Opleiding Informatica

Psychology-Inspired Memory Sampling in a Deep Q Network

Samuel J. Meyer

Supervisors:
Dr. Ir. J. Broekens
Dr. W.A. Kosters

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

16/07/2019

Abstract

Deep Q-Learning uses a method called experience replay which re-uses past experiences of an agent in learning to complete some task. This increases the learning speed and information efficiency of the agent. Several psychology-inspired methods for memory sampling from replay memory were implemented. Performance of these was compared to two established methods for an exploration task in a partially observable environment. The psychology-inspired methods performed approximately as well as the standard method of random sampling, but worse than prioritized memories sampling. Due to being computationally much more expensive this makes them not recommendable.

Contents

1 Introduction

In recent years, reinforcement learning (RL) has grown in popularity. A popular example of an application is in Google's AlphaZero, which has used RL to master playing games like chess, shogi and go [1]. For these games, RL-based algorithms are already at human level performance. Although RL has proven to be a very effective technique, it has many aspects ripe for improvement. One of these is experience replay [2], which we explain in this introduction.

In reinforcement learning, an agent learns to perform a task through interacting with its environment. Desired outcomes will result in awarding the agent positively, while undesired outcomes result in rewarding the agent negatively. By relating these rewards with the actions the agent performs, and the states in which they are performed, the agent learns an optimal action to perform given a state. In the most basic form these associations get stored in a state-vector space encompassing all possible action-state pairs. The learned decision-making of the agent based on these action-state pairs is called its *policy*.

In deep reinforcement learning, specifically Deep Q-Learning (DQN), the optimal behavioral policy is approximated by a neural network (NN). This gets rid of the requirement of explicitly having to store all action-state pairs and allows for generalization. However, the training of the network requires a lot of data, and RL is not particularly data efficient. Without the use of any other methods, every action-state pair is only used once to learn from. To make DQN more data efficient and speed up its learning, methods like experience replay have been introduced.

Experience replay is a method of making use of data more than once, by storing experiences and using a subset of these to update the NN at some interval. The selection of the experiences to be sampled is called memory sampling. AlphaZero (discussed above) is already very successful using random memory sampling. However, efforts have already shown that by using methods prioritizing certain experiences over others in selection for experience replay, can lead to the finding of an optimal solution in even fewer steps. This is explored in a thesis by De Jong [3], which this thesis builds on.

Similarly to how RL has roots in theories about trial and error animal learning, such as operant conditioning, further efforts for improving RL also take inspiration from the field of psychology. An example of this is the use of episodic memory tables to increase the amount of information gained from samples, among other improvements [4]. This thesis aims to explore this same spirit of psychology-inspired methods, specifically for sample selection in experience replay.

The following section will discuss some necessary background information about the techniques applied in this bachelor thesis. In Section 3 the research question and the underlying theories upon which the proposed and tested methods are based will be discussed. Section 4 explains the methodology of the tests that have been conducted. This is followed by the explanation of the method for testing in Section 4, the results and discussion in Section 5, and the conclusions and future research in Section 6.

2 Background Information

This following subsections introduce important topics and techniques for the understanding of the rest of this thesis.

2.1 Machine Learning

Machine learning is the study and implementation of systems that learn to perform tasks through experience, without being programmed with pre-specified methods for selecting between possible outputs. This is useful for classification problems where doing so is unthinkable. An example for this is computer vision, where a system might have to learn how to recognize ducks in images. Ducks have many features that specifically makes them ducks, as opposed to other animals and objects. The system also has to account for all possible locations and rotations of the ducks that may be in the image. The parameters and relations between them that have to be taken into account when explicitly writing code to achieve perfect duck-hunting, are too many to be feasible. Besides computer vision, machine learning is useful for a range of other applications such as email filtering and speech recognition. Other interesting uses are also in the works for topics such as genetics [5] and cancer prognosis [6]

There are four main branches of machine learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning (which will be discussed in the next subsection) [7]. Supervised learning is used when, for a system that needs to produce an output based on a specific input, a set of inputs and correctly corresponding outputs are available. This is essentially learning by example. The system learns to relate inputs to outputs, and then needs to provide correct outputs for inputs which were not in the set used for training. Going back to the duck example, the system would be fed a large set of photos of which a subset contains ducks. For this training set it would also be told which ones contain ducks and which ones do not. The system has to use this data to learn to recognize ducks in future photos of ducks that are unlabelled.

In unsupervised learning, there is no initial set of output data. The system has to discover relationships between the points of input data, in the hope that new relationships or patterns are discovered. Given a large set of photos of ducks, it may learn how to categorize different kinds of ducks, or group them based on the type of environment the ducks are in.

Semi-supervised learning [8] is somewhere between supervised and unsupervised learning. Typically, most of the data set is unlabelled, but the small subset of data that is labelled helps increase learning accuracy. This would be used in cases where labelling data is expensive or hard to do.

2.2 Reinforcement Learning

Reinforcement learning distinguishes itself from supervised and unsupervised learning in that a system is not provided with input or output sets, but learns based on evaluations of its interactions with the environment.

The way a reinforcement learning algorithm interacts with its environment is through a Markov Decision Process (MDP). This is a process based on the Markov Property, stating that the information prior to the current state is not more informative when inferring about the future than the current state by itself. In an MDP there is a set of possible states $\mathcal{S}$ an agent can be in, and a set of actions $\mathcal{A}$ the agent can perform. For each time step t, the agent has to make a decision a_t from $\mathcal{A}$ within its current state s_t from $\mathcal{S}$ in order to get to the next state s_{t+1}. The selection of the action is done based on the agent's current policy $\pi(\mathcal{S}_t)$. This policy maps states to the probabilities of taking the available actions from within those states. When the agent has moved to a new state, it gets a reward r_t based on whether and how beneficial the new state is. For example, upon reaching a specific goal, the agent could get a reward of $+1$ or get a reward of -1 when landing in a fail-state. To adhere to the Markov Property, the next state in an MDP should depend only on the current state, not any past states.

To learn what actions to take in which states, the values of being in specific states is calculated. This is defined as the cumulative reward of all following states. The total discounted return J at time t is:

$$J = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1}$$

Here $\gamma \in (0, 1]$ is a discount factor that determines the weight of future rewards. J then gives the discounted sum of all future rewards, i.e. the total discounted return. However, the sum of future rewards is not a certainty, and depends on the current policy. Therefore, to calculate the value of being in a state, the value function $V_\pi(s_t)$ calculates the *expected* discounted return starting from that state.

$$V_\pi(s_t) = \mathbb{E}[J|S_t = s_t] \tag{2}$$

For the agent to then make the decision of which action to take, each possible action has a probability of being taken. This is based on the values of taking that action at a specific time, while in a specific state, given the current policy. This value to determine the quality of such an action-state pair is called the Q-value:

$$q_\pi(a, s) = \mathbb{E}_\pi \left[J|A_t = a, S_t = s\right] \tag{3}$$

To get the best possible total return for each state, an optimal policy is needed. This uses the optimal Q-policy q_*, which returns the highest possible expected return from all actions that can be taken within a state. The optimal policy needs to satisfy the Bellman optimality equation. This equation states that the expected Q-value for a specific state action pair (a, s) needs to be the reward for taking action a in state s, added to the discounted optimal Q-policy for all future steps using the current policy:

$$q_*\left(a, s\right) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(a', s'\right)\right] \tag{4}$$

Here a' is the next action and s' the next state. With the optimal Q-function, the RL algorithm can determine the best policy by maximizing the optimal policy for every possible state. There are several methods that can do this, one of which is Q-learning.

The process of Q-learning is as follows: Initially, the current policy has not mapped any action-state values, so these are randomly initialized. Through ϵ-greedy strategy, actions are first taken exclusively by random selection, and over time lean more to using the policy. This is the shifting of exploration of what can be done in the environment, to exploiting the gained information to get the best results. Throughout all of this, the policy is updated through the following formula:

$$q(a, s) = (1 - \alpha) * q(a, s) + \alpha(R_{t+1} + \gamma \max_{a'} q_* (a', s')) \tag{5}$$

Here α is the learning rate, which determines to what extent the current Q-value for the action-state pair is overwritten by the newly determined value. This is repeated after every action the agent takes, always using the current estimation of the values of states as mapped in the policy. Throughout the learning process, the Q-values given by the policy will approach those given by the Bellman equation. Once they are equal, the optimal policy has been reached.

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) [9], or simply neural networks, are sets of nodes organized into layers. Nodes of each layer propagate signals to nodes in following layers, based on the connection strengths between them. This is inspired by the functioning of the brain, where these nodes model the function of neurons (as they can also be called when describing ANNs). A neural network always has an input layer, an output layer, and a set of hidden layers between them. The input layer has a node for every component of input data that is received, the output layer has a node for each possible desired output, and the amount of nodes in hidden layers is determined by any of various selection methods. Given a set of input values, each node passes on these values multiplied by the specific weight of the connection between those two nodes. The sum of all the values a node receives, is transformed through some activation function before passing on the result of this function. The ANNs used in this paper use the rectified linear unit (ReLU) activation function:

$$f(x) = \begin{cases} 0, & \text{if } x < 0. \\ x, & \text{if } x \geq 0. \end{cases} \tag{6}$$

Here x is the summation of the weighted inputs for a node. The network is trained by changing the weights between its nodes. This is done by minimizing a loss function. A loss function gives a measure of difference between a network's output and the target value it is trying to predict. The loss function is minimized by calculating the gradient of this loss function with respect to each weight in the network through backpropagation, and then updating them through stochastic gradient descent (SDG). By continuously adjusting the weights after receiving sets of inputs, the network learns a generalized method for getting to a certain output, based on some trend in the input.

2.4 Deep Q-Learning

As the action-state space grows, a Q-learning algorithm has to iterate through increasingly more Q-values for action-state pairs to calculate expected cumulative return. For a problem with a small action-state space Q-learning works well, but as the problems get more complex, this iteration becomes less feasible. This is especially the case if the agent would have to learn in a real-world environment, where the amount of possible states and actions can be immense. To solve this, Q-learning has been combined with ANNs in a technique called Deep Q-Learning.

As opposed to using an action-state table to store and calculate Q-values, Deep Q-learning uses an ANN to estimate the Q-values of action-state pairs. This ANN that serves as function approximator has the agent inputs as input nodes, and the possible actions as output nodes. For every situation the agent evaluates, it gives an estimate of the Q-value for every possible action that can be taken. Unless otherwise determined randomly through the ϵ-greedy strategy, the action with the highest estimated Q-value is selected.

In order to update the ANN, there has to be some sort of loss-function to calculate how its weights need to be adjusted. To do this, the Bellman equation (Equation 4) is used. Because the Bellman equation gives the Q-value that each action-state pair should have, the loss can be calculated by looking at the difference between the Bellman equation and the output of the ANN. The Q-value of the next state is needed to estimate the value of the Bellman equation. Using the current (online) policy to calculate this value is unstable, because it is constantly updated. To remedy this, a periodic copy of the online ANN is made to what is called the target network. By using the target network to calculate the Q-value of the next state, the learning process becomes much more stable. The loss is calculated as follows:

$$\text{loss} = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_* \left(a', s'\right)\right] - \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]. \tag{7}$$

The target the target network is used to get the value $\max_{a'} q_* \left(a', s'\right)$, and the online policy network estimates $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$.

2.5 Experience Replay

In experience replay, in each step an informative set of data is stored as an experience in a memory bank called the replay memory. Such an experience contains the following:

$$e_t = \left(s_t, a_t, r_{t+1}, s_{t+1}\right) \tag{8}$$

This shows that the current state s_t and action a_t have led to the following state s_{t+1} and reward r_{t+1}. Every time an experience is added to the replay memory, a batch of experiences is sampled and used to train the policy network. When only using the consecutively collected samples to directly train the ANN, the correlation between these consecutive samples leads to overfitting; the changes made to the network from prior experiences become undone [10]. Also, rare but important experiences only get used once, which is a waste of useful information. By sampling experiences from all over the memory and thus breaking the mentioned correlation, less information is lost due to overfitting, and rare experiences get sampled more often. There is an overall increase in data efficiency.

3 Research Question

The standard method of selecting which experiences to add to a sampling batch in experience replay is random selection. The efficacy of this has invoked the investigation towards a series of more targeted methods for selecting these experiences, to see if experience replay could be improved even further. This paper proposes several psychology-inspired methods through various implementations, and aims to compare these to some methods that have already been shown to be effective. The following question is addressed:

> How do several psychology inspired memory sampling methods affect the performance of agents using deep reinforcement learning?

3.1 Psychology Inspired Methods

There are two main psychological ideas that have inspired the methods that were tested. Both of these ideas are based on comparing the current observation with stored experiences. The first one of these is the theory of ecphory, which describes the act of recovering memories based on a trigger that is similar to the memories to be recovered [11]. When trying to learn something new, one of the most important factors in the learning process is repetition [12]. Part of such learning is implicit; we do not necessarily have to think of exactly what went wrong ten tries ago to do better next time. However, because memories of past tries get cued on later tries, our learning is sped up. The re-using of past memories serves as extra repetition, thus re-learning information from variants of the same situation. Translating this to experience replay, the ANN is trained slowly through every new consecutive experience that is added. Combining this with the addition of sampling past experiences from the experience replay that are similar to the current experience may speed up the learning of performing the desired tasks.

Another psychological theory is the theory that novel experiences are remembered better than familiar ones. This is based on the idea that novel stimuli draw more attention than familiar stimuli [13]. Consequently, this would lead to the extent to which the new stimulus is remembered to be a greater learning experience than one repetition of a familiar stimulus. Novelty also relates to an important aspect of RL: exploration. Especially in the beginning of the learning process for an RL agent, the agent uses an ϵ-greedy strategy to slowly shift from exploration of the environment to exploitation of learned information. By increasing the importance of novel experiences in memory sampling, the finding of new situations during exploration can be reinforced in memory, much like how this is done in human learning.

In this thesis, several methods to implement these two ideas are proposed. The selection of experiences based on similarity to the current experience, is done through the measure of Euclidean distance of the states in these experiences. First a method was implemented to do this with direct comparison to every sample in the replay memory, then two different methods of grouping experiences were used to determine from which group memories there was to be sampled[14]. For the k-means clustering method of these two, the creation of a new cluster has been interpreted as the finding of a novel experience, and as a result using this experience heavily for training the ANN.

3.1.1 Simple Euclidean Distance

When deciding about methods for comparing samples in a way that corresponds to the theories discussed above, the simplest structure would be to compare the current observation to all observations currently in the experience replay memory. Because the states in the experiences were all vectors of observations, the chosen method of comparison was Euclidean distance between the current state of the agent and every state in the replay memory experiences. The Euclidean distance between two vectors was calculated as follows:

$$\sqrt{\sum_{i=0}^{n-1}(c_i - q_i)^2} \tag{9}$$

with c_i and q_i being corresponding indexes for vectors of length n in the current vector c and any vector q in the replay memory. A new vector was filled with each Euclidean distance, with the index corresponding to the index of the experience to which the current one was compared. The distances in this vector were then subtracted from the maximum state values in order to effectively invert the distances, and then normalized to sum up to 1. This allowed for this vector to be used as the vector of sampling probabilities for the corresponding experiences. Because the distances were inverted, the experiences closest to the current experience has the highest probability of being selected for the sampling batch.

3.1.2 K-means Cluster Sampling

A common theory about the organization of episodic memory is schema theory [15]. Schema theory states that similar memories are grouped together in memory, in structures called schemata. Being in a specific situation, the schema related to the current event gets activated. It has been shown that events congruent to a currently activated schema are remembered better later on [16]. This relates directly to the idea of ecphory, but introduces an organizational memory structure.

To implement this in memory sampling, experiences were organized into clusters through two different methods: k-means clustering and fuzzy c-means clustering (discussed in the next subsection). This would organize the experiences into clusters around *centroids* based on similarity, which could be seen as analogous to schemata. Typically, the number of clusters is predetermined. However, the best choice for this number depends on the amount of experiences, and in memory sampling the amount of possible experiences to select grows until the memory store is full. To take this into account, the decision was made to start k with a value of one, and then increase the number of clusters as the amount of experiences would grow. A similar method has been used in RL to divide the state-space into sub-tasks [17], but not for memory sampling.

Whenever a batch was selected for experience replay, the centroid closest to the most recent experience was found. To determine whether the number of clusters needed to be increased, a threshold value was used. This value was experimentally determined by observing what the average distance between all existing experience states in the memory store were after a run. Once the distance between a new experience state and any existing centroid was greater than the threshold, a new centroid was created with the same coordinates as the experience state.

Having found the closest centroid, all samples which were closest to this centroid were grouped together. From this cluster, a batch of experiences was selected randomly. The centroid of the sampled cluster is then re-adjusted to be located at the mean location of all its members.

3.1.3 K-means Cluster Sampling + Novelty

The k-means cluster sampling method with the addition of novelty works identical to the method without novelty, except for one change. Just as in the non-novelty version, whenever an experience state was considered different enough from all other states in the replay memory beyond a threshold value, a new centroid was created. In the novelty version of k-means sampling such an experience was considered to be novel. Because a novel experience should be more impactful than any other experience, the next batch of experiences was filled with copies of this one experience. Any other batches were put together as described for the non-novelty k-means sampling method.

3.1.4 Fuzzy C-Means Cluster Sampling

In fuzzy c-means clustering, instead of dividing experiences into concrete clusters, each experience is a member of every cluster to a certain degree. The same method for growing the number of centroids as used in k-means sampling was used here. After finding the centroid closest to the most recent experience and possibly adding a new cluster if needed according to the threshold, a membership matrix was filled. This matrix showed the degree of membership to each centroid, for every experience. This degree of membership was based on their Euclidean distances to the centroids, and were normalized so the sum of all membership values for each centroid added up to 1.

The batch was randomly selected using the membership values for the cluster of the centroid closest to the most recent experience as sampling probabilities. After this, the centroids were re-calculated using the following equation in order to take the degree of membership of each experience into account when shifting the centroid:

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}} \tag{10}$$

Here u_{ij} is the value in the membership matrix for cluster i and experience j, x_j is experience state j, and $m \in [1, \infty)$ is a weighting exponent.

3.2 Established Methods

To get a sense of how the proposed methods performed, they were compared to two established methods. The first of these is the standard sampling method of random sampling, the other is prioritized memories sampling.

3.2.1 Random Sampling

As was explained in the introduction, random sampling is effective because it decorrellates subsequent samples from each other. The downside to random sampling is that, since all experiences have equal probabilities of sampling, there may be experiences that are more useful than others that do not or rarely get selected for the experience replay batch.

3.2.2 Prioritized Memory Sampling

De Jong proposed a method based on the standard method for prioritized sampling, called prioritized memories sampling [3]. In prioritized sampling, the probability of sampling experiences depends on some measure of their size. The greater their measure in size, the greater the probability of being sampled. In Q-learning one of the measures of size of an experience that can be used is the difference between an estimated Q-value and the estimated optimal Q-value as per the Bellman equation. This difference, used as the loss for learning in Deep Q-learning, can also be called the temporal difference error (TD error). Using TD error for sampling priority can lead to a lack of diversity, which is addressed by using a stochastic sampling technique based on proportional prioritization to determine the priority of picking samples. Since the selection of the samples is not identically distributed this introduces some bias, which is addressed with a method called importance sampling. Importance sampling scales the TD errors so that their priority decreases the more often they are sampled. De Jong introduced three methods to improve in prioritized sampling, and then added a new technique to improve it even further, thus creating prioritized memories sampling.

The first of the improvements on prioritized sampling is the recalculating of the TD error after optimizing the policy once a batch has been sampled. This TD error is then used to calculate the probability distribution for the next iteration, in order to prevent re-picking of experiences of which not much more can be learned. The second improvement is done by not normalizing sampling weights as is usually done in prioritized sampling. This is done to prevent some experiences from not being sampled anymore towards the end of learning, where their TD error can approach zero. This conserves the distribution of sampling. The third improvement is the addition of the most recent experiences are added to every batch to make sure every experience is used at least once.

Prioritized memories sampling works the same as prioritized sampling, with the addition of a second actor, and by storing the entire state of the environment alongside every experience. After some interval of experiences received by the replay memory, an experience is selected from the replay memory through prioritized sampling. The second agent is then loaded into the environment a few steps prior to the selected experience, then takes steps in this separate environment. The experiences the agent collects are all sent to the replay memory. This fills the replay memory with experiences that surround particularly large/interesting experiences based on TD error.

4 Method

To investigate whether the proposed memory sampling techniques would lead to a better performance than other known memory sampling techniques, a set of simulations was run. While there is a gap between the level of detail and variety of situations in the real world and a more simple simulation, running simulations for the sake of comparing the memory sampling methods suffices for the scope of this thesis. The simulated test environment and a large part of the DQN code was originally written for the master thesis *The effect of sampling methods on Deep Q-Networks in robot navigation tasks* [3]. Considering this code was written for the goal of comparing established sampling techniques with sampling techniques proposed by De Jong, the only significant changes made to the code were the additions of the implementations of the sampling techniques proposed by this paper. This was done to allow for a direct as possible comparison to the established methods, de Jong's methods, and the methods proposed here.

4.1 Simulated Environment

The simulated test environment was inspired by the Atari 2600 game Freeway by David Crane. The main goal of this game is for the player(s) to get their character, a chicken, to the other side of a freeway while avoiding getting hit by cars. In the simulation used for this thesis, the agent is trying to cross a 40 meter long two-lane road from one sidewalk to the other, while one car repeatedly and continuously moves along the road from the left to right side of the screen, while on the right lane. This car moves at a speed of 1.4m/s. Each lane is three meters wide, and each sidewalk is another 2 meters wide.

The agent starts on one sidewalk, and gets a reward of $+1$ when reaching the other sidewalk. Once the other sidewalk has been reached, the goal-sidewalk is switched, and a new reward can be received by crossing the road again. This continues for the duration of the simulation. If the agent gets hit by he car, it is reset to its starting position and receives a reward of -1. The agent has been modeled after the actual robot Pepper (SoftBank Robotics), a humanoid robot designed to socially interact with humans [18]. The top speed of 3km/hr, and a turning speed of 90^o of Pepper have been translated to the general moving and turning speed of the agent in the simulation. Pepper perceives its surroundings through a set of 20 lasers horizontally split over a range of -120^o to 120^o on its front side. The simulated versions of these lasers have a range of 15 meters. The agent can take one of three actions at every time step. It can move forwards, turn left, or turn right. To incentivize the agent to move forward and explore during the learning process, moving forward gives a reward of 0.01. This is called an intrinsic reward.

Because Pepper has an average of four readings of its environment per second, every time step represents 0.25 seconds. Every instance of speed has also been divided by four to accurately represent the appropriate speed per step.

4.2 Architecture

The architecture is made up of several different components that run in parallel. This was designed this way by the original creator De Jong to reduce training time. These three components are: the actor, the learner, and the replay memory.

The actor acts as the agent as per the description of the simulated environment. The decisions of what actions to take within this environment are made through the ϵ-greedy strategy. Initially, all actions are chosen randomly. Over time, as the value of ϵ decreases, more and more decisions are made based on the agent's current policy. When deciding based on the policy, the action with the highest Q-value as given by the policy network is performed. This results in a received reward and the next state. In the case of a collision with the car, the next state is marked as a crash. After every 4 actions taken, the 20-input states of these actions are stacked to represent the 80-input state for the current experience. This is because the amount of information that can be deduced from a single state is limited. Stacking these states allows for taking factors into account such as direction and speed of movement. This is especially useful in a partially observable environment as is the case in this test-scenario. The tuple representing the current experience is then placed in a queue from which the replay memory can access it. In the original implementation, the actor would place experiences in this queue as fast as it could. This has been changed so that the agent does not continue as long as there is more than one experience in the queue. This change was made because when the sampling methods in the replay memory are more expensive, this can result in the agent placing experiences in the queue faster than the replay memory can process them. This was already the case for some of the sampling methods originally implemented for this architecture, but not to a degree leading to dramatic performance decrease. For the methods implemented for this thesis, the performance decrease was much higher. Since comparisons of the agent learning performance over a set of steps is more fair if sampling frequency is the same for all methods, the discussed change has been made.

The replay memory contains the 50000 most recent experiences. Every time the replay memory receives an experience from the agent, a batch of 32 experiences is created through one of the sampling methods. This batch always includes the most recent 4 experiences. The prioritized memories method associates a TD error with every experience. The psychology-inspired methods compare the current experience with the experiences within the replay memory. The clustering methods can create a maximum of 5 clusters to reduce computational cost.

The learner receives the batches of experiences from the replay memory, which it uses to calculate the changes to the policy network. Both the policy network and the target policy network are part of the learner. The TD errors for all the experiences in the batch are used for a loss function called Huber loss:

$$f(\theta) = \begin{cases} \frac{1}{2}\theta^2, & \text{if } |\theta| \leq 1 \\ |\theta| - \frac{1}{2}, & \text{if } |\theta| > 1 \end{cases} \tag{11}$$

This decreases the effect of outliers on learning because it uses an absolute loss for large errors instead of a squared loss [19]. The policy network is then updated through backpropagation. The policy network has 79 hidden nodes, and uses a rectified linear unit as activation function. Every 100 batches the learner receives the policy network is copied to the target network, and every 4 batches the policy network is copied to the agent.

4.3 Performance Measures and Testing

Four different performance measures were used. Three of these are the same methods used for De Jong's thesis [3] to allow for direct comparison. A fourth one was added to give insight into the learning rate of the sampling methods separately from each other. All methods are based on the performance measure of moving average. Because the task for which the agent is being trained is continuous instead of episodic, a measure has to be used that can give an impression of performance throughout this process instead of getting results per episode. This is done by taking the average of rewards over a fraction of the theoretical maximum. This maximum was calculated based on the shortest possible path between the two sidewalks in the simulation environment.

The first measure is the final moving average μ_r^{final}. To take fluctuations into account after reaching optimal performance, the average of the last 100,000 experiences (out of a total of 800,000 experiences) was taken. This gives an idea of what the general final performance will end up being, using a specific sampling strategy.

The second measure is the maximum moving average μ_r^{max}. If at any point during the process a significantly better performance is reached than what eventually ends up being the final performance, this could give information about the stability of learning caused by a used method.

The third method is the fraction of the test at which a method reaches its own final moving average, over the course of the entire run μ_r^{rate}. If different methods reach their final moving average after approximately the same fraction of the test but have significant differences in final moving average, this leads to clear distinction between those methods.

The final method is the learning rate of methods $\mu_r^{0.8}$ in relation to the best final moving average. This represents at which fraction of the run a method for the first time reached 0.8 * the best final learning rate of all methods. This allows for more direct comparison of learning rates.

Using the training agent for recording the performance measures could lead to several problems. As the *epsilon* value decreases, it can be unclear whether improvements in performance are due to a decrease in random-action based exploration or due to increased use of the policy. Also, the reward for moving forward motivates the agent to learn, but it is not a true measure of performance. Lastly, if initial Q-value estimates are bad, the agent could get stuck in a sub-optimal region and not explore further [20].

To solve this issues, a test agent is run in parallel with the training agent, receiving the same policy updates. Every corresponding issue is addressed through the differences that the test agent's ϵ value is constant at a low value of 0.001, it does not receive an intrinsic reward for moving, and gets reset every 1000 steps.

Table 1 gives an overview of the hyper-parameters for testing:

Hyper-Parameter	Value	Explanation
Steps	800000	Amount of steps
ϵ start	1	Start value ϵ
ϵ end	0.01	value ϵ
Exploration steps	200000	Step where ϵ has become ϵ end
Policy-network update	4	Frequency policy-network update
γ	0.99	Discount factor
Learning rate	0.001	Based on previous work
Target network update	100	Based on previous work
Replay memory size	50000	Based on previous work
Batch size	32	Based on previous work
Sample rate	4	Based on previous work
Prioritized α	0.6	Based on previous work
Prioritized β_0	0.4	Based on previous work
Prioritized β_{end}	1	Based on previous work
Max clusters	5	Balance computational cost with accuracy
Cluster threshold	3.8	Average distance between samples in a run
m	2.1	Matrix weight exponent
Prioritized agent interval	100	Interval for prioritized memories agent
Prioritized agent step reversion	25	Steps loaded back into history
Prioritized agent step number	30	Steps taken after being loaded into history

Table 1: Hyper-parameters

Testing was done over a period of 48 hours on the GPU servers of the Leiden Institute of Advanced Computer Science (LIACS). By running tests for every different method in parallel, different numbers of runs were accomplished depending on the methods. For random sampling, prioritized memories sampling, and simple Euclidean sampling, 5 tests were completed. For each k-means sampling method 2 runs were completed. For the fuzzy c-means sampling method, only a single run was completed over this two day span.

5 Results

The averaged results of all performed trials are shown in Figure 1. The x-axis represents the fraction of the total amount of steps taken. The y-axis represents the moving average as a fraction of the theoretical maximum as described in Section 4. Shadows show the difference between the maximum and minimum values reached for every method over the course of the trials.

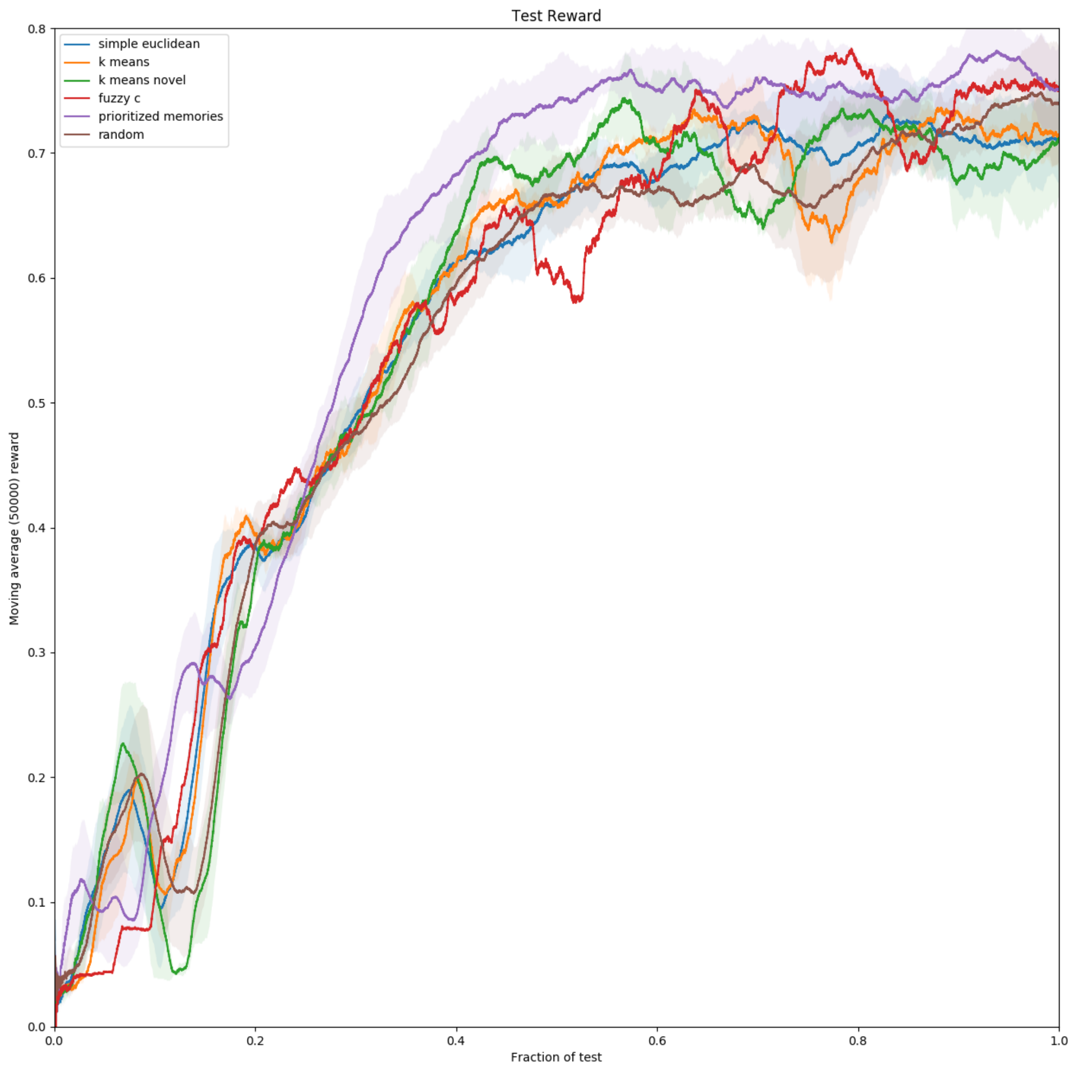

Figure 1: Moving average reward over course of averaged trial runs. Standard deviations for the trials of each method were used as error bands.

The averaged trials for the simple Euclidean, prioritized memories, and random methods clearly show smoother lines due to the greater amount of trials that was possible in the testing time span (5) compared to the k-means methods (2) and the fuzzy c-means method (1).

All methods show a sharp initial increase, followed by a short decrease or stagnation in performance. This occurs a second time to a much smaller extent. After this, there is a slow increase to the final performances. Around the first dip, both prioritized memories sampling and fuzzy c-means sampling don not increase performance much in the first place, and dip in performance only slightly. Prioritized memories sampling also has an earlier second dip, while fuzzy c-means sampling does not seem to have much of a second dip. After the second performance dip the prioritized memories method gets to a steady high performance, and the fuzzy c-means method gets to a very high peak, but quite late in the run.

Because of the small number of trials, polynomial trend lines of the data were plotted. This allows a degree of insight into what the averages of performance might have looked like over more trials.

	μ_r^{final}	μ_r^{max}	μ_r^{rate}	$\mu_r^{0.8}$
simple Euclidean	0.7068	0.7304	0.7812	0.4545
k means	0.6671	0.7363	**0.5615**	0.4709
k means novel	0.7037	0.7441	0.6495	0.4656
fuzzy c	0.7410	**0.7837**	0.8102	0.5304
prioritized	**0.7473**	0.7821	0.6165	**0.3768**
random	0.7248	0.7487	0.9528	0.4145

Table 2: Performance measures. Bold font shows best values.

Table 2 gives an overview of some important performance measures. For final performance (μ_r^{final}) and maximum performance (μ_r^{max}), the highest values are the best values. For the rate at which a method reaches its own final performance (μ_r^{rate}) and the rate at which a method reaches 0.8 times the best final performance ($\mu_r^{0.8}$), the lowest values are the best values.

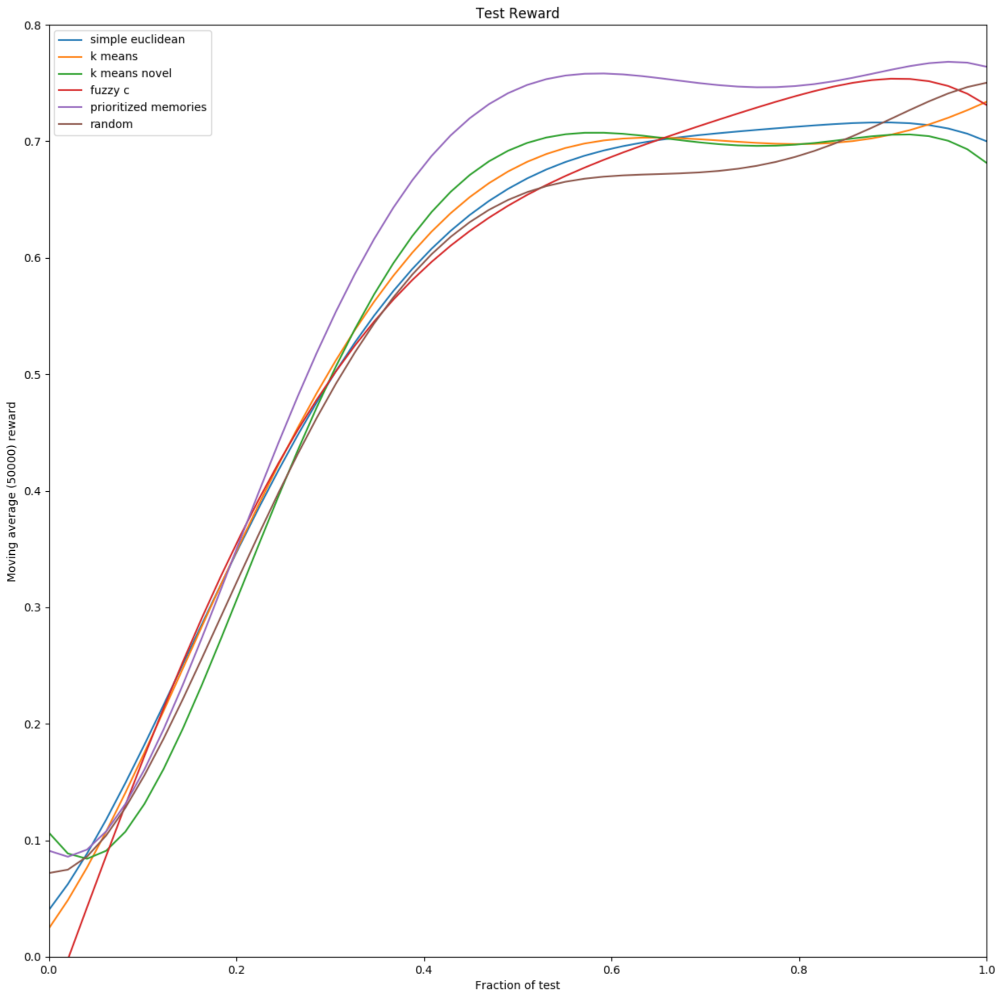

Figure 2: Polynomial trends of moving average reward over course of run

The trend lines as shown in Figure 2 for all methods show near identical performance throughout the first part of learning, and eventually diverge during final improvement and performance. The priority memories method shows the best performance, but after some time the graph shows the fuzzy c-means method and random sampling method coming close. The k-means methods almost overlap with each other through most of the learning.

Range of steps	Random Sampling Frequency	Simple Euclidean Sampling Frequency
0 - 100k	974510	972581
100k - 200k	779052	800059
200k - 300k	800732	799699
300k - 400k	799688	799907
400k - 500k	800055	799613
500k - 600k	800132	800285
600k - 700k	799239	799719
700k - 800k	625249	626797

Table 3: Summation of number of occurrences of states in memory samples per 100,000 steps

Table 3 shows how often specific states throughout the trial have been parts of experiences that have been sampled. This is done as a summation of the sampling frequencies per every 100,000 steps. Both sampling methods show near identical values and follow the same trend. For the first 100,000 steps the states are sampled significantly more often than those during the rest of the run. For steps 200,000 to 700,000 the summed sampling frequencies are all very close too 800,000. For the last 100,000 steps, the frequencies lower to around 625000.

5.1 Discussion

There are two main tasks in this simulation that need to be learned: Getting to the other side and turning around, and evading the car. The better the agent gets at the first task, the more quickly positive rewards are gained. The better the agent gets at the second task, the less quickly negative rewards are gained. When the training agent initially explores, the test-agent's performance slowly increases from the information gained through this random decision making. For initial gain of reward, the policy is first focused on getting to the other side. As the training agent shifts from exploration to using its policy, and thus also increases focus on getting to the other side, it will get hit more often by the car. Due to the new experiences coming in based on the car crashes, the focus of the policy shifts more towards car evasion. This change in the policy initially decreases the agent's ability to get to the other side, thus showing the performance decrease between the 0.1 and 0.2 fractions of the test for many of these methods. Once car evasion has been mastered, a re-shifting of the focus causes a similar but much smaller performance decrease around the fraction of 0.2 of the test. After this, the balanced optimization of both tasks steadily increases for all methods until they reach their optimum. This replicates the same pattern as was observed by De Jong in his comparison of memory sampling methods using the same architecture [3].

The prioritized memories sampling method shows the best performance. This is due to the importance sampling technique making sure the experiences that are sampled are identically distributed. This is what the SDG method for updating the ANN method assumes [21], thus allowing for the updates to be more effective. Also, the training of the ANN with experiences surrounding those with large TD errors increases learning speed.

Figure 1 shows a lot of similarity between the random sampling method and simple Euclidean sampling. This can be explained by their sampling distributions. Random sampling is effective because by its nature it produces identically distributed experiences. The simple Euclidean sampling method places higher priority on samples closer to the most recent samples. This means that events that occur more often during the simulation, also have more surrounding experiences in the replay memory. While these may get sampled more often, thus increasing the probabilities for all these experiences, the fact that there are more of these experiences in turn decreases the probability of any single one of those memories being sampled. This balances the overall frequency that any single experience may be sampled, leading to identical distribution. Table 3 shows how the sampling distributions are both near identically distributed, and identical to each other. Because the amount of states that can be used for experiences to be sampled is initially small, the sampling frequency is higher. Once the experience replay is filled, all of the next states are sampled quite equally. The most recent experiences have been sampled less because they have not been around long enough to be sampled as much as older experiences have.

Both k-means methods show a relatively low final performance. Also, a distinct similarity between their learning becomes apparent when looking at their polynomial trends in Figure 2. Their similarity can be explained by comparing the novelty implementation of the k-means method with the cluster creation implementations of both methods. The k-means novel method fills an entire batch with any experience that is considered novel. An experience is considered novel when it is different from any other cluster centroid beyond a certain threshold, and is then chosen as the cluster centroid of the new cluster. As soon as this new cluster is formed, whenever it is sampled it would initially only contain a single experience, thus filling a batch with that experience. Thus, the first sampling of a new cluster has the exact same result as the explicit novelty implementation of the k-means novel method. This does mean that both methods have an implicit novelty mechanism. Any experiences added to a new and thus novel centroid initially have a higher probability of being sampled.

The relatively low performance of both k-means methods could be explained by the strictness of their categorization. When the current experience is only a little bit closer to one cluster than it is to another, and every sample in its cluster has an equal probability of being sampled, fringe experiences that are far from the centroid but also far from all other clusters are sampled more often. Considering these are not samples that occur often, nor do they resemble any other scenario particularly well, they most likely aren't very valuable. Thus, less is learned. This is especially the case with a low number of clusters as was done in this research (5), considering that with more clusters each would be smaller, and all samples in a cluster are much closer to the current experience. However, the k-means method is the value that is quickest to reach its own final performance. This is quite likely due to its final performance being so low, but could serve useful in a scenario where the speed of learning is more important than the final performance.

Fuzzy c-means sampling appears to initially be the slowest method to learn, but increases in performance towards the end to eventually have the highest peak. The fuzzy c-means method avoids the fringing problem due to the fact that experiences closest to specific centroids have the highest probability of being sampled. It could be that over time these more central experiences represent core experiences, and may thus have more to be learned from. However, due to its data coming from a single trial, there should not be much weight on inferences about its performance.

An additional consideration for any of these methods is the time needed to train them. Considering that the fuzzy c-means method showed similar performance to the other methods but take far longer to train, it does not seem like the best choice. However, if further testing would show a marked increase in performance over other methods, the extra time could be worth it.

5.2 Related Work

Research article [4] has a similar spirit to this thesis, in that it applies new methods to reinforcement learning based on human memory organizational patterns. The article introduces Episodic Memory Deep Q-Networks, which uses an episodic memory store to store and use the most highly rewarding experiences for sampling. This lowers the amount of required samples for an optimal policy when tested with a collection of Atari games.

Paper [22] puts a focus on using novelty to increase exploration and performance in RL. However, instead of trying to detect novelty in collected experiences, novelty is used as a driving factor for exploration. A method called novelty search is hybridized with evolution strategies of RL. The results show this helps in avoidance of local optima and increased performance in a collection of Atari games and simulated robot tasks.

6 Conclusions and Further Research

This thesis explores various methods of memory sampling in replay memory, during the training of an agent with Deep Q-learning. Four of the six methods proposed in this paper are based on psychological theories. Random sampling and prioritized memories sampling were tested to give an established relative measure of performance. The performance of these memory sampling methods was tested in a partially observable environment, in which an agent had to repeatedly cross a road while avoiding a consistent flow of traffic.

None of the psychology-inspired methods tested in this thesis perform significantly worse than random sampling, but there is not enough evidence to suggest that any of them perform significantly better either. Considering the increase in training time due to the extra amount of calculations required per time step, none of them can currently be recommended over simple random sampling.

Prioritization based on temporal difference show to be the most effective memory sampling method as long as it is assisted with methods like importance sampling to make up its shortcomings.

Several concessions were made to reduce training time to feasible lengths for the scope of this thesis. These include limiting the amount of clusters for the cluster sampling methods to five, pre-setting the threshold value based on average distance between samples in the replay memory instead of updating this with every sample, and the amount of trials run per sampling methods. Future research that can employ more time and/or computing resources could investigate the effects of changes in these regards.

The effects of a variety of other parameters is also still unknown. Investigation into other measures of novelty and implementations of its effects could be interesting. For measure of similarities between experiences there is also the possibility of taking more parts of an experience into account, such as the action taken. There may also be other, more suited methods for the measure of similarities between states other than Euclidean distance.

While this thesis hasn't resulted in a breakthrough discovery of memory sampling based on psychological theories, it gives insight in the effect of psychologically inspired methods for experience sampling and replicates the effectiveness of prioritized TD-based sampling with importance sampling.

References

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[2] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon University Pittsburgh PA School of Computer Science, 1993.

[3] T.J.L de Jong. The effect of sampling methods on deep Q-networks in robot navigation tasks, 2019. Master Thesis, Delft University of Technology.

[4] Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. *CoRR*, abs/1805.07603, 2018.

[5] Maxwell W. Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16:321 EP–, May 2015. Review Article.

[6] Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8 – 17, 2015.

[7] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[8] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 2010.

[9] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[10] Borislav Mavrin, Hengshuai Yao, and Linglong Kong. Deep reinforcement learning with decorrelation. *ArXiv*, abs/1903.07765, 2019.

[11] E Tulving, M E. Le Voi, D A. Routh, and Elizabeth Loftus. Ecphoric processes in episodic memory [and discussion]. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 302, 08 1983.

[12] Robert Bruner. Repetition is the first principle of all learning. 2001. University of Virginia, Darden School of Business.

[13] Emily Mather and Kim Plunkett. The role of novelty in early word learning. *Cognitive Science*, 36(7):1157–1177, 2012.

[14] Khaled Hammouda et al. A comparative study of data clustering techniques. Technical report, University of Waterloo, 2000.

[15] Frederic C. Bartlett and Walter Kintsch. *Remembering: A Study in Experimental and Social Psychology*. Cambridge University Press, 2 edition, 1995.

[16] Marlieke T.R. van Kesteren, Dirk J. Ruiter, Guillén Fernández, and Richard N. Henson. How schema and novelty augment memory formation. *Trends in Neurosciences*, 35(4):211–219, 2012.

[17] Xiaobo Guo and Yan Zhai. K-means clustering based reinforcement learning algorithm for automatic control in robots. *Int. J. Simul. Syst. Sci. Technol*, 17:24, 2016.

[18] A. K. Pandey and R. Gelin. A mass-produced sociable humanoid robot: Pepper: The first machine of its kind. *IEEE Robotics Automation Magazine*, 25(3):40–48, 2018.

[19] Felix Leibfried, Jordi Grau-Moya, and Haitham Bou-Ammar. An information-theoretic optimality principle for deep reinforcement learning. *ArXiv*, abs/1708.01867, 2017.

[20] Vincent Franois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.

[21] Alfredo V. Clemente. Decoupling deep learning and reinforcement learning for stable and efficient deep policy gradient algorithms, 2017. Master Thesis, Norwegian University of Science and Technology.

[22] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *NeurIPS*, 2018.