adjacent to the fish at a given time. Adjacent states which contained a wall were not considered as possible states to move and of the states that could be moved to the state with the best value and if all those states had the same value a random move was chosen. This process of move selection can be considered the agents policy.

6.1 Parameter values

For the implementation of Nexting suitable parameter values and timescales had to be chosen. Six distinct time scales, or values of γ, were used for the running of experiments and these were not changed due to time constraints and computational resources.

$$\gamma^i = [0, 0.2, 0.5, 0.8, 0.90.975]$$

The other parameters that were set include the learning rate α the TD value λ and the weight vectors.

$$\alpha = 0.1$$

$$\lambda = 0.9$$

$$\theta^i = \vec{0}$$

For the implementation of the other TD methods identical values of α and λ were used but the value of γ was set to 0.8. These values were chosen to speed up the convergence of the value functions as proposed by (Richard Sutton 2015).

Each Implemented method was given similar conditions to that of the river guppies navigating the construed maze in that they got five attempts to navigate through the maze, simulating the 5 days

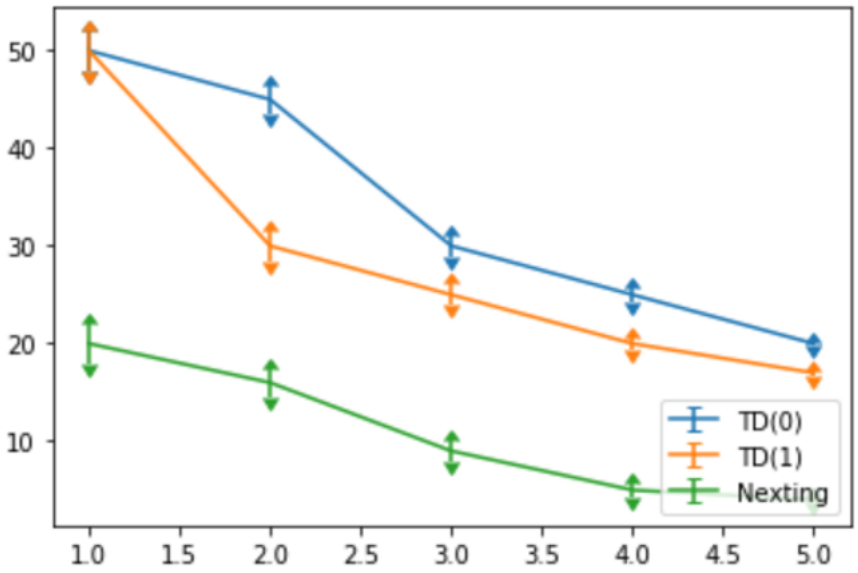

Figure 5: Number of errors made by agent on each trial

that the fish had to learn, in which the value functions were persistent, and for simulating multiple fish, the trials were repeated with slightly different starting values. There were a total of 4 different starting states and each agent was was put through a 5-trial simulation 6 times to account for error.

a wrong turn or mistake was implemented in exactly the same way as the real life experiment in that if a fish crossed a boarder at a turn it would count as an error. The total number of errors was recorded for each trial and stored in a CSV file.

7 Results

From all the trials run it was apparent that Mutli-timescale Nexting performed much better than the other TD methods. Simply from manually observing how the different methods reacted it appeared that nexting agent learned much more quickly from the mistakes it made than the other TD methods. As seen in 5 Nexting starts with a much lower initial amount of errors and although it does not reduce the amount of errors as quickly it still finishes with the least amount of errors overall. Nexting started with an average of 20 errors which slowly reduced to as low as 3 after the 5th trial.

As there was no meaningful way to record the time to solve the maze the number of errors made was the primary metric used for comparison. As there was only one optimal path through the maze errors can be seen as deviations from this optimal path.

In terms of evaluating how the value function performed for Multi-timescale Nexting 6 shows how

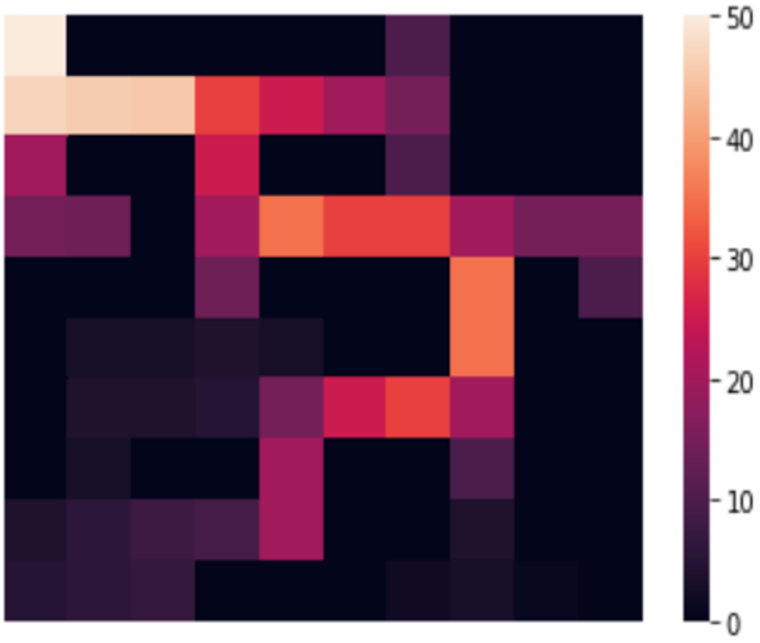

Figure 6: Value function for each state, lighter colour represents more valuable state

that the optimal path is clearly outlined with the corners slightly less valuable as they lead to bad states.

8 Discussion

While the results seem to indicate that multi-timescale Nexting can be comparable with fish learning about their environment the experiments that were run are still in a state such that it would not be easy to fully generalise to a real world experiment. While a number of constraints were place on possible exploration of parameters and environment due to the limited time period and lack of computer resources there are still some issues when directly applying what was meant to be a static prediction model in an active case.

8.1 Exploration-Exploitation issue

When taking a purely greedy approach to selecting states as while that agent must select states it also needs to learn about its environment. In general, this is a non-trivial problem as not exploring the environment means agent will just repeated that first decent solution it finds whereas lack of exploiting the environment will mean the agent will not strive to find an actual solution and even if it does may not even know it.

one proposed solution for this problem is Greedy in the limit of infinite exploration (GILE) (**gile**) that uses an optimistic prior to essentially give the agent the idea that unknown is better than a 'bad' state and essentially over time makes the agent take a more aggressive policy. Given that the environment used for testing was sufficiently small it is unlikely that it would make any meaningful different to the results, but for generalising to a real world scenario may prove invaluable.

8.2 Feature representation

One of the key advantages of Nexting is its ability to handle vast amounts of features due to only having to learn the weights of various value functions. The feature representation chosen for this project was fundamentally simple for ease of implementation and as such did not take advantage of the rich potential for parallel computing that Nexting offers. As such, looking into computer vision approaches for state representation as well as adding features relating to the internal state of the agent could lead to a robot being able to learn to navigate a maze, and work out weather it has enough power to do so.

9 Conclusion

It may very well be possible to teach a machine to learn like a fish, this project has demonstrated that a multi-timescale reinforcement learning model on top of a simple simulated environment can produce behaviour very similar learning patterns to real life fish put in a similar scenario. Perhaps with more tools and environments being made, reinforcement learning may provide a means for a different understanding of fish behaviour in a different way an hopefully a way to model their behaviours going into the future.

References

Bellman, R. (2003). *Dynamic programming*. Princeton University Press.

Gilbert, D. (2006). *Stumbling on happiness*. Knopf Press.

Mckeag, T. (2012). *How one engineer's birdwatching made japan's bullet train better*. Retrieved from https://www.greenbiz.com/blog/2012/10/19/how-one-engineers-birdwatching-made-japans-bullet-train-better

Modayil, J., White, A., & Sutton, R. (2011). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, *22*. doi:10.1177/1059712313511648

Munday P., D. D., Cheal A. (2014). Behavioural impairment in reef fishes caused by ocean acidification at co2 seeps. *Nature Climate Change*, *4*, 487–492. Retrieved from https://doi.org/10.1038/nclimate2195

Neftci, E. O., & Averbeck, B. B. (2019). Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, *1*(3), 133–143. doi:10.1038/s42256-019-0025-4

Rescorla, R. (1980). Imultaneous and successive associations in sensory precon-ditioning. *Journal of Experimental Psychology: Animal Behavior Processes*, *6*(3), 207–216.

Richard Sutton, A. B. (2015). *Reinforcement learning:an introduction*. MIT Press.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593–1599. doi:10.1126/science.275.5306.1593. eprint: https://science.sciencemag.org/content/275/5306/1593.full.pdf

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Nature Machine Intelligence*, *55*(4), 189–208. Retrieved from https://doi.org/10.1037/h0061626

Tyrone, L.-X., & Bisazza, A. (2017). Complex maze learning by fish. *Animal Behaviour*, *125*, 69–75. doi:10.1016/j.anbehav.2016.12.022

Vincent, J. F. V. (2006). Applications - influence of biology on engineering. *Bionic Engineering*, *3*.