The Role of Reward Signal in Deep Reinforcement Learning

Rafal Muszynski

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Philosophy of
UCL

Declaration

I, Rafal Muszynski, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Acknowledgements

This work was supported by Microsoft Research through its PhD Scholarship Programme.

Abstract

The goal of the thesis is to study the role of the reward signal in deep reinforcement learning. The reward signal is a scalar quantity received by the agent, and it has a big impact on both the training process of a reinforcement learning algorithm and its resulting behaviour.

Firstly, we study the behaviour of an agent that is learning with different reward signals in the same environment with the same learning algorithm. We introduce and measure agents' *happiness* as a relation between agents' actual reward obtained from the environment, as compared to the possible maximum and minimum rewards in a given setting. The experiments show that the rewards intended to result in a given behaviour during training do not result in the same behaviour when agents interact with each other.

Secondly, we use these observations to investigate the role of the reward signal further. Namely, we explore the space of all possible reward signals in a given environment through an evolutionary algorithm. Through experiments, we demonstrate that it is possible to learn complex behaviours of winning, losing, and cooperating through reward signal evolution. Some of the solutions found by the algorithm are surprising, in the sense that they would probably not have been chosen by a person trying to hand-code a given behaviour through a specific reward signal.

The results presented in the thesis indicate that the role of the reward signal in reinforcement learning is likely bigger than indicated by its current coverage in the literature and is worth investigating in greater detail. Not only can it lead to programmes with less overfitting, but it can also improve our understanding of what reinforcement learning algorithms are really learning. This in turn will give us more robust, explainable, and overall safer systems.

Impact statement

The results presented in the thesis can be of benefit for the Reinforcement Learning community in both academia and business.

In academia, for the field of Deep Reinforcement Learning specifically, we argue that our findings indicate that more focus should be placed on investigating the role of the reward signal. Firstly, we showed that the agents trained on different rewards in the same environment do not transfer their intended behaviour to situations when they interact with each other. Secondly, our experiments revealed that simply manipulating the reward signal in an evolutionary manner can be enough to obtain agents that are able to manifest a variety of complex behaviours.

Our hope is that such results will spark more research on the theoretical guarantees of reinforcement learning algorithms, mostly in a multi-agent setting. Additionally, combining our approach with population-based training could potentially result in agents with a more diverse set of strategies to choose from that are less prone to display unwanted behaviour. We see that as a potential contribution to the wider field of AI safety.

As far as the world outside of academia is concerned, the findings of our work will have an incremental impact on businesses that use machine learning in their products. Our experiments showed how unpredictable deep reinforcement learning algorithms can be, when trained on different rewards. Robustness of systems and their safety are of crucial importance in many practical applications such as self-driving cars, and more generally in each situation where we have autonomous agents interacting. Understanding what the agents are really learning is of vital importance in those industries. Hopefully, our work will be seen as a warning sign to those not testing their models properly, but also as a way forward, towards safer and more robust algorithms that offer a higher variety of solutions with less overfitting.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is becoming evident that artificial intelligence (AI) plays an increasingly more important role in our lives - from drug design, credit scoring, to providing us with the news we see on our social media. The more prominent AI's position in the world is, the more scrutiny and ethical considerations it attracts. As a result, there seems to be a bigger and bigger need in business and academia for models that are reliable, have predictable performance with no bias and are explainable. This issue is relevant for both the developers in companies creating the algorithms, and also for policymakers and customers. As we put more and more trust into AI, can we really say that our understanding of its inner workings improves satisfactorily?

The situation described in the previous paragraph typically involves supervised learning (SL) problems, which have enjoyed a rapid progress in the last few years due to the black-box class of algorithms - deep neural networks (DNNs). DNNs have achieved an incredible performance on many tasks such as image recognition, language translation and speech recognition. However, even the best algorithms can still be quite easily fooled as shown by the many adversarial examples present in the literature. And that is despite the well established know-how that the supervised learning practitioners enjoy from years of both academic theory and business applications. For example, in a supervised learning setting we typically have a well defined performance metric that we strive to optimize. Moreover, dividing the dataset into training, development and test sets allows us to perform model selection and avoid overfitting.

Reinforcement learning (RL) might soon play an equally important, if not even a more important, role in our lives as supervised learning does today. By contrasting RL with the typical SL setting, we can tell that RL problems can be even seen as more challenging, in the sense that they lack some of those well established good practices, and theoretical guarantees. For example, it is often the case in RL, that one would train and test the model in exactly the same environment. Hence, the familiar dataset split from supervised learning is absent from the RL pipeline. As

a result, we end up not only using black-box algorithms to learn, but one might be tempted to say that our evaluation of a single algorithm is also black-box and not fully understood.

Any university course introducing reinforcement learning usually involves presenting a diagram similar to Figure 2.5. It shows an agent taking an action in some environment and receiving a reward for it. Typically, most of the course is spent explaining the algorithms needed to learn actions leading to high rewards, ideally maximizing some cumulative reward function. By contrast, little time is usually spent discussing the reward signal itself and its impact on the learning process as a whole. Hence, it seems that the role of the reward signal might be a potentially interesting, yet often neglected, part of the reinforcement learning set-up. As it forms an integral part of the general RL "black-box", we have decided to explore it in greater detail in this thesis.

Do the reinforcement learning algorithms really learn what we think they learn? *Not always*, as shown - among others - by our experiments. Can we use the reward signal in order to achieve higher diversity of the resulting behaviours of the agent? Our other experiments suggest the answer is *yes*. Our findings indicate that there is potential in exploring generating *good* reward signals. This can lead to greater exploration of the state space, higher number of resulting strategies for the agent to choose from and, possibly, less overfitting and more predictability.

The structure of the thesis is as follows. In Chapter 2 we introduce the key concepts in reinforcement learning and evolutionary computation which we draw on in subsequent chapters. Chapter 3 describes unintuitive behaviour of the agents trained in the same environment, with the same learning algorithm, but with a different reward signal. The questions posed by the results of that simple experiment are studied further in Chapter 4, in which we develop an algorithm for exploring the state space of all possible reward signals, for a given goal function, using evolutionary computation. Conclusions spanning both studies, as well as open questions left to be answered in the future, are presented in Chapter 5.

Chapter 2

Preliminaries

2.1 Learning and its types

Marvin Minsky published "The Society of Mind" ([45] around the time the author of this thesis was born. The term *artificial intelligence* (AI) had only been introduced three decades earlier. And only in the last 30 years researchers have managed to develop AI systems better at chess than Garry Kasparov, or at Go than Lee Sedol. Hopefully, in the near future computers will become proficient at driving a car - saving around 1.25 million lives per year ([57]).

All the above examples testify to an enormous progress we have made and describe a situation when thinking humans try to create thinking machines. This distinction led to the creation of the term *machine learning*. Machine learning was born as a field out of a need for computer programmes that can perform tasks that seem easy for humans, such as recognizing your friends in a crowded room, yet are almost impossible to be described as a set of step by step rules for the algorithm to follow. A simple exercise of trying to describe precisely in words what our best friend looks like quickly showcases this issue.

It turns out, that in the case of this problem, known as *facial recognition*, it is far better to collect many photos of faces (a *training set*) along with the names of people in them (*labels*). Then, one might apply an algorithm with adjustable *parameters* to try to make as close a match as possible of the labels to the original photos. As a result, we receive the *learned* parameters, which we then want to use on a new set of previously unseen photos and labels (a *test set*) to make sure that the system has a good *generalization* power. Getting as high a generalization as possible on new sets of data can be seen as the main goal of machine learning [12].

As usually is the case, there are many types of problems that we can encounter in life, and many ways to solve them. In machine learning this has led to the following three main classes of problems [2].

In *supervised learning*, as in the example of face recognition mentioned above, we are essentially given the input data along with the correct output, all the data on the clients of the bank (e.g. their salary, credit history, etc. - known as *features*) and whether they paid their loan back or not - the label.

Reinforcement learning is characterized by the fact that the algorithm is given input data, but only some output with a grade for that output. A self-driving car can receive data from many sensors and cameras, but we cannot tell it if every single minuscule change it makes while driving is good or bad, we can only give that feedback periodically.

In *unsupervised learning* the algorithm receives the input, but no output. That means it receives no correct labels. It has to figure out, *learn* those labels by itself. Trying to categorize books into genres can be seen as a practical example of this problem.

There are many other approaches to learning from data, or to AI in general (cf. [67]). Lastly, we want to mention *evolutionary computation*. It comprises algorithms inspired by biology, and uses the ideas of trial-and-error, random mutations and inheritance to create better and better solutions to a given problem.

In the following subsections of this chapter we spend more time describing the concepts from both reinforcement learning and evolutionary computation, as they form the basis of the main studies presented in the thesis. Concretely, DQN - a deep reinforcement learning algorithm is used in both Chapters 3 and 4, and evolutionary computation is used in Algorithm 2.

First of all, we describe the necessary concepts from *deep learning*, a powerful supervised learning technique that has helped accelerate the progress in reinforcement learning in the recent years.

2.2 Deep Learning

2.2.1 Key building blocks

Deep learning [41, 72] has emerged in recent years as a powerful technique to solve many supervised learning problems for which we have big amounts of labeled data. It has turned out, that as we have bigger and bigger datasets, using algorithms with more and more parameters can actually allow us to learn the function transforming the data into labels ($f : X \to Y$) better. As opposed to *shallow learning* in which we learn the $f : X \to Y$ in one step, deep learning learns a *deep representation* of $f : X \to Y$ through a composition of many functions. As a result, each level of composition learns representations at different level of abstraction. This approach is represented in Equation 2.1, where the input x is transformed through function

h_1 along with parameters $\mathbf{w}_1$, which is then transformed through function h_2 along with parameters $\mathbf{w}_2$, and so on. Eventually, we get the output y, passed to the *loss function* l in the last step to compare predictions with true labels and give us a sense of how good our model is.

$$\begin{array}{ccccccccccc} x & \longrightarrow & h_1 & \longrightarrow & \dots & \longrightarrow & h_n & \longrightarrow & y & \longrightarrow & l \\ & & \uparrow & & & & \uparrow & & & & \\ & & \mathbf{w}_1 & & \dots & & \mathbf{w}_n & & & & \end{array} \tag{2.1}$$

The process presented in Equation 2.1 is also known as the *forward pass*. It is possible to learn good values of the *weights* (or parameters) by using the *chain rule* known from calculus, and backpropagating the error calculated at the end of the forward pass to the parameters earlier in the graph, and making small tweaks to them. This is known as the *backward pass*, and the algorithm used for updating the weights is called *back-propagation*. The process is shown in Equation 2.2.

$$\begin{array}{ccccccccccc} \frac{\delta l}{\delta x} & \xleftarrow{\frac{\delta h_1}{\delta x}} & \frac{\delta l}{\delta h_1} & \xleftarrow{\frac{\delta h_2}{\delta h_1}} & & \dots & & \xleftarrow{\frac{\delta h_n}{\delta h_{n-1}}} & \frac{\delta l}{\delta h_n} & \xleftarrow{\frac{\delta y}{\delta h_n}} & \frac{\delta l}{\delta y} \\ & & \downarrow{\scriptstyle \frac{\delta h_1}{\delta w_1}} & & & & & & \downarrow{\scriptstyle \frac{\delta h_1}{\delta w_n}} & & \\ & & \frac{\delta l}{\delta \mathbf{w}_1} & & & \dots & & & \frac{\delta l}{\delta \mathbf{w}_n} & & \end{array} \tag{2.2}$$

There are different choices for the forms that the function h can take.

- Linear transformations

  $h_{t+1} = Wh_t$

- Nonlinear activation functions

  $h_{t+1} = \sigma(h_t)$, where $\sigma(\cdot) = max(0, \cdot), \frac{1}{1 + exp(\cdot)}, ...$

The nonlinear activation functions play a crucial role, as they allow to capture and model the complex structure of real-world data with higher accuracy.

The loss function l can be calculated as Mean Squared Error, but it can also take different forms, depending on the details of our problem.

Finally, since it is computationally expensive to calculate the cost function (loss over all training examples, **J**) for the full dataset, different optimization algorithms can be used to update (or learn) the weights **w** in a more efficient way. All of them work by adjusting the value of **w** by a small scalar value known as the *learning rate* (α) multiplied by the gradient of the cost function.

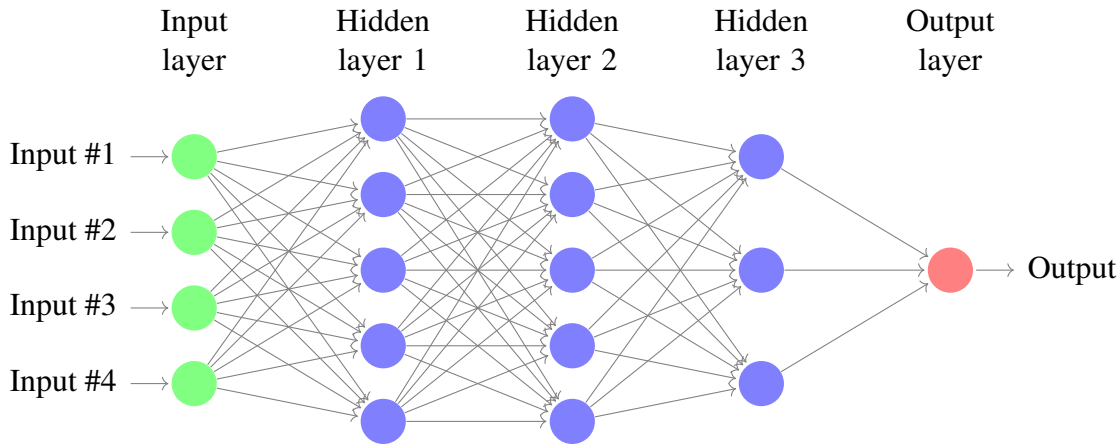

Figure 2.1: An example of a Multilayer Perceptron architecture with 4 input neurons, 3 hidden layers of sizes 5, 5, and 3, and 1 output neuron.

$$\mathbf{w} := \mathbf{w} - \alpha \nabla \mathbf{J} \tag{2.3}$$

Typically, the update is performed based on the gradient calculated on a random mini-batch of i examples ($i > 1$ and smaller than the size of the training set). This approach is known as *mini-batch gradient descent*. Other choices include ([65]): gradient descent with *momentum* [61], *Root Mean Square Prop* algorithm (RMSprop)[32], and *ADAM* [37], which is the combination of the former two.

2.2.2 Deep Learning models

We can use the building blocks introduced in the previous section, along with some further techniques, to create a vast array of architectures [25]. Arguably, the three main ones are *multilayer perceptron* (MLP), *convolutional neural network* (CNN), and *recurrent neural network* (RNN).

An example of an MLP network is presented in Figure 2.1. It consists of an *input layer*, *output layer*, and a number of *hidden layers*. Each hidden layer consists of *neurons* that perform linear and nonlinear calculations described in the previous section. If all the neurons in the layer have a connection to all the neurons in the next layer, we say that the layer is *fully connected*. Edges in the figure represent the flow of information through the layer and have weights w associated with them. All the neurons in the first hidden layer use the information from all the inputs, transform it according to their specified transformations, and send the information to all the neurons in the following layer, until it reaches the output. The weights can be initialized randomly, and are later learned through the process of back-propagation.

If we think about an image of dimensions 1000 pixels by 1000 pixels by 3 RGB channel values as an input to the fully connected layer, we can immediately

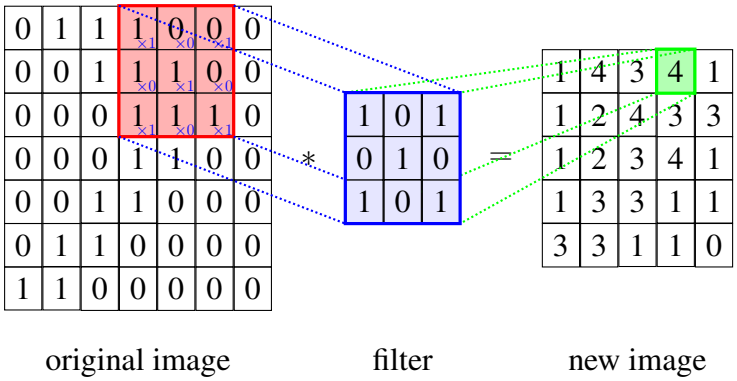

Figure 2.2: An example of a convolution of an image with a 3x3 filter.

notice that the number of weights needed to describe even one neuron is enormous. Learning such a network would require a lot of computational power and many examples of images in order to avoid overfitting.

However, the *convolution* operation can help streamline the process and leads to a second important class of deep learning architectures - a convolutional neural network [42], in which weights are shared between local regions of our data (typically visual data). Sharing of weights, and reducing the number of parameters needed by the network is achieved through an operation of *convolution*. A convolution of 2d matrix of numbers with another 2d matrix is presented in Figure 2.2. On the left we see a 7x7 image with its pixels encoded with numbers. It is convolved with another 3x3 matrix, known as *filter* or *kernel* to produce a new 5x5 image on the right. Each entry in the resulting image is obtained by calculating a sum of elementwise multiplications of the entries in the original image with the filter. As an example, number 4 (highlighted in green) is obtained by calculating $1*1+0*0+0*1+1*0+1*1+0*0+1*1+1*0+1*1$. In reality, the values of the numbers in the filter are not known and are learned through back-propagation.

Depending on the size of the original image we can use *padding* if the filter size is not a full multiple of the original image. We can also calculate the filter at different *strides* by skipping the desired amount of pixels. To build a full CNN, we usually combine the convolution calculation (and calculate its activation) with *pooling* and fully connected layers. Instead of performing convolution operation of a size given by the filter, pooling operation typically returns a max of a given region (this is known as *max-pooling*, *average pooling* - where the average of a given region is returned - is used less often). An illustration of an example of a full CNN is provided in Figure 2.3. It shows a representation of an input image of 128x128 pixels. The image is convolved with 8 filters of the size 8x8 in the first layer, which is followed by another convolutional layer with 24 filters of the size 16x16. The next two layers perform max-pooling operation, after which the results

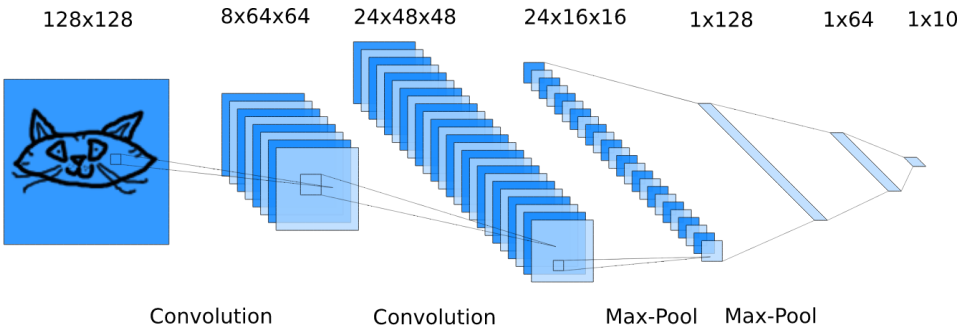

Figure 2.3: An example of a Convolutional Neural Network and its main ingredients: convolution, pooling and fully connected layers.

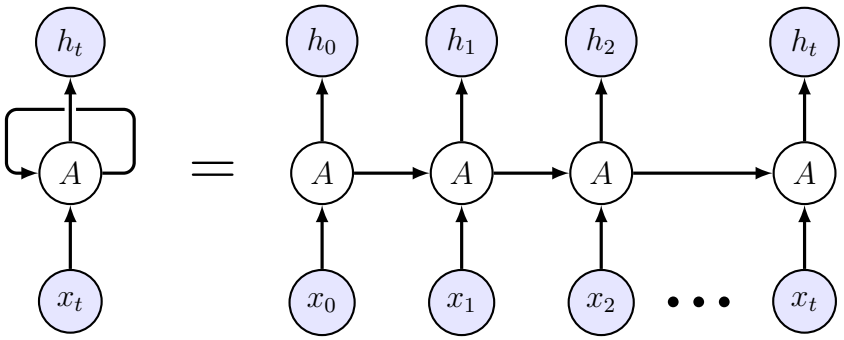

Figure 2.4: Recurrent Neural Networks share parameters over timesteps.

are passed to two fully connected layers and produce ten outputs, e.g. 10 different handwritten digit categories.

The last main class of deep learning architectures is a recurrent neural network [66] which also uses the idea of sharing the weights - similarly to CNN. However, the weights are shared between timesteps of the computation, not between local regions of our data. Figure 2.4 shows a basic RNN architecture and illustrates the fact that parameters A are used to produce not only the output of h_0 when applied to input x_0, but also for all future outputs h_t. Since we do not concentrate on RNNs in this thesis, we refer the reader to [25] for further details.

2.3 Reinforcement Learning

Reinforcement learning (RL) [78] is an area of machine learning that describes the learner as an agent in an environment. The dynamics of a classical RL setting are depicted in Figure 2.5. The agent is embedded in an environment in state S_t and is given a reward R_t for visiting that state. Subsequently, the agent takes an action A_t only to find itself in the new state S_{t+1} with a reward R_{t+1} - the loop keeps repeating.

Consequently, the agents in RL settings face two crucial problems. Firstly, the agent has to find the right balance between *exploration* - visiting states it never

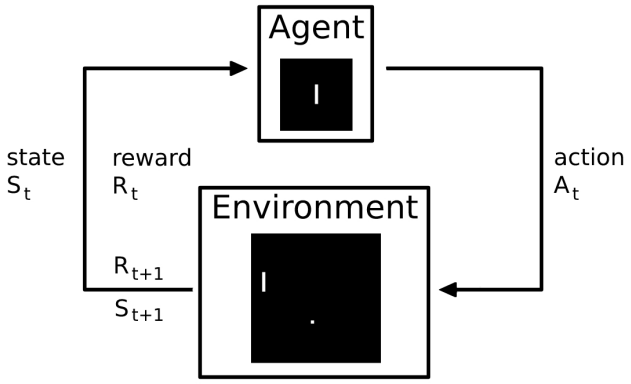

Figure 2.5: Typical Reinforcement Learning setting.

visited before, trying out new actions - and *exploitation* - using the learned "knowledge" in order to maximize its cumulative reward (defined in the next section). Secondly, the agent has to solve a *credit assignment problem*. This means that the algorithm has to find a way to connect the dots between the actions it took and the rewards that were consequences of those actions, and the two are often delayed in time - making the whole problem even more difficult.

The above constitutes the classical view of RL. We can also think about it along the following axes. Firstly, the number of agents interacting in the environment yields single or multi-agent RL. Secondly, the complexity of the environments and the techniques used allow us to distinguish between classical and deep learning approaches to RL.

2.3.1 Single agent reinforcement learning

Classical techniques

A key concept on which RL hinges is the Markov property which states that only the current state affects the next state for the agent. This means that the future is conditionally independent of the past, given the current state. As a result, a case when there is a single agent within a RL framework is usually formalized as a *Markov decision process (MDP)*, a 5-tuple, where:

- S is a finite set of states,
- A is a finite set of actions,
- $P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$, called the transition function,
- $R_a(s, s')$ is the immediate reward received after transitioning from state s to state s', as a consequence of action a (we will refer to this quantity as r_t for

simplicity),

- $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards, and keeps the cumulative reward finite.

The goal of the agent, the goal of learning in MDP, is to find a policy π that maximizes the cumulative reward by best mapping states to action selection probabilities. The cumulative reward is described by the sum of immediate scalar reward r over t timesteps: $\sum_1^t r_t$.

In the case when the model of the environment is not available, reinforcement learning can be used to find π. *Q-learning* [17] is arguably the most famous example of an RL algorithm. It is a model-free temporal difference algorithm. In Q-learning, a value function $Q(s, a)$ is calculated over state-action pairs. It is updated based on the immediate reward and the discounted expected future reward in the following way:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)].$$

In the update rule above, γ is the discount factor for future rewards, and $\alpha \in [0, 1]$ is the learning rate that determines how quickly Q is updated based on new reward information. Q-learning is proven to converge to the optimal policy π^*, given "sufficient" number of updates for each state-action pair, and a decreasing learning rate α. Additionally, the first two terms in square brackets, form the temporal difference target (TD target), which is the basis of temporal-difference learning algorithms. Q-learning is a specific TD algorithm used to learn the Q-function.

Of course, countless other approaches have been developed to tackle different scenarios. For a comprehensive treatment of the subject we refer the reader to [78]. We included a brief description of Q-learning as it is the most relevant algorithm in the context of the work presented in subsequent chapters.

Deep Reinforcement Learning techniques

The growing amount of data and much better technology to work with it, have both contributed greatly to the recent reemergence of interest in neural networks. As a result, neural networks have grown from "shallow" models of a few layers to "deep" models that have quickly shown impressive practical power in the realm of image recognition and beyond [41, 72]. This new approach has been called deep learning, and was extended to the reinforcement learning setting in [46] as a Deep Q-learning algorithm (DQN) - and has given birth to Deep Reinforcement Learning (DRL). Deep learning techniques can be seen as excelling at low-level intelligence - work-

Algorithm 1 Deep Q-Network algorithm. Adapted from [46]

initialize replay memory D
initialize action-value function Q with random weights $\boldsymbol{\theta}$
initialize target action-value function $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
for $episode = 1$ to M **do**
  initialize sequence of images $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1$ to T **do**
    select $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ argmax_a Q(\phi(s_t), a; \boldsymbol{\theta}) & \text{otherwise} \end{cases}$
    execute a_t and observe r_t and x_{t+1}
    set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    store $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    // experience replay
    sample $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \boldsymbol{\theta}^-) & \text{otherwise} \end{cases}$
    perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \boldsymbol{\theta}))^2$ w.r.t. the network parameter $\boldsymbol{\theta}$
    // periodic update of target network
    every N steps, reset $\hat{Q} = Q$, i.e., set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
  end for
end for

ing with visual of audio data. High-level intelligence requires making decisions and planning - which have traditionally been a goal of reinforcement learning. Hence, in DRL we can naturally merge the two by using DL to learn the representation part of the problem, and use RL to learn the objective.

Many other approaches besides DQN (and its extensions) have been developed in DRL. For a brief summary see [8], or [43]. Here, we specifically concentrate on the original DQN, as one of the building blocks behind the experimental results presented in the following chapters.

DQN is a value-function based algorithm. What was groundbreaking about it was the fact that it was the first algorithm shown to work from scratch, only with pixels as an input, on the challenging domain of Atari 2600 games [11]. No changes to the architecture were made, and it was able to master many of the games at a super-human level. The full algorithm introduced in [46] is presented in Algorithm 1.

Instead of using the information about the true state of the game contained within 128 bytes of Atari 2600 RAM, the DQN directly learns from the pixels on the screen. This constitutes the state s_t of the environment encoded by a 210x160 pixel 8-bit RGB image. This makes the possible state space incredibly big and

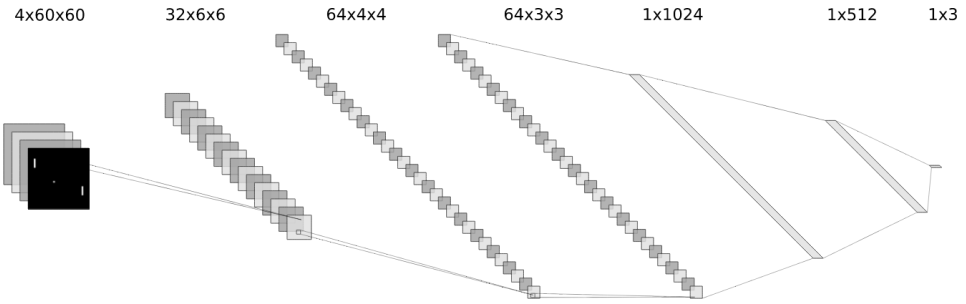

Figure 2.6: Diagram presenting DQN architecture used in the experiments in the following chapters.

impractical to use to build the Q-table of state-action pairs to look up.

The novelty of the DQN was in its ability to learn in this high-dimensional setting and to learn a Q-function by using deep neural networks. The DQN also addressed the issue of instability of function approximation in RL by using experience replay and target networks.

Experience reply acts as a memory bank of the tuple $(s_t, a_t, s_{t+1}, r_{t+1})$. As a result the agent can sample from this experience without further interacting with the environment. This is used for offline training of the algorithm. There are a few advantages to such an approach. Firstly, by sampling batches of experience we reduce the variance of learning updates. Secondly, since the typical memory buffer is large, sampling from it breaks the temporal correlations that can adversely affect RL algorithms. Last but not least, there is a practical advantage to using a memory buffer, as modern hardware is well-suited for working with batches of data, which speeds up the training process.

Introduction of the target network has led to further stabilization of the algorithm. Initially, the target network contains the weights of the network enacting the policy, but the weights are kept frozen for a large period of time. The weights are only updated during training to make the target network match the policy network after a fixed number of steps.

The DQN algorithm has been extended in many ways since its introduction [8].

Lastly, we present the DQN architecture used in the experiments in the subsequent chapters in Figure 2.6.

2.3.2 Multi-agent reinforcement learning

In the MDP formulation we assume that there is only one agent interacting with the environment. If we want to extended this to model multi-agent reinforcement learning (MARL), where many agents may interact with each other and learn simultaneously, Markov games, or stochastic games offer a generalization of MDPs [13]. The joint action space now consists of all the actions taken by n agents. The

same goes for the rewards and transitions between states - they all depend on joint actions taken by the agents.

Learning in a multi-agent setting is much more complex. Agents may compete for a limited reward, they may have different, opposite goals, they can impede and shape actions of one another. The environment becomes non-stationarity and the Markov property does not hold anymore from the perspective of a single agent. What follows is a brief description of key approaches to learning in multi-agent settings from the classical perspective and in the context of DRL.

According to [13] there are two main approaches to learning in multi-agent RL setting, and one approach which can be seen as a mix of the two.

The first, and maybe obvious approach to working with MARL, is the one classified as *independent learning*. The approach simply consists in reducing the MARL problem to the single agent RL setting. Basically, each agent treats other agents just as a part of the random environment with which it interacts. The clear advantage of this approach is that we can just try using the algorithms developed for the single agent RL. However, we lose the convergence properties when actually working with MARL. The previously discussed Q-learning is and example of an algorithm that falls within this category.

The second approach to MARL - *joint-action learning* - takes the other agents present in the environment into account. Agents learn from the space of joint actions, not the individual actions as in independent learning. The fact that agents need to be able to observe actions of others is one of the drawbacks of this approach. Additionally, as the number of agents grows, the complexity of the algorithm grows exponentially.

Lastly, *gradient ascent optimization* can be seen as a mix of the two approaches discussed above. The key concept is to use gradient descent and have the agents' policies follow the gradient of their individual expected rewards. For examples of joint-action learning and gradient ascent optimization algorithms used in MARL see [13].

Learning in multi-agent environment has been studied extensively [31], but only recently has the use of DRL in that setting been gaining attention. Perhaps the first example of using DRL in the context of multi-agent setting was [79], which studied cooperation and competition by manipulating the rewards received by agents. Since then, the field of DRL has really taken off, incorporating more and more techniques, including ideas from evolution. We discuss such studies in more detail in Chapter 4.

The experiments presented in the following chapter adopt an independent learning approach.

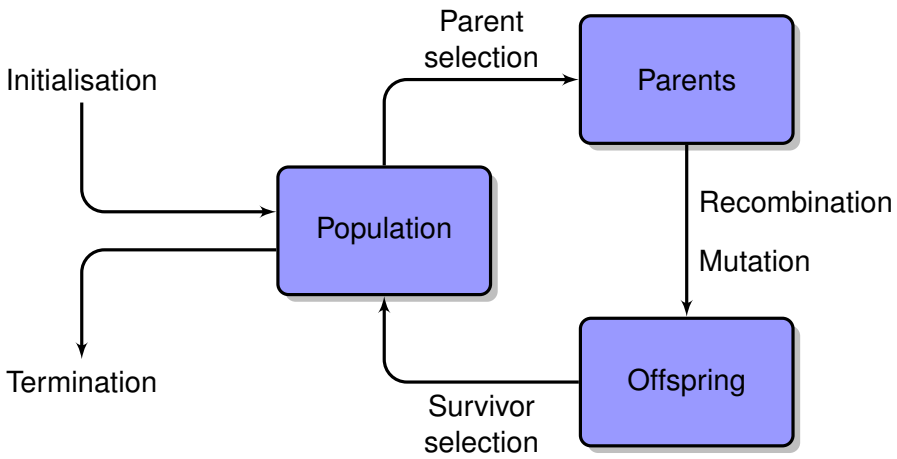

Figure 2.7: The general Evolutionary Algorithm flow. Adapted from [20].

2.4 Evolutionary computation

Reinforcement learning methods presented earlier in the chapter focus on estimating value functions in order to solve reinforcement learning problems. However, there are other approaches to solving those same problems that do not rely on the idea of the value function at all. Evolutionary computation is an example of one such approach. The following treatment is based on [20].

Evolutionary computation is a biologically inspired class of algorithms. Its main building block is an *evolutionary algorithm*. The way an evolutionary algorithm works is similar to the way the actual biological evolution takes place. Namely, we have a population in a specific environment with limited resources, which causes a pressure for natural (or in a computational setting - artificial) selection. This leads to the survival of the fittest. Consequently, the overall fitness of the whole population increases.

More concretely, the main components of evolutionary algorithms are [20]:

- representation (definition of individuals)
- evaluation function (or fitness function)
- population
- parent selection mechanism
- variation operators, recombination and mutation
- survivor selection mechanism (replacement)

A typical evolutionary algorithm has all of the above ingredients specified, usually along with some initialization and termination conditions (see Figure 2.7).

Initialization is usually performed at random, whereas the termination conditions usually involve [20]: specifying the maximall CPU time, fitness improvement or population diversity dropping under some specified threshold.

To explain the flow in Figure 2.7 in more detail, imagine we are given a quality function that we want to maximize. During initialization we can randomly generate a set of candidate solutions. Each candidate has its own fitness, as measured by the *fitness function*. The higher value means higher fitness. Those values can help us choose *parents*. By applying *recombination* and/or *mutation* to them we create the *offspring*. The distinction between recombination and mutation is that in recombination we choose two or more parents to produced one or more children, whereas mutation typically is applied to one parent and results in one child. Children enter the population and have their fitness assessed. At this point the cycle repeats until a desired output is found. It is worth noting that many components of the described process are stochastic in their nature.

Recombination and mutation can be seen as the driving forces behind increasing the *diversity* of the solution, while the selection process concentrates on ensuring the *quality* of it.

Popular evolutionary algorithm variants along with data structures that they work on include [20]:

- Genetic Algorithms (strings over a finite alphabet)
- Evolution Strategies (real-valued vectors)
- Evolutionary Programming (finite state machines)
- Genetic Programming (trees)

According to Eiben and Smith [20], the emergence of the differences between these algorithm classes are mainly historical in nature, and the choice of the approach depends on the suitability of the data representation of our problem.

Sutton and Barto [78] give many contrasting points between evolutionary computation and reinforcement learning algorithms. First of all, evolutionary methods do not really produce agents that learn while interacting with the environment, in a typical machine learning sense. Secondly, evolutionary computation methods ignore the variety of the useful information given by the reinforcement learning problem, such as the clear relationship between the states and actions. Thirdly, pure evolutionary computation can be effective when the space of policies is small, we have a lot of time for search, or the agent cannot sense the complete state of its environment. As a result, Sutton and Barto "[...] do not consider evolutionary methods by themselves to be especially well suited to reinforcement learning problems [...]."

The authors do, however, note that evolutionary computation can be a useful component of a bigger reinforcement learning system.

Chapter 3

Obvious Reward Choice Leads to Unexpected Behaviour: Personality Learning in a Society of Agents

3.1 Summary

The original goal of this paper was to explore the possibility of modeling different personalities in artificial agents. However, it became apparent that the results of the experiments gave evidence of the fact that hand-designed reward signals are not always predictive of the intended behaviour of the agent, and that the problem of overfitting quickly becomes pronounced when we move from independent reinforcement learning to a setting with more than one agent. At the time of working on this part of the thesis, other researchers reported similar problems with undesired behaviours of their RL algorithms and the issue of overfitting (most notably [6], which can be seen at https://openai.com/blog/faulty-reward-functions/ and [40]).

Modeling personality is a challenging problem with applications spanning computer games, virtual assistants, online shopping and education. Many techniques have been tried, ranging from neural networks to computational cognitive architectures. However, most approaches rely on examples with hand-crafted features and scenarios.

Here, we approach learning a personality by training agents using a Deep Q-Network (DQN) model on reward signals inspired by psychoanalysis. The agents are trained against hand-coded AI in the game of Pong. As a result, we obtain 4 agents, each with its own "personality." Then, we define happiness of an agent as a relation between the actual reward obtained by it and the potential minimum and maximum values of the reward. The happiness measure can be seen as a measure of alignment with the agent's objective function, and we study it when agents play

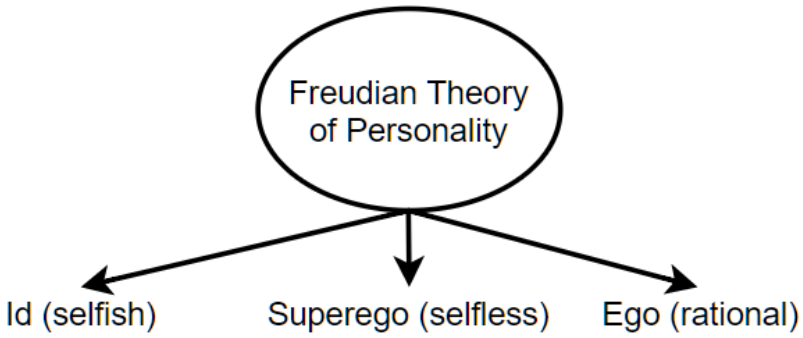

Figure 3.1: Components of personality according to Freud.

both against hand-coded AI, and against each other, in a small society of agents. We find that the agents that achieve higher happiness during testing against hand-coded AI, have lower happiness when competing against each other. This suggests that higher happiness in testing can be seen a sign and measure of overfitting in learning to interact with hand-coded AI, and leads to worse performance against agents with different personalities.

3.2 Introduction

Personality is defined in [73] as the "unique, relatively enduring internal and external aspects of a person's character that influence behavior in different situations." Theories of personality strive to explain and describe different types of personalities among people.

Freud [23, 22] was the first to develop a modern theory of personality based mostly on clinical observations. In [73], Freudian structure of personality is described as composing of three elements: id, ego, and superego (presented in Figure 3.1). The first element - id - is concentrated on following basic instincts. It is interested in instant gratification, knows no morality and is selfish. On the opposite side, superego constitutes the moral aspect of personality. It wants to act according to parental and societal values and standards. Finally, ego is the rational element of the Freudian model. Its role is to resolve the conflicts between the demands of id and the moralizing superego by using defense mechanisms such as denial or repression.

Many alternative approaches describing personality have been proposed since (for a detailed discussion see [73]). Moreover, computational models of personality have also been implemented (see section 3.3). Most of them concentrate on intrapersonal aspects of personality and work with hand-crafted features, in relatively simple settings.

In the last few years, the advances in Deep Learning (DL) [41, 72] and the development of Deep Q-Networks (DQN) [46] have made it possible to work with increasingly more complex environments, e.g. Atari 2600 games [11], using only pixels as input. Hence, the need for specifying the features for the model to use has lessened.

In this chapter, we explore the applicability of DQN to computational psychology on the intrapersonal level. We concentrate on id and superego components of personality. The key idea is to employ DQN to learn a Freudian-inspired personality model from raw input in the game of Pong and use the resulting agents with different "personalities" to study their performance both in training and in a society of bots. We want to underline, that we refer to agents trained on different reward signals as having different "personalities" and our goal is not to emulate humans' behaviour, but to study such artificial agents in the context of a reinfrocement learning framework.

Firstly, to achieve agents with different personalities, we formulate and train a DQN-based model using Freudian-inspired reward functions (section 3.4). We let one DQN learn from a reward signal strengthening selfish behavior - in Pong this can be reflected by scoring a point against an opponent - which is a typical characteristic of id [73]. Another DQN learns superego - based on rewards corresponding to morality, cooperation. In the game, this can be represented by trying to win, but without getting ahead of our opponent by more than one point. Secondly, we define a "happiness" measure of a resulting id/superego agent based on its performance in relation to its objective function (section 3.4). Finally, when the trained id and superego agents play against each other, we find that the agents that are less happy after the training session - or less aligned with their intended reward objective - are more happy when they compete against each other (section 3.5). This suggests that less overfitting during training, leads to a more successful, resilient behavior of an agent when encountering other agents in the society.

3.3 Related work

To our knowledge, the first attempt to computationally describe the psychoanalytic theory of mind is [55] (cf. [54]). The model involves ego, superego and id which correspond to reality, morality and pleasure respectively. Defensive mechanism is assumed to play an important role and inductive probability is used for modeling it. Id, superego and ego are each assigned an energy level which sum to one. Additionally, separate parameters for repression and anxiety are set. An agent updates its parameters based on the interactions with the environment. The agent reacts to a trigger, which leads to a conflict between id and superego. The conflict leads to

anxiety of ego, and if the critical anxiety levels are reached - defense mechanisms are invoked. The authors give a numerical example demonstrating how the repression mechanism works in predefined conditions. We did not find any extensions of the model to computer simulations.

The above approach relies on basic probability to describe the Freudian theory of personality. More recently, researchers have used neural networks to learn personality models based on more modern theories of personality. A computational model of personality using some of the traits present in arguably the most widely accepted interindividual theory of personality - the Big Five (cf. [73]) - was demonstrated in [64]. The authors note that the Big Five (also called the Five Factor Model - FFM) does not provide a model of psychological dynamics. In other words, it describes the structure of personality with five factors, not the dynamics of the structure. The authors describe personality by configurations of resources, goals and behaviors given different situations. The model is a hierarchical neural network. A list of situational features feeds into the goal list (avoidance or approach behaviors), which in turn - together with the layer of available resources - feed into the final layer describing different behaviors. A hidden layer is present both between the situational features and the goal list, and between the goal list with resources and the final behavior layer. Other examples of simpler models and scenarios using neural networks to learn a personality include [62, 63]. Another probabilistic approach is given by [39], which uses a Bayesian Belief Network with the FFM to build a multilayer personality model. The three layers used are emotion categories, mood, and personality. The model is used in a simulation of personality for an emotional virtual human in a chat application.

Yet another approach to modeling personality is to use cognitive architectures. Cognitive architectures stem from cognitive science and play a similar role to intelligent agents in AI community. Newell introduced the idea of *Unified Theories of Cognition* (UTC) - a single set of mechanisms supposed to characterize all of cognition [48]. Many such mechanisms have been proposed since, concentrating mostly on the rational aspects of cognition (cf. [10]). However, there exists a line of work extending the cognitive architectures to studying personality. The fact that you can replicate many of the previous experiments used in computational models of personality, e.g. [64, 63], within a CLARION cognitive architecture was demonstrated in [77]. This work shows that a cognitive architecture by itself is enough to simulate a personality without making too many changes and can incorporate individual invariance of an agent and the variability of behavior. Its inclusiveness of previous simulations indicates that the CLARION architecture is more general than other computational models of personality. The CLARION architecture relies on both Q-learning and neural networks to learn the model. As far as other cogni-

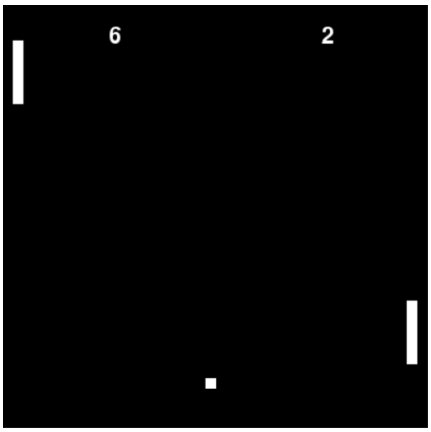

Figure 3.2: The goal of the original Pong game is for either of the players to bounce the ball past the opponent's paddle.

tive architectures are concerned, [36] use one trait of the Five Factor Model in the ACT-R architecture and demonstrate the behavior of their model (PIACT) in a soccer simulation environment. The BDI architecture has also been used to simulate personality in [58], or more recently with the FFM in [4].

The biggest difference in our approach to modeling personality is that it only involves specifying new reward signals for the id or superego components of the personality model to achieve complex behavior in a challenging environment.

We also acknowledge that there is a big difference between personalities in people and the computational models of personality described in the literature mentioned above (including ours). The focus in this work is not to emulate a person's behavior, but rather to study the behaviour of artificial agents trained on the rewards signals inspired by Freudian theory of personality.

3.4 Problem description

Motivated by the Freudian theory of personality [23, 22], we formulate two types of rewards and agents corresponding to the id and superego elements of personality.

Definition 1. *ID reward (r_{ID}) is a predefined, scalar reward signal sent to the agent at each time step, encouraging selfish behavior.*

Definition 2. *SE reward (r_{SE}) is a predefined, scalar reward signal sent to the agent at each time step, encouraging social behavior.*

Definition 3. *Cumulative ID reward (R_{ID}) is a total reward achieved by an agent in a Markov Chain of length n, ending with a terminus event e:*

$$R_{ID} = r_{ID1} + ... + r_{IDn} \tag{3.1}$$

Definition 4. *Cumulative SE reward (R_{SE}) is a total reward achieved by an agent in a Markov Chain of length n, ending with a terminus event e:*

$$R_{SE} = r_{SE1} + ... + r_{SEn} \tag{3.2}$$

Definition 5. *ID is an agent with an objective of maximizing R_{ID}.*

Definition 6. *SUPEREGO (SE) is an agent with an objective of maximizing R_{SE}.*

The last thing we need defined is happiness. Happiness of people can be measured through surveys, e.g. [44], and later captured as a scalar value on a scale. More recently, it has been described mathematically in computational neuroscience as a relation between certain rewards, expected values of given gambles, and the difference between expected and actual rewards in individual [69], and more broadly in social context [68]. Our definition of happiness is for artificial agents and takes into account maximum and minimum values they can obtain in an environment in relation to their actual reward, and is independent of the happiness of other agents (but of course is influenced by the performance of the opponent encountered).

Definition 7. *Happiness of Agent X (H_X) is defined as the quotient:*

$$H_X = (R_X - R_X^*)/(R_X^{**} - R_X^*) \tag{3.3}$$

where the cumulative reward obtained by agent X in a Markov Chain, ending with a terminus event e is denoted as R_X, and its potential maximum and minimum values as R_X^{**} and R_X^*, respectively.

Using the above definitions, we train ID and SE agents in the game of Pong. We study their happiness both when training against hand-coded AI, and when they play against each other in a society of bots. Again, we want to acknowledge that the measure of happiness presented in this section serves as a measure of performance of an artificial agent in its environment, and not as a proposed method to study happiness in humans.

3.5 Experiment set-up and results

In order to demonstrate ID and SE agents in practice, we modified a simple Pong game (example of screen setting seen by each agent is presented in Figure 3.2). The game has 2 players, each controlling one of the paddles on either side of the screen. The goal of the game is to bounce the ball in such a way, so that it goes past the opponent's paddle - for which the player is awarded 1 point. The game ends with one of the players scoring 11 points.

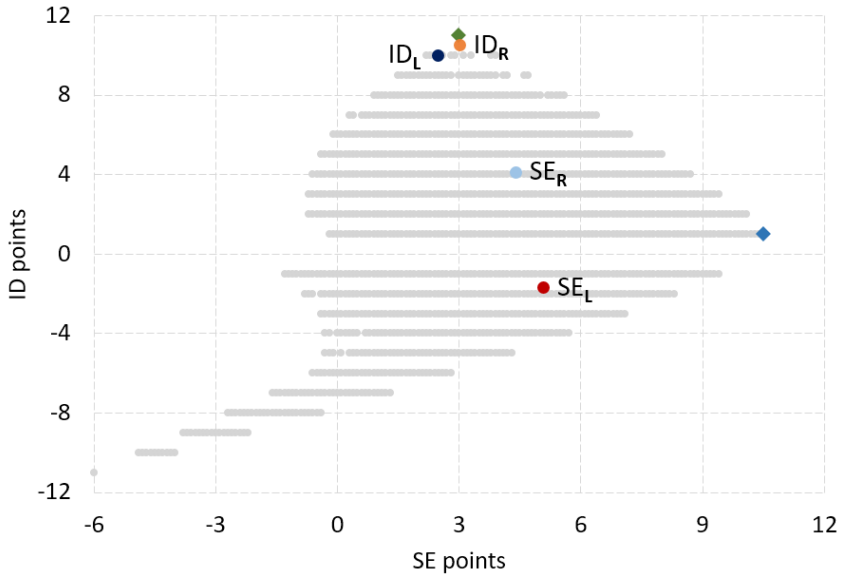

Figure 3.3: Possible R_{ID} and R_{SE} values combinations at the end of the game and the actual average values obtained by the agents during 1000 test games against hand-coded AI.

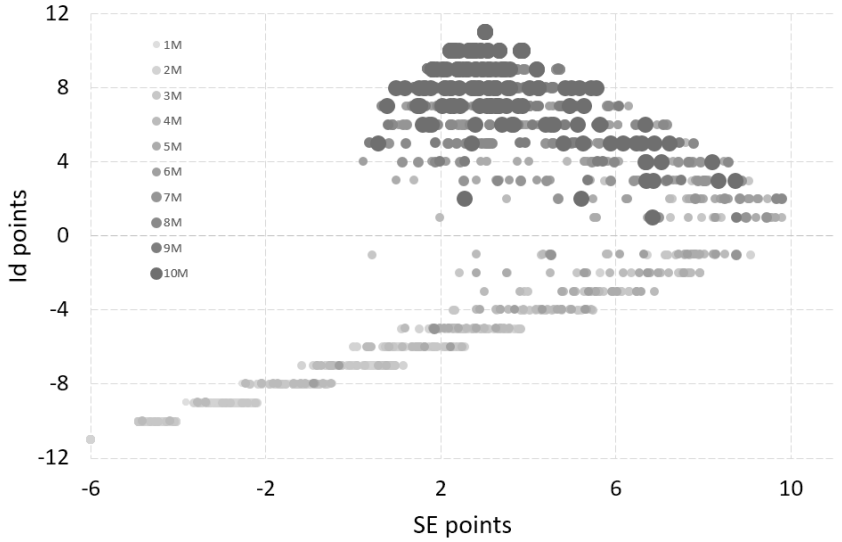

Figure 3.4: The evolution of training of the ID_R agent plotted against the possible outcomes of each game in terms of the cumulative R_{ID} and R_{SE} rewards, shows the successful convergence of the scores closer and closer to the agent's optimal goal.

In terms of the terminology from section 3.4, scoring 11 points is the terminus event (e), after which a new match begins. A subjective reward (r_{ID}) used to train ID relates to the id component of the Freudian theory of personality that is selfish. Here, we set r_{ID} to +1, if the ball goes past the opponent (ID scores a point), -1 if it goes past ID (ID loses a point), and 0 otherwise. As a result, the minimum value of R_{ID} (i.e. R^{*}_{ID}) in one match is -11 (losing every single point), and the maximum is +11 (R^{**}_{ID}). On the other hand, r_{SE} captures the social aspect of the environment, in line with the superego component of personality. Here, the rules for getting the reward are more involved. In short, the goal of SE is still winning, ideally 11:10, and taking turns in scoring the points with the opponent.[1] Hence, R^{*}_{SE} is -6, and R^{**}_{SE} is 10.5. Fig. 3.3 shows possible values of R_{ID} and R_{SE} that an agent can obtain at the end of each match (for clarity, the values of R_{SE} are rounded). Note the natural trade-off line between the optimal ID (pair $R_{SE} = 3$, $R_{ID} = 11$, the green diamond at the top) and SE values (pair $R_{SE} = 10.5$, $R_{ID} = 1$, the rightmost blue diamond).

3.5.1 Training phase

To train ID and SE agents, we implement DQN [46] in TensorFlow [1] (used also for the experiments in the following chapter) with r_{ID} and r_{SE} as the reward signals. Both ID and SE agents are first trained separately against a hand-coded AI (which simply tries to follow the ball so that it is in the middle of the paddle at each point of the game), and strive to maximize R_{ID} and R_{SE} respectively. We train ID to control the paddle on the left and right on 10 million frames. We do the same for the SE agent. As a result we obtain 4 models: ID trained to control the paddle on the left (ID_L), right (ID_R), and the SE agents respectively (SE_L and SE_R). Note that ID_L did not perform well when used to control the paddle on the opposite end of the screen (the same was observed for other agents), hence the 2 agents per side. Figure 3.5 shows a learning curve for the ID_L and SE_R agents during training. For clarity, the lines on the graph show the moving averages of R_{ID} and R_{SE} over the last 10 matches. The learning curves of ID_R and SE_L look similarly and are therefore omitted. Note that performance of SE_R starts to deteriorate slightly around 4 million frames. We observe a similar behavior when training SE_L. Hence, in the following experiments we froze the weights calculated at 4.2M frames for SE_L, 4.15M for SE_R, and 10M for both ID agents. Different way to visualize the training of the agent is to plot the evolution of its score against all the possible reward values as training progresses. As an example, we show that for ID_R agent in Figure 3.4. One can note the positive progress of training, as the

[1] To see the full code and the videos showing the performance of agents, go to https://git.io/vbT1v.

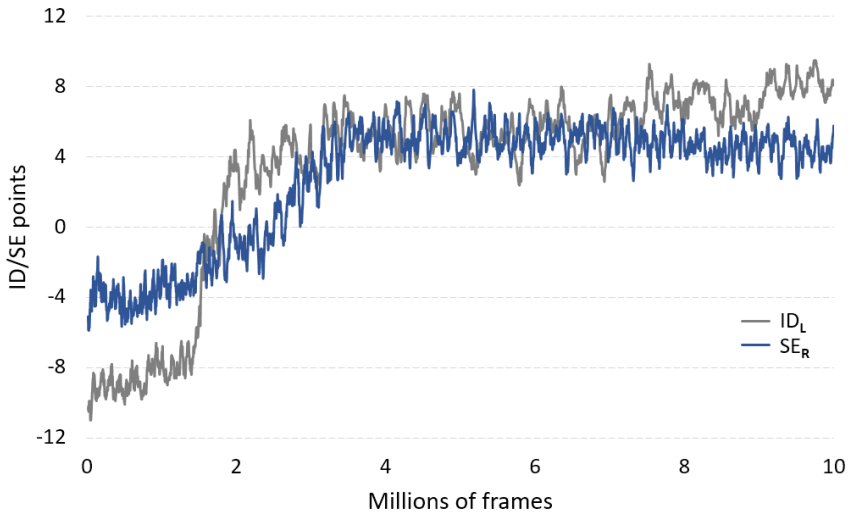

Figure 3.5: Training ID_L and SE_R agents against hand-coded AI for 10 million frames.

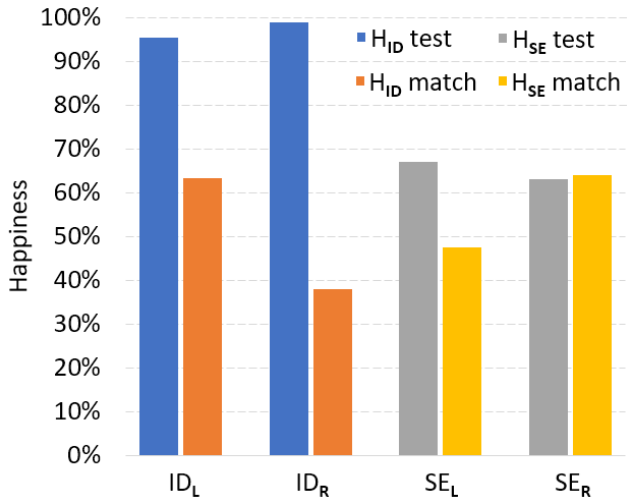

Figure 3.6: Lower happiness score during testing correlates with smaller drop in happiness in a society.

subsequent outcomes of each match are closer and closer to the maximum value for the agent (the top of the graph).

3.5.2 Testing phase

In the test phase, we set the epsilon value to 0, and run each model for 1000 matches against the hand-coded AI (see Table 3.1). One can note, that ID agents learned to win every single time (% won column in Table 3.1), with ID_R agent beating the hand-coded opponent by a larger number of points than ID_L, as can be seen by its higher average R_{ID} value. SE_L agent did not learn to win consistently, but it achieves the highest score among all the agents on its core objective - maximizing the R_{SE} value. The numbers indicate that r_{SE} is a harder signal to learn, as both

Table 3.1: Performance of agents during 1000 test games against hand-coded AI.

Agent	Average score	% won	Average R_{ID}	Average R_{SE}
ID_L	11.00	100	9.99	2.48
ID_R	11.00	100	10.76	3.03
SE_L	8.84	16	-1.69	5.09
SE_R	10.78	91	4.08	4.41

Table 3.2: Summary statistics of 100 matches between agents.

Match	Average score L	Average score R	% won L	% won R
ID_L vs. ID_R	10.80	5.29	91	9
SE_L vs. SE_R	3.50	10.93	3	97
ID_L vs. SE_R	9.44	8.96	51	49
SE_L vs. ID_R	8.92	9.05	40	60

SE agents did not get as close to their potential maximum ($R_{SE} = 10.5$, $R_{ID} = 1$) as both ID agents. The visual confirmation of this is presented in Fig. 3.3, where the average values of R_{SE} and R_{ID} obtained by the agents during testing are plotted against all the possible scenarios. Indeed, the averages of both ID agents are closer to their respective maximum (the green diamond at the top), than those for the SE agents (the rightmost blue diamond). We find that r_{ID} is easier to learn as the same actions lead to the same rewards. The SE agent has a more difficult function to learn, as the same actions result in different reward depending on the score. ID rewards are independent of the score.

Subsequently, the 4 trained agents played 100 matches against each other (see Table 3.2). The ID_L agent - which performed worse than ID_R during testing - won 91% of the games against its opponent, and also beat SE_R by a small margin. SE_R also achieved lower R_{SE} than SE_L in testing, yet it greatly outperformed SE_L in a series of matches. Finally, happiness statistics (see Figure 3.6) further indicate, that agents achieving lower happiness on their respective objective during testing, show higher happiness when interacting with other agents during the matches. This suggests that achieving high performance on the given reward objective against the hand-coded Pong AI is not enough to generate a resilient strategy. Agents may need more variability during training to better respond to the changing environment later on.

3.6 Conclusions

In this chapter we describe the computational model of personality based on Freudian psychoanalysis and DQN, and define "happiness" of an agent. Through the experiments, we show that the agents that are less aligned with their intended objective after the training period, exhibit more alignment when interacting with other agents. Here, we acknowledge the weaknesses of this work and mention possible further improvements. Firstly, due to the nature of the DQN algorithm, it is impossible to compare our work with other computational models of personality as was done in [77]. Secondly, this study relies on the psychoanalytic theory of personality. However, it appears that extending this work to the FFM could further improve the ability of the model to capture human personality. Lastly, in order to calculate the happiness, one needs to know the minimum and maximum values of R_{SE} and R_{ID}, which may be impossible in more complex environments. We also note, that we see the potential for extending this work to multi-agent settings with individual personalities in computer games (e.g., [60, 5]) or even for using this approach to study life changing events being either a result of one big "reward" (e.g., from Holmes and Rahe stress scale [33]) or a summation of many events with smaller "rewards." Furthermore, experiments presented in this chapter suggest that the algorithms with hand-coded reward signals do not always learn the desired behaviour, and that there is no structured approach for telling how the agent is going to perform in new situations.

Chapter 4

Learning to Win, Lose, and Cooperate through Reward Signal Evolution

4.1 Summary

The problem of designing an optimal reward signal for a Deep Reinforcement Learning (DRL) algorithm has been gaining some popularity as we are facing a point when some DRL systems may enter our daily lives at a big scale in the next few years, as in the case of self-driving cars [38]. Despite this practical advances, we still do not have many theoretical guarantees predicting the performance of such systems, and we see more and more examples in the literature of DRL systems that learned something different than what the person who built it had initially intended [6]. In this chapter we concentrate on the problem of exploring the space of possible reward signals in a DRL setting. We propose an evolutionary algorithm for optimizing the reward signal at each timestep of agents' interaction with the environment. We show that it is possible to use goal functions as fitness functions evaluating the quality of a given solution. Through experiments, we demonstrate that the algorithm can work in the domain of the game of Pong. Moreover, the algorithm leads to a simultaneous emergence of a population of agents learning complex behaviours of winning, losing, and cooperating. Some of the solutions found by the algorithm are surprising, in the sense that they would probably have not been chosen by a person trying to hand-code a given behaviour through a specific reward signal. We hope that the algorithm presented in the chapter will spark more research in this area and ultimately lead to the development of safer, more predictable systems.

4.2 Introduction

The reward signal plays a defining role in reinforcement learning. After all, the most important goal of the agent is to maximize a cumulative reward over the long run, based on the immediate reward it receives at each timestep from the environment (see Chapter 2). Consequently, the reward is the main source of information for the algorithm allowing it to decide whether a given interaction within its setting was positive or not. In a way, the reward plays a similar role for an agent, as the biological sensations of pain, pleasure, etc. do for humans [78].

What is more, the reward is also the main driver behind changing the agents' policy π - actions followed by low or negative rewards may be replaced by other actions later on. This defines the immediate role of the reward signal. Over the long run, the agent decides what is good through its value function. The value function indicates what cumulative reward an agent can expect by starting from a given state. As a result, a value function can allow us to sacrifice the immediate reward and steer us towards future states that yield a higher cumulative reward overall. This forms an interplay of sorts, between these two concepts. The algorithm starts in an unknown environment, sampling the reward signal in a way by performing the actions, uses that information to form a value function, only to use it to increase the total reward received. Yet, the rewards guide the process of learning over the long run and they are harder to estimate, compared to the rewards, which are just given by the environment. As a result, most of the focus in RL research has been on efficiently determining the value function. In fact, [78] note that "[t]he central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades."

However, in this chapter of the thesis we want to focus precisely on the reward, not the value function. The main reason for that comes from the observation in many practical applications that the choice of the reward signal strongly influences the results, and there are more and more examples in the literature highlighting undesired behaviour of the RL algorithms with misspecified rewards - even when the intentions of the designer of the given system seemed intuitive. We give examples of such studies below, and also in the following section.

Typically, each RL algorithm is specified with some goal function in mind. Oftentimes, in simple environments, it is straightforward to translate the desired intention (our goal) into the reward signal. Yet, one might argue that as we move towards more difficult environments, especially those that modern deep reinforcement learning tries to study, this process can quickly become unpredictable and lead to surprising results [47]. This situation is often exacerbated by the the problem of sparse rewards. To make things worse, RL algorithms have been shown to be able to

"hack" their own goal functions to deliver the reward in unintended and unexpected ways [6].

Hence, it seems that the reward signal is an important part of any RL system, and it is worth investigating it in greater detail. One might say, that typically a reward signal is hand-designed to reflect the goal its creator had in mind. In practice, this usually comes down to trying many variants of the settings, and observing the process of algorithm's improvement on the goal along the way. To conclude, [78] state that "[s]ome more sophisticated ways to find good reward signals have been proposed, but the subject has interesting and relatively unexplored dimensions."

When deciding on the choice of the reward signal, do we assume one unique reward signal to achieve a given goal? Can the optimal goal choice depend on a specific moment at which we stop the training phase for a given reward signal? In this chapter we approach designing the reward through an evolutionary scheme in a highly dimensional environment of the game of Pong to try and find answers to these questions.

4.3 Related work

We present the literature connected to our work by looking through the following lenses: approaches to learning a reward signal and evolutionary methods, with the focus on deep reinforcement learning.

4.3.1 Reward learning

Over the years, many approaches to learning a reward have been proposed. *Inverse RL* [50] has been introduced to investigate the setting in which we do not know what the reward should be, but we can observe an expert performing the task, and can try inferring the reward signal from the demonstrations. As a result we can receive an approximation of the reward, and can try and improve the performance of the agent on the task based on it. Generally, these methods of learning the reward can also be known as "imitation learning," "learning from demonstration," or "apprenticeship learning."

Inverse reward design (IRD) [30] is based on the observation that the specified reward function can be treated as approximation of what the designer actually had in mind. As a result, the reward should be interpreted in the specific context in which it was created. IRD uses approximate methods for solving the IRD problem, and the experiments show that it can help lessen the problems of both misspecified rewards and reward hacking.

Another approach based on learning a reward function from *human preferences*

is [15]. Learning is achieved by comparing pairs of trajectory segments, and has been shown to work in the complex DRL setting of Atari games and simulated robot locomotion.

Yet another dimension of looking at devising the reward signal, is that of deciding whether the agent should learn some specific task in the environment in which it happened to find itself, or should it try to learn to optimize some non-task-specific reward. This approach is called *intrinsic motivation*, and can be seen as doing something "for its own sake" [78]. A prominent example is [71] where the agent's reward is a function of how quickly its environment model is improving as far as predicting state transitions is concerned. A recent example of training a single agent using a non-task-specific reward scheme in a DRL setting is presented in [59].

Another approach of designing a reward is by using the experience gained by the agent during its lifetime. The approach is approximating the optimal reward online by gradient ascent [76] and generalizes the standard policy gradient method. The approach has been extended to learn non-linear reward-bonus function parameters in [29], to learn intrinsic motivation [82], and [81] approach learning return function's parameters through a gradient-based meta-learning algorithm.

Lastly, the designer can also carefully engineer a reward function such that it gives the agent intermediate reward for progressing towards the desired goal. This approach is known as *reward shaping* [49]. Potential-based reward shaping adds a reward for moving from state s to s', based on the difference in potential functions of the given states [49]. The approach has been extended to plan-based reward shaping [28], hierarchical reinforcement learning [24], and multi-agent RL [18, 19]. Recently, the approach has been further extended to use natural language instructions as the reward shaping machanism [26]. Reward shaping can still be a difficult and time-consuming endeavour [26].

Reward evolution

Perhaps the most closely related works to that presented in this chapter, are those concentrating on the evolution of the reward. The observation leading to this approach can be the fact, that the reward can be seen as one of the variables in the general RL setting. As a result, it can be optimized, and evolutionary algorithms are often a good choice for any optimization task. The proposed solution can then be evaluated against the designer's goal, and the worst performing solutions eliminated, leading to an improved result.

To the best of our knowledge, the work presented in [74] is the closest in spirit to ours. The authors approach designing a reward through an evolutionary algorithm. The paper formulates an optimal reward function given a fitness function and some distribution of environments. Through the experiments, the rewards are discovered

through exhaustive search rather than prespecified. The authors find that the reward found through this process can differ from the one intended by the fitness function, but can still be advantageous.

Our work differs in the sense that we present the evolution of the reward in a complex domain of a Pong game, in an environment with an opponent present. Whereas [74] concentrate on experiments in a setting with predefined features to define the combinatorial space of reward functions, we consider the mutations of the reward at each timestep. We also pay more attention to the subject of overfitting and possible role of evolutionary algorithms in countering it. The work of [74] presents an emergence of reward functions, our approach additionally shows an emergence of complex behaviours of an agent.

The work of [74] is discussed further in [75], however the main focus of that work is the problem of intrinsic versus extrinsic motivation. More recently, [27] extend the framework presented in [74] to the space of reward functions spanned by a given set of feature states and multi-objective evaluation function.

In [51, 53, 52], the authors used Genetic Programming in order to evolve a population of reward functions. In order to search for reward functions, the search is preformed over the space of programmes generating the rewards. The approach is proposed as a method to "alleviate the difficulty of scaling reward function search" and replace the exhaustive search used in [74].

The authors of [3] concentrate on using evolutionary computation to evolve a population of agents behaviour, allowing them to inherit the rewards as a secondary optimization process. The rewards are mutated, hence the fitness can increase over time. There are two neural networks involved. First evaluation network is fixed and uses inherited weights to produce a scalar reward. Second - action network - is also inherited, but trainable after initialization.

4.3.2 Evolutionary methods in deep reinforcement learning

Lastly, we broadly describe the context of using evolutionary methods specifically in a deep reinforcement learning setting.

Perhaps the first study to present the applicability of evolutionary methods in the context of complex environments was [70]. It showed that using evolutionary strategies and enough computational power was enough to learn to perform well in the complex domain of Atari games. The research mixing the worlds of deep reinforcement learning and evolution has quickly followed. Ideas from evolutionary computation have been mostly applied by creating populations of agents which undergo mutations and increase the diversity of strategies created for the agents. Then, when agents interact, the process of elimination guarantees high quality of

the resulting solution. Such an approach was part of an algorithm mastering a difficult domain of Starcraft II [80], and the evolutionary aspects of that system have been investigated in [7]. Another study drawing on ideas from evolution in the DRL setting is [34] with an objective of "the emergence of complex cooperative agents." The goal is achieved by applying a variant of population-based training [35] to DRL agents.

Recently, the reward search has been automated by adding an additional, evolutionary layer over standard reinforcement learning layer, to find the reward that maximizes the task objective (through proxy rewards [14] - parameterized versions of the standard environment rewards) in a continuous control task [21].

To the best of our knowledge, ours is the first study to concentrate specifically on the problem of learning a reward signal in a complex, multi-agent setting through an evolutionary algorithm without any prespecified features used as combinations to mutate the reward function. We create a population of agents by training them separately on different mutations of the reward signal. The reward signal is not prespecified, but its evolution allows us to achieve a set of complex and diverse behaviours.

4.4 Reward signal evolution

In this section we concentrate on the details of our approach. If we think about the interplay between the number of reward signals and goal functions, we can distinguish the following three main situations.

1. 1 to 1
2. n to 1
3. n to N

Usually, in RL we have a situation in which there is one predefined reward signal, designed with one assumed goal in mind - we call it the 1 to 1 case. One can also have a situation, when many reward signals (n) can be explored to find the one best optimizing our goal (n to 1). The last combination is the one in which we have n reward signal candidates, and use them to optimize for many different goals N (n to N). The Algorithm 2 presented in this section can be used to cover these cases, and can be seen as a general framework of sorts.

Concretely, the proposed approach (see Algorithm 2) requires us to choose our learning algorithm, 1 or many goal functions we are interested in (can be seen as fitness functions), and 1 or many unique random reward sequences (here, the goal is

Algorithm 2 Reward signal evolution algorithm

initialize learning algorithm A
initialize i goal functions $G_i(r)$
initialize j unique random reward sequences r_j
store $\mathbf{r} = \{i \| r_j\}_{i \times j}$
burn-in: train A on r_j for M timesteps
repeat
    train $A(r_j)$ until M timesteps
    test $A(r_j)$ for m timesteps
    calculate $G_i(r_j)$ on test data
    for i **do**
        remove $argmin\bar{G}_i(r_j)$ from **r**
        add p unique random mutations of r_j to **r**
    end for
    $M := M + M$
until $convergence$

a function representing our intended behaviour of the agent, whereas the reward is the instantaneous scalar quantity received by the agent from the environment at each timestep). The algorithm of choice is then trained on the initial reward signals, and its performance on the specified goal functions is checked periodically. At that point two things happen - firstly, we eliminate the rewards leading to the worst result for each of the goals, and secondly, we add the required number of new reward signal mutations to the population. In the example provided in Section 4.5 the convergence means that we have eliminated all the possible reward signal mutations, and are left with one winner per each goal. In general, in many examples one could create almost infinitely many reward signals. That could be seen as a *mutation* phase. Once you do not mutate any new reward signals, the algorithm enters an *elimination* phase, until we are left with top k solutions that we require.

4.4.1 Choosing r_j

Lastly, we distinguish the following ways to choose r_j in Algorithm 2:

- **Fixed** - the typical way for setting a reward signal in RL
- **Semi-evolutionary** - the algorithm is free to mutate the reward for the set of rules specified by the expert (e.g. position of the ball in a pong game, or for interacting with specific elements of the environment)
- **Evolutionary** - the algorithm is free to mutate the reward at any timestep t

Finally, we note that the evolutionary approach for choosing r_j can be seen as a *reward sampling scheme*, sampling from the distribution of all possible reward

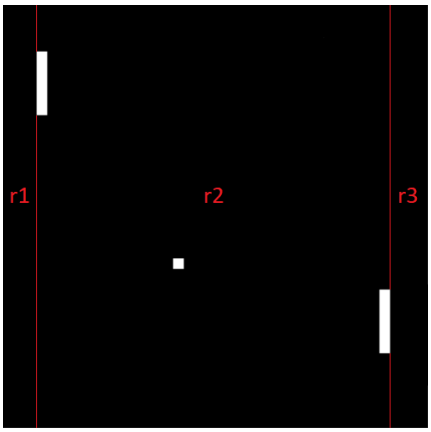

Figure 4.1: The Pong game screen showing the regions in which the reward signal can be mutated.

signals.

4.5 Experiment

4.5.1 Set-up

To illustrate how Algorithm 2 could work in practice, we chose the Pong game environment, as shown in Figure 4.1.

In all experiments we concentrate on controlling the paddle on the right side of the screen. We use DQN [46] as the base algorithm A in Algorithm 2. The architecture of the DQN used is presented in Figure 2.6. Typically, RL algorithms have one reward function to optimize for. Here, we give the algorithm 3 reward functions ($i = 3$). Concretely, the goals are:

- **Winning**, maximizing the cumulative number of points scored by the agent ($G_1(r)$)
- **Losing**, maximizing the cumulative number of points lost by the agent ($G_2(r)$)
- **Cooperation**, maximizing the time spent playing until the point is lost by either of the players ($G_3(r)$)

We use the semi-evolutionary way to choose r_j by introducing *mutation regions* to lower the computational cost of the algorithm and have a better intuition and control of the obtained results. To generate the reward sequences, r_j, the algorithm can choose from the values of either 0 or 1, in any of the reward regions $r1$, $r2$, $r3$ depicted in Figure 4.1. That means that there is one unique reward signal when the ball passes either of the paddles, and another one when the ball is between the paddles. The three regions, combined with a 0 or 1 reward signal in each of them,

yield a total of 8 unique reward signals that the algorithm can discover through mutations. We set the initial value of j to 3. Furthermore, we initialize the parameters M and m to have the values of 1 million and 100,000 respectively. That means that we perform the elimination and mutation step of the algorithm after each 1 million timesteps, based on the test results from further 100,000 timesteps. Lastly, after each elimination, we add two new unique reward signals to the population ($p = 2$).

4.5.2 Results

First, we randomly generated 3 reward signals, and obtained the following values of r_j: $r_1 = 000$, $r_2 = 001$, and $r_3 = 011$. As an example, $r_2 = 001$ corresponds to a reward signal of 1 when the ball passes the paddle on the right, and 0 otherwise (see Figure 4.1). The DQN algorithm was trained on each of the reward signals r_j for 1 million timesteps, as a burn-in phase. Now, the algorithm entered its main loop. After the next 1 million timesteps of training, each $A(r_j)$ was tested for 100,000 timesteps. The values of $G_i(r_j)$ were calculated, and the reward signals corresponding to the lowest value for each goal were eliminated from **r**. Lastly, two unique reward signal mutations were added: $r_4 = 101$ and $r_5 = 110$. Hence, the state of **r** after the first iteration is given by:

$$\mathbf{r} = \left\{ \begin{array}{ccc|cc} 1\| \cancel{000} & 001 & 011 & 101 & 110 \\ 2\| 000 & 001 & \cancel{011} & 101 & 110 \\ 3\| \cancel{000} & 001 & 011 & 101 & 110 \end{array} \right\} \tag{4.1}$$

The algorithm continued to run for six more iterations. We present the steps of that process in a graphical form in Figures 4.2, 4.3, 4.4. The figures capture the creation of each new mutation of the reward signal, its score for a given goal, and the reward signals eliminated along the way. Ties (in case when different reward signals led to the same outcome for a given goal) were decided at random.

Additionally, we present the details of each of the iterations in a matrix form, to show the outcome of each loop in more detail:

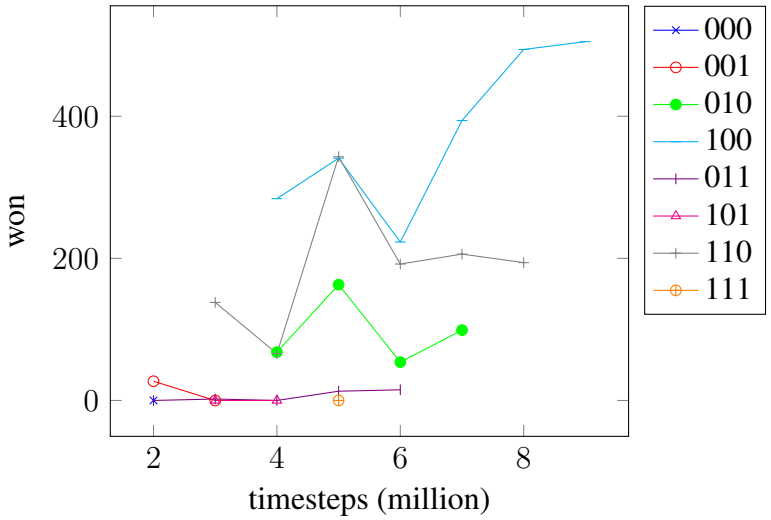

Figure 4.2: Survival and fitness of a given signal when the goal is **winning**.

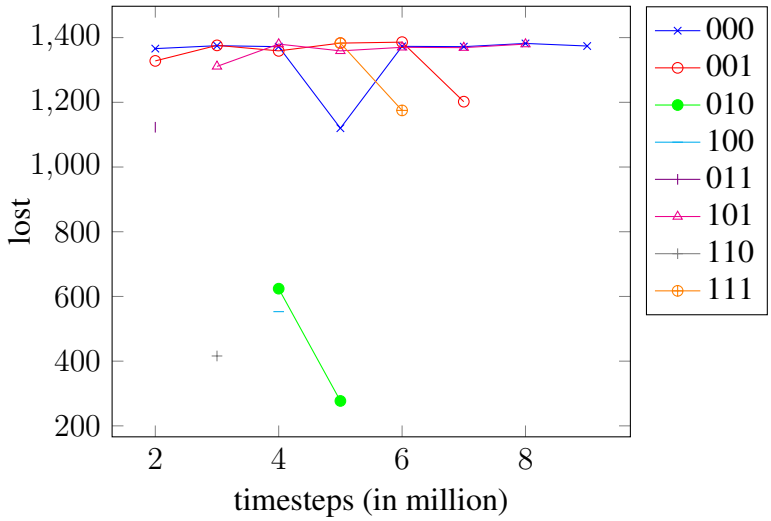

Figure 4.3: Survival and fitness of a given signal when the goal is **losing**.

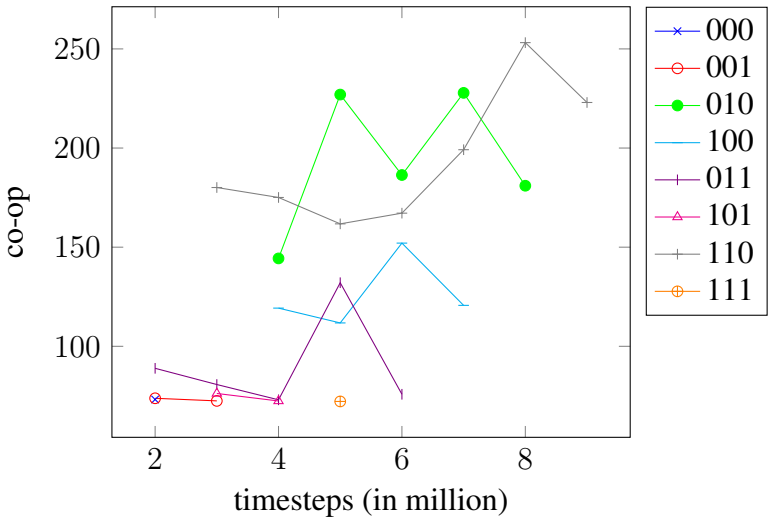

Figure 4.4: Survival and fitness of a given signal when the goal is **cooperation**.

$$
\begin{aligned}
\mathbf{r} = &\left\{\begin{array}{llll|ll} 1\|\cancel{000} & 001 & 011 & & 101 & 110 \\ 2\|000 & 001 & \cancel{011} & & 101 & 110 \\ 3\|\cancel{000} & 001 & 011 & & 101 & 110 \end{array}\right\} \\
\xrightarrow{3M} &\left\{\begin{array}{llllll|ll} 1\|\cancel{001} & 011 & 101 & 110 & & & 010 & 100 \\ 2\|000 & 001 & 101 & \cancel{110} & & & 010 & 100 \\ 3\|\cancel{001} & 011 & 101 & 110 & & & 010 & 100 \end{array}\right\} \\
\xrightarrow{4M} &\left\{\begin{array}{lllll|l} 1\|011 & \cancel{101} & 110 & 010 & 100 & 111 \\ 2\|000 & 001 & 101 & 010 & \cancel{100} & 111 \\ 3\|011 & \cancel{101} & 110 & 010 & 100 & 111 \end{array}\right\} \\
\xrightarrow{5M} &\left\{\begin{array}{lllll} 1\|011 & 110 & 010 & 100 & \cancel{111} \\ 2\|000 & 001 & 101 & \cancel{010} & 111 \\ 3\|011 & 110 & 010 & 100 & \cancel{111} \end{array}\right\} \\
\xrightarrow{6M} &\left\{\begin{array}{llll} 1\|\cancel{011} & 110 & 010 & 100 \\ 2\|000 & 001 & 101 & \cancel{111} \\ 3\|\cancel{011} & 110 & 010 & 100 \end{array}\right\} \\
\xrightarrow{7M} &\left\{\begin{array}{lll} 1\|110 & \cancel{010} & 100 \\ 2\|000 & \cancel{001} & 101 \\ 3\|110 & 010 & \cancel{100} \end{array}\right\} \\
\xrightarrow{8M} &\left\{\begin{array}{ll} 1\|\cancel{110} & 100 \\ 2\|000 & \cancel{101} \\ 3\|110 & \cancel{010} \end{array}\right\}
\end{aligned}
\tag{4.2}
$$

Eventually, the surviving reward combinations for each of the goals were:

- **Winning**: $r_7 = 100$
- **Losing**: $r_1 = 000$
- **Cooperation**: $r_5 = 110$

4.5.3 Sensitivity analysis and comparisons

In this section, we concentrate on analysing the sensitivity of the results presented in the previous section and comparisons with baselines.

Table 4.1 shows the intermediate score on each of the goals for each of the reward signals every 1 million timesteps. This allows us to evaluate the sensitivity of the presented solutions, as we can investigate what could have happened, had the algorithm made different choices along the way. The table also includes the

scores on the said 1 million checkpoints for three baseline DQN algorithms ('b100', 'b010', 'b001' in Table 4.1) and a random play baseline (rand in Table 4.1). The random play baseline takes random actions at each timestep. With equal probability it can move up, down, or stay it the same place. The baseline DQN algorithms have the same architecture as the DQN used in Algorithm 2, however the reward signals on which they learn differ. The rewards for the baseline DQNs are based on the score. This means that 'b100' receives a reward of 1 only when it scores a point, 'b001' when it loses a point, and 'b010' receives a reward of 1 as long as neither of the players scores. The baseline algorithms 'b100', 'b010', 'b001' are closest - it would seem - to the semi-evolved algorithms that used '100', '010', and '001' as their respective reward signals. The main difference is in the frequency of the timeframes for which the algorithm receives the reward.

The results of the random play, as shown in Table 4.1 are rather consistent as far as all the three proposed goal functions are concerned.

Learning to lose

Losing every single point seems to be the easiest goal to learn, as expected. Semi-evolved rewards '000' and '111' did not lead to any significant learning, even over 9 million timesteps of training. The final results on all the goals for the said reward signals where close to that of '011' and '101', which proved too confusing for the DQN to learn based on the same pixel input. The reward signal '001' led, as expected, to a very good result on the goal of losing the maximum number of points, but its performance was close on that goal to many other reward signals that just proved too difficult to learn, and similar to the baseline score 'b001'.

Learning to win

Three out of the eight semi-evolved reward signals led to an algorithm that was able to score more points than random play. Namely, '010', '100', '110'. It seems that as long as the reward is given for keeping the ball between the paddles, and there is no incentive to lose - the signal is good enough to learn to score some points. Interestingly, the signal '110' led to the second best performing algorithm on the goal concerning scoring as many points as possible. It was only toped by perhaps the "obvious" reward signal choice of '100'. The baseline algorithm 'b100' proved better on average than '100' across the 8 test checkpoints, however it did suffer from less consistency, and even got stuck in a local extremum at 3 million timeframes (the algorithm kept the paddle in the lower part of the screen and only occasionally managed to hit the ball). It seems that the performance of the algorithm trained on the '100' signal was more consistently improving on its goal, and achieved the highest

Table 4.1: Performance of DQN on reward signals with no elimination.

		2M			3M			4M	
r_j	won	lost	co-op	won	lost	co-op	won	lost	co-op
000	0	1366	73.17	0	1375	72.66	0	1372	72.84
001	27	1328	73.78	0	1376	72.47	0	1359	73.54
010	79	624	142.06	79	541	160.97	68	624	144.33
100	312	586	111.05	141	523	150.04	284	553	119.26
011	0	1123	88.90	2	1235	80.72	0	1369	73.01
101	0	1380	72.42	0	1311	76.21	0	1380	72.42
110	137	579	139.36	138	416	180.09	66	504	175.07
111	0	1275	78.41	2	1369	72.90	1	1385	72.11
b100	62	773	119.54	0	1311	76.24	256	895	86.64
b010	152	855	99.18	244	344	169.92	110	644	132.37
b001	0	1378	72.50	0	1378	72.47	0	1392	71.92
rand	62	997	94.24	75	965	96.07	62	1006	93.46
		5M			6M			7M	
r_j	won	lost	co-op	won	lost	co-op	won	lost	co-op
000	3	1120	88.84	0	1373	72.68	0	1372	72.79
001	1	1383	72.14	0	1386	72.10	51	1202	79.72
010	163	277	226.97	54	481	186.40	99	339	227.83
100	341	552	111.79	223	431	152.05	394	434	120.65
011	13	743	132.07	15	1302	75.81	0	1310	76.23
101	0	1359	73.51	0	1370	72.92	0	1369	72.90
110	343	272	161.76	192	405	167.16	206	294	199.16
111	0	1383	72.19	41	1175	82.08	0	1231	81.20
b100	433	304	135.44	550	355	110.37	577	267	118.3
b010	262	412	148.15	64	547	163.49	102	509	163.11
b001	0	1376	72.57	0	1373	72.78	0	1384	72.21
rand	72	990	94.06	49	986	96.54	68	933	99.84
		8M			9M			Avg.	
r_j	won	lost	co-op	won	lost	co-op	won	lost	co-op
000	0	1382	72.27	0	1374	72.70	0	1342	74.74
001	0	1375	72.62	0	1385	72.14	10	1349	73.56
010	48	503	180.99	98	338	228.74	86	466	187.29
100	494	499	100.44	505	385	112.05	337	495	122.17
011	2	1381	72.23	0	1382	72.23	4	1231	83.90
101	0	1380	72.40	0	1382	72.33	0	1366	73.14
110	194	200	253.17	161	286	222.98	180	370	187.34
111	0	1370	72.94	0	1381	72.38	6	1321	75.53
b100	528	289	122.09	482	289	129.45	361	560	112.26
b010	140	331	211.9	83	559	155.55	145	525	155.46
b001	0	1392	71.76	0	1382	72.31	0	1382	72.32
rand	58	999	94.49	65	975	96.07	64	981	95.60

score at 9 million timesteps, with the highest score on G_1 of all the algorithms. It seems it was also learning faster than baseline in the beginning.

Learning to cooperate

As far as cooperation is concerned, two reward signals led to a very similar average performance for that goal. Concretely, running an algorithm to learn on the reward signal '010' and '110' would give comparable performance. We find it interesting, as we would argue that '110' is not an obvious choice to set-up a priori with cooperation in mind. Both reward signals led to a better cooperation than the baseline 'b010'. It is worth noting, that not only did the reward signal '110' lead to the highest average cooperation, but also to the third highest score on the "winning" goal (including the baseline 'b100').

Finally, Algorithm 2 led to the following choice of reward signals: '100', '000', '110' for G_1, G_2 and G_3, respectively. The results are sensitive mostly with respect to G_2, as many reward signals led to losing many points quickly. The reward signal '100' led to a clearly superior performance on G_1 quickly, and only the signals '010' and '110' led to a high performance on the goal of cooperation.

For clarity, we summarize the main results of the experiment in the list below:

- Algorithm 2 proved to work on highly dimensional input space of pixels
- many rewards can lead to a good performance of the "losing" goal
- fewer reward signals lead to learning to cooperate
- the goal of "winning" proved the most difficult to learn, with the smallest number of reward signals leading to the desired outcome
- our experiments give further evidence of the fact that there exist multiple reward signals to learn a given goal
- interestingly, the performance during training of the baseline 'b100' seemed much less stable and predictable than for the algorithm trained on the reward signal '100', possibly due to the frequency of timesteps for which each algorithm "experienced" the reward
- lastly, the sensitivity of the results shows that it is not obvious at which point the training phase should stop, in order to have the agent performing at its best with respect to the goal.

4.6 Conclusions

The problem of reward signal design is an interesting open area of research in reinforcement learning, especially as we move from simple toy examples, into highly dimensional input spaces of video games, and beyond. That increase in complexity requires us to spend more time considering the problems of AI safety, as we are going to rely on this systems more and more, but still have little understanding of what they actually learn, and how predictable these systems are.

In this chapter we have investigated the role of the reward signal in a deep reinforcement learning setting. We proposed an algorithm based on evolutionary computation to search through the space of possible reward signals in the game of Pong. We showed that it is possible to use the proposed algorithm in order to learn complex behaviours of winning, cooperation and losing by specifying high level goal functions, but without providing a concrete reward signal for any of them. The reward signals were discovered from scratch, and allowed us to investigate their role in the proposed system. We found that often there is no unique reward signal to achieve a certain level of performance on a given goal. The number of reward signals leading to a given goal also varies, depending on the complexity of the goal.

The algorithm we propose is highly adaptable, i.e. the designer can change the algorithm used, modify the goals, change the statistic used as the fitness function, control the number of mutations, etc. We hope that the results could lead to a better understanding of such systems, and eventually help us create safer systems, more aligned with our intended goals.

Chapter 5

Conclusions and Future Work

In order to conduct a thorough investigation of the role of the reward signal in deep reinforcement learning, we began by introducing the necessary concepts from the field of machine learning, with the focus on reinforcement learning, in which the reward signal plays a key role. Additionally, we described the ideas from a powerful supervised learning technique called deep learning, which led to the development of deep reinforcement learning. Finally, we also included a discussion of the role of evolutionary computation in the field or reinforcement learning, and its usage as a component in designing reward signals.

Through our first study, we have shown that the a priori specification of the reward signal, which seems "obvious" to the designer, can lead to an unexpected behaviour of agents trained on it when the said agents can interact with each other. Namely, the agents trained on different rewards in the game of Pong did not translate their intended behaviour from training with a hand-coded AI opponent to a situation when the agents were playing against each other. For example, training the agent to win every single point did not result in an agent winning with another agent that was trained to win only slightly ahead of the opponent. The results suggested the problem of overfitting to the environment and the opponent in a highly-dimensional setting of the game.

In the second study we investigated the behaviour of agents trained on different reward signals even further, by introducing a flexible evolutionary reward signal algorithm, with the aim of sampling from the space of all possible reward signals. The results obtained from the experiments suggest, that it is possible to evolve a range of reward signals, which when combined with high-level goals, can lead to the emergence of complex behaviours of the agents, such as winning, losing, and cooperating. Furthermore, the results can be seen as another piece of evidence towards the fact, that many reward signal specifications can be used to achieve a given goal. Additionally, it seems that the more complex the goal is, the smaller the number of reward signals leading to the desired outcome. Lastly, the results are

consistent with our previous study - and other studies in the literature (presented in Chapter 4) - in terms of the fact that it is not always obvious how to choose the reward signal a priori, in order to achieve the behaviour intended by the designer.

There are many possible directions in which further work based on this results can go. They can be broadly categorized into the extensions of the algorithm for reward signal evolution presented in the last chapter, and joining it with other results from the literature to explore adjacent topics. Firstly, the proposed algorithm allows for the mutation of the reward at each timestep. This seems computationally quite expensive, but would be worth investigating further (at least by relaxing the number of reward regions used), how the algorithm behaves in such a setting, in even more complex environments with longer episodes. Sampling a reward from a continuous, rather than discreet set of values needs more research as well. Additionally, it would be interesting to research the possibility of sending "good" reward signal sequences to the environment, and trying to learn those that result in consistent, predictable behaviour. Finally, we propose that measuring the complexity of the goal in relation to the number of reward signals that lead to learning it in a satisfactory manner is also a potential research direction.

In the context of the broader literature, we see the potential of joining our approach with the work in deep reinforcement learning that uses population-based training to achieve a diverse set of strategies of the resulting agents. Evaluating the performance of agents has received relatively little attention, but there are studies emerging that challenge the classical train-test split [9, 56], especially as the policies learned using independent reinforcement learning have been shown to overfit to the other agents' policies during training [40, 47]. It would be interesting to investigate the role that the reward signal evolution could play in those problems, as currently most of the focus is on evolving the agents to increase the diversity of the solution. In other words, it seems to be the case that using population-based training results in less overfitting and better solutions. Hence, we argue that joining reward evolution with population-based training could improve such solutions to an even bigger degree. Lastly, our work can be seen as an example of a quality and diversity optimization [16], and perhaps new measures of diversity of the population of agents can be of benefit to the broader field as well.

In general, the thesis indicates that there is a need for more work on the theory of reinforcement learning, especially in the context of AI safety of the systems based on deep reinforcement learning. As we outsource more and more of our important daily tasks to these algorithms, it is vital that we can reliably and with high degree of confidence predict their behaviour. The results presented in this thesis and in the literature indicate that this goal is possible to achieve. Hopefully, in the not too distant future we can all benefit from such safe systems.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yaser S Abu-Mostafa. *Learning from data*, volume 4. AMLBook, 2012.

[3] David Ackley and Michael Littman. Interactions between learning and evolution. 1991.

[4] Sebastian Ahrndt, Johannes Fähndrich, and Sahin Albayrak. Modelling of personality in agents: from psychology to implementation. *Proceedings of the HAIDM*, pages 1–16, 2015.

[5] Christopher Amato and Guy Shani. High-level reinforcement learning in strategy games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 75–82. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[6] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[7] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. *arXiv preprint arXiv:1902.01724*, 2019.

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[9] David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. Re-evaluating evaluation. In *Advances in Neural Information Processing Systems*, pages 3272–3283, 2018.

[10] Tina Balke and Nigel Gilbert. How do agents make decisions? a survey. *Journal of Artificial Societies and Social Simulation*, 17(4):13, 2014.

[11] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[13] Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. Evolutionary dynamics of multi-agent learning: A survey. 2015.

[14] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end to end. *arXiv preprint arXiv:1809.10124*, 2018.

[15] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[16] Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018.

[17] Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[18] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 225–232. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[19] Sam Devlin and Daniel Kudenko. Plan-based reward shaping for multi-agent reinforcement learning. *The Knowledge Engineering Review*, 31(1):44–58, 2016.

[20] AE Eiben and James E Smith. Introduction to evolutionary computing. 2015.

[21] Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning. *arXiv preprint arXiv:1905.07628*, 2019.

[22] Sigmund Freud. *The ego and the id*. WW Norton & Company, 1962.

[23] Sigmund Freud. *Inhibitions, symptoms and anxiety*. WW Norton & Company, 1977.

[24] Yang Gao and Francesca Toni. Potential based reward shaping for hierarchical reinforcement learning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[26] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

[27] Ricardo Grunitzki, Bruno C da Silva, and Ana LC Bazzan. A flexible approach for designing optimal reward functions. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1559–1561. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[28] Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, volume 2, pages 10–22. IEEE, 2008.

[29] Xiaoxiao Guo, Satinder Singh, Richard Lewis, and Honglak Lee. Deep learning for reward design to improve monte carlo tree search in atari games. *arXiv preprint arXiv:1604.07095*, 2016.

[30] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

[31] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[32] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012.

[33] Thomas H Holmes and Richard H Rahe. The social readjustment rating scale. *Journal of psychosomatic research*, 11(2):213–218, 1967.

[34] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281*, 2018.

[35] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[36] Sara Karimi and Mohammad Reza Kangavari. A computational model of personality. *Procedia-Social and Behavioral Sciences*, 32:184–196, 2012.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Will Knight. Reinforcement learning: 10 breakthrough technologies 2017. *MIT Technology Review*, 120(2), 2017.

[39] Sumedha Kshirsagar. A multilayer personality model. In *Proceedings of the 2nd international symposium on Smart graphics*, pages 107–115. ACM, 2002.

[40] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Julien Perolat, David Silver, Thore Graepel, et al. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4193–4206, 2017.

[41] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[42] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[43] Yuxi Li. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*, 2018.

[44] Sonja Lyubomirsky and Heidi S Lepper. A measure of subjective happiness: Preliminary reliability and construct validation. *Social indicators research*, 46(2):137–155, 1999.

[45] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[47] Rafał Muszyński and Jun Wang. Happiness pursuit: Personality learning in a society of agents. *arXiv preprint arXiv:1711.11068*, 2017.

[48] Allen Newell. Unified theories of cognition, 1994.

[49] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping.

[50] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning.

[51] Scott Niekum. Evolved intrinsic reward functions for reinforcement learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[52] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

[53] Scott Niekum, Lee Spector, and Andrew Barto. Evolution of reward functions for reinforcement learning. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 177–178. ACM, 2011.

[54] Tohru Nitta. A computational model of personality'. *No Matter, Never Mind: Proceedings of Toward a Science of Consciousness: Fundamental approaches, Tokyo 1999*, 33:315, 2002.

[55] Tohru Nitta, Toshio Tanaka, Kenji Nishida, and Hiroaki Inayoshi. Modeling human mind. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 2, pages 342–347. IEEE, 1999.

[56] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. {\alpha}-rank: Multi-agent evaluation by evolution. *arXiv preprint arXiv:1903.01373*, 2019.

[57] World Health Organization. Infographics on global road safety 2013, Oct 2018. Accessed: 2019-07-20.

[58] Lin Padgham and Guy Taylor. A system for modelling agents having emotion and personality. In *Intelligent Agent Systems Theoretical and Practical Issues*, pages 59–71. Springer, 1997.

[59] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.

[60] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[61] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[62] Mike Poznanski and Paul Thagard. Changing personalities: towards realistic virtual characters. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(3):221–241, 2005.

[63] Michael Quek and DS Moskowitz. Testing neural network models of personality. *Journal of Research in Personality*, 41(3):700–706, 2007.

[64] Stephen J Read, Brian M Monroe, Aaron L Brownstein, Yu Yang, Gurveen Chopra, and Lynn C Miller. A neural network model of the structure and dynamics of human personality. *Psychological review*, 117(1):61, 2010.

[65] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[66] David E Rumelhart and Geoffrey E Hintonf. Learning representations by back-propagating errors. *NATURE*, 323:9, 1986.

[67] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[68] Robb B Rutledge, Archy O De Berker, Svenja Espenhahn, Peter Dayan, and Raymond J Dolan. The social contingency of momentary subjective well-being. *Nature communications*, 7, 2016.

[69] Robb B Rutledge, Nikolina Skandali, Peter Dayan, and Raymond J Dolan. A computational and neural model of momentary subjective well-being. *Proceedings of the National Academy of Sciences*, 111(33):12252–12257, 2014.

[70] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[71] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

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

[73] Duane P Schultz and Sydney Ellen Schultz. *Theories of personality*. Cengage Learning, 2016.

[74] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. 2009.

[75] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[76] Jonathan Sorg, Richard L Lewis, and Satinder P Singh. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, pages 2190–2198, 2010.

[77] Ron Sun and Nicholas Wilson. A model of personality should be a cognitive architecture itself. *Cognitive Systems Research*, 29:1–30, 2014.

[78] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[79] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[80] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo

Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[81] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pages 2396–2407, 2018.

[82] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 4644–4654, 2018.