MAIA: Instinkters roll vid blomplockning i Minecraft med Q-inlärning

MAIA: The role of innate behaviors when picking flowers in Minecraft with Q-learning

Henrik Siljebråt

handledare / supervisor

Christian Balkenius

KOGM20

2015-09-15

Masteruppsats i kognitionsvetenskap
Avdelningen för kognitionsvetenskap
Filosofiska institutionen
Lunds Universitet

Master's thesis (2 years) in Cognitive Science
Lund University Cognitive Science
Department of Philosophy
Lund University

MAIA: The role of innate behaviors when picking flowers in Minecraft with Q-learning

Henrik Siljebråt
henrik@siljebrat.se

Recent advances in reinforcement learning research has achieved human level performance in playing video games (Mnih et al., 2015). This inspired me to understand the methods of reinforcement learning (RL) and investigate whether there is any basis for those methods in neurobiology and animal learning theories. The current study shows how RL is based on theories of animal conditioning and that there is solid evidence for neurobiological correlates with RL algorithms, primarily in the basal ganglia complex. This motivated a simple perceptron-based model of the basal ganglia called Q-tron, which utilizes the Q-learning algorithm. Additionally, I wanted to explore the hypothesis that adding an innate behavior to a Q-learning agent would increase performance. Thus four different agents were tasked with picking red flowers in the video game Minecraft where performance was measured as quantity of actions needed to pick a flower. A "pure" Q-learner called PQ used only the Q-tron model. MAIA (Minecraft Artificial Intelligence Agent) used the Q-tron model together with an innate behavior causing it to try picking when it saw red. Two mechanisms of the innate behavior were tested, creating MAIA1 and MAIA2, respectively. The fourth agent called random walker (RW) chose actions at random and acted as a baseline performance measure. We show that both MAIA versions have better performance than PQ, and MAIA1 has performance comparable to RW. Additionally, we show a difference in performance between MAIA1 and MAIA2 and argue that this shows the importance of investigations into the precise mechanisms underlying innate behaviors in animals in order to understand learning in general.

1 Introduction

How would you teach a dog to play fetch? All dogs are not the same, but you might start with handing the dog its favorite toy and then take it back in exchange for a treat. Repeat this a few times and you can throw the toy on the ground and when the dog picks it up and hands it to you, it gets a treat again. You can then throw the toy farther away and step by step the dog will learn to play fetch.

In computational modelling terms, this problem is commonly called reinforcement learning. As a cognitive scientist interested in artificial creatures and robots, the primary aim with the current study was to understand the fundamentals of reinforcement learning by simulating a robot that learns to pick flowers in a video game. The secondary aim was to investigate the animal learning basis of reinforcement learning and potential neurobiological evidence for the algorithms. The problem will thus be framed as one of an animal foraging for food. But when building a robot model we need to consider not only the learning mechanism but also how it will see and act.

Recent advances in the use of neural networks for reinforcement learning has been shown to achieve human level performance in game playing (Mnih et al., 2013; 2015). The authors trained a "Deep Q-network" (DQN) to play 49 two dimensional video games as well as, or better, than an expert human player. This was accomplished with only pixel values and game score as input. The method was based on the Q-learning algorithm (Watkins, 1989; Watkins & Dayan, 1992) combined with deep learning (Bengio, 2009). This type of reinforcement learning resembles the operant conditioning used for teaching animals like our dog; for every action there is reward or punishment (Staddon & Niv, 2008).

While the work of Mnih et al. (2015) is impressive, especially the ability of the agent to learn many different types of games with the same parameter settings, there are some issues. First, they state that the performance of their model is highly dependent on experience replay, a memory of previous experiences from which random samples are made to update the behavior of the agent between training episodes. They compare this behavior to the function of sleep and consolidation of memory in the hippocampus and state proudly how their work utilizes "biologically inspired mechanisms" (Mnih et al., 2015, p.532). However, experience replay as a method has existed for more than twenty years (Lin 1991; 1992), so this exemplifies how machine learning research often uses neuroscientific findings as an afterthought more than a goal. Second, Schmidhuber (2015) explains, somewhat bitterly, how their claim of having done something truly new is debatable as something very similar was done by Koutník et al. (2013). Though to be fair, Mnih et al. did present preliminary results in their 2013 article. And third, while the performance of the Atari playing agent may be comparable to a human, the learning rate is surely not. The DQN required 38 days (912 hours; Mnih et al., 2015, supplementary methods) to learn one game, while the professional human games testers used as controls had two hours of practice. Even though being a professional games tester grants a significant amount of experience that is difficult to account for in a direct comparison of learning rate, I am humbly willing to bet that a human ten-year-old with no experience with video games would match the performance of the DQN agent in shorter time than 38 days. It would be an interesting study to test children at different ages on how fast they can learn to play at "expert level" for these games, as we would then have a reference for the cognitive level of an AI agent.

This difference in learning rate can be partly explained by the approach, which utilizes vast amounts of training data to learn the patterns of previous experience (called model-free learning, which will be further discussed below). When the DQN agent fails, it does so because it has not previously experienced that particular game state and the algorithm has not accurately generalized the past experiences into useful information. Humans can learn much quicker because we are able to generate predictions - have hypotheses - about the future based on prior knowledge, grounded in a view of the brain as a pattern predictor where top down connections try to predict the incoming sensory input and learning utilizes the errors in these predictions (Clark, 2013; Friston 2010). The common framework to explain such predictions is Bayesian statistics and is used for example in systems for self driving

cars (Petrovskaya & Thrun, 2009) and models of motor control (Wolpert & Ghahramani, 2000). The important difference between the Bayesian predictions and the ones made by the DQN agent is that the Bayesian approach also takes the uncertainty of reward into account (O'Doherty, 2012).

However, the complexity of the calculations in Bayesian network models have been shown to be NP-hard, meaning that it is simple to confirm if the correct solution has been reached but the time to get there is non-deterministic (Gershman et al., 2015). Computational models therefore need to exploit the structure of the agent's environment in order to solve the two main issues that face these agents; action specification - identify the most useful set of actions - and action selection - find a policy indicating what action to take (Whiteson et al., 2007). These two processes, action specification and action selection, occurs concurrently and dynamically with environmental interaction and has given rise to what is called embodied cognition (Clark, 2013; Wilson, 2002).

Despite the above criticism of the work by Mnih and colleagues, they provided the seed that inspired me to understand the principles of their methods by attempting a similar but simplified Q-learning algorithm. Can this approach be useful to robots seen as embodied creatures in a three dimensional environment?

1.1 Scope and Hypotheses

We will start off with some basic theory of animal learning, then explain reinforcement learning followed by its neurobiological basis. This will show evidence of correlates for reinforcement learning algorithms, primarily in dopamine response of the basal ganglia system. We use these correlates as motivation for a model that combines Q-learning with the perceptron, creating what we call Q-trons. Each Q-tron represents an action and together, a collection of Q-trons thus form a very simple model of the basal ganglia, used as the brain in our agents. Functionality of the Q-tron model was pilot tested in a simple scenario with an agent finding its way through a house to a goal room in as few steps as possible.

While Mnih et al. (2013; 2015) models an agent that plays video games, we instead adapted their approach to model agents as creatures that forage for food. In order to simulate an environment more similar to the one a robot or animal would navigate, the three dimensional video game Minecraft was chosen. Its game world consists of cubes that are combined to create fields, forests, hills, plants and animals similar to how Lego works. The player sees this world in a first person view and move around by using the mouse and keyboard. Interaction mainly consists of "chopping" (hitting) blocks to collect materials which allows for placing these blocks to build structures.

The Minecraft agents were placed in an enclosed pasture filled with red flowers and their goal was to learn how to find and pick as many red flowers as possible, using only game screen pixels as input and picked flowers as the reward signal.

Four agents were set loose in the pasture; the first was a "pure" Q-learner (model PQ), equipped with Q-trons. The second, a random walker (model RW) that chose actions completely randomly and was used as a baseline performance measure. The third and fourth models also used Q-trons but were additionally equipped with innate behaviors, causing them to "chop" if they saw red in the center of the screen. The difference between them were the details of the innate mechanism, and we call these models MAIA1 and MAIA2, where MAIA is the acronym for Minecraft Artificial Intelligence Agent. The addition of innate behavior is a way of exploiting knowledge of the environment similar to how animals evolve such behaviors (see below).

The main hypothesis then, is that the MAIA agents will perform better than the PQ agent. The reasoning behind this is that MAIA only has to learn how to reach the "innate behavior trigger" (seeing red) while the PQ agent also needs to learn how to identify a flower and connect that object with the "chopping" action.

2 Background

"We have a brain for one reason and one reason only, and that's to produce adaptable and complex movements." (Wolpert, 2011)

As an example of this "reason for brains", Wolpert (2011) uses the sea squirt, which in its larval stage has a primitive brain and eyes, swims around and then as it reaches adulthood, plants itself on a rock and consumes its brain. In other words, it is very similar to the more well known couch potato. Now, one may have objections to Wolpert's assumption, but it is a useful starting point to understand learning as a basic process allowing the creature to adapt to its environment.

Learning has been shown in very simple animals like fruit flies, nematodes (Shettleworth, 2013) and even single-cell organisms (Armus & Montgomery, 2010). This is nice, as it gives us a valid reason for using learning as a basic behavior in our artificial agent. But if learning exists at such a basic level, one may ask what role innate behaviors - behaviors that do not require learning - play in producing movements? It boils down to the age old question of genes versus environment, or as Shettleworth (2013) expresses it "attempting to classify behavior as learned as opposed to innate is meaningless" (p.13-14). She uses as example a study of New Caledonian crows, investigating if their use of tools to "fish" for food is something learned or innate. Interestingly, young crows not allowed to see any form of tool use still picked up sticks and poked them into holes around the same age as crows allowed to see tool use, indicating some form of built in tendency to poke with sticks. But learning by way of trial and error still needed to occur for all of the crows to become successful in obtaining food. For this reason, the "innate" behaviors of our MAIA agents - the specifics of which will be described in the methods section - work in a similar fashion; the agent is predisposed to "chop" when it sees red but learning is still required to become efficient in finding and picking flowers. This behavior is further motivated by insects that forage flowers. For example, it has been shown that bumblebees not only use color and scent, but also exploit electric fields around flowers to decide which ones to visit (Clarke et al., 2013).

Adaptable behavior can be framed as the problem of optimal decision making. We want to maximize reward while minimizing punishment, often also referred to as the principle of maximum expected utility (MEU; Von Neumann & Morgenstern, 1947). This principle is behind what Gershman et al. (2015) call "computational rationality", where they see MEU as the goal of perception and action under uncertainty. What this means is that in chess for example, every possible move is evaluated according to its expected utility and you choose the move with the highest value. However, this is what computers do, not humans. Because our brains have limited resources, we approximate the MEU, but this approximation

is in turn bounded by the same limited resources and so is thought to be why humans sometimes make decisions that cannot fully be explained by rational rules (Gershman et al., 2015).

2.1 Training animals with conditioning

As the utility of behavior is closely connected to the question of choice or decision making - how, then, do we make optimal choices to maximize rewards and minimize punishments? The well established theory of conditioning provides some answers. But we should first define reward (also called reinforcer) as an object or event that "elicits approach and is worked for" (Wise, 2004). The value of these reward objects or events are given through either an innate mechanism like food and sex (also called primary rewards) or through learning (Schultz, 2006).

There are two main forms of conditioning; classical (or Pavlovian) and instrumental (or operant) (Cerruti & Staddon, 2003). In classical conditioning, the animal learns to associate a neutral stimulus with an upcoming reward. To use Pavlov's (1927) classic experiment as an example, hungry dogs were given food (unconditioned stimulus, US) following a tone (conditioned stimulus, CS). At first, the food caused salivation (unconditioned response, UR), but after a number of trials the dogs began to salivate when they heard the tone. At this point, the salivation response is called conditioned response (CR). In operant conditioning, the animal is required to perform a wanted behavior in order to receive the reward which is not the case in classical conditioning. In other words, rewards cause changes in observable behavior in both cases of conditioning but in classical conditioning the behavioral reaction does not affect outcome while instrumental conditioning requires a behavioral response for reward to occur. We can now see that our introductory question of teaching a dog to play fetch is a form of operant conditioning.

In both forms of conditioning, the common factor is that the animal learns to predict outcomes. The dogs in Pavlov's experiments learned that the tone predicted the upcoming food reward and our fetch playing dog learns to predict that by performing a behavior sequence, it will receive a reward. Learning is mediated through the errors of these predictions and it follows then that as prediction error decreases, so does learning (for that specific case). This has been formalized in the Rescorla-Wagner learning rule (Rescorla & Wagner, 1972; here adapted from Schultz, 2006):

$$\Delta V = \alpha\beta(V - \lambda) \qquad (1)$$

where ΔV is the change in associative strength (i.e. learning) between the CS and reward and is proportional to the prediction error $(V - \lambda)$; V is the actual value and λ is the predicted value. α and β are parameters that depend on aspects of the stimuli and modifies the learning rate. So what this means is that a reward that is fully predicted does not contribute to learning. It is also important to add that this learning rule works both ways; associative strength can also decrease if reward is withheld. This way of removing an association is called extinction.

What is needed for a reward to become a goal? Balleine & Dickinson (1994; 1998) believes that the goal should be mentally represented in the animal when the behavior is being prepared and executed. This representation should contain a prediction of future reward and the associative strength connecting behavioral action to reward. The animal should also have an established internal representation of reward that

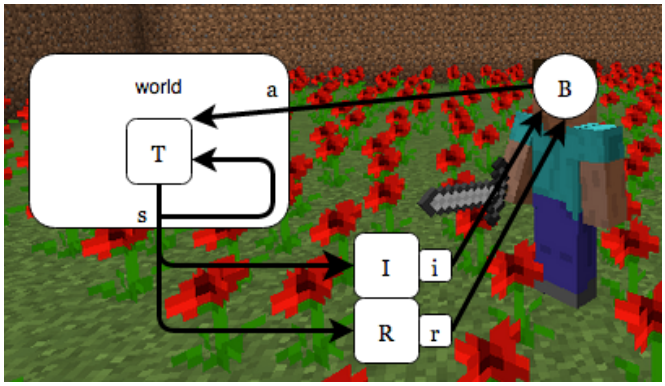

Figure 1. A standard reinforcement model set in our pasture. See text for explanation. The player's face has been covered to avoid privacy concerns.

updates when the reward changes value. The importance of these principles will be shown further below as we describe the neurobiology of reinforcement learning and prediction error.

So, in goal-directed behavior we need to make choices. Since these choices cannot be observed directly and internal states have historically been seen as irrelevant in operant conditioning experiments, choice has been measured indirectly with response strength of the animal and the value of choice alternatives (Cerruti & Staddon, 2003). But we are interested in modelling the internal states and so need to look to reinforcement learning for answers.

2.2 Training machines with reinforcement learning

Reinforcement learning (RL) is a form of unsupervised learning, meaning that the agent explores the environment and the evaluation of outcomes are made concurrently with learning. This can be contrasted with supervised learning, where the system's outcomes are compared to known examples of correct input and output and the system parameters are modified according to any discrepancies found. When RL agents explore their environment, they transition from one state to another by taking actions and is rewarded or punished after each action taken or at some point in the future. This allows for learning to occur without specifying exact behavior patterns and the use of rewards makes it possible to model both forms of conditioning (Kaelbling et al., 1996).

There are two main ways of solving RL problems (Kaelbling et al., 1996). One is to search the behavior space for an optimal behavior, commonly applied with genetic algorithms. In broad terms, genetic algorithms spawn an initial generation of randomized behaviors and the most successful ones are combined (the official academic term for "have sex") to spawn new generations and so on. Another way to solve the problem is to estimate the utility of taking actions in states of the world (the MEU mentioned above), which is the approach we shall focus on presently. It is worth mentioning here that many successful attempts have been made to combine the two approaches (e.g. Whiteson et al., 2007).

The standard reinforcement model can be seen in Figure 1. It assumes a stationary environment - it does not change over time. Neither does the probability of receiving a reward r (also called reinforcement signal). Both of these assumptions are broken in our case, since flowers disappear when picked causing the environment to change and therefore also changes the probability of reward. The innate behavior and exploration

strategy (explained below) can compensate for this. Referring back to Figure 1 again, the behavior *B* should choose actions according to a policy π that maximizes the sum of *r* over the total run time of the simulation. *T* is the state transition function, defining how the agent's action *a* transitions the world from one state *s* to the next. Because the agent does not necessarily know what causes *r* and may not be able to observe the entire world state, *I* and *R* refer to the "true" functions of input and reward, where *i* is the observed input.

RL algorithms now face three main challenges when trying to find the optimal behavior policy π* (Kaelbling et al., 1996). The first is the exploration/exploitation problem. How should we balance exploration of the environment with exploiting the knowledge already gathered? Most techniques progressively decrease the exploration towards zero as time increases, but if the environment is dynamic some exploration must always occur. The exploration/exploitation strategy is usually denoted ε, and a common strategy for dynamic environments is ε-greedy. It chooses random actions (exploration) with a probability *p*, which can be adjusted during the agent's training. When exploiting its knowledge, it always chooses the action with the highest value, hence the name greedy. In more dynamic environments, it may be that several actions have values that are very close which the ε-greedy policy does not consider. So a more refined version would be the Boltzmann distribution for exploration as demonstrated by Mnih et al. (2015).

The second problem is how the algorithm should take the future into account, since the agent has no way of knowing how many steps there will be until reward is received. Building upon the above mentioned MEU, a basic and common optimality model is the infinite-horizon discounted model (Barto, 2007; Kaelbling et al., 1996). The expected utility is the sum of all future rewards geometrically discounted with the parameter γ ($0 \leqslant \gamma < 1$). So for each state we have a value, $V(s)$, like so:

$$V(s) = E(\sum_{t=0}^{\infty} \gamma^t r_t) \qquad (2)$$

The E is for expected value, so this means that with gamma close to 1, future rewards will be more important than immediate rewards and vice versa, so that we can adjust the importance of those future rewards for the the expected utility. Equation (2) forms the basic reasoning behind the more complex algorithms we will soon discuss. Interesting to note here is that regardless of the value of gamma, early rewards will be worth more than later ones, which is precisely how I like my marshmallows.

The third problem is the question of convergence and performance. Some algorithms can be proved mathematically to converge to the optimal policy π*, but for more complex environments and agent behaviors this is not always possible. In those cases, the performance of the chosen optimality model can instead be measured by the speed with which it converges to optimality or near-optimality and we can also measure the level of performance after a given time (Kaelbling et al., 1996). The assumption of (2) is also that there are regularities between the state signals (*i* in Figure 1) and reward values, if there are no such regularities the predictions become random guessing (Barto, 2007). We can model these possible regularities using Markov chains, a method that calculates future probabilities based on the current state without relying on previous ones (Sutton & Barto, 1998).

A common family of methods building on Markov chains is temporal difference (TD) learning (Sutton & Barto, 1998) upon which many different variants have been developed. By utilizing a specialized form of Markov chain called Markov Decision Process (MDP), they can be described as extending the Rescorla-Wagner rule (Niv, 2009). We shall not explain Markov chains or MDP's in detail, but very briefly, and as mentioned above, Markov chains and MDP's allow us to model the probabilities of the state transition function and the probability of reward for a state (as seen in Figure 1). The important take-away here is that these methods assume having a model of the world, but model-free (see below) versions exist that build on the ability of the agent to explore and take samples of the environment (*i* and *r* in Figure 1). With increasing amounts of samples, the agent can approximate the true functions *I*, *R* and *T* (Figure 1). TD learning managed to explain something that the Rescorla-Wagner rule could not; second order conditioning, where a learned association like tone-food could be built upon to create for example light signal-food by coupling the light signal to the tone (Niv, 2009).

Now, to further complicate the matter, these methods also rely on the assumption that state transition probabilities are fixed. This means that they describe Pavlovian conditioning and not instrumental conditioning, which is what we are interested in for our agent. As Niv (2009, p.8) puts it: "Since the environment rewards us for our *actions*, not our *predictions* (be they correct as they may), one might argue that the ultimate goal of prediction learning is to aid in action selection." Indeed, this leads us to the "temporal credit assignment" problem (Sutton & Barto, 1998) - how do we figure out what actions led to reward?

2.2.1 Finding the optimal policy

If we have a model of the environment we can find the optimal policy in two ways; value iteration or policy iteration (Sutton & Barto, 1998). Each state has a value $V(s)$ and the optimal value for a state, $V^*(s)$, is the expected infinite discounted reward (equation (2)) that will be gained in that state given a policy π. The optimal policy $\pi^*(s)$ is the optimal way of choosing actions given values *V*. By iterating over values and using mathematical relations between V^* and π^* we can find one given the other. So by iterating over values, we find the policy by successive approximations of V^*. In policy iteration, given an initial action, all *V* are calculated for each state. Then the initial action is changed to see if the total *V* is better and if so the policy is updated to choose that initial action instead and progressively π^* is approximated.

When we do not have a model of the environment, which is the case for our flower picker MAIA, there is still hope. Either we learn the behavior *B* (Figure 1) without a model (model-free methods) or learn an approximation of the model and use that to choose the behavior (model-based methods). These model-based methods should not be confused with the previously described case where we actually have a model of the environment. Model-free methods have the disadvantage of using gathered data extremely inefficiently and so need lots of experience to achieve good performance, while model-based methods are faster in learning time but instead require significantly more computational power (Kaelbling et al., 1996). There are however ways of combining the two methods for improved performance over using either one and also indications that a similar combination exists in the brain (Gershman et al., 2015; also see below section on biological basis for reinforcement learning).

An interesting observation to make here is that the DQN agent by Mnih et al (2015) uses a model free algorithm (Q-

learning; see below) which is part of the reason for why it needs such a long training time as 38 days. But even then, their use of computational power to do that training is massive. They unfortunately do not mention how much power they used, but it is a safe bet that they did their training on clusters of graphic processing units (GPU). One such GPU uses around 200 watts while the human brain uses around 20 watts (Versace & Chandler, 2010). So if a model-free algorithm requires at least ten times the power a brain does, a model-based method would require - as just mentioned - significantly more.

2.2.2 Q-learning

Q-learning (Watkins, 1989; Watkins & Dayan, 1992) is a model-free method that builds on TD-learning. It combines policy and value iteration by directly approximating the optimal policy and looks like follows:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a)) \quad (3)$$

where *s* is the current state, *a* is the action chosen in that state, *s'* the new state after *a* is taken and *a'* the predicted optimal action in state *s'*. As in equation (1), *r* is the reward and γ the discount parameter. The α refers to the learning rate, a parameter used to adjust how much the Q-value can change every update. It should also be noted that the reward value can have positive and negative values, reflecting reward and punishment, respectively. Adding negative values for actions that do not lead to reward can be interpreted as adding cost for actions.

So what happens here is that the agent is in a state *s* and chooses an action *a* according to some strategy. That action is performed and the agent observes the new state *s'* and its reward *r*. The trick lies in using the predicted optimal action *a'* for the state *s'* (*maxQ* in (3)) to update the Q-value for the state-action pair (*s*, *a*). In other words, the agent stores estimations about the value of possible actions for each state and as actions are taken, the actual received reward is compared to the estimation. The difference is the prediction error which is used to update the estimation function. Thusly, the prediction error for Q-learning is:

$$error = \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a)) \quad (4)$$

Progressively, as the prediction error approaches zero, the estimation function in turn approaches the true values of actions. To compare with the Rescorla-Wagner rule in equation (1); *r* is the actual reward given and *Q*(s, a) is the predicted value. The Q-learning algorithm has been proven to converge with probability 1 as long as each action is executed in each state an infinite number of times on an infinite run (Sutton & Barto, 1998). In other words, convergence of the algorithm is unaffected by the exploration strategy (*ε*, as mentioned above).

However, the strategy of exploration/exploitation still needs to be carefully chosen for learning rate to be as quick as possible. Especially as there are very few cases where we can realistically visit every state-action pair an infinite number of times. The parameters are also important to choose wisely. Bayer & Glimcher (2005) found in their study that an α of .7 concurred with their physiological recordings. In practice, however, it is feasible to use a constant α of 0.1 (Sutton & Barto, 1998). For γ, we want to choose a high value in our case of picking flowers; since there may be many steps before a reward is received, we need to take into account potential actions and rewards far into the future.

2.2.3 Continuous states and partially observable environments

So far, the methods discussed have assumed that we have discrete states and actions, so that the predicted values can be stored in lookup-tables. When we move to continuous or very large spaces, like our agent MAIA walking around in the flower filled pasture, we need to use some other representation of mapping state-action pairs to values. This is called function approximation for which there are many methods but we will focus on the use of artificial neural networks (described in more detail below). The side effect of using such a function approximation is that convergence can no longer be guaranteed (Kaelbling et al., 1996).

Another difficulty is when the agent cannot observe the entire state of its environment, only parts of it (as described with the relation between I and i in Figure 1). This is the case for MAIA since it only observes what is in front of it. If you, dear reader, would be out picking flowers in a pasture, you would also observe only part of your environment. To fully observe the environmental state, you would have to know exactly where all flowers in the field are at all times, like having a live satellite feed. In short, most real world problems need to handle the case of partially observable environments and these are commonly modelled by partially observable Markov decision processes (POMDP), though it is also possible to ignore the problem and rely on the randomness in the exploration/exploitation implementation (Kaelbling et al., 1996).

However, one can also utilize knowledge of the world and implement predefined knowledge into the agent. A nice example of this is Tesauro's (1995) TD-gammon, a backgammon playing agent. He had a basic version with minimal prior knowledge and an improved version with explicit knowledge of backgammon pre-programmed. The basic version was not nearly as effective as the improved version which achieved performance comparable to backgammon masters. As previously mentioned, this is the approach MAIA uses with its "innate" behavior.

2.2.4 Methods to improve performance and convergence

The DQN agent by Mnih et al. (2015) was highly dependent on the method of experience replay. The DQN agent stores every experience - a sequence of state-action-reward-state - and during the regular update of the Q-value these experiences are randomly sampled and used for smaller updates of the same value. This is slightly different from Lin (1991; 1992) who instead uses lessons - a collection of experiences from starting state to end state - and at the end of one lesson, the experiences are iterated over recursively to update the Q-value. The experience replay method is itself based on a similar mechanism used in Sutton's (1990) Dyna architecture called relaxation planning, where the model generated hypothetical experiences to update the Q-function. Lin (1991) states that experience replay is faster and does not require a model to be learned (since Dyna is model-based).

Another method is the eligibility trace (Barto, 2007). Like experience replay, it also saves state-action pairs and updates them according to their eligibility trace parameter, which decides how many steps back these updates should be made, the size of the agent's short term memory if you so will. More recent state-action pairs have a larger impact than those farther back in history. The eligibility trace also has the advantage of being capable of on-line updates, without explicitly storing previous states (Kaelbling et al., 1996).

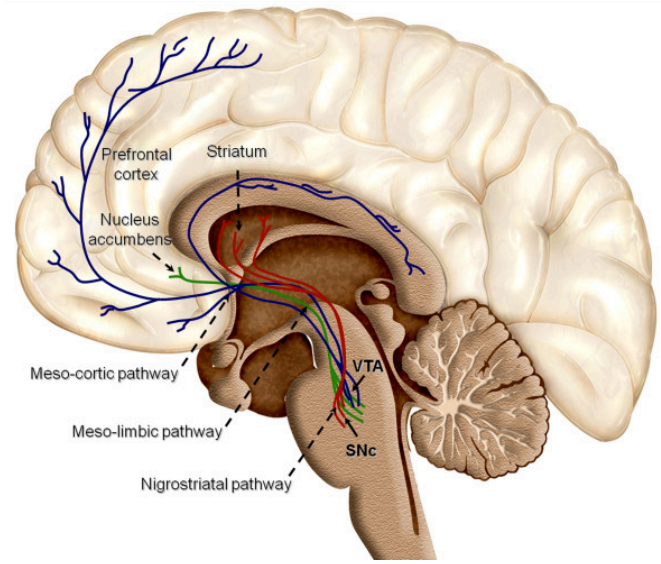

Figure 2. It is very difficult to find a picture showing all areas we mention, but this one shows the VTA and SN pathways; both of which will be shown to be important in relation to reinforcement learning algorithms. Adapted from Arias-Carrión et al. (2010).

2.3 Biological basis for reinforcement learning

As we have seen, reinforcement learning is a big and complex set of tools and theories. But since we are interested in finding something out about animals - and by extension, humans - is there any relevance to reinforcement learning? What biological connections are there, if any?

Whatever one's opinion on behaviorism's failure to account for internal states, the approach has given us loads of behavioral data on classical and instrumental conditioning that we can use for neurobiological investigations of reward. We will here briefly review findings on established and possible connections for RL as a computational model of learning in animals. But first, some general observations on reward and dopamine.

Reward in neurobiological studies is often used as a collective term for effects not only of reinforcement but also motivational arousal, the latter meaning that an animal given reward is "energized" before and during the next trial (Wise, 2004) or more formally; preparatory or approach behavior toward reward (Schultz, 2006). It seems to only affect "extra" arousal however, as dopamine blockage (usually done with neuroleptic drugs) still allows normal responses for previously rewarded stimuli (Wise, 2004). One should also keep in mind that reward is not always easy to separate from attention in these kinds of studies (Schultz, 2006; also discussed below).

Nevertheless, the neurotransmitter dopamine has been identified with both motor and motivational functions (Wise, 2004), making it highly interesting in relation to our previously mentioned RL algorithms that build on reward for action selection. It is, however, difficult to clearly distinguish between these roles in animal experiments. Since we cannot speak to the animals, we cannot differentiate between "wants to but cannot" and "can but does not want to" as Wise (2004) puts it. This difficulty in boundary distinction is reflected in the neuroanatomy as we shall see presently.

As seen in Figure 2, most dopamine cells develop in the mesencephalic-diencephalic junction (VTA and SNc area) and project to forebrain targets (Wise, 2004). Motor function is mostly connected to the nigrostriatal system (substantia nigra (SN) and striatum) and motivational function with the ventral tegmental area (VTA). The VTA has two main systems, the mesolimbic and mesocortical dopamine systems, where the former mainly projects to nucleus accumbens and olfactory tubercle and less so to the septum, amygdala and hippocampus and the latter projects to prefrontal, cingulate and perirhinal cortex (Wise, 2004). There is considerable overlap between the mesolimbic and mesocortical systems and they are therefore often referred to as the mesocorticolimbic systems. To add to this complexity, there is no clear boundary between the nigrostriatal system and VTA which makes it, as mentioned previously, difficult to distinguish between the motor and motivational dopamine systems (Wise, 2004).

This complexity for dopamine's role in goal-directed behavior is summed up by Wise (2004, p.9): "a conservative position would be that dopamine acts in nucleus accumbens, dorsal striatum, amygdala, frontal cortex and perhaps other sites to reward immediate behavior and to establish conditioned motivational cues that will guide and motivate future behavior." Though details have been added since then the larger picture of their integration into a general theory largely remains unanswered (Schultz, 2015; Dayan & Berridge, 2014; but see below on basal ganglia). We shall thus not go much further into the rabbit hole of non-conservative positions, but will add that Schultz (2006) reports that for primary rewards, activations of most of these structures have been shown to reflect the actual reward and not the sensational experience itself (that is, the reward of food and not the taste of it) and also that there are indications that different parts of these structures may code for different dimensions of stimuli (like vision, olfactory or spatial information, all of which can be interpreted as the factors α and β in equation (1)).

When it comes to reinforcement, dopamine modulates many aspects of conditioning and seems to be most crucial in positive reinforcement (Wise, 2004; Schultz, 2006). When blocking dopamine function, for example with neuroleptics, both Pavlovian and instrumental conditioning is impaired in different ways (Wise, 2004). But the role of dopamine is still very unclear, as is evidenced by Wise (2004) who reports; first, that dopamine blockage is functionally equivalent to intermittent omission of reinforcement showing similar progressive decline in responding to unrewarded trials; and second, that dopamine blockage can have the opposite effect - that animals are unable to learn that they are not given rewards.

We thus have many conflicting findings regarding dopamine's role in reward and learning, and should keep in mind that other neurotransmitters and brain areas also are involved in learning. But the dopamine system is the most studied in regards to reward, and it seems to have such an important role for reinforcement learning that it leads Schultz (2006) to formulate the following general rule:

$$DA\ response = reward\ occured - reward\ predicted \quad (5)$$

Again, we have a rule that is similar to the Rescorla-Wagner rule in equation (1) and also the one for Q-learning in equation (4). Can this really be true? Let's find out.

2.3.1 Reward prediction error hypothesis of dopamine

The first clear evidence of dopamine neuron activation representing prediction error came in the early 90's (Montague et al., 1996). Schultz et al. (1997) added to this by showing how previous studies on recordings from dopamine neurons in monkeys during conditioning tasks showed a

striking similarity with the error functions described by TD-methods (Figure 3).

Compare the results shown in Figure 3 to the story of Pavlov's dogs told above. If given food at a random time, dopamine neurons in the midbrain will fire more rapidly to reflect that something unexpected happened as in Figure 3 (a). After being trained with receiving food at a set length of time after a tone, the reward will no longer be unexpected but the tone will still be unexpected, so the prediction error reaction in the dopamine neuron now happens after the tone as in Figure 3 (b). And even more intriguingly, in Figure 3 (c) we can see that if the reward is withheld from the trained animal, the neuron will decrease its firing rate at the precise time when reward was predicted to occur, indicating that the neuron also encodes the timing of reward. However, Bayer & Glimcher (2005) found evidence of dopamine neurons encoding only positive reward prediction errors, meaning the case in Figure 3 (c) might be encoded by different neurons and they suggest serotonin as a possibility, since it has been implicated to have a role in other aversive events. It could also be the case that the divergent results can be explained by the different recording sites used, as Bayer & Glimcher (2005) recorded from the substantia nigra (SN, see Figure 2) area and Schultz et al. (1997) from the ventral tegmental area (VTA; see Figure 2). It has later been suggested that the SN and VTA circuits may be functioning in accordance with the actor/critic TD-model, where the SN circuit works as the actor - learning the action-selection policy - and the VTA circuit works as the critic - learning predictions of the world, essentially the model of the world (Niv, 2009).

What this means is that the previous hypothesis of dopamine representing reward occurrence was not the whole story, it seemed to instead (or rather also; see mention of motivational arousal above and attention below) represent a prediction error precisely in line with the logic behind the learning rules we have discussed. This has become known as the reward prediction error hypothesis of dopamine (Montague et al., 1996), and many other aspects of TD models have been shown to be reflected in dopamine neurons (Niv, 2009; Schultz, 2007; Schultz, 2006; Wise, 2004; Schultz, 2015), one of which is the previously mentioned second order conditioning. If the monkey in Figure 3 (b) is trained with an additional CS before the tone, the prediction error response will move from the tone to that additional CS. Another important example would be the dopamine response reflecting that the most recent reward is more important for the error function than rewards farther back in time (Bayer & Glimcher, 2005). This is similar to the function of eligibility traces and Pan et al. (2005) show that they may indeed be essential to dopamine error functioning.

Most of those results however were based on Pavlovian or very basic instrumental conditioning tasks, so what of more complicated cases of action selection, for example one where all available actions lead to reward but of different magnitude? Morris et al. (2006) conducted a study that elucidated in more detail what kind of TD-related method could best explain the dopamine neuron response in such a task. Monkeys saw four kinds of stimuli presented to the left or right of a screen and were trained to respond with a left or right movement of the arm, corresponding to the placement of the stimuli. The four stimuli were followed by reward with different probability ranging from .25 to 1 in .25 steps. The results from SN neuron recordings indicated that the dopamine prediction error response reflected an action choice already made, instead of reflecting the selection process itself. This detail is not in line

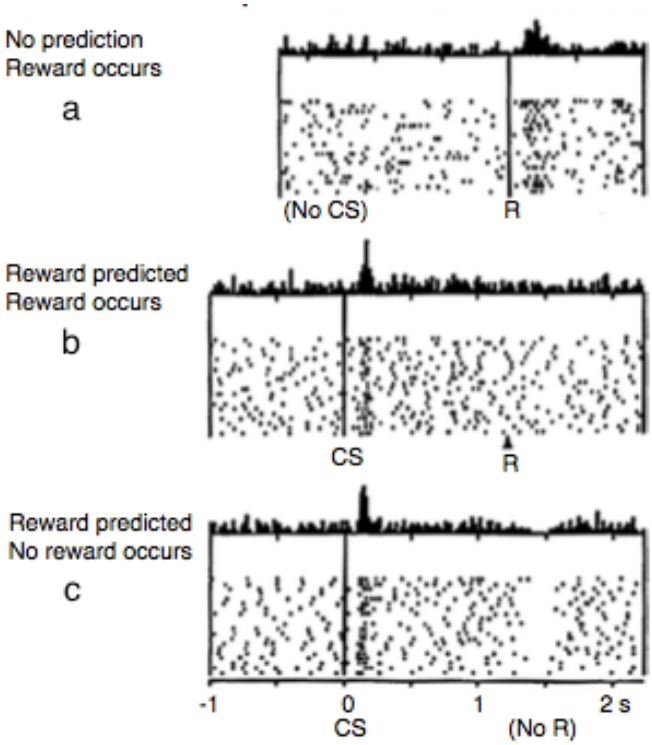

Figure 3. Each row of dotted line represents a single trial, where each dot is the neuron firing. The bars on top of each graph represents the average over those trials. The conditioned stimulus CS was a tone and the reward R was fruit juice. In (a) the reward R occurs without CS and the neuron increases activity to report that something unpredicted happened. (b) shows how after training, the prediction error response has moved to just after the CS and since the reward R has been learned to be predicted, there is no change in activity when it is given. And in (c) we can see how the neuron activity is decreased when a trained animal expects a reward, but it is not delivered. This last example shows how information about the timing of reward delivery also seems to be encoded in the neuron's activity. Adapted from Schultz et al. (1997).

with the actor/critic model but is the difference between how Q-learning and its related method SARSA works, the latter being the one found in the monkeys (Morris et al., 2006). In Q-learning, the prediction error is determined by the action with the highest Q-value (*maxQ* in equation (3)) and not the action that was actually chosen, while in SARSA the prediction error is determined by the chosen action (Niv et al., 2006). However, in a similar task but done with rats making odor decisions and recordings made in VTA dopamine neurons, Roesch et al. (2007) found their results to be more similar to the Q-value representations in Q-learning than those in SARSA (or other methods).

Additionally, some dopamine neurons in dorsolateral prefrontal cortex seem to represent different behavioral outcomes in that one neuron may activate for reward and moving left, while another activates for the same reward but for moving right to receive that reward (Schultz, 2006). However, these activations may also be related to some form of "excitement" or even attention as mentioned previously.

This may all seem very promising, but as with all neuroscience research, the deeper you go the more complex a picture arises. O'Doherty (2012) mentions how there seems to be two different behavioral mechanisms underlying action selection, in that one is is model-based and used for goal-directed selection of actions based on their value (which is based on later reward) and the other a model-free mechanism for stimulus-response associations. We have previously

mentioned the actor/critic method and also how combinations of model-free and model-based methods can perform better than either alone (Gershman et al., 2015), but as we saw these cannot explain all aspects of neurobiological findings.

There is furthermore the issue of finding evidence for RL in humans, as we have until now mostly discussed animal findings. We shall not go into much detail, as the model MAIA uses is not complex enough to warrant any comparisons to human functioning. Also, human behavior is often so different between individuals that it is difficult to draw comparisons to specific algorithms (Shteingart & Loewenstein, 2014). But it is worth mentioning that there are indications of neural correlates for the dopamine prediction error in humans (Niv, 2009).

2.3.2 The basal ganglia and action selection

Several brain areas have so far been mentioned, like the VTA (ventral tegmental area) and SN (substantia nigra). We shall now present these in their larger context - the basal ganglia. Interestingly, the basal ganglia system seems to be very similar across vertebrate species, suggesting that its role is very old in evolutionary terms (Redgrave et al., 1999; Redgrave, 2007). That role seems to be, as we have seen in our previous discussion, related quite closely to learning and action selection in a reinforcement learning framework. Though we should mention that the basal ganglia system is also involved in other functions like fatigue and apathy (Chakravarthy et al., 2010).

Before we move on, we should mention here the role of attention in action-selection and action specification. Perceptual attention mediates both action-selection and action specification, so there is a problem of knowing exactly what is selected and where it happens (Prescott et al., 2007). Action selection may be a global property of the brain and body in their environment (as per embodied cognition mentioned in the introduction). But if there in fact are specialized components for this problem then we need to determine what is required by them. Redgrave et al. (1999) does exactly this, and they have four criteria for such a system. Its inputs should provide information about relevant internal and external cues; each available action should have an associated utility value calculated by a mechanism internal to the system; conflicts between available actions should be resolved based on those values; and the system's output should allow the winning action to be performed. Luckily, the basal ganglia seem to match these criteria (Redgrave et al., 1999; Redgrave, 2007; Prescott et al., 2007; Chakravarthy et al., 2010), but as we've seen in the discussion on dopamine error signals, the details are unclear. Those details can be very important and as an example of this, Prescott et al. (2007) describes a study with computational modelling showing that the location of synapses on input neurons to the basal ganglia have significant effects on network function.

2.4 Synthesis

For the implementation of MAIA, we will not pay attention to attentional processes. We will instead focus on action selection within the reinforcement learning framework as modelled by Q-learning. Perception is handled by taking screenshots of the game screen, so what the agent will receive as input is roughly similar to retinotopic maps found in LGN/V1 of the visual processing stream (Wandell et al., 2007). Motor output is also not handled in detail, our system will select an action and that action is performed by sending the appropriate key or mouse click to the game. Since the basal ganglia is likely to play an important role in action selection and because of its old and stable evolutionary history, we claim that our implementation is a highly simplified model of the basal ganglia. Additionally, in order to keep the model simple, only positive reinforcement will be used and there will be no cost for taking actions. This has the added benefit of fitting the neurobiological findings better, because the dopamine error hypothesis mainly has been proven for positive reinforcement. Moreover, the behavior meets the four criteria of a specialized action selection function as defined by Redgrave et al (1999). In this way, MAIA is able to capture the essence of the "Wolpert premise" presented earlier - that the reason for brains is to produce adaptable movements.

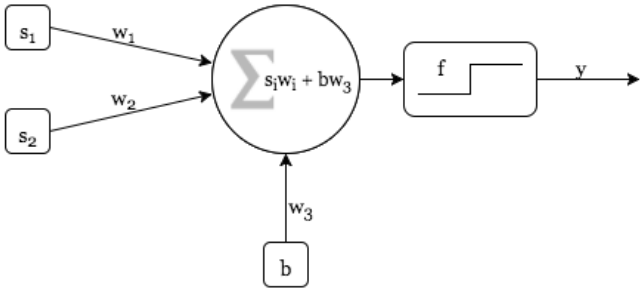

Figure 4. The perceptron. Inputs s_1, s_2 and b are first multiplied by their respective weights w_1, w_2 and w_3. They are then summed and put through an activation function f, creating the output y.

3 Methods

Earlier we mentioned that the MEU (maximum expected utility) needs to be approximated. We do this with the Q-learning algorithm, which in turn needs to be approximated for use in continuous state spaces. There are many methods for such function approximation, but we will do this with an artificial neural network (ANN). One can question how plausible ANN's are as models of biological neuronal networks, but that is a question outside the scope of this work. Besides, as Balkenius (1995) argues; because we try to build a model of a creature, ANN is the most reasonable method.

Briefly, an ANN is a simplified model of biological neural circuits, implementing an algorithm similar to Hebbian learning (Dayan & Abbott, 2001). A neuron is represented as a node and synapses by connections between nodes. Each connection has a "weight" (w) associated with it - usually a real number - that represents synapse strength. Connection inputs (s) to a node are multiplied by their respective weight and put through an activation function f which creates that node's output as seen in Figure 4.

3.1 The perceptron

The simplest type of node to understand is the perceptron (Rosenblatt, 1958). It works as just described and can mathematically be described like so:

$$y = f(\sum_{i=1}^{n} w_i s_i + wb) \qquad (6)$$

where y is the perceptron's output and b is the bias input which has its own weight. The bias can be understood spatially as allowing the function to move away from origo, similar to b in the simple linear equation:

$$y = ax + b \qquad (7)$$

Due to its simplicity, the perceptron is limited to solving linear discriminations. Meaning that, if we continue the analogy with our linear equation (7), the perceptron can find a line separating two types of data among a collection of data points. So how do we teach the perceptron to do useful things? We use supervised learning as an example and the following step function (also seen in Figure 4) as the activation function:

$$y = \begin{cases} 1 \; if \; ws + b > 0 \\ else \; 0 \end{cases} \quad (8)$$

We can teach the system to perform AND and OR operations by inputting examples and compare the perceptron's output with the known correct output. We then use the error of the output to adjust the weights. In Table 1 you can see training examples for the AND operation.

Table 1. The AND operation is true (represented by y as 1) if inputs s_1 and s_2 are the same. If they are not the same, the AND operation is false (represented by y as 0).

s_1	s_2	y
0	0	1
0	1	0
1	1	1
1	0	0

Usually, the weights are initialized randomly, so if we take the example from the first row of Table 1 and have weights w_1 as -0.5, w_2 as 0.5, w_3 as 0.2, with b as 1 and use equation (6) we get:

$$y = f(s_1, s_2) = s_1 w_1 + s_2 w_2 + w_3 b = 0.2$$

Which will, according to (8) give 1 as output. Here we can see the role of the bias. With the chosen activation function the perceptron would not be able to solve the problem if not for the bias, since regardless of the value of weights 1 and 2 the function would always be 0 without the bias and so not be able to output the correct 1 as we would like for an AND operation.

If we go to the second row of Table 1 we instead get:

$$y = f(s_1, s_2) = s_1 w_1 + s_2 w_2 + w_3 b = 0.7$$

This will give 1 as output according to (8), which is incorrect; it should be 0. The weights will need to be updated in order for the perceptron to output the correct value. How do we do this then?

We use an error function that calculates an error value, commonly called the delta rule. In its simplest form it is:

$$error = correct \; value - calculated \; value \quad (9)$$

If we call the calculated value a "prediction" we can see that (9) is the same principle as the prediction errors we have discussed above, like the Rescorla-Wagner rule in (1) and the dopamine response in (5). So for our second example above it becomes:

$$error = 0 - 1 = -1$$

In order to avoid large changes in weights (too large of a change may "step over" the solution), we adjust the error with a learning rate α. As mentioned in the Q-learning section above, we can choose α as 0.1 and calculate the new weight values:

$$w_i = w_i + \alpha(-1 \cdot w_i)$$

Meaning that every weight's new value is its own value multiplied by the error added to its old value. This goes on and on for the examples in Table 1 until we always receive correct output (or as close to always as we can; to converge on the correct solution). Important to note here is that since there may be many possible values for w that will solve the problem, we are not guaranteed to converge on the optimal configuration of weight values. To summarize our example, we can now see that the perceptron approximates the function in (8). The "knowledge" of this approximation is stored in the weights, similar to how the memory mechanism of long term potentiation is represented by synapse strength (Kandel, 2001).

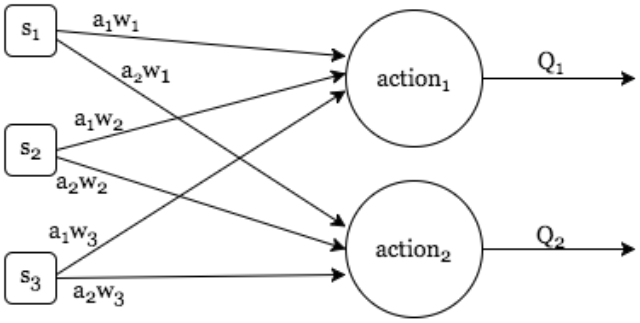

Figure 5. A collection of Q-trons, creating our simple basal ganglia model. Every action is represented by a Q-tron ($action_1$ and $action_2$), each with its own weights. The output value, Q_1 and Q_2, is stored by the Q-tron, ready to be exploited.

We should also note that since the perceptron is a linear classifier, it can only learn to distinguish between two categories of data if a line can be drawn between them. To again take an example from logical operations, the perceptron cannot learn the XOR (exclusive or) operation. A regular OR operation would return y as 1 for rows two, three and four in Table 1. XOR, however, is not true if both s_1 and s_2 are true. This makes it impossible to draw a straight line between true and false for XOR. This can, however, be solved by using several "layers" of perceptrons. In that case, inputs are sent to a layer of one or more perceptrons, all of which send their outputs to one or several "hidden" layers and finally to the end output.

Such a network of perceptrons is usually what is meant by "artificial neural networks". To again interpret this spatially, additional layers of perceptrons can be seen as creating additional lines so we can make a finer grained classification between data points. Convolutional neural networks, as used by the DQN agent, are in principle many such layers organized in a hierarchical structure that progressively recognize features of an input image in a similar fashion as the hierarchical structure of the visual cortex.

3.2 The Q-tron

Now then! How can we use the perceptron to handle states and actions with Q-learning? Why, with a Q-tron of course!

The Q-tron represents an action that the modelled agent can perform, for example "take one step forward". So the basic assumption for the Q-tron is that the action space for the agent is discrete, while the state space can be continuous. Referring to Figure 5, the agent observes the world as an array of arbitrary size, represented by the state s. The state does not have to be an RGB (Red, Green, Blue) image, it can be any numerical array. When a Q-tron is created, it sets its initial Q-value to zero and initializes its weight array with random values between -0.5 and 0.5. The Q-tron itself only handles the storage and update of Q-values, meaning it handles equation (3). Action selection (exploration/exploitation),

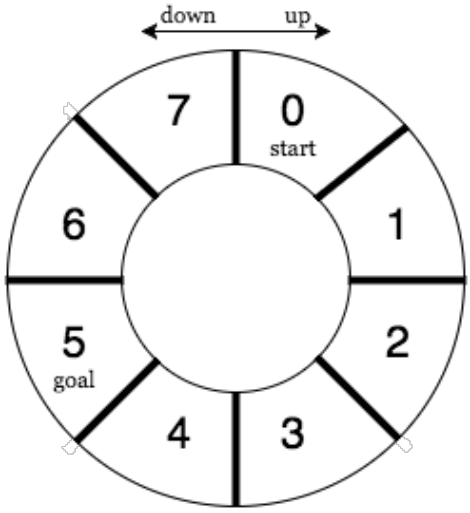

Figure 6. A circular house. The agent always starts in room 0, from where it is possible to move "up" to room 1 or "down" to room 7. The other rooms work the same, so the agent can always move one room "up" or one room "down". The goal is always room 5.

terminal state, etc. is done in the main program itself. This is so that the network of Q-trons can more easily be extended with hidden layers in the future. During a forward pass - calculation of the Q value - the weights are multiplied by the state as in (6) and put through a sigmoid activation function (equation 10), which limits output between 0 and 1.

$$f = \frac{1}{1+e^{-x}} \qquad (10)$$

The Q-tron updates its weights with a method called back-propagation (Rumelhart et al., 1988) which uses the delta rule with gradient descent to approach the wanted value. This is where we adapt the error for the Q-learning equation from (4) to be used in the same approach as in (9).

Gradient descent uses the derivative of the output value, which can be spatially interpreted as finding an optimal value (minimum or maximum) by taking steps in the direction of the slope of the function. The principle of this can be understood by imagining a three dimensional landscape with valleys and hills. So when an action has been taken, the error tells us if that step took us towards the top of the hill that we want to climb or not. With derivative of the sigmoid function being:

$$\frac{dy}{dx} = sigmoid(x) \cdot (1 - sigmoid(x)) \qquad (11)$$

we get a gradient descent step where "error" is equation (4) and looks like:

$$step = error \cdot Q(s,a)(1 - Q(s,a)$$

and the weights are then updated with:

$$w_i = w_i + w_i step$$

From Figure 5 it may seem like the Q-trons are unaware of each other, but even though only the Q-tron representing the action performed is updated, it takes the other actions' Q-values into account through the *maxQ* function. Also, as mentioned above on RL and Q-learning, we cannot be certain of convergence for the Q-trons. Specifically, we have two main problems here. The first is that gradient descent may find a local optimal value, meaning we find a hill but not the tallest hill in the world. There is also the possibility that we find the

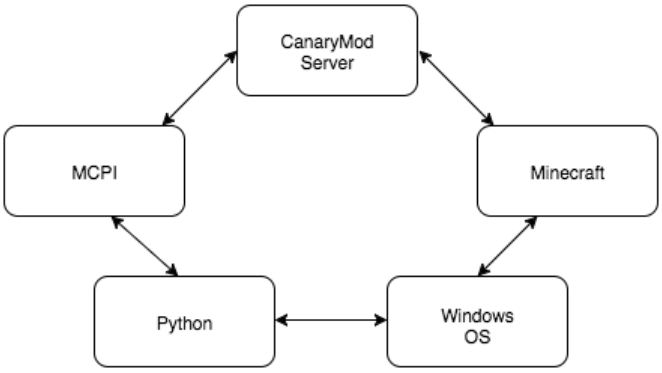

Figure 7. General layout of the software used. The program itself was written in Python, which communicated with MCPI to send and receive information from the CanaryMod server. Python also communicated with Windows to send keyboard and mouse commands and receive information on where the Minecraft game window was located and coordinates for screenshots. Minecraft connected to the CanaryMod server, which ran the plugin CanaryRaspberryJuice (not shown) allowing it to communicate with MCPI. Technically - if you are so inclined - everything communicated with the Windows operating system since all software was run on the same machine.

tallest hill, but it takes significant amounts of time to get there. This is where our innate behavior comes in; if we implement it properly, it may allow for faster convergence up that hill.

3.3 Getting through the house

As we soon shall see, the simulations in Minecraft take many hours to complete. So in order to confirm that the Q-trons work as expected, a small pilot test was constructed. This test of the Q-trons was made in an environment constructed as a circular house, exemplified in Figure 6. The simulation used 40 rooms instead of the eight rooms in the figure, but the principles are the same. The agent always started in room 0 with the goal of finding the shortest route to the goal; room 5. The state *s* of the house was programmatically represented as a one dimensional array where the agent was represented by the value *10* and empty rooms by the value *0.1*. So, in the case seen in Figure 6, the shortest route to the goal is to go "down" from 0 in three steps. In the simulation case, however, the shortest route would be to go "up" in five steps.

As previously described, the agent has one Q-tron for each possible action so in the house environment there are two Q-trons; one for "up" and one for "down". Training of the Q-trons was done in the following fashion:

Algorithm 1.

```
Initialize Q-trons with value of zero and random weights
Repeat (for each episode):
        Observe starting state s
        Repeat (for each step of episode):
                With probability epsilon, select random action a,
                otherwise select optimal action as a
                Take action a, observe reward r and new state s'
                Update Q-tron representing a, using given r, s, s'
                Set s' as s
        until s is goal state
```

3.4 Software

The main applications that were used and their relations can be seen in Figure 7. The complete source code for MAIA with

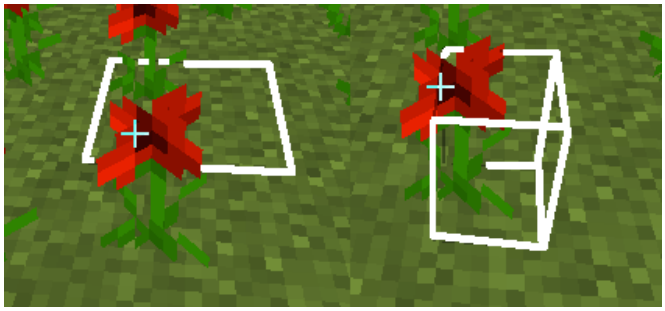

Figure 8. The cross is the center of the viewport and the white outline (exaggerated for clarity, in the game the outline is a thin black line) is the "block" that is currently selected and will be interacted with if a mouse button is pressed. As clearly seen, a very slight change in perspective changes what block is selected.

installation instructions can be found on GitHub[1]. It also contains the CanaryMod server (see below) used for the project, since that server distribution is no longer maintained so providing it allows for easier replication (and improvement!) of the results. Initially an attempt was made to create a cross platform implementation but due to performance speed with screenshot capture and functionality issues with keyboard and mouse input, the final implementation is Windows only.

3.4.1 Minecraft

Minecraft[2] is a 3D computer game created by Markus Persson and released in 2011 by Mojang, later sold to Microsoft. The game world is similar to our own, with hills, rivers, valleys, fields, animals and plants. There are also portals to other worlds and a wide variety of monsters, all of which may or may not reflect reality. The game's environments are made from blocks, creating a distinguishable "blocky" look that makes features fairly big and non-detailed. This makes the game a good test environment for our intelligent agent.

The player sees the world in first person view and can collect materials by "chopping" blocks (destroying them) and combine those materials to craft items with which to build structures. Movement is made with the keyboard and the camera view is changed by moving the mouse around. Interaction with the world works through clicking the mouse buttons.

It is not without its problems, however, as can be seen in Figure 8. To select a block for interaction, it needs to be centered in the view. But smaller objects like flowers have smaller "selection boxes" which means that a flower may be centered in view but not selected. The opposite is also true; a flower may be selected without being central in the view. This of course complicates our task.

3.4.2 CanaryMod Server

The CanaryMod[3] Server 1.2 (development of which unfortunately seems to be discontinued as of summer 2015 but can still be downloaded) is a custom version of the official Minecraft server and allows the use of plugins. One such plugin is CanaryRaspberryJuice[4] and it opens a port allowing the Python module MCPI to communicate with the CanaryMod server. This way, we can manipulate the player and the world blocks with Python code. Furthermore, the

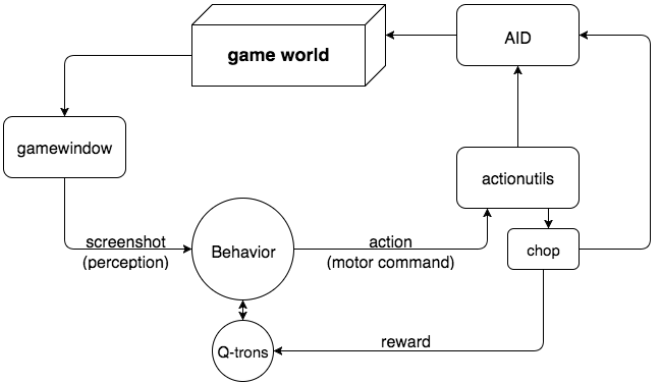

Figure 9. Starting with the 3D game world, the gamewindow component observes the game state by taking a screenshot. This percept is forwarded to the Behavior component, which communicates with the Q-trons to decide on an action. The selected action is sent to actionutils, which in turn calls on AID (Artificial Interface Device) to send the action command to the game. In case the chop action is selected, the chop module will check the selected game block to see if it is a flower and if it is gone after the chop action has been performed. The reward signal is then sent back to the Q-trons for updating.

CanaryMod server allows many custom settings and the main ones used for MAIA was turning off monsters, health and hunger and set long lasting daytime so the input colors were held at constant luminance.

3.4.3 Python and its many libraries

Python[5] 2.7 was used since the availability of libraries are still better than the now seven years old Python 3. The previously mentioned MCPI module allowed Python to communicate with the CanaryMod server. The other main libraries used was Numpy for mathematical operations, Pillow for taking screenshots and PyWin32 to access the Windows operating system in order to get coordinates for the Minecraft game window and send keyboard and mouse input. A full list of libraries used can be found on MAIA's GitHub page.

3.5 MAIA - Minecraft Artificial Intelligence Agent

MAIA consists of three main components; eyes, brain and motor control together with several subcomponents to allow communication between the main components and provide the possibility of saving data for analysis. An overview of the main components can be seen in Figure 9 and will presently be presented in more detail along with additional modules and components. We should mention here that the other agents of course use most of these modules as well.

3.5.1 gamewindow

These functions utilize the win32api to find the Minecraft game window, focus it so action inputs are fed to Minecraft and not some other active window and also returns the screen coordinates for the center of the game window itself. These coordinates are used by the previously mentioned Pillow library to grab frames of a chosen size from the center of the game screen. These frames are grabbed as RGB tuples, converted to a 1D Numpy array and returned for use as state input to the Q-trons. For example, a grabbed frame of size

[1] https://www.github.com/fohria/maia
[2] http://www.minecraft.net, version 1.8 for the Windows operating system
[3] https://github.com/CanaryModTeam/CanaryMod
[4] https://github.com/martinohanlon/canaryraspberryjuice
[5] http://www.python.org

50x50 pixels will be returned as a Numpy array of size 50x50x3.

3.5.2 Behavior

This algorithm is similar to the one presented for "getting through the house" but with a few additions, mainly the innate behavior to chop if seeing red. We classified something as red in two ways; the first (MAIA1) counted how many pixels in a captured frame matched exactly the RGB values of the four light red pixels on the flower rings. If at least five such pixels were found, the frame was classified as red. The second approach (called MAIA2) looked at the average amount of red in the observed state using the following formula:

$$\frac{R}{0.5(G+B)} > 2 \qquad (12)$$

where R, G, B are the average values of those components in the grabbed frame. For both versions, if a frame was classified as red, there was a probability of 0.8 that the chop action would be selected. With one flower chopped as the goal state and exploration/exploitation strategy ε-greedy, the algorithm is thus:

Algorithm 2.

```
Initialize Q-trons with value of zero and random weights
Repeat (for each episode):
        Observe starting state s
        Repeat (for each step of episode):
                If s is red, with p=0.8 select chop as action a,
                otherwise;
                        With probability epsilon, select random
                        action a,
                        otherwise select optimal action as a
                Take action a, observe reward r and new state s'
                Update Q-tron representing a, using given r, s, s'
                Set s' as s
        until s is goal state
```

3.5.3 Q-trons

The actions made available were step left, right, forward or backward, look left or right and chop. This makes for seven actions and thus seven Q-trons. Since we used RGB frames of size 50x50, the total number of weights was 7501, where the last one is for the bias.

3.5.4 actionutils

The take_action.py function receives the selected action and calls on the AID module to perform the key press or mouse click associated with that action. Most importantly, get_block_position.py and chop.py work together to enable counting of the amount of flowers that have been picked. They use the MCPI event function pollBlockHits which listens to right clicks made when the player has a sword equipped. Yes, that is correct, this only works with a sword. Only peasants pick flowers, real knights of the round table cut them with sharp swords because that way the flower will last longer in their loved ones' vase. If a block is in range, the event function returns the type of block clicked. This event listener required many hours of careful testing in order to get working as intended for the current implementation, since it needed to synchronize with in game animations and AID calls to actually return something.

Every move action - step left, right, forward and backward - was hard coded as sending the key press for 0.25 s. Looking left or right was defined as moving the mouse cursor 150 pixels to the left or right, respectively.

3.5.5 AID

When humans interact with computers, input devices are commonly called Human Interface Devices so here we use the AID - Artificial Interface Device. In short, it uses win32api calls with C structures to send key presses and mouse clicks to Windows and in turn Minecraft, if that is the currently focused window.

3.5.6 API

This is a small nodejs[6] websocket server that listens to incoming traffic on one port from the running Maia application and redirects that information to another port that the client connects to. The client is run in a browser and uses morris.js[7] to display graphs of the current Q-value for the different actions. The API server was mainly used for debugging purposes and turned off during the simulation runs.

3.5.7 counterupdaters.py

Contains functions to update performance measure counters and save these to disk both during the simulation and when the program exits.

3.5.8 prepareworld.py

This is run before every new round (see below for explanation of rounds) and uses MCPI to create a pasture with flowers. It finds the current position of the player, places a grass block directly below the player and additional grass blocks in a square with sides of 20 blocks around that grass block. On the grass it then places red flower blocks and around the grass blocks a dirt wall two blocks high so that the player cannot get out of the pasture. It also places air blocks around and above the player to avoid that any part of the pasture is in shadow from the in game sun, as that would change the shadowed flowers' colors. This means that the agent always starts in the center of the pasture every new round.

3.6 Hardware

Virtual machines could unfortunately not be used, as the graphics capabilities were insufficient. Instead, the simulations were run on three desktop computers, two of which used Windows 7 and one used Windows 8.1.

3.7 Analytical tools

Analysis was made with the programming language R[8]. Actions taken per episode were averaged over simulation runs and plotted with the ggplot package, using the generalized additive model (Hastie & Tibshirani, 1990) for house simulations and local polynomial regression fitting (Cleveland et al., 1992) for the Minecraft simulations.

[6] http://www.nodejs.org
[7] http://morrisjs.github.io/morris.js
[8] https://www.r-project.org/

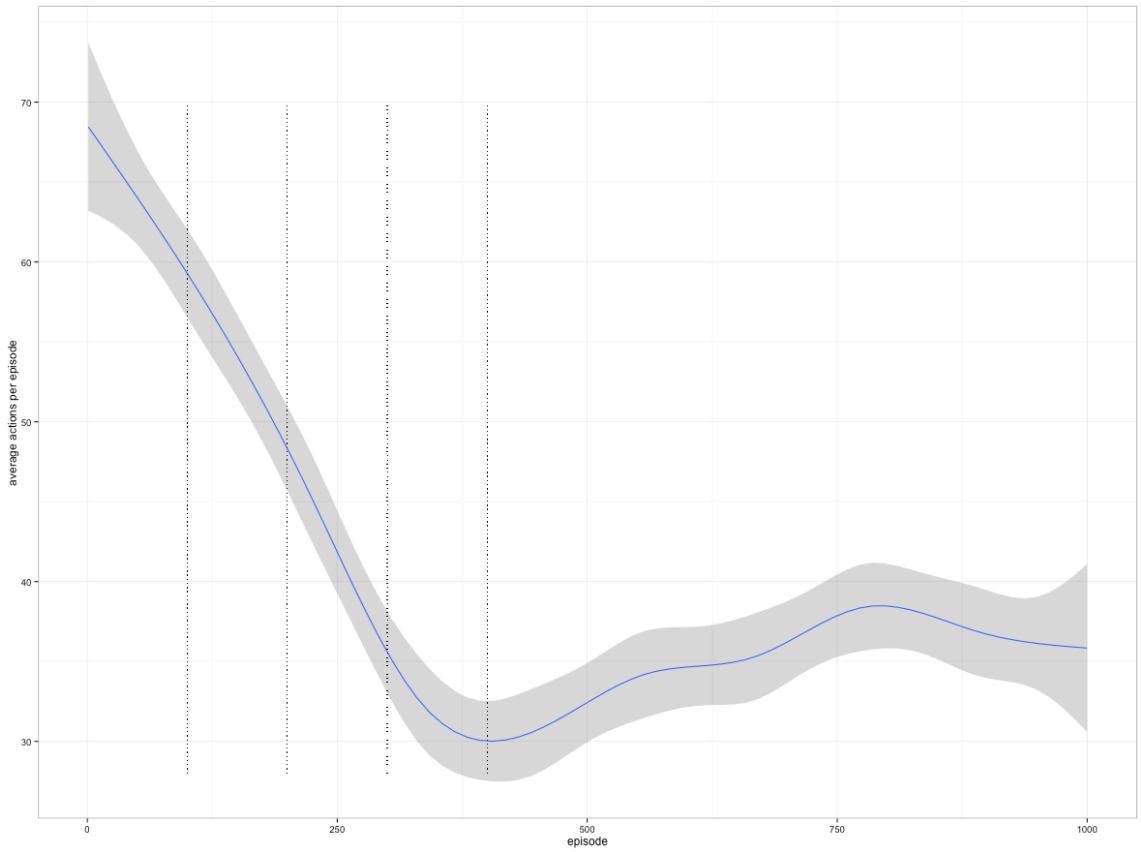

Figure 10. Average number of actions per episode for the house agent, aggregated over 50 simulation runs. The solid line represents the mean and the shadow is the standard error. Vertical dotted lines indicate when learning rate α was decreased.

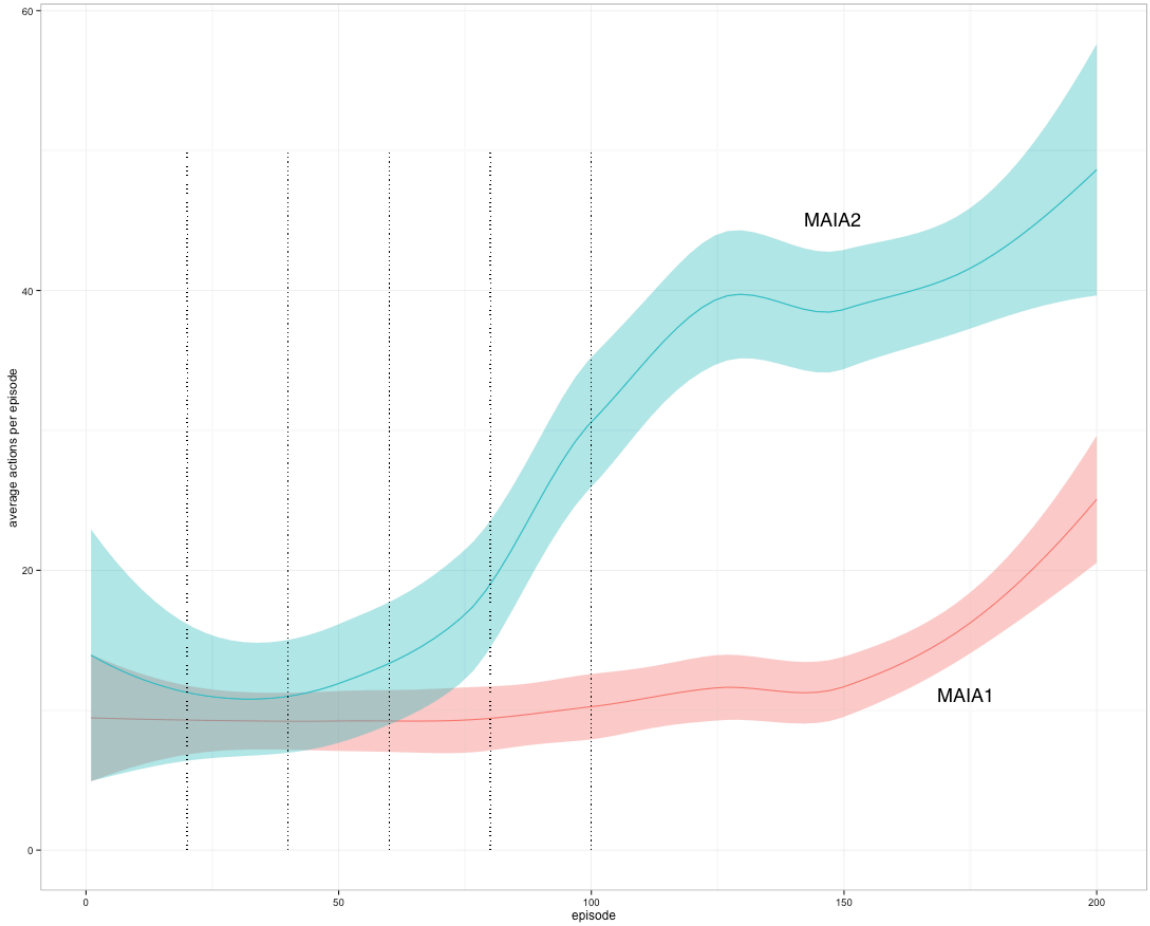

Figure 11. Average actions per episode for the two versions of MAIA that have different red check mechanisms. Unfortunately, no learning seems to occur. The solid lines are the mean values and shadows represent standard error. Vertical dotted lines indicate when learning rate α was decreased.

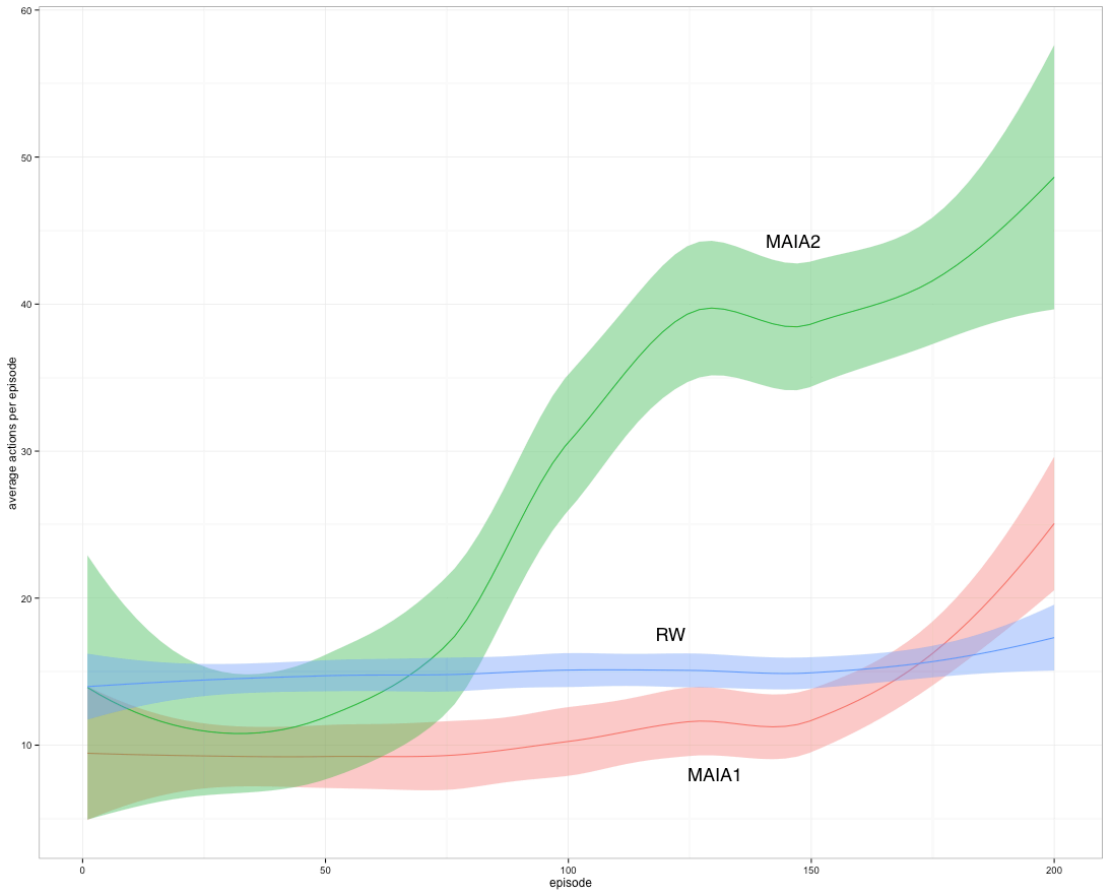

Figure 12. Average actions per episode for the random walker RW contrasted with the two versions of MAIA. MAIA1 is slightly better than RW until after around 150 episodes. Solid lines are the mean values and shadows represent standard error.

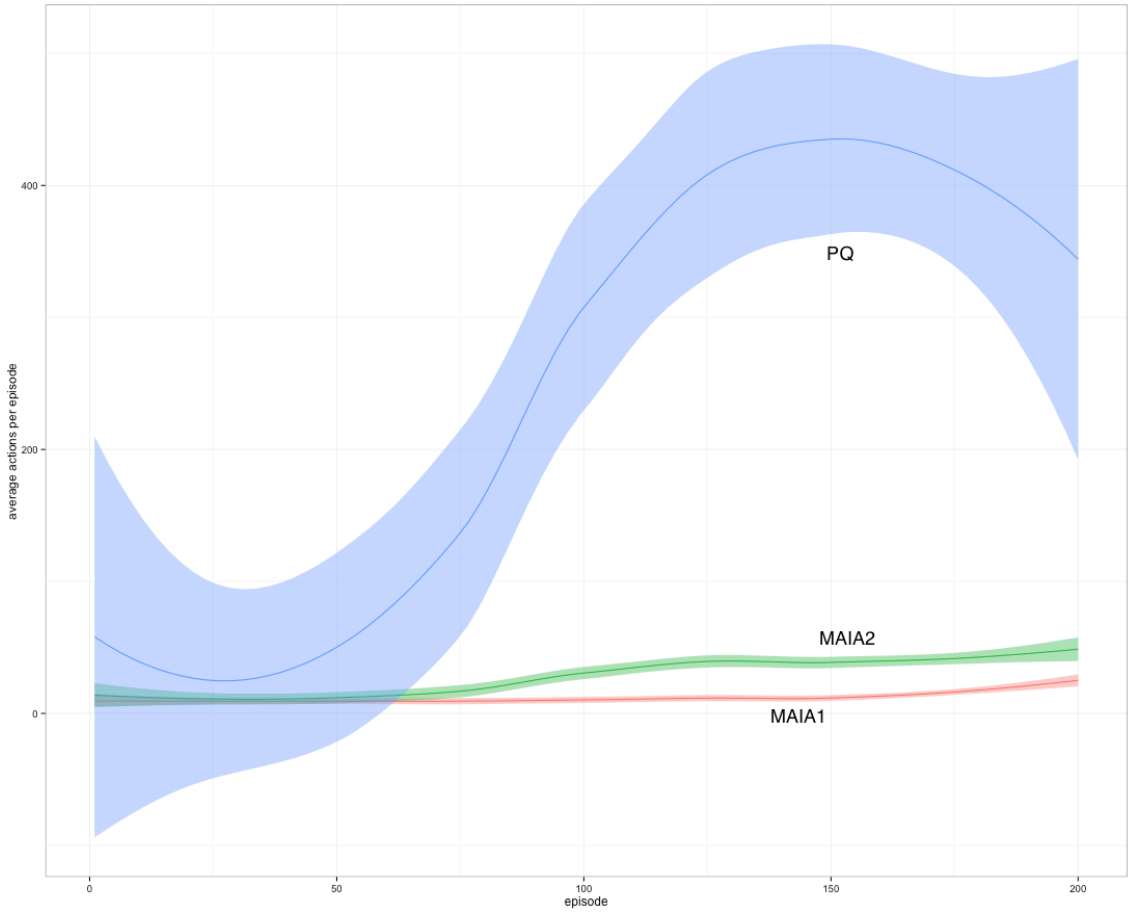

Figure 13. Average actions per episode for the pure Q-learner PQ contrasted with the two versions of MAIA. We can clearly see that PQ performance is worse. Solid lines are the mean values and shadows represent standard error.

4 Experiments and Results

For clarity, we will here first explain the experimental procedures and then the results of the experimental simulations.

4.1 Getting through the house

Training of the house agent (Algorithm 1) was performed with 1000 episodes for one simulation run. The parameters used for the house agent were α=0.1 and γ=0.9. ε was set to an initial 0.5 and was progressively decreased in steps of 0.1 every 100 episodes and then kept at 0.1 for episodes 401-1000.

Figure 10 shows average number of actions taken each episode for an aggregate of 50 simulation runs for the house agent. As can be seen, performance increases (fewer actions per episode) until around 400 episodes where performance starts to decrease again (increasing actions per episode). Kaelbling et al. (1996) mentions how such oscillatory behavior can occur even in simple environments and Mnih et al. (2013) uses experience replay to solve this issue. It may also be an artefact of the low exploration rate (0.1) used after 400 episodes in combination with the randomness inherent in the problem, causing the agent to alternate between choosing up and down as the optimal route. In other words, only in some of the runs will the agent learn to go up in five steps and in some runs it will go down in 35 or more steps, explaining the average being between 30 and 40 steps for episodes after the 400 mark.

Nevertheless, we can clearly see that performance in general increases during the simulation run so it seems that our Q-trons work. It is also worth mentioning that the decreasing performance at the "tail" was mainly due to the influence of one out of the fifty runs.

4.2 Minecraft simulations

As mentioned previously, one episode equaled one flower picked. Each simulation run used 200 episodes and was also divided into rounds of 20 flowers picked. When one round was complete, a new pasture was created and ε was decreased by 0.1. ε was initially set to 0.6 for all q-learner versions and held at 0.1 for episodes 101-200. γ was always 0.9 and α=0.1. These parameters should ideally be tested with different values for different simulation runs in order to find the best combination. Unfortunately, time did not allow for this, so even though these values are reasonable, they have been chosen somewhat arbitrarily. Due to the same time constraints, initial attempts with simulation runs of more than 200 episodes (and different round lengths) were abandoned.
The random walker RW used only random actions of the seven available each step. The "pure" Q-learner PQ used Algorithm 1 while MAIA1 and MAIA2 used Algorithm 2. The difference between MAIA1 and MAIA2 was the mechanism for seeing red, explained in section 3.5.2 above.

4.2.1 MAIA1 versus MAIA2

MAIA1 was successfully run 13 times, with a mean run time of 0.84 h (SD=0.25). MAIA2 was successfully run 14 times, with a mean run time of 1.95 h (SD=1.25). Like for the house test, an aggregate over simulation runs was averaged for both MAIA versions and are presented together in Figure 11. We can see that MAIA2 seems to decrease in performance as the chance for random actions, ε, decreases. MAIA1 on the other hand does not decrease in performance until around 150 episodes, which is mainly caused by a few outliers. Unfortunately, neither version shows any sign of learning, like the house agent in Figure 10. This will be further discussed below, but briefly, there are two main reasons for this discrepancy. First, the house agent observes the entire world state while MAIA only observes parts of it. Second, the house state is reset each time the goal is reached, while in the MAIA simulations the world is reset every twenty flowers. This makes the rewards occur randomly, which can cause performance issues as discussed in section 2.2.

As an interesting side note, version 1 was the only agent (except for a random walker) that successfully completed runs of 1000 episodes in less than 24h.

4.2.2 Random walker versus MAIA siblings

Ten random walker (RW) simulations were run with a mean run time of 1.01h (SD=0.05h) and is presented together with the MAIA siblings in Figure 12. We can see that the RW agent has steady performance over all episodes as is to be expected. Also, it may at first seem like a win for MAIA1, but most likely, a random walker with a red check function would at least match its performance, meaning that version 1 of MAIA is at random levels of performance until around 150 episodes when it gets worse.

4.2.3 PQ agent versus MAIA siblings

The "pure" Q-learner (PQ) successfully completed five runs with a mean run time of 18.7h (SD=22, shortest run time was 4.9h and longest 57h). The average actions per episode over these five runs are contrasted with the two MAIA versions in Figure 13. The pure Q-learner is not performant compared to both MAIA siblings, which is in line with expectations.

4.2.4 General observations

Animals should be observed in their natural environment and since the two MAIA siblings have never known anything else than the flowered pasture, it is as much their habitat as anything else. It was not feasible to observe all simulations all the time of course, but some observations worth mentioning were made.

The most successful simulations had look left or look right as the dominant behavior, causing the agent to chop innately, turn, chop and so on. The downside of this strategy was that the agent created a circle around it without flowers, but a few successful exploration steps towards uncut areas allowed it to begin circling again. For the same reason, sometimes this did not work at all, because the few exploration actions taken either made the agent for example go forward and then backward again or it stepped towards areas already chopped.

Another common strategy, but less successful, was for one of the step movements to become dominant. This works fine early in a round, as it steps left for example, chops innately, then steps left again and chops another flower. But soon it steps left into a wall and the agent then in the best case started sliding along the wall with repeated left movements and chopped innately as it passed flowers near that wall. In the worst cases though, the agent moved "straight" into the wall and not sliding along it (or moved into a corner), so only the specific action of look right or look left would save it.

Moreover, it was not uncommon for the innate behavior to cause the action count to increase. Its function works nicely when going from an empty space to where there are flowers, but as seen in Figure 8, sometimes the innate behavior caused

the agent to continue chopping many times without reward because the selection was a ground block instead of a flower. In those cases, there is only a 20% chance it will go on to explore/exploit actions.

5 Discussion

This project was born of a desire to understand the principles of the DQN agent by Mnih et al. (2015) and learn the fundamentals of programming an agent with reinforcement learning. In that regard, I believe it was a success. The other goal was to investigate the biological basis of reinforcement learning and whether an innate behavior would improve the performance of our agent. As seen in Figure 13, that is certainly the case.

But what kind of animal is MAIA supposed to be? Depending on what "level" of species we believe the model corresponds to, the results would be impressive for a single cell organism but abysmal for a dog. Both cases are worthwhile though, since by studying common features of neural biology – in this case the basal ganglia (which the single cell organism lacks) – between species we can figure out how other areas work, because evolution often builds on previous structures. A possible objection to the approach taken here is that the innate behavior is motivated by behavior in both vertebrates such as dog and monkey as well as foraging insects. I believe the reasoning is sound though, from the perspective of a creature in a pasture of red flowers, however small and contained that world may be.

More concerning is that the current results are not in agreement with the role of the basal ganglia system as an "action selector", since the general impression from Figure 12 is that we might as well use a random walker. The pattern we would have liked to see is the one from Figure 10, but both versions of MAIA seem to decrease in performance as the exploration rate goes down. In other words, the high degree of randomness in early rounds "saves" the MAIA agents from getting stuck. A silly interpretation would be that all previous research is wrong and the basal ganglia has nothing to do with reinforcement learning or action selection. A more reasonable interpretation is that our model does not work as intended. Why is this?

Most importantly, there is no cost for performing an action. A real creature uses energy when it moves and thus cannot walk perpetually into a wall without dying of thirst or hunger. Cost could be implemented as a negative reward given for those actions that do not lead to the goal state of a chopped flower. That way, the "optimal" behavior of walking into a wall or corner would more quickly result in that action's Q-value to decrease and so allow another action to become optimal and chosen when the exploit chance is high. Also, since there are four actions for taking steps and two for looking around, there is a higher chance of a step action being chosen initially, and if the new state causes an innate chop the agent is likely to continue the step movement into a wall.

Furthermore, there is the simplicity of our model. We mentioned how several layers of perceptrons allows the network to solve more difficult problems. And as we can see in Figure 8, there are tiny differences between having a flower or the ground selected. The effect is that the input to the Q-tron network is basically the same for both of those cases, making the problem almost impossible to solve. Every added layer in a neural network allows for finer features to be detected, and since those selection lines in Figure 8 are so small we would probably need a deep convolutional network in order to detect them.

Additionally, the Q-tron implementation does not utilize Markovian methods in any way. MAIA can only see one screenshot at a time, making the task partially observable because it is not possible to know the context from that single screenshot. The DQN agent (Mnih et al., 2015), for example, constructed MDP's out of series of state-action-reward chains and used those series as states in the Q-function. In that way, it could avoid MAIA's problem.

As mentioned, the innate behavior was meant to partly remedy the implementational weaknesses of the Q-tron. But there were cases where the innate behavior wrongly identified flowers, because of the mentioned "selection problem". There are ways we could construct a better innate behavior, for example; check every quadrant of the grabbed frame for equal amounts of red which would allow the innate behavior to trigger for centrally located flowers. We could also train a neural network with a genetic algorithm, teaching it to identify selected flowers. It would probably have to be quite large, but that training could be done off-line and then put in as a module in the program. This method would be the most reasonable for an innate behavior, since natural creatures are likely to have evolved their behaviors in a similar way.

The downside of using an innate behavior is that the implementation becomes task specific. If we wanted MAIA to pick yellow flowers, performance would fall to levels on par with the pure Q-learner. The DQN agent, on the other hand, is capable of learning to play many different games with the same settings and parameters. However, it still needs to train each game separately. There is no transfer of learning, i.e. by training on one game it also becomes better at another. Animals, and especially humans, do transfer some kind of general knowledge from experience; if you have played many video games you are likely to have a better initial performance in a new game compared to a video game naïve human. We can compare this line of thought with the example of New Caledonian crows from the background section. They apparently have some sort of innate ability to sample their environment for state-action correlations and learning mechanisms guide this behavior to successful implementation. Humans have more generalized ways of sampling both their environments and bodies, just look at babies wildly flailing their limbs around. So innately speaking, there seems to be a difference in degree of generalization of innate behaviors between species, underlying the sampling necessary to update the prior distribution of Bayesian models (Wolpert & Ghahramani, 2000; Clark, 2013; Friston, 2010).

Which takes us to the questions of what an innate behavior is and where does it come from? Is it a specialized function or a general one? If specialized, does it work like showed here with some manipulation or control of input that works in coordination with action specification in the basal ganglia or is it a separate process that bypasses regular action selection? In the background section, we referred to Shettleworth's (2013) statement that differentiating innate and learned behavior is "meaningless", because they interact in ways that make them very difficult to disentangle. But what the results here show, especially those in Figure 11, is that the specifics of the innate mechanism play a very important role for behavior. It is therefore not beneficial to call the question of innate versus learned behavior for a meaningless one. It is rather a highly meaningful question to ask because we need to find the answer in order to build proper models. For if we

cannot build such models it is doubtful we have truly understood anything at all.

5.1 Future work

With the assumption that the action selection role of basal ganglia is correct, how do we increase performance while conforming to biological findings?

There are many possibilities for improving the Q-tron model of the basal ganglia, with three main approaches. First; to expand the perceptual input with preprocessing or an ANN with more capacity similar to the discussion above on innate mechanisms, i.e. a deep network with hierarchical layers similar to visual cortex structure. Second; motor control would ideally be expanded with motor signals instead of input to the game, as it would allow for easier transfer of the model to a physical robot. For a nice example of a combination of these two "expansion slots", see Levine et al. (2015). The third option is to expand the complexity of the current model.

It would be simple to state that to improve our Q-tron model, we would add deep convolutional networks with experience replay, Markov methods and aversive reinforcers or costs for performing actions. But as mentioned, it is not self-evident where these methods fit into the neurobiological picture. If an MDP series of state-action-reward sequences is used as one state as in the DQN agent, would we still be talking about the basal ganglia? Or is it rather something that happens in conjunction with short and/or long-term memory in frontal cortex and hippocampus? And since the biological evidence mainly points to a role for basal ganglia components in positive reinforcement, where does cost of performing actions come from – as a specialized component in the learning algorithm or a general property of the entire system that is refilled with food and drink?

Similar questions can be asked for eligibility traces and experience replay. It is likely that both are necessary, since one is made on-line and explains some neurobiological findings (eligibility trace; Pan et al., 2005) while the other looks similar to memory consolidation during sleep (experience replay; Mnih et al., 2015). But in the latter case, if hippocampus is to be involved, we also need to consider the spatial maps created there (Hafting et al., 2005).

So an interesting way of improving MAIA would be to add a hippocampus module that dynamically creates a spatial map, in which MDP series of state-action-reward sequences are stored. If successful, this could generate predictions testable in real creatures and physical robots. In the latter case, it would be especially interesting to see if training can be done in Minecraft and transferred to a physical robot. Another interesting venue would be to combine the theory behind the prediction errors shown throughout our previous sections with the more general prediction error theory of Friston (2010) in an effort to find common ground.

There is a long road ahead before we reach the point of robot dogs that can play fetch. As we have shown, much of today's cutting edge research is based on algorithms many decades old. But the union of machine learning and neuroscience can help us find new answers to the questions we have and the connections between algorithms and biology for reinforcement learning show this particularly well. Is it not exciting that correlates for learning algorithms are found in dopamine neurons? I think so, and I intend to look deeper.

Acknowledgements

This work is dedicated to Sloke, the only dog that ever won my hearth. May you and your family rest in peace. Many thanks to my friends and family for discussions, coffee and support. And last, but certainly not least, I want to thank my supervisor Christian Balkenius for being generally awesome and inspiring me with his enthusiasm and curiosity.

References

Armus, H.L., Montgomery, A.R., & Jellison, J.L. (2010). Discrimination learning in paramecia (P. caudatum). *The Psychological Record*, *56*(4), 2.

Arias-Carrión, O., Stamelou, M., Murillo-Rodríguez, E., Menéndez-González, M., & Pöppel, E. (2010). Dopaminergic reward system: a short integrative review. *International archives of medicine*, *3*(1), 24.

Balkenius, C. (1995). *Natural intelligence in artificial creatures* (Vol. 37). Lund University.

Balleine, B.W., & Dickinson, A. (1998). Goal-directed instrumental action: contingency and incentive learning and their cortical substrates. *Neuropharmacology*, *37*(4), 407-419.

Barto, A.G. (2007). Temporal difference learning. *Scholarpedia*, *2*(11), 1604.

Bayer, H.M., & Glimcher, P.W. (2005). Midbrain dopamine neurons encode a quantitative reward prediction error signal. *Neuron*, *47*(1), 129-141.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

Cerruti, D.T., & Staddon, J.E.R. (2003). Operant Conditioning. *Annual Review of Psychology*, *54*, 115-144.

Chakravarthy, V.S., Joseph, D., & Bapi, R.S. (2010). What do the basal ganglia do? A modeling perspective. *Biological cybernetics*, *103*(3), 237-253.

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, *36*(03), 181-204.

Clarke, D., Whitney, H., Sutton, G., & Robert, D. (2013). Detection and learning of floral electric fields by bumblebees. *Science*, *340*(6128), 66-69.

Cleveland, W.S., Grosse, E., & Shyu, W.M. (1992). Local regression models. Statistical models in S, 309-376.

Dayan, P., & Abbott, L.F. (2001). *Theoretical neuroscience* (Vol. 806). Cambridge, MA: MIT Press.

Dayan, P., & Berridge, K.C. (2014). Model-based and model-free Pavlovian reward learning: revaluation, revision, and revelation. *Cognitive, Affective, & Behavioral Neuroscience*, *14*(2), 473-492.

Dickinson, A., & Balleine, B. (1994). Motivational control of goal-directed action. *Animal Learning & Behavior*, *22*(1), 1-18.

Friston, K. (2010). The free-energy principle: a unified brain theory?. *Nature Reviews Neuroscience*, *11*(2), 127-138.

Gershman, S.J., Horvitz, E.J., & Tenenbaum, J.B. (2015). Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, *349*(6245), 273-278.

Hafting, T., Fyhn, M., Molden, S., Moser, M.B., & Moser, E.I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801-806.

Hastie, T.J., & Tibshirani, R.J. (1990). Generalized additive models (Vol. 43). CRC Press.

Kaelbling, L.P., Littman, M.L., & Moore, A.W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 237-285.

Kandel, E.R. (2001). The molecular biology of memory storage: a dialogue between genes and synapses. *Science*, *294*(5544), 1030-1038.

Koutník, J., Cuccu, G., Schmidhuber, J., & Gomez, F. (2013, July). Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (pp. 1061-1068). ACM.

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). End-to-End Training of Deep Visuomotor Policies. *arXiv preprint arXiv:1504.00702*.

Lin, L.J. (1991, July). Programming Robots Using Reinforcement Learning and Teaching. In *AAAI* (pp. 781-786).

Lin, L.J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*(3-4), 293-321.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.

Montague, P.R., Dayan, P., & Sejnowski, T.J. (1996). A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of neuroscience*, *16*(5), 1936-1947.

Morris, G., Nevet, A., Arkadir, D., Vaadia, E., & Bergman, H. (2006). Midbrain dopamine neurons encode decisions for future action. *Nature neuroscience*,*9*(8), 1057-1063.

Niv, Y., Daw, N.D., & Dayan, P. (2006). Choice values. *Nature neuroscience*,*9*(8), 987-988.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, *53*(3), 139-154.

O'Doherty, J.P. (2012). Beyond simple reinforcement learning: the computational neurobiology of reward-learning and valuation. *European Journal of Neuroscience*, *35*(7), 987-990.

Pan, W.X., Schmidt, R., Wickens, J.R., & Hyland, B.I. (2005). Dopamine cells respond to predicted events during classical conditioning: evidence for eligibility traces in the reward-learning network. *The Journal of neuroscience*, *25*(26), 6235-6242.

Pavlov, I.P. (1927). Conditioned reflexes. *An Investigation of the physiological activity of the cerebral cortex*.

Petrovskaya, A., & Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, *26*(2-3), 123-139.

Prescott, T.J., Bryson, J.J., & Seth, A.K. (2007). Introduction. Modelling natural action selection. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, *362*(1485), 1521-1529.

Redgrave, P., Prescott, T.J., & Gurney, K. (1999). The basal ganglia: a vertebrate solution to the selection problem?. *Neuroscience*, *89*(4), 1009-1023.

Redgrave, P. (2007). Basal ganglia. *Scholarpedia*, *2*(6), 1825.

Rescorla, R.A., & Wagner, A.R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In *Classical conditioning: Current research and theory*.

Roesch, M.R., Calu, D.J., & Schoenbaum, G. (2007). Dopamine neurons encode the better option in rats deciding between differently delayed or sized rewards. *Nature neuroscience*, *10*(12), 1615-1624.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1988). Learning representations by back-propagating errors. Cognitive modeling, 5, 3.

Schmidhuber, J. (2015). Retrieved 2015-08-30 from: http://people.idsia.ch/~juergen/naturedeepmind.html

Schultz, W., Dayan, P., & Montague, P.R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593-1599.

Schultz, W. (2006). Behavioral theories and the neurophysiology of reward. *Annu. Rev. Psychol.*, *57*, 87-115.

Schultz, W. (2007). Reward signals. *Scholarpedia*, *2*(6), 2184.

Schultz, W. (2015). Neuronal Reward and Decision Signals: From Theories to Data. *Physiological reviews*, *95*(3), 853-951.

Shettleworth, S.J. (2013). *Fundamentals of comparative cognition*. Oxford University Press.

Shteingart, H., & Loewenstein, Y. (2014). Reinforcement learning and human behavior. *Current opinion in neurobiology*, *25*, 93-98.

Staddon, J.E., & Niv, Y. (2008). Operant conditioning. *Scholarpedia*, *3*(9), 2318.

Sutton, R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning* (pp. 216-224).

Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning: An introduction*. Cambridge: MIT press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58-68.

Versace, M., & Chandler, B. (2010). The brain of a new machine. *IEEE spectrum*, *47*(12), 30-37.

Von Neumann, J. (1947). Morgenstern, 0. *Theory of games and economic behavior*, 19-7.

Wandell, B.A., Dumoulin, S.O., & Brewer, A.A. (2007). Visual field maps in human cortex. *Neuron*, *56*(2), 366-383.

Watkins, C.J. C.H. (1989). Learning from delayed rewards. Doctoral Dissertation.

Watkins, C.J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279-292.

Whiteson, S., Taylor, M.E., & Stone, P. (2007). Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, *15*(1), 33-50.

Wilson, M. (2002). Six views of embodied cognition. *Psychonomic bulletin & review*, *9*(4), 625-636.

Wise, R.A. (2004). Dopamine, learning and motivation. *Nature reviews neuroscience*, *5*(6), 483-494.

Wolpert, D.M., & Ghahramani, Z. (2000). Computational principles of movement neuroscience. *nature neuroscience*, *3*, 1212-1217.

Wolpert, D.M. (2011). Retrieved 2015-08-30 from: http://www.ted.com/talks/daniel_wolpert_the_real_reason_for_brains