Reinforcement learning from internal, partially correct, and multiple demonstrations

Mao Li

Doctor of Philosophy

University of York
Computer Science

July 2019

Abstract

Typically, a reinforcement learning agent interacts with the environment and learns how to select an action to gain cumulative reward in one trajectory of a task. However, classic reinforcement learning emphasises knowledge free learning processes. The agent only learns from state-action-reward-next state samples. The learning process has the problem of sample inefficiency and needs a huge number of interactions to converge upon an optimal policy. One of the solutions to deal with this challenge is to employ human behaviour records in the same task as demonstrations for the agent to speed up the learning process.

Demonstrations are not, however, from the optimal policy and may be in conflict in many states especially when demonstrations come from multiple resources. Meanwhile, the agent's behaviour in the learning process can be used as demonstration data. To address the research gaps mentioned above, three novel techniques, including; introspective reinforcement learning, two-level Q-learning, and the radius restrained weighted vote, are proposed in this thesis. Introspective reinforcement learning uses a priority queue as a filter to select qualified agent behaviours during the learning process as demonstrations. It applies reward shaping to give the agent an extra reward when it performs similar behaviours as demonstrations in the filter. The two-level-Q-learning deals with the issue of conflicting demonstrations. Two Q-tables (or Q-net in function approximation) for storing state-expert value and state-action value are proposed respectively. The two-level-Q-learning allows the agent not only to learn a strategy from selected actions but also to learn to distribute credits to experts through trial and error. The Radius restrained weighted vote can derive a guidance policy from demonstrations which satisfy a restriction through a hyper-parameter radius. The Radius restrained weighted vote applied the Gaussian distances between the current state and demonstrations as weights of the votes. Softmax was applied to the total number of weighted votes from all candidate demonstrations to derive the guidance policy.

Acknowledgements

First of all, I would like to thank my supervisor Dr. Daniel Kudenko, who guided me to the field of reinforcement learning and showed me what a good researcher is. I really appreciate his contribution of time and ideas to make my Ph.D. experience so valuable and stimulating. He continuously gave me patient guidance and support. His rigorousness and enthusiasm for research were so motivating for me. I am also thankful to Dr.James Cussens, my internal auditor, who helped me to review every milestone of my Ph.D. and who provided constructive suggestions on my thesis. My sincere gratitude is also given to Prof. Ann Nowé, who spent her time reading my thesis and attend my thesis viva.

My wife, Dr.Wei Zheng, gave me meticulous care in daily life and spiritual encouragement. Her love, understanding and encouragement supported me along this journey. My parents, my first teachers, have contributed immensely to my personal development; they have encouraged me to explore science and engineering since I was very young. I thank my family for their love and support. This thesis is dedicated to them.

My sincere gratitude also goes to my friends in and outside York, who cheered me up and encouraged me to move forward.

Finally, thanks Dr.Bertie Dockerill for proofreading the thesis.Thanks also to those who contributed to online-course projects. Without the equal right of education by internet, I could not share the opportunity to learn such good training courses and explore more in my research field.

Thank you!

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Some of the material contained in this thesis has appeared in the following published or awaiting publication papers:

1. Li, M., Brys, T., Kudenko, D. (2018). Introspective reinforcement learning and learning from demonstration. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, (pp. 19921994).

2. Li, M., Kudenko, D. (2018). Reinforcement learning from multiple experts demonstrations. In Workshop on Adaptive Learning Agents (ALA) at the Federated AI Meeting, volume 18.

3. Li, M., Brys, T., Kudenko, D. Introspective Q-learning and learning from demonstration. The Knowledge Engineering Review. 2019.

4. Li, M., Yi, W., Kudenko, D. Two-level Q-learning: learning from conflict demonstrations. The Knowledge Engineering Review. 2019.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, an overview of the thesis is given. It includes a definition of reinforcement learning (RL), the motivation for the research, identification of a research gap and my contribution to addressing this. This chapter also includes an outline of the structure of the rest of the thesis.

1.1 General approaches to AI from RL

One definition of artificial intelligence (AI) is that it is an agent that is able to observe the environment and execute actions to maximise its chance of approaching its goals successfully (Russell and Norvig, 2016; Poole et al., 1998; Nilsson, 1998). Turing, the creator of modern computing opined, in a 1950 paper, that, "instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain" (Turing, 1950).

Machine learning(ML) constructs a mathematical model based on training data, in order to make predictions or decisions without being explicitly programmed to perform the task. Performance of an ML algorithm is improved using experiences with the training time (Friedman et al., 2001). Machine learning has been divided into three categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses algorithms to train an agent to learn the mapping from a feature space to a target space through a labelled dataset (Mohri et al., 2018, chapter 2). Unsupervised learning agents have been trained on datasets without labels. Unsupervised learning is generally used, as Ghahramani (2003) notes, to discover hidden patterns or groupings in a dataset. In contrast, reinforcement learning (RL) is the paradigm which addresses sequential decision making in a task. Unlike supervised learning, the goal of reinforcement learning is to find the optimal decision sequence to maximise the cumulative utilities of the agent (Barto, 1997).

To achieve the goal of creating an intelligence agent, AI researchers draw lessons from psychologists and biologist to educate agents through giving them reward and punishment signals. Skinner (1990) considers reinforcement to be the key to learning. The main principle of RL is giving positive reinforcement and negative reinforcement signals to an agent based on all of its actions. As the name implies, positive reinforcement gives a reward signal to the agent. For instance, training a mouse to touch a board in its cage so that food is given when it touches the board. The giving of food is, in this instance, the positive reward. Negative reinforcement is akin to a punishment. Therefore, to use the same example, giving the mouse an electric shock to prevent it from performing actions which are considered to be bad behaviours would be a negative reinforcement. Physiological evidence of RL has been found in recent years with Niv (2009) reporting that the temporal difference of error of reward, a key signal of reinforcement learning, has been recorded in the functional imaging of humans and animals in their decision-making processes.

Figure1.1 shows the loop of agent-environment interaction. In a loop of agent-environment interaction, the agent observes the state of the environment then selects an action according to its policy. After the selected action has been executed, the state of the environment is changed. Meanwhile, the reward function gives a number by which to evaluate the performance of the action. Then, the agent goes to the next loop, observing the environment again and so on.

The loop keeps circulating until it approaches a special state, the terminal state. The terminal

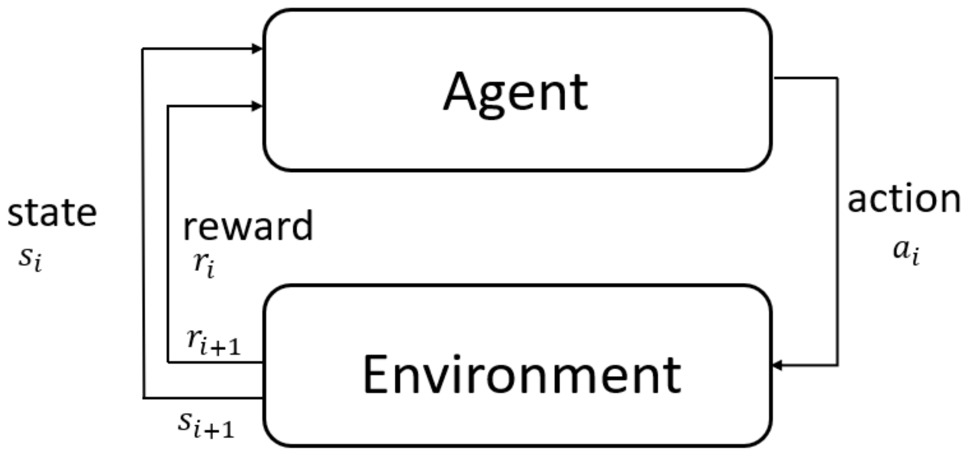

Figure 1.1: Agent environment interaction

state is the sign that the task has come to an end. For instance, in a chess game, the opponent could be considered to be like the environment. An action is represented by the making of one move during the game. If one side's King has been checkmated, then the agent has achieved the terminal state for he has won the game and thus the task has come to an end. Obviously, however, losing pieces in the same game of chess is not good. Therefore, an agent should be given a negative reward when it loses pieces. If the agent takes the opponent's pieces, it should be given a positive reward. A big positive reward should be given to an agent if it checkmates the opponent's King. The environment represents the real world, like the rules and opponents in a game. The reward function, defined by humans, represents subjective wishes, the goal of a task, for example, to checkmate the opponent's King.

1.2 Research Gap

Although the framework and perspective of reinforcement learning is very ambitious, RL algorithms still face many challenges:

1.2.1 Delayed feedback and credit allocation

The goal of RL is to find an optimal policy that maximises the expectation of cumulative rewards. The agent only gets the cumulative reward at the end of an episode. However, long-delayed rewards make it extremely hard to trace back what sequence of actions contributed to giving of the rewards. RL needs a mechanism to allocate the total reward of a trajectory into each state-action in the trajectory. This mechanism has been called the credit allocation problem (Sutton, 1988). The details of this are discussed in Chapter two.

1.2.2 Representation of state space

Like supervised learning, RL faces the challenge of the curse of dimensionality (Bellman, 2015). In a toy task, the number of states is small and all states can be stored in one tabulation, called tabular RL. However, the number of states increases exponentially with the number of features. When it comes to a complex task, the number of features is too large to store all the states in a Q-table. Furthermore, if state features are continuous, the number of states is infinite. Function approximation techniques are used to deal above issues. It uses parameterized models of features to fit the values. Current years, Deep learning is one active research area which used an artificial neural network (ANN) to propose an end-to-end learning system which could use raw data, such as images, as input data and thereafter learn the features of the raw data via multiple level transformation. Reinforcement learning combines with deep learning is called deep reinforcement learning (DRL) in literature.

Due to the fact that the sample of RL came from trajectories, those samples were highly related. To satisfy the condition of independent and identically distributed (I.I.D) samples, memory replay has been proposed. The memory replacement technique addresses the challenge of correlation of samples by using a buffer to collect each sample online, and then resamples a batch sample from memory to fit the model. Memory replay breaks the sequential relationships of samples - Chapter Two gives more review details.

DRL has achieves some success in lots of domain. The most successful example of deep reinforcement learning is from Google DeepMind, which invented the deep q-learning method to play the Atari Game 2600 from an image without prior knowledge. It used raw images (80x80 pixels) as input states, with multiple levels of convolutional and full connection neural networks as function approximations to learn the optimal actions performed in the game. Mnih et al.

(2013, 2015) and Hessel et al. (2017) showed that after millions of steps the agent outperformed human players in a majority of games. This was the first implemented end-to-end model-free control (reinforcement learning from raw data). After Mnih et al. (2013, 2015), Atari Game 2600 became a new benchmark for reinforcement learning research.

1.2.3 Explore and exploit the balance

An agent has no prior knowledge of a task, it only estimates and improves its model of Q-values or agent's policy from samples. Therefore, the agent faces an explore-exploit dilemma.

Demonstrations from external experts or agent itself could be used to guide the agent with biased exploration. Chapter 3 will review previous research in this topic and chapter 4 chapter, chapter 5 and 6 shows our contributions of this area.

The key contribution of this study is biased exploration using demonstrations from inside and outside sub-optimal policies.

1.2.4 Sample inefficiency

Reinforcement learning generates samples from agent-environment interactions and then uses these samples to estimate and improve the policy. In the original reinforcement learning algorithm, the sample is used only once and then discarded. It follows, that information about the sample has not been completely employed. As we know, interacting with the environment involves computation time and energy costs, so improving sample efficiency is a key research topic in RL.

An off-policy RL algorithm learns the value of the optimal policy independently of the agent's policy. Therefore, with off-policy learning, samples can be reused multiple times to increase the utilisation rate of samples.

1.2.5 Sparse Rewards

In most tasks, the reward of most state-action pairs is zero. This phenomenon has been called 'sparse rewards' (Osband et al., 2014, 2016). Due to the lack of feedback from the environment, the RL agent needs to explore for a long time to find the optimal policy. Intuitively, we can add extra signals to reward or punish the agent during its learning. Changing the reward function has been called reward shaping (Ng et al., 1999). However, a change of reward function may result in a different optimal policy. For instance, a tasks only have one positive rewards, when the agent approaches the terminal state and rewards 0 for all non-terminal states. If we simply give an extra reward for a non-terminal but good state, the optimal policy will change. In such an instance, the agent will learn a policy to repeatedly enter and leave the state with extra reward rather move forward to the terminal state. This is because continually going in and leaving the extra reward state is a better choice than only getting a one-time reward.

1.3 Research Motivation

Improving the efficiency of samples is the main research focus of this thesis. One method to address this issue is guiding the RL agent to biased exploration of the state-action space with demonstration. As far as we know, all current research in this area has focused on using demonstrations from human players to guide the agent to biased exploration. In our research, we consider that demonstrations could come from the agent's own performance in the RL loop. A filter has been applied to select qualified demonstrations to reuse. The filter will keep Currently superior demonstrations and sweep out poor demonstrations. As the agent learn from environment, the performance of demonstrations in filter have been improved continuously and finally converge into optimal demonstrations.

Conflicts between demonstrations from different resources have been ignored in some research. This thesis, therefore, focuses on how to improve the sample efficiency via demonstrations. The demonstrations come from agent-environment interactions or human experts. It has following characteristics:

1. Sub-optimal A human expert is not an optimal agent, so the demonstration data we collected is not optimal, but is much better than random policies.
2. Conflict: For demonstrations coming from multiple sources, in a same state, it may have multiple demonstrations have different actions.
3. High-dimensional: In recent years, the deep learning technique has been introduced to reinforcement learning. It uses raw data as a state. The demonstration data may, therefore, develop a high-dimensional raw dataset.
4. Huge size: The cost of data collection continues to reduce. For instance, numerous game corporations have collected a bulk of player behaviour data. That data could be used to help train the agent to perform a complex task.
5. Imbalanced: Demonstrations may be imbalance that major demonstration distributed on part of state space.

Considering these characteristics, we think there is a research gap that requires the building of an algorithm to deal with the challenges.

1.4 Hypotheses

In this thesis, the following hypotheses are proposed:

1. In Q-learning, experience samples and demonstrations can be filtered and reused to speed up the learning process via reward shaping compared the performance of state-of-art algorithm such as similarity based shaping (SBS) (Brys et al., 2015) on Super Mario Bro and cartpole.
2. An agent using a Two-level Q-learning approach will improve the performance when dealing with conflicting demonstration with sub-optimal experts, compared to state of the art Reinforcement learning from demonstrations (RLfD) method such as Confidence Human Agent Transfer(CHAT) (Wang and Taylor, 2017).

3. Constraining the distance between state and demonstration, called as radius in this thesis when selecting demonstrations and derive a policy from majority voting improves the performance compared to state-of-art approaches such as SBS and kNN-transfer (Wang et al., 2018).

The following assumptions are made in this thesis: The demonstration dataset is sub-optimal. The performance of the polices which produce the demonstration are better than the performance of a uniformly and randomly chosen policy.

1.5 Contribution

1.5.1 Introspective Q-learning

To deal with the sample inefficiency problem, this thesis extends the technique in Brys et al. (2015) which used Gaussian similarity between state and demonstrations to reshape the reward function. Brys et al. (2015) is based on the assume that similar states have the same optimal action and that these widely exist in many domains such as control tasks. Our approach, Introspective Q-learning succeed this assumption.

In Brys et al. (2015), the demonstrations come from human experts. However, in this research, high value samples generated during learning were considered to be demonstrations. Monte Carlo method has been used to estimate the Q-value of each demonstrations. Demonstration is recorded as a triple, noted as $< s_t, a_t, \hat{Q}_t >$. The triples were filtered using a priority queue. This filter keeps high estimate values as demonstrations and keeps weeding out low value items to improve the qualification of the demonstrations. As time goes by, the triple that was kept in the filter had a high estimated value.

Introspective Q-learning is different from $TD(\lambda)$ algorithm. $TD(\lambda)$ algorithm propagate TD error to previous samples in a same trajectory (vertical propagation). The hyper-parameter λ is the sense of distance of propagation, used to balance variance and bias. However, Introspective Q-learning propagates values around state-actions through reward shaping and can be considered to be lateral propagation, propagating information to the surrounding state-action.

Due to filter updates, the demonstration during the learning process, at the end, the agent only keeps samples from the optimal policy. Therefore, introspective RL could converge into the same optimal action as normal Q-learning.

1.5.2 RL from conflicting demonstrations

Demonstrations from multiple sources such as human experts, heuristic rules and policy transferred from other domains and so on, may be in conflict (Li and Kudenko, 2018). This means that there can be multiple demonstration that have same state but possess different actions. If we ignore those conflicts, the agent cannot know which demonstrations to trust. To address conflicting demonstrations, a novel algorithm named Two-level Q-learning (TLQL) is proposed. The key idea of TLQL is that the agent learns not only the optimal policy for selecting actions, but also learns a policy of selecting which demonstrations to follow for every state.

1.5.3 RL from massive and imbalanced demonstrations

In the same domain (e.g. online games), there are a massive number of demonstrations available. This brings about two challenges. First, the distribution of demonstrations is imbalanced which means there are massive demonstrations in some particular states while there are no available demonstrations in some states. If the agent only applies its nearest demonstrations, it will tend to have the overfitting issue in the sparse demonstration regions. Second, for the states with massive demonstrations, demonstrations cover all types of actions and are in conflict. The TLQL algorithm proposed in this chapter which can narrow the action space, is not suitable for massive demonstrations.

To deal with the challenges mentioned above, a radius-restricted weighted vote algorithm (RRWV) is proposed. This approach introduces a hyper-parameter radius to select demonstration candidates. The selected demonstration candidates then vote for demonstrations' actions.

In the algorithm, votes on each action are counted and a policy via softmax function on the votes is produced accordingly. Compared with previous studies on RL with demonstrations, our approach provides new techniques to address complicated demonstration settings.

1.6 Structure of the Thesis

As Figure 1.6 illustrations, this thesis is separated into six chapters.

Chapter Two reviews the definitions of the Markov decision process, terminology, and mathematical notations. This chapter also reviews the classic algorithm to evaluate and improve the policy via state-action value. Policy gradient algorithms and derivative algorithms are also included. Current reinforcement learning is well developed, combined with deep learning. In Chapter Two, we review deep Q-learning and its derivative algorithms: Double-DQN, Dual-DQN, prioritised replay, noise DQN. The policy gradient method and value function base are combined in an actor-critic framework. State-of-the-art techniques, such as TROP and PPO, are also included in this chapter.

Chapter Three is a review of related research in which external information to speed up the RL process is used. External information could come from human expert demonstrations, heuristic rules, online agents advising, and transferring policy from other tasks. In this chapter, we review the characteristics of the external information sources. How to combine RL with that information is a well-studied research topic, and there are some techniques to inject external information into RL loops, such as reward shaping, probability reuse, pseudo-action and so on. These are reviewed and commented upon.

Three classes of methodology pertaining to injecting human expert demonstrations into the agent's learning process are reviewed first. We also review behaviour clones and human-involved agent-environment interactions. Finally, discussions on the differences between that research and the work in this thesis is presented.

Chapter Four focused on Introspective Reinforcement Leaning. In the chapter we present how

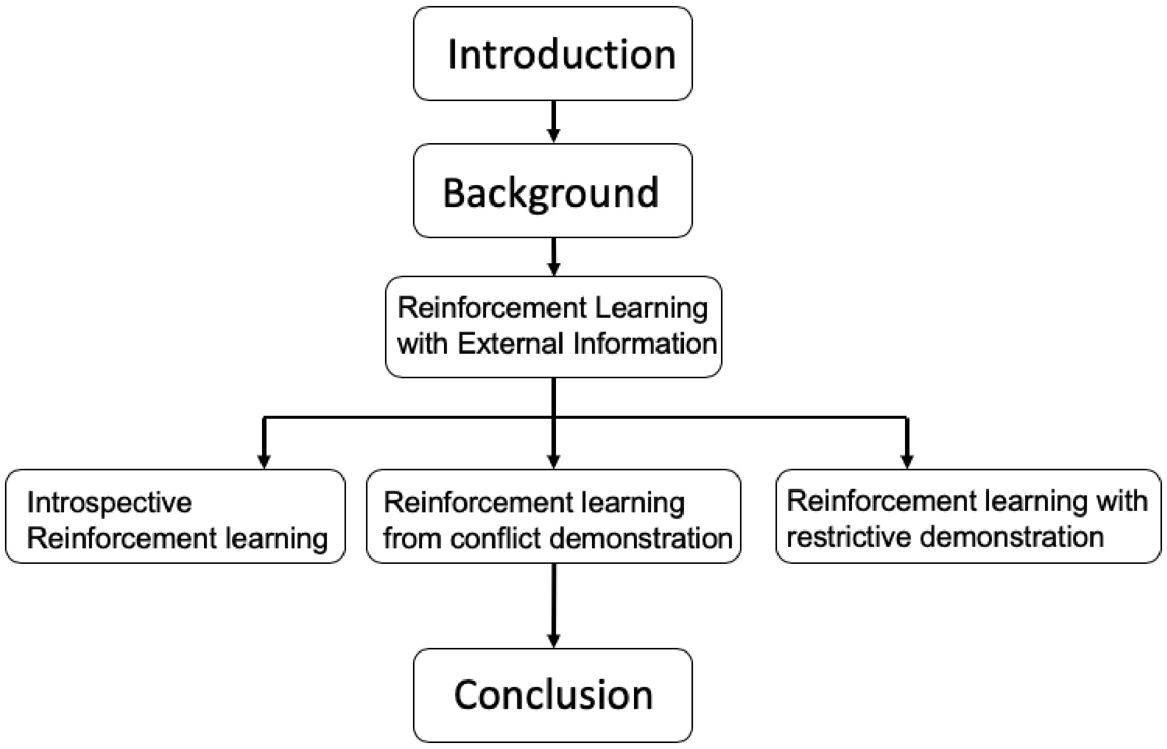
Introduction
Background
Reinforcement Learning with External Information
Introspective Reinforcement learning
Reinforcement learning from conflict demonstration
Reinforcement learning with restrictive demonstration
Conclusion

Figure 1.2: Thesis structure

the introspective RL algorithm works. It applies a priority queue to filter high-value samples during the learning process and uses the reward shaping technique to supply extra reward signals to the agent.

Chapter Five: Q-learning with multiple conflicting demonstrations, describes the technique details of Q-learning with multiple conflicting demonstrations. This novel method can make an agent not only learn the policy of selecting actions, but also select experts to trust in.

In Chapter Six, we study and analyse reinforcement learning from a mass demonstration setting. It proposes to derive a policy from demonstrations with a radius. This technique helps avoid over-fitting in sparse demonstration regions. It also addresses demonstration conflicting issues via softmax voting weighted by Gaussian distance.

In the final chapter, conclusions are drawn based on our work. In addition, this chapter also presents further research directions drawing on the work presented in the thesis.

1.7 Chapter Summary

In this chapter, an overview of reinforcement learning has been given. In addition, this chapter has identified that improving sample efficiency is one of the challenges and difficult issues faced by reinforcement learning. Existent research gaps and our hypotheses were also presented in this chapter. Finally, the contributions that this thesis makes to the furtherance of existing academic knowledge were noted as well as an overview of the thesis' structure.

Chapter 2

Reinforcement Learning Background

In this chapter, we provide the background to reinforcement learning and define the terminology used in later chapters. We also review state-of-the-art algorithms in RL that are compatible with deep neural networks.

2.1 Overview

Reinforcement learning (RL) is a paradigm of a behavioural learning model (Sutton et al., 1998). The RL agent receives feedback from the environment, guiding the agent to the optimal policy. Unlike other types of supervised learning, RL is not trained on labelled training data, but learns a policy via receiving a reward signal on each step. The solution to the task, called 'the policy', is defined as a mapping from the state space to the action space, denoted as $\pi(s)$ (Szepesvári, 2010; Wiskott, 2016). Unlike the one-shot decision of supervised learning, the goal of RL is to search for an optimal policy to gain the maximum accumulated reward from the whole trajectory. RL is a memoryless process that can be modelled, as described in the next section, as a Markov Decision Process (MDP).

Early research included applying RL to classical control problems such as mountain-car (Boyan and Moore, 1995), cartpole (Geva and Sitte, 1993) and pendulum (Anderson, 1989). Tesauro

(1995) trained an RL agent, TD-Gammon, to play Backgammon. After a simulation with self-play, it was able to beat a human backgammon champion. The past thirty years have seen tremendous achievements in the field of RL. In 2015, Google DeepMind proposed a Go player agent, AlphaGo (Silver et al., 2016), and beat the top ranked human player. This attracted wide public attention to the progress that had been made in the field of RL. Following this, AlphaGo combined various techniques such as deep learning, Monte Carlo Tree search and memory replay as a general solution to board games and cards. This solution was called Alpha zero (Silver et al., 2017b), and it won shoji (Japanese chess) (Silver et al., 2017a) and other games. RL also achieved successes in other domains such as Dota (OpenAI, 2018), finance (Nevmyvaka et al., 2006; Van Roy, 2001), recommendations (Golovin and Rahm, 2004), and robotics (Kober and Peters, 2012).

2.2 The Markov Process

Markov processes are the foundation of RL. This chapter introduces Markov Chains, Markov Reward Processes and, Markov Decision Processes.

2.2.1 Markov Chains

A Markov Process or Markov chain is a memoryless process which is represented as two-tuples $\langle S, T \rangle$. S is a set of states, represented as a vector; T is a transition probability matrix of states. If the probability of the current state depends only on the probability of the previous state, it is regarded as having a Markov property (Kaelbling et al., 1996). Markov processes can be described by equation 2.1:

$$P_{s,s'} = P[S_{t+1} = s' | S_t = s] \tag{2.1}$$

Markov properties were introduced to model the environment-agent interactions because RL is a memoryless process. For a finite state set S, the transition function T can be expressed as a matrix shown in equation 2.2

$$\begin{bmatrix} V(s_1|s_1) & T(s_2|s_1) & ... & V(s_n|s_1) \\ V(s_1|s_2) & T(s_2|s_2) & ... & V(s_n|s_2) \\ ... & & & \\ V(s_1|s_n) & T(s_1|s_n) & ... & V(s_3|s_n) \end{bmatrix} \tag{2.2}$$

2.2.2 The Markov Reward Process

A Markov reward process (MRP) consists of a reward function and a Markov chain. The reward function is a utility function which maps from a $< state, action >$ pair to a real number, noted as $R(s, a) \mapsto \mathbb{R}$. A positive reward means an award is given whereas a negative reward represents a punishment. The cumulative reward is a discounted sum of rewards from time step t to the horizon.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \tag{2.3}$$

Equation 2.4 defines the value of state as the expected return from the start in state s noted as:

$$V(s) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...] \tag{2.4}$$

γ is a hyper-parameter, a real number between 0 and 1, to balance the short-term reward and the long-term reward. If the discount is 0, the agent only learns the reward of one step; this is equivalent supervised learning. If the discount is 1, all rewards in the present and in the long term are considered to be equally important.

2.2.3 The Markov Decision Process

A Markov Decision Process (MDP) is defined as a five-tuple (S, A, R, T, γ). Specifically, **A** is the action space in which an agent interacts with the environment. The policy π of a RL agent is a mapping from a state-action to a probability, denoted as $\pi : (S, A) \mapsto [0..1]$. Applying the policy π in the MDP results in a Markov Reward Process (MRP) $(S, A, T^{\pi}, R^{\pi}, \gamma)$ where $R^{\pi}(s'|s) = \sum_{a \in A} \pi(a|s)T(s'|s,a)$, $T^{\pi}(s'|s) = \sum_{a \in A} \pi(a|s)T(s'|s,a)$. Equation 2.5 shows the Bellman equation, which is the foundation of dynamic programming and temporal difference algorithms.

$$V^{\pi}(s) = R^{\pi}(s) + T^{\pi}(s'|s)V^{\pi}(s') \tag{2.5}$$

Equation 2.5 indicates that a MDP can be decomposed into sub-problems. These sub-problems can be reused in the algorithm.

2.3 Model-based policy estimation

The term ‘model-based’ used here refers to a setting in which the transition probability function is known. Because of this, both the analytical solution and the iterative algorithm can be applied to solve the task.

According to the definition of a state value, it can be decomposed into two parts: the immediate reward and the discounted sum of future reward: $V(s) = R(s) + \gamma \sum T(s'|s)$. The finite states of a MRP could be expressed as the matrix below:

$$\begin{bmatrix} T^{\pi}(s_1) \\ T^{\pi}(s_2) \\ ... \\ T^{\pi}(s_N) \end{bmatrix} = \begin{bmatrix} R^{\pi}(s_1) \\ R^{\pi}(s_2) \\ ... \\ R^{\pi}(s_N) \end{bmatrix} + \gamma \begin{bmatrix} T^{\pi}(s_1|s_1) & ... & T^{\pi}(s_n|s_1) \\ T^{\pi}(s_1|s_2) & ... & T^{\pi}(s_n|s_2) \\ ... & & \\ T^{\pi}(s_1|s_n) & ... & T^{\pi}(s_3|s_n) \end{bmatrix} \begin{bmatrix} V^{\pi}(s_1) \\ V^{\pi}(s_2) \\ ... \\ V^{\pi}(s_N) \end{bmatrix} \tag{2.6}$$

Equation $V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi}$ shows that an MRP can be explained by the linear equation:

$$\begin{aligned}
&V^\pi - \gamma T^\pi V^\pi = R^\pi \\
&(I - \gamma T^\pi)V = R \\
&V^\pi = (I - \gamma T^\pi)^{-1} R^\pi
\end{aligned} \tag{2.7}$$

The computational complexity of this solution is $O(N^3)$, as a matrix inverse operation was involved.

Dynamic programming is another solution through iteration, as shown by Algorithm 1. The computational complexity of the dynamic programming is $O(N^2)$ for each iteration.

Algorithm 1 Dynamic programming of MRP

```
procedure DYNAMIC PROGRAMMING OF MRP UNDER POLICY $\pi$
    for k=1 until convergence do:
        for all s in S do:
            $V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in S} T^\pi(s'|s) V_{k-1}^\pi(s')$
        end for
    end for
end procedure
```

2.4 Model-free policy estimation

In most circumstances, the transition probability function is unknown or is too complex to be represented. Model-free estimation does not require a known transition function (Sutton et al., 1998). Rather, it estimates the expectation of state values by sampling data through agent-environment interactions.

2.4.1 The Monte Carlo estimation

Monte Carlo techniques (MC) estimate the expectations of state values using rewards from trajectories as Algorithm 2 shows.

Algorithm 2 Monte Carlo model-free estimation

```
procedure General policy iteration
    for each state s visited in episode i do:
        total first visits N(s) = N(s) + 1
        Increment total return $S(s) = S(s) + G_{i,t}$
        Update estimate $V^\pi = S(s)/N(s)$
    end for
end procedure
```

The process of mean estimation can be incremental.

$$V^\pi(s) = V^\pi \frac{N(s)-1}{N(s)} + \frac{G_{it}}{N(s)} = V^\pi + \frac{1}{N(s)}(G_{it} - V^\pi(s)) \tag{2.8}$$

The α is defined as a learning step in the MC estimation:

- $\alpha = \frac{1}{N(s)}$ is identical to every MC visit.
- $\alpha > \frac{1}{N(s)}$ opens a memory window to forget older samples. This is useful in non-stationary domains.

$$V^\pi(s) = V^\pi \frac{N(s)-1}{N(s)} + \frac{G_{it}}{N(s)} = V^\pi + \alpha(G_{it} - V^\pi(s)) \tag{2.9}$$

Due to the fact that MC estimates the cumulative reward from the whole episodes, it does not require either Markov properties or the known transition function. Although MC is an unbiased estimation of the policy value, the variance of trajectory samples is very large, especially in tasks with a long trajectory. MC needs numerous samples from a complete trajectory to confidently predict an estimated value.

2.4.2 Temporal difference estimation

Temporal difference learning (TD) adheres to the properties of MDPs, and reuse the estimated value of the next state to estimate the value of the current state. It is called the bootstrap technique. TD estimates the cumulative reward of the trajectory G_t by $\gamma V^{\pi}(S_t)$ plus one-step reward r_t. Therefore, unlike the MC, the TD is a biased estimation of the policy value.

$$V^{\pi}(s) = V^{\pi} + \alpha([r_t + \gamma V^{\pi}(s_{t+1})] - V^{\pi}(s)) \tag{2.10}$$

The $V^{\pi}(s)$ is called the TD prediction, and $R_t(s) + \gamma V^{\pi}(s_{t+1})$ refers to the TD target. The TD error will be continuously reduced until the agent approaches an acceptable performance level.

Compared with MC, TD algorithms have a number of benefits. First, the TD algorithm updates the value step-by-step, while the MC updates the value trajectory by trajectory. This means the TD does not require the task to have a finite horizon. With the discount of less than 1 , an infinite-horizon task also converges to its state value. Second, with the bootstrap technique in the TD, variance is reduced. Theoretically, the TD algorithm will converge to a true state value when every state is visited unlimited times, according to the law of large number.

2.5 Policy improvement

2.5.1 Generalised policy improvement

The goal of RL is to find an optimal policy so as to obtain the maximum cumulative reward.

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \tag{2.11}$$

In reality, however, the policy space is extraordinarily large. For instance, even when the state set $\mathbf{S}$ and action set $\mathbf{A}$ are finite and discrete sets, the size of policy space will be $|A|^{|S|}$. To use the principle of Bellman equation, state values $\mathbf{V}$ are decomposed into Q-values of $< state, action >$ pairs. As shown in Equation 2.12, the Q-value $< s, a >$ equals an immediate reward plus the **V** value of the next state following the policy.

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s) V^{\pi}(s') \tag{2.12}$$

The policy π_i on state s could be monotonously improved to π_{i+1} by a maximisation operation as Equation 2.13 shows.

$$\begin{aligned} &\max_a Q^{\pi_{i+1}}(s,a) = \\ &\max_a R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^{\pi_i}(s') \\ &\geq R^{\pi}(s) + \gamma \sum_{s' \in S} T^{\pi}(s'|s) V^{\pi_i}(s') \end{aligned} \tag{2.13}$$

Model-free policy search is a contextual multi-armed bandit problem that needs an exploration-exploitation balance. In this thesis, we use the $\epsilon - greedy$ strategy to improve the policy. Greedy here means that the agent applies the current policy by selecting the action with the maximum Q-value. To balance exploration and exploitation, the agent randomly chooses an action with a uniform distribution under a small probability ϵ. Algorithm 3, which combines policy evaluation and policy improvement, is called the generalised policy iteration (GPI); it facilitates the convergence to the optimal policy.

Algorithm 3 Generalised policy iteration

```
procedure GENERALISED POLICY ITERATION
    while i==0 or $|\pi_i - \pi_{i-1}| > 0$ do:
        policy estimation: $V^{\pi_i}$
        policy improvement: $\pi_{i+1} = \epsilon - greedy(V^{\pi_i})$
    end while
end procedure
```

Policy estimation and policy improvement can operate in one iteration, called the value iteration, as Algorithm 4 shows.

Algorithm 4 Value interaction

```
procedure DYNAMIC PROGRAMMING OF MRP UNDER POLICY $\pi$
    while $\Delta < \theta$ do:
        $\Delta = 0$
        for all s in S do:
            $v \leftarrow V(s)$
            $V(s) \leftarrow \max_a \sum_{s'} T(s'|s,a)[r + \gamma V(s')]$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
        end for
    end while
end procedure
```

2.5.2 TD-learning

TD-learning follows the same structure as the value interaction algorithm (Algorithm 4). Instead of interacting with a known transition function, the TD-learning agent estimates and updates the value function with samples. For each sample, the Q-value is then updated by the temporal difference error (TD-error) between the target Q-value $R + \gamma Q(s', a')$ and the current Q-value $Q(s, a)$.

SARSA

State-Action-Reward-State-Action (SARSA) (Rummery and Niranjan, 1994) is a model-free and on-policy algorithm that can update the policy based on the action from its current policy. With an exploration-exploitation balance like ϵ-greedy, SARAS selects action **a** from **s** using the policy derived from the Q-value function. The updated SARSA rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma Q(s', a') - Q(s,a)]$$

Q-learning

Q-learning is an off-policy algorithm because the policy generating samples and the policy being updated are different. The policy generating samples is derived from the Q-value function via $\epsilon - greedy$, but the target policy comes from the maximum Q-value actions. It updates the

Q-value with TD-error between the current Q-value: $Q(s,a)$ and the target Q-value: $R + \max_a \gamma Q(s', a)$. The updated Q-learning rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \max_a \gamma Q(s', a) - Q(s,a)]$$

Eligibility Traces

TD-learning can reduce variance using bootstrapping samples. However, it is a biased estimation of true value. Monte Carlo, on the other hand, is an unbiased estimation but has variance. An eligibility trace (Sutton et al., 1998; Kaelbling et al., 1996) can mix both TD-learning and MC. The integration of TD-learning and MC can be recognised as a special case of an eligibility trace.

An eligibility trace can record the previously experienced occurrences of state-action pairs. Samples from environment interaction is recorded in the trace with an eligibility of one. In an eligibility trace, a hyper-parameter λ, where $0 < \lambda < 1$ is introduced as the decay rate. The eligibility of samples is decayed by multiplication with λ. The eligibility trace algorithm remove samples that less than a threshold. Only state-action pairs in the trace are updated when a reward is received. If $\lambda = 0$, the TD(0) will be equal to the TD algorithm, because it only has a one-step update. However, when $\lambda = 1$, it will update the Q-value using a discounted sum of rewards in the whole trajectory; this is equivalent to the Monte Carlo approach. An eligibility trace for the state **s** and action **a** is updated as follows:

$$e(s,a) = \leftarrow \begin{cases} 1 & s = a_t, a = a_t \\ \gamma\lambda(s,a) & otherwise \end{cases}$$

An eligibility trace can be presented as a vector **e**. The parameters of the Q-value approximation, θ, is updated based on Equation 2.14, where δ denotes the TD-error between the target value and the current value.

$$\theta \leftarrow \theta + \alpha e \delta \tag{2.14}$$

2.6 Function approximation

When the domain is small, a look-up table can store all state-action pairs (Busoniu et al., 2010). However, the size of the look-up table grows exponentially with the number of state features. Bellman (1961) called this phenomenon 'the curse of dimensionality'. Due to the limitation of memory and computational time, a tabular representation cannot hold a complex task and a function approximation must be involved. In addition, for continuous state and action spaces, a look-up table is not suitable because the number of state-action pairs is infinite.

The first step of a function approximation is to represent the state-action pairs as features. This is denoted as in the equation 2.15.

$$X(s,a) = \begin{pmatrix} X_1(s,a) \\ X_2(s,a) \\ ... \\ X_n(s,a) \end{pmatrix} \tag{2.15}$$

The goal of a function approximation is to minimise the loss error between the true state-action value function $Q^\pi(s,a)$ and the approximate state-action value function $\hat{Q}^\pi(s,a,\mathbf{w})$. $\mathbf{w}$ is parameters to learn. The loss function is defined through the approach noted in equation 2.16:

$$J(\mathbf{w}) = \mathbb{E}_\pi[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathbf{w}))^2] \tag{2.16}$$

In TD-learning, TD-error is used as the loss function, as shown in equation 2.17. Weight $\mathbf{w}$ is updated according to its gradient $\nabla J_{TD}(\mathbf{w})$ as equation 2.19 and 2.18 shows.

$$J_{TD}(\mathbf{w}) = \frac{1}{2}(r + \gamma max_{a'}\hat{Q}(s',a',\mathbf{w}) - \hat{Q}(s,a,\mathbf{w}))^2 \tag{2.17}$$

$$\nabla J_{TD}(\mathbf{w}) = (r + \gamma max_{a'}\hat{Q}(s', a', \mathbf{w}) - \hat{Q}(s, a, \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s, a, \mathbf{w}) \tag{2.18}$$

$$\mathbf{w} = \mathbf{w} + \nabla J_{TD}(\mathbf{w}) \tag{2.19}$$

In this thesis, tile coding and deep neural network were employed as function approximators. Tile coding is a piece-wise constant approximation that is particularly well suited for continuous state space (Sutton et al., 1998). In tile coding, the feature space is grouped into partitions. As Figure 2.1 shows, each partition is defined as a tiling and each item in the partition is a tile. Each tile is represented with binary features. If the given state falls into the region represented by a tile, the binary record is 1. If not, the record is 0. The state value represented via tile coding is the sum of weights:

$$V(s) = \sum_{i=1}^{n} b_i(s)w_i \tag{2.20}$$

where n is the number of tiles; $b_i(s)$ is record of the i^{th} tile; and, w_i is the weight of each tile.

The gradient of the value function is:

$$\nabla V(s) = \max_a(R(s, a) + \gamma V(s')) - V(s)$$

The updated rule is:

$$w_i \leftarrow w_i + \frac{\alpha}{m} b_i(s)\nabla V(s)$$

Smaller tiles produce fewer approximation errors but their generalisation abilities are reduced. Adding more layers of tiling can improve the precision of learning. The number of tiles is defined as a hyper-parameter that can be used to balance learning speed and precision (Whiteson et al., 2007).

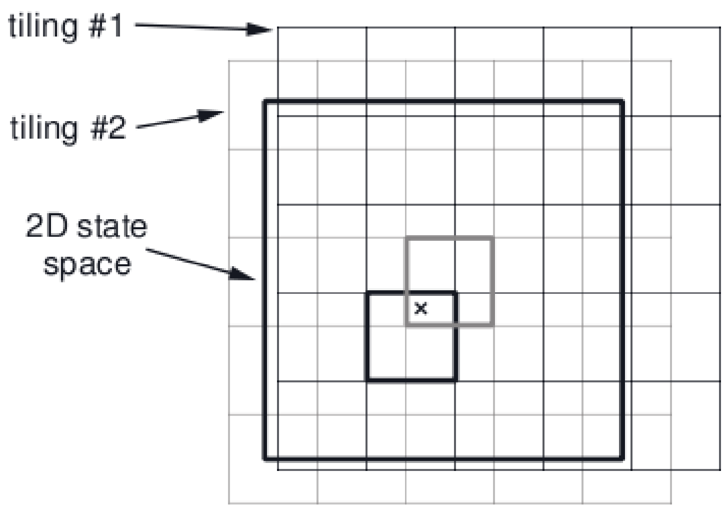

Figure 2.1: Tile coding (Sutton et al., 1998)

2.6.1 Deep Reinforcement learning

Deep learning is a type of machine learning method based on learning data representations, as opposed to task-specific algorithms with feature engineering. In a general way, it is based on the layers used in artificial neural networks (ANNs) (Bengio et al., 2009). Deep learning and RL are integrated in Deep Reinforcement learning (DRL), which has been developed rapidly since 2015 when Mnih et al. (2015), proposed an end-to-end player agent to Atari. In this chapter, Deep Reinforcement Learning (DRL) and its derivation algorithms are introduced.

Deep Q-learning

ANNs are models vaguely inspired by the biological neural networks in human brains (Schmidhuber, 2015). Classic techniques, such as the Multi-layer Perceptron (MLP) and the Convolutional Neural Networks (CNN), were well developed in the 1980s and the 1990s. Benefiting from data blooming and the increase in computational power today, ANN techniques have been revived in the past decade. AlexNet (Krizhevsky et al., 2012) shows that the deep learning CNN to have high power in the ImageNet competition. CNNs use filters to detect margin information from images and automatically extract features using multiple iterations of convolution operations. This process is called end-to-end learning or representation learning.

The integration of RL with an ANN was examined in earlier research. Riedmiller (2005) first applied an ANN to fit Q-values. Lange et al. (2012) proposed the deep fitted Q-Learning(DFQ) algorithm to build a vehicle control system. Mnih et al. (2013, 2015) combined Q-learning and the CNN technique to use raw pixels as input and produce the Q-values of actions. This was the first end-to-end control system, which outperformed human expert players in most Atari 2600 games. AlphaGo is a Go play agent which defeated the human champion Lee Se-dol and demonstrates the power of deep reinforcement learning.

In Mnih et al. (2013), the inputs to the Deep Q-Network (DQN) include the last 4 frames of images. The transition of images was via 3 layers of a CNN, and then 2 fully-connected layers which produced Q-values for each action. Another important contribution of DQN is memory replay. Function approximation must assume that training data satisfies independent and identically distributed (I.I.D) sampling. However, in Q-learning, data comes from episodes of interactions. Step samples from the same episode are strongly related. To break the correlations of samples, DQN introduces a replay buffer called memory to collect samples deriving from interactions and re-sample batches of datasets from memory to train the neural network. The vanilla DQN is shown in Algorithm 5

Algorithm 5 Vanilla DQN

```
procedure DQN
    Initialise Q-net $\theta$,target-Q-net $\theta$, Memory D.
    for episode=1...M do
        for t=1...T do
            exploit-explore with Q-net $\theta'$
            store transition $< s_t, a_t, r_t, s_{t+1} >$ into D
            sampling minibatch of transitions from D
            set $y_i = \begin{cases} r_j & \text{if terminate at j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta) & \textit{otherwise} \end{cases}$
            perform a gradient descent of $\theta$ on $(y_j - Q(\phi_j, a_j; \theta))^2$
        end for
    end for
end procedure
```

2.6.2 Improvement of DQN

After the great success of Mnih et al. (2013), different types of DQN modified algorithms have been proposed in the last three years to improve its performance from different perspectives. The following algorithms are the most influential.

Target net Freeze

In Mnih et al. (2013), the agent updates the neural network after each interaction. To avoid oscillations of the neural network in small sets of samples, Mnih et al. (2015) introduced a network freezing technique to separate the Q-network into two identical nets called current net and target net. The target network provides actions during the learning process and the parameters of the current network are optimised at every step. After N updates (N is a hyper-parameter), the parameters of the current Q-network are copied to the target Q-network. Therefore, compared with the vanilla DQN, the policy updating frequency is decreased and the algorithm is more stable (Mnih et al., 2015).

Double DQN

Using an $\epsilon - greedy$ policy to estimate the Q-value yields a maximisation bias (Hasselt, 2010). To avoid the bias, Van Hasselt et al. (2016) applied two independent networks to the unbiased estimation of Q-values. One Q-network, noted as Q_1 was applied to select an action and another Q-network, noted as noted as Q_2 was employed to estimate the target Q-value alternatively as Equation 2.21 shows.

$$\begin{aligned} Q(s,a) &= Q_2(s, \arg\max_a Q_1(s_i, a)) \\ Q(s,a) &= Q_1(s, \arg\max_a Q_2(s_i, a)) \end{aligned} \tag{2.21}$$

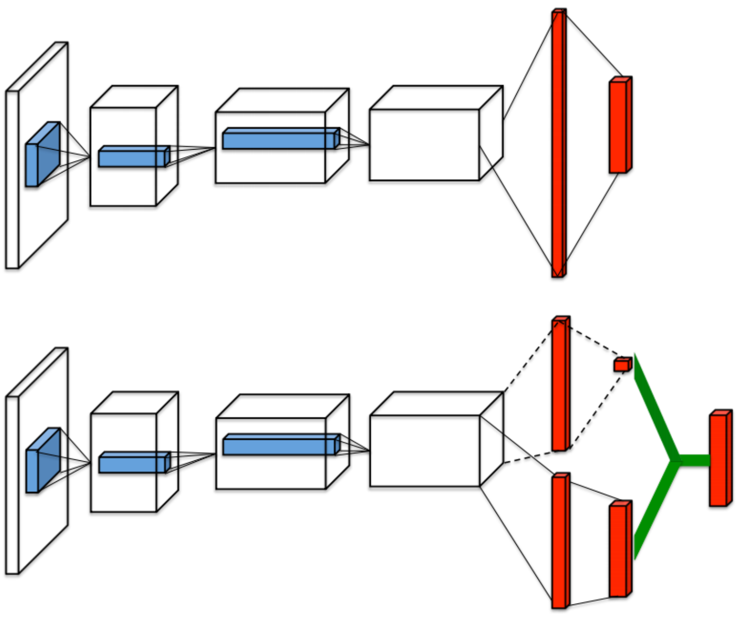

Figure 2.2: A popular single stream Q-network (top) and the dueling Q-network (bottom) (Wang et al., 2015)

Dueling DQN

In some domains, such as Atari game Enduro, actions do not affect the environment in a relevant way (Wang et al., 2015). Therefore, calculating values of state-action pairs is unnecessary. Dueling DQN (Wang et al., 2015) estimates the state-dependent action advantage function rather the Q-value function of state-action pairs. The advantage function of (s, a) pairs is defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

As figure 2.2 shows, the value of state $V^{\pi}(s)$ and the advantage of the state-action $A^{\pi}(s,a)$ are decoupled into two output tensors in the neural network. Through the decoupling, the DQN agent can learn which states are valuable without learning the effect of each action on each state.

Memory prioritised replay

The idea of memory prioritised replay is that some samples in memory should be more important than others. However, these more important samples might be selected less frequently. In vanilla DQN, the agent re-samples steps from its memory using a uniform distribution. Schaul et al. (2015) considered that the agent should extract samples from memory using different weights. The TD-error indicates how surprising the sample is. Therefore, high TD-error transitions should be sampled with high probabilities. Schaul et al. (2015) re-organised samples in memory using a priority queue. Each sample in the priority queue was selected based on its priority.

Distributional DQN

The key idea of Distributional DQN (Bellemare et al., 2017) is to model the distribution of the cumulative reward rather than to model the expected value. If the environment is stochastic and the Q-value follows a multimodal distribution (e.g. bi-modal distribution), the action from $\arg\max_a(s, a)$ may lead to a sub-optimal outcome. The authors proposed the distributional Bellman equation, as shown in Equation 2.22. This uses $Z(s, a)$ to replace $Q(s, a)$ in the Bellman equation. $Z(s, a)$ is the distribution of the cumulative reward. The Wasserstein Metric is applied to measure the distance between $Z(s, a)$ and $R(s, a) + \gamma Z(s', a')$. Bellemare et al. (2017) given the mathematical proof of the convergence of the distributional Bellman equation.

$$Z(s, a) \overset{D}{=} R(s, a) + \gamma Z(s', a') \tag{2.22}$$

There are two main benefits of the Distributional DQN. Firstly, the agent can take the distribution of Q-value into account in selecting an action rather than considering only the maximum Q-value. Secondly, even if the expected cumulative rewards are the same, their variances might be very different. In the literature on finance, the variance is regarded as the risk; people are generally risk-averse which means that actions with lower variances should be selected, if their expected values are the same.

Noisy DQN

Fortunato et al. (2017) proposed a new exploration method that adds zero-mean Gaussian noise to the neural network. This is a new approach to balance exploration and exploitation. Compared with $\epsilon - greedy$, the noisy network could yield substantially higher scores for a wide range of Atari games.

Rainbow

Hessel et al. (2017) examined 7 scaling DRL algorithms in Atari Games. The combination of all 7 improved techniques outperformed each algorithm. Hessel et al. (2017) provided an overview of the current development of DQN. DQN is the general framework for optimal control with raw data. This algorithm can improve a good performance in most of the 57 Atari games, but not in every game. The games that are unable to benefit from DQN in Atari, such Montezuma's Revenge, indicate that learning without knowledge has limitations under some circumstances.

DQN and its improved algorithms have a low sampling efficiency. For example, the DQN agent needs to learn over 4 million frame images to approach the optimal policy in Atari games. In a real application, samples are limited because of the scarce resources of agent-environment interactions. Therefore, reducing the number of interactions with the environment is one of the pressing topics in the field of DRL.

2.7 Policy gradient

Another approach in RL is to directly search for the policy with a maximised cumulative reward. Compared with value-based approaches, policy gradient algorithms do not keep the value function. The parameterised policy is able to deal with continuous state space and action space. Policy gradient algorithms regard RL as an optimisation problem.

2.7.1 REINFORCE

Note the policy as $\pi_\theta(s)$. τ denotes the episode, a state-action sequence $< s_0, a_0 > ... < s_H, a_H >$. The definition of cumulative reward is $R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$. In policy gradient methods, the goal of RL is to find a parameter θ that can maximise the expectation of the cumulative reward of episodes, which is defined by equation 2.23:

$$\theta = \arg\max_{\theta} \sum_{\tau} P(\tau;\theta)R(\tau) \tag{2.23}$$

The parameters of the policy, θ, can be optimised by gradient ascent methods. The gradient w.r.t θ is defined as Equation 2.24:

$$\begin{aligned}
\nabla_\theta \pi_\theta =& \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \\
=& \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_{\tau} \nabla_\theta P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
=& \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau) \\
\approx& \frac{1}{m} \sum_{i=1}^{m} \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned} \tag{2.24}$$

REINFORCE (Williams, 1992) is an earlier algorithm for gradient descent methods. It can generate an episode with policy $\pi(a|s,\theta)$. For each episode, it uses the cumulative reward G to derive the gradient of the current policy π_θ as per Equation 2.25.

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{2.25}$$

2.7.2 Reducing the variance of the gradient

The weakness of using REINFORCE is the high variance of the gradient which results in unstable training. An effective method to reduce variance is to introduce a constant baseline.

When the cumulative reward subtracts the constant, the gradient stays the same whilst the variance of the gradient can be reduced.

Proof:

$$\nabla_\theta E[R(\tau) - b] = \nabla_\theta E[R(\tau)] - \nabla_\theta E[R(b)] \tag{2.26}$$

$$\nabla_\theta E[R(b)] = \nabla_\theta \sum_\tau P(\tau;\theta) \cdot b = b \cdot \nabla_\theta \sum_\tau P(\tau;\theta) = b \cdot \nabla_\theta 1 = b \cdot 0 = 0 \tag{2.27}$$

Another method to reduce the variance of the gradient is rooted in the idea of 'reward to go' (Wu et al., 2018). This comes from the fact that $\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$ should only depend on the step after $< s_t, a_t >$. The gradient $\nabla_\theta J(\theta)$ has been modified as Equation 2.28 shows.

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) Q_\pi(s_t^{(i)}, a_t^{(i)}) \tag{2.28}$$

2.7.3 Actor-Critic

In the REINFORCE algorithm, a whole trajectory needs to be collected to get the cumulative reward and to compute the gradient of the policy. Therefore, TD-learning can be introduced into REINFORCE to improve learning efficiency. The Actor-Critic algorithm is a strategy combining policy gradient and TD-learning.

The actor is a stochastic policy, π_θ, which delivers actions to be executed. The critic, parameterised by w, is the value function used to estimate the value of the current policy. Figure 2.3 shows the Actor-Critic algorithm. For each $< s, a >$ step sample from interaction with the environment, the algorithm fits the value function as Equation 2.29 shows.

$$min_\phi (r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2 \tag{2.29}$$

It then uses the estimated value function to derive the gradient of the policy as Equation 2.30

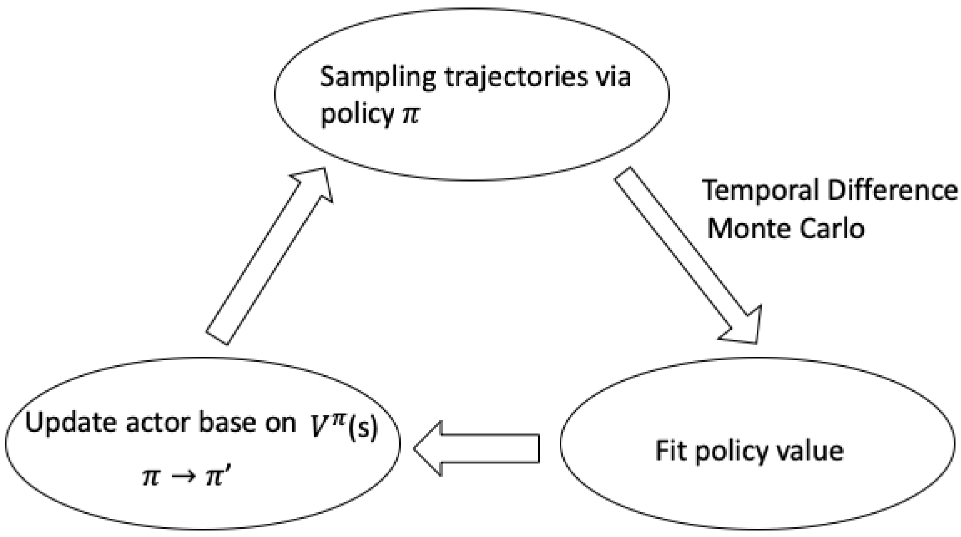
Sampling trajectories via policy π
Temporal Difference
Monte Carlo
Fit policy value
Update actor base on $V^{\pi}(s)$
$\pi \to \pi'$

Figure 2.3: Actor-Critic learning process

shows. $A_\pi(s_t, a_t) = Q_\pi(s_t, a_t - V_\pi(s_t))$ is called the advantage function of $< s_t, a_t >$.

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})A_\pi(s_t^{(i)}, a_t^{(i)}) \tag{2.30}$$

Finally, the policy π is updated by the policy gradient $\nabla_\theta J(\theta)$.

2.7.4 Off-policy gradient descent

The obvious weakness of the on-policy algorithm is its low sample efficiency. When the agent updates policy parameters from θ to θ', it needs to renew all samples that are used to estimate the policy gradient. The current technique to deal with this challenge is off-policy learning. To reuse samples, the off-policy idea is applied in policy gradient algorithms.

Importance sampling is the foundation of the mathematics of off-policy learning. Equation 2.31 exhibits how importance sampling estimates the mean of distribution p(x) by samples taken from another distribution function q(x).

$$E_{x\sim p}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}] \tag{2.31}$$

Although the expectations of $E_{x \sim p}[f(x)]$ and $E_{x \sim q}[f(x)\frac{p(x)}{q(x)}]$ are the same, their variances are different.

$$Var_{x \sim p}[f(x)] = E_{x \sim p}[f^2(x)] - (E_{x \sim p}[f(x)])^2$$

$$Var_{x \sim q}[f(x)\frac{p(x)}{q(x)}] = E_{x \sim q}[f^2(x)(\frac{p(x)}{q(x)})^2] - (E_{x \sim q}[f(x)])^2 = E_{x \sim p}[f^2(x)\frac{p(x)}{q(x)}] - (E_{x \sim p}[f(x)])^2$$

Therefore, when the distance between $p(x)$ and $g(x)$ increases, the variance of $E_{x \sim q}[f(x)\frac{p(x)}{q(x)}]$ tends to become higher.

An off-policy algorithm estimates a policy using samples from different policies. The distance between these different policies is too large to obtain a correct estimated value. Therefore, in off-policy gradient algorithms, the learning rate cannot be too high, as it would results in a huge gap between the old policy and the updated policy. To control the learning rate, Schulman et al. (2015) proposed the trust region optimisation algorithm (TRPO) which refines the loss function through adding a regular term, the KL distance between two policies.

$$\begin{aligned} &E_{s \sim \theta_{old}, a \sim q}[\frac{\pi_\theta(a|s))}{q(a|s)} Q_{\theta_{old}}(s,a)] \\ &\text{subject to } E_{s \sim \theta_{old}, a \sim q}[\frac{\pi_\theta(a|s))}{q(a|s)} D_{KL}(\pi_\theta(s,a)||\pi_{\theta_{old}}(s,a))] \end{aligned} \tag{2.32}$$

Building on the TRPO, proximal policy optimisation (PPO) (Schulman et al., 2017) reduces TRPO's complex computation using the clip loss function defined in Equation 2.33.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta))\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)] \tag{2.33}$$

2.7.5 DPG and DDPG

In some control domains, the action space is continuous. Thus, the $\max_a Q(s,a)$ operator cannot be used. Deterministic policy gradient (DPG) (Silver et al., 2014) is an actor-critic style algorithm for continuous action tasks. Equation 2.34 represents DPG's policy gradient.

$$\nabla_\theta J(\theta) \approx E_\beta[\nabla_a Q_\phi(s_t, a)|_{a,\pi(s_t)} \nabla_\theta \pi_\phi(s_t)] \tag{2.34}$$

Lillicrap et al. (2015) combined DPG with neural network, memory replay and target network to develop a refined algorithm that is called deep DGP (DDPG).

2.8 Summary

This chapter provides a detailed review of Markov properties and the Markov decision process (MDP). Agent-environment interaction is defined as an MDP. For an MDP with a known transition function, dynamic planing could be used as the solution.

In a model-free RL setting, the transition function is unknown. The agent estimates the value of the current policy using samples from agent-environment interactions. The current policy could be improved with a greedy operation. Policy estimation and improvement operation are executed alternately. This method is called generalised policy iteration. The value iteration integrates estimation and improvement in one step. After enough interactions, the agent approaches the optimal policy.

Following the structure of value iteration, TD-learning updates the temporal difference between values of the target policy and the current policy to estimate and improve the current value in one step. In the SARSA algorithm, the estimated policy and the policy that generates samples are the same. Accordingly, the SARSA algorithm is also called on-policy learning. However, in Q-learning, an off-policy algorithm, the estimated policy is the optimal policy while the policy generating samples is an $\epsilon - greedy$ policy. Policy gradient, a policy-based RL algorithm, directly estimates the gradient of a policy using sampling and can approach the local optimal policy by a gradient descent algorithm. Actor-Critic combines TD-learning and policy gradient to improve the sample efficiency by reducing the variance of the gradient.

To deal with challenges from the curse of dimensionality of state and action space, function approximation techniques are introduced in reinforcement learning. Classic function approximation techniques include tile coding and linear regression. In recent years, deep learning has become incorporated into the field of reinforcement learning to integrate the benefits of deep learning into RL, which has developed as DRL.

DRL provides an end-to-end technique to learn human-level control in a dynamic environment from raw data; for example, using images as states. The state-of-the-art algorithms in DRL have been reviewed in this chapter. The table below summarises the algorithms discussed in this chapter.

Approach	On/off policy	Value/policy based	State space	Action space
Tabular Q-learning	Off policy	Value-based	Discrete	Discrete
DQN and derives	Off policy	Value-based	Discrete/Continuous	Discrete
REINFORCE	On policy	Policy based	Discrete/continue	Discrete
TRPO/PPO	Off policy	Actor-critic	Discrete/continue	Discrete
DPG/DDPG	Off policy	Actor-critic	Discrete/continue	Continue

Research above improve the knowledge free RL algorithm in various perspectives. In practice, there are numbers of external knowledge such as demonstrations from human experts can be used to improve the RL. In Chapter three, studies of improve samples efficiency an reduce the number of the agent-environment has been reviewed.

Chapter 3

Reinforcement Learning with External Information

3.1 Key challenges

There are two key research questions in RL with external information:

- What kinds of information can be used?
- How can information be injected into the RL loop?

In this chapter, previous studies that have examined these two questions are reviewed. We summarise different sources of knowledge that could be applied in RL, including demonstrations of human experts, heuristic rules, online advisers and transfer policies from similar tasks. We also review relevant techniques that can inject knowledge into the RL loop.

There are two main challenges for RL with external information. First, the quality of the external information is unknown and may be sub-optimal. Specifically, heuristic rules may not be applied to all states; demonstrations from experts have errors and noise; and, transfer policies may not be suitable for some states. Second, external information from multiple sources may be in conflict. In some states, different external information suggest different actions. Our goal is to leverage sub-optimal information for biased exploration at an early stage of the learning process and avoid misguiding the RL agent when it approaches the goal. A well-developed knowledge injection technique would not change the optimal policy of the original task. Rather it would significantly improve sampling efficiency.

Imitation learning, as a relevant area, is also reviewed. In existent research, supervised learning has been widely applied to learning the mapping from a state to an action (Lee, 2017; Argall et al., 2009). However, this approach ignores the relationship of state-action pairs in one trajectory and breaks the assumptions of independence and identical distribution (I.I.D) of samples (Argall et al., 2009). At the same time, supervised learning is not able to discover the true intentions of demonstrators.

3.2 Sources of Information for RL

3.2.1 Heuristic rules

Exploring the environment based on the knowledge of our predecessors is one of the most important characteristics of human beings. Information from heuristic rules is knowledge that is derived from similar tasks that have been undertaken by previous humans. It is used to teach inexperienced newcomers and guide them in learning the lessons of our predecessors, so that newcomers do not need to start from scratch.

Most heuristic rules originate from perceptions, and there is no strict mathematical proof. Therefore, they may not be correct. A heuristic rule can be considered as a mapping from a state to an action. It may be sparse, which means that not all states have suggested actions from heuristic rules. There are many studies focusing on heuristic rules for RL. Bianchi et al. (2004) and Celiberto et al. (2007) encoded heuristic rules to map from $< state, action >$ to a real number. Devlin et al. (2011) investigated the role diversification of players in RoboCup Soccer by encouraging specific behaviours in the agent. Efthymiadis and Kudenko (2013) employed human strategies in StarCraft game.

3.2.2 Online Advice

When one or more human experts and agents online provide actions to the RL agent, there will be online advice. 'Teaching Agents Manually via Evaluative Reinforcement + Reinforcement Learning' is a framework that the RL agent could greatly benefit from a human trainer's feedback in learning process (Knox and Stone, 2010). Loftin et al. (2016) demonstrated that considering the strategy of the online trainer could learn with less feedback than algorithms based on numerical feedbacks.

3.2.3 Policy transfer from similar tasks

Transfer learning is a sub-area of machine learning that applies knowledge of one problem to a different but related problem (Taylor and Stone, 2009). Transfer reinforcement learning, as a sub-topic of transfer learning, is about reusing policy from one task to another task. As transfer RL can gain benefits from the similarities which exist between the source task and the target task, it can avoid learning from scratch each time and speed up the learning process. For example, in the classic control domain, the mountain car (2D) and the extended 3D mountain car share similar characteristics; so the learning of the mountain car can benefit that of the extended 3D mountain car using transfer RL (Taylor and Stone, 2009).

Taylor and Stone (2009) proposed there are three steps involved in transfer reinforcement learning :

- Select one or a set of source tasks appropriate to the special target task.
- Build the relationship between the source task and the target task.
- Transfer knowledge from the source task to the target task.

To measure the performance of transfer RL, There are propose 5 metrics (Taylor and Stone, 2007).

- Jumpstart: The initial performance of the agent improved by transfer learning
- Asymptotic Performance: The final performance of the agent enhanced by transfer knowledge
- Total Reward: The area under the learning curve
- Transfer Ratio: The ratio of the total reward with transfer learning to that without transfer learning.
- Time to Threshold: The learning time in achieving the specified performance

3.2.4 RL from Demonstrations

Demonstrations are records of agents' behaviour by human experts and other agents. In some complex applications such as an auto-driving system or a video game, the state-action space is very large. As a result, it is difficult to find a policy with acceptable performance using limited computational resources. Demonstrations from human domain experts can be used to guide the RL agent in a biased exploration of the state-action space and improve the sample efficiency. However, demonstrations are sub-optimal and in conflict; they may not cover all the state-action space. Therefore, RL cannot rely only on demonstrations, it also needs to learn knowledge from interacting with the environment.

Early research on demonstrations such as Schaal et al. (2003) focused on supervised learning, also called learning from demonstration (LfD) or behavioural cloning. It means learning via a mapping from a state to an action. However, because these approaches ignore the sequential information of demonstration data, they represent a short-term view on a single step and cannot discover any real intentions of demonstrators. Therefore their generalisation ability is very weak. Additionally, the quality of demonstrations limits the performance of LfD (Atkeson and Schaal, 1997; Argall et al., 2009).

The reinforcement learning from demonstration (RLfD) algorithm combines RL and LfD, and thus learns from both agent-environment interactions and demonstrations. Brys et al. (2015)

encoded demonstrations as a potential function to reshape the reward function. They computed a Gaussian distance between the current state-action and the nearest demonstration with the same action, and used the Gaussian distance as a potential function. This approach can speed up the learning process in a domain with sparse rewards and does not change the optimal policy of the original task. Suay et al. (2016) applied the relative entropy IRL proposed by Boularias et al. (2011) to learn an estimated reward function, which was then used as a potential function to reshape the original reward function.

To break correlations of different steps, state-action pairs sampled from memory are applied to update the neural network under a deep RL scenario. Reinforcement learning from imperfect demonstrations (RLfID) by Gao et al. (2018) sample state-action pairs from a demonstration set and use those samples to update Q-value in the first k learning loop. It then returned to doing the sampling from its own memory. The constant k is a hyper-parameter to balance learning from demonstrations and learning from agents' own experience. Hester et al. (2018) proposed the method of deep q-learning from demonstrations (DQfD) which linearly combined supervised loss, 1-step TD loss, n-step TD loss and L2 regularisation as the total loss function as equation 3.1 illustrates. a_E is the action suggested by an expert E.

$$J_E(Q) = \max_{a \in A}[Q(s,a) + l(a_E, a) - Q(s, a_E)] \tag{3.1}$$

3.3 Injection of information into RL

How to inject external information into the RL loop is one of the most widely studied topics in the field. Taylor et al. (2011b) summarised three directions to transfer exterior policy knowledge into the RL process. The main idea of the first is to give extra value bonuses to an agent when the agent performs the same action as the expert. The second is the 'Extra Action' method, in which the agent can choose a pseudo-action as following external policy (Taylor and Stone, 2007). The pseudo-action is added into the trade-off between exploration and exploiting, thus has a probability of being selected. The last method is 'probabilistic policy reuse', which means

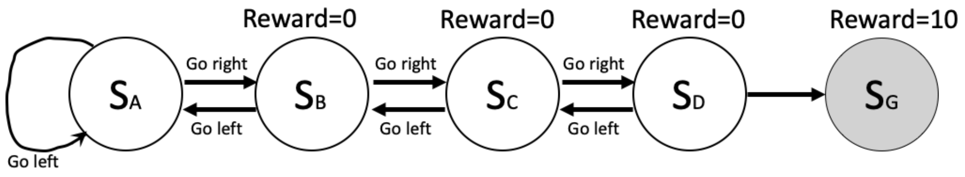

Figure 3.1: Sparse reward MDP

reusing the source policy from the exterior with a decreasing probability (Fernández and Veloso, 2006). Via decreasing and probabilistic policy reuse, the agent's own policy will be replaced by the source policy to select an action. In this section, we follow the three directions to review previous studies on knowledge injection techniques.

3.3.1 Reward shaping

In 1938, the psychologist Skinner first proposed the concept of 'shaping', which was thought to be able to synthesise relatively complex behaviours by guiding animals to perform simple functions (Ferster and Skinner, 1957). By continuously giving a reward (food) to behaviour that is constantly approaching the desired behaviour (e.g. pigeons move to a selected location), the pigeons could be directed to a selected location for foraging.

In fact, the reinforcement of learning agents also needs to be guided especially in reward sparse space. For example, some domains only give non-zero reward signals when the task is completed. Without feedback during the process, the agent can only perform uniformly random explorations for a long time. Figure 3.1 shows a sparse reward MDP. It has two actions: going right and going left, either of which will result in the corresponding state with 100% probability. The non-zero reward $+10$ can only be given if the agent approaches the terminal state s_G of the goal. In fact, if the agent explores the MDP uniformly from the first episode, a large number of explorations will be needed to approach completing the goal.

External tips can greatly enhance the learning efficiency of the agent. One intuitive method to deal with reward sparsity is to provide the agent with a small extra reward according to external tips with the purpose of encouraging the agent to reach the goal. The result of this is that reward function has changed as shown in Equation 3.2.

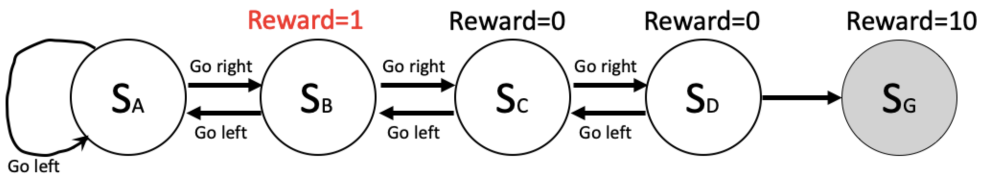

Figure 3.2: Sparse reward MDP

$$R'(s,a,s') = R(s,a,s') + F(s,a,s') \tag{3.2}$$

$R'(s,a,s')$ is a new reward function for the agent. The $F(s')$ is called the shaping function. Changing the reward function, however, may result in a failure to converge to the same policy as the original problem. For example, as Figure 3.2 shows, an extra reward is given to the agent to encourage it to approach the goal state s_G. In order to maximise the cumulative return, the agent will repeatedly jump between state s_A and state s_B. It seems that the agent is cheating and the policy that the agent has learned is not a solution to the original task.

Addressing the challenges above, Ng et al. (1999) proposed the application of the potential function to shape the reward. This defines the shaping function F, as a mapping from state-action-state tuple (s,a,s') to a real number. It follows that one define the potential function, noted as Φ, as a mapping from a state s to a real number:

$$F(s,a,s') = \gamma\Phi(s') - \Phi(s) \tag{3.3}$$

γ is the discount of the MDP. The new reward is $R'(s,a,s') = R(s,a,s') + F(s,a,s')$. The main contribution of the paper by Ng et al. (1999) is that it demonstrates potential reward shaping to keep the optimal policy the same.

The characteristics of reward shaping make it suitable for injecting external knowledge into the RL loop. Brys et al. (2015) applied the Gaussian distance between state-action pairs to encode demonstrations as a potential function. Building on that idea, novel methods called Introspective Reinforcement Learning are proposed by the thesis, which are discussed in subsequent chapters of this thesis. Introspective Reinforcement Learning is about not only using

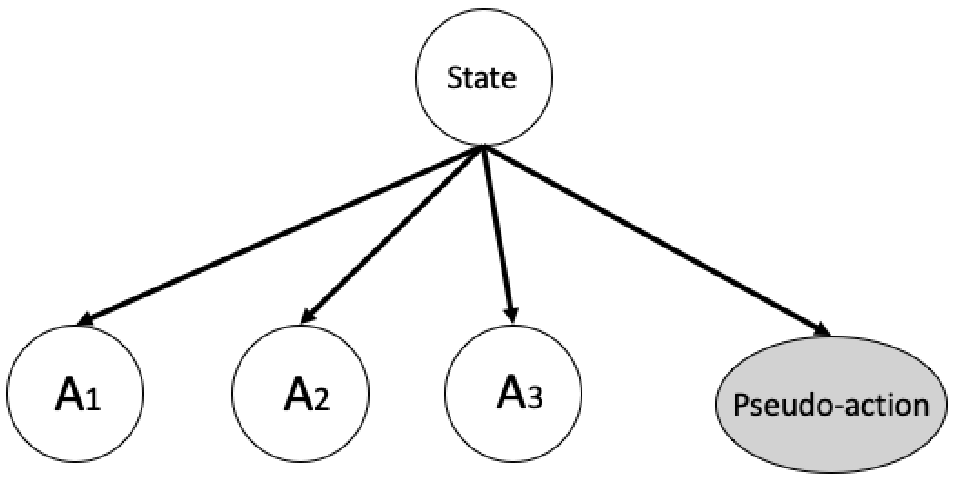

Figure 3.3: A pseudo action with real actions

external demonstrations, but also keeping and updating high Q-value samples by filtering out low Q-value samples from memory. The internal demonstrations are used to shape the reward functions in accordance with the approach furthered by Li et al. (2018).

3.3.2 Pseudo-action

The key idea of pseudo-action is adding an action to the original MDP to represent the action that follows the suggested actions of the external policy (Taylor and Stone, 2007). As Figure 3.3 shows, the Q-table has one more action to represent the pseudo-action.

The disadvantage of a pseudo-action is that it only deals with one policy. In a real application, multiple conflicting policies always exist. Within the thesis Chapter 5 addresses the concept of RL with conflicting domains and extends the idea of pseudo-actions by proposing a novel algorithm of two-level-Q-learning. Each external policy is considered to be an action consisting of a pseudo-action table (High-Q-table in the algorithm). The RL agent learns the Q-value of pseudo actions using trial and error.

3.3.3 Probabilistic reuse

Reusing the past policy with a probability ϕ is a method of transfer knowledge in new task (Fernández and Veloso, 2006). ϕ is a hyperparameter to balance exploration, expert policy and the agent's own policy. To reduce reliance on the expert policy, the probability ϕ is discounted by a decay rate ν for each iteration as equation 3.5 shows.

$$action = \begin{cases} \pi_{past}(s) & prob.\phi \\ \epsilon - greedy(\pi_{current}(s)) & prob.1 - \phi \end{cases} \tag{3.4}$$

$$\phi \leftarrow \phi * \nu \tag{3.5}$$

Wang et al. (2018) employed both probabilistic reuse and reward shaping to improve RL. They transferred a policy from demonstrations via the k-NN classification to find the k-nearest neighbour demonstration and injected the derived policy into the RL loop by policy probabilistic reuse. In this thesis, continuous space has been discretised into cells. Each cell has one or more demonstrations, each of which votes for its action. The number of votes is then used to encode a potential function to reshape the reward function. These aspects are addressed in Chapters 6.

3.3.4 Pre-training

Yosinski et al. (2014) demonstrated that applying parameters from a previously trained model to initialise a new model helped to speed up the learning process of the new model. Deep RL applies a convolutional neural network (CNN) to offer an end-to-end approach which directly learns the policy from raw images pixels. One drawback of Deep RL is that the agent must learn feature representations from raw images, in addition to policy learning. Therefore, the agent needs to spend a lot of time in training in order to approach a reasonable performance. A pre-training method could be applied to pre-train the Q-value/policy network using an externally labelled dataset. To help the agent quickly learn features of raw images, Cruz Jr et al. (2017) proposed including pre-training hidden layers of the Deep RL using small demonstrations from human experts. After pre-training, the algorithm goes back to interacting with the environment to refine the neural network through real samples. Rajeswaran et al. (2017) also applied small demonstrations from humans to pre-train dexterous manipulation. Since the task was in a high-dimensional space, with 24 dimensions, it was a great challenge to train the agent in a model-free environment. Pre-training can facilitate a reduction sample complexity in a scientific way.

3.4 Imitation learning

Imitation learning is not RL but a similar approach. The goal of imitation is to mimic human behaviour in a given domain. Imitation learning does not require the reward function which, as noted, is difficult to design in a complex task. In this section, we discuss techniques for imitation learning, including behaviour cloning, interactive learning and inverse reinforcement learning.

3.4.1 Behavioural cloning

Earlier research focused on mapping states to actions by supervised learning algorithms that applied demonstrations from human experts as training datasets. These approaches have been called behaviour cloning. The goal of behavioural cloning is to find parameters of policy that minimise loss function L.

$$\hat{\pi} = \arg\min_{\pi \in \Pi} E_{s \sim d_{\pi^*}}[L(\pi(s), \pi^*(s))] \tag{3.6}$$

For each step t, $\pi^*(s_t)$ is the action demonstrated by an expert (class label); $\pi(s_t)$ is the current policy learned from demonstrations; and L is the loss function, like the hinge loss function in support of the vector machine (SVM); it measures the similarity between the learning policy π and the demonstrator policy π^*. d_{π^*} is the demonstration dataset. The candidate policy set, Π, is represented as a parameterised model (e.g. a neural network). This approach does not work well because it suffers from the distribution mismatch problem. The current policy influences the distribution of trajectories. Therefore, every small error in the supervised learning above could cause a distribution mismatch between the expert's trajectories and the agent's trajectories. Online experts could help to correct the distribution mismatch problem.

3.4.2 Interactive demonstrator

Interactive demonstrator algorithms assume that online experts (humans or agents) that can be queried at any state and in any environment are available. In general, interactive demonstrator algorithms always have a loop including three steps.

- Step 1: Conduct supervised learning on demonstrations.
- Step 2: Collect samples from interacting with the environment based on the current policy.
- Step 3: Collect interactive feedback from the online expert.

Ross and Bagnell (2010) proposed the Stochastic Mixing Iterative Learning algorithm (SMILe) to mix policies learned from each iteration. π_i' is the policy trained by the labelled dataset from Step 3. At the end of each iteration, update the agent's policy as $\pi_{i+1} \leftarrow \beta\pi_i' + \pi_i$. Ross et al. (2011) proposed the Dataset Aggregation algorithm to aggregate feedback samples from experts into the training dataset. For each loop, the new labelled dataset, noted as D_i in Step 3, is aggregated into the main training dataset D.

$$D \leftarrow D \cup D_i$$

3.4.3 Inverse reinforcement learning

Inverse reinforcement learning (IRL) is about learning the reward function from demonstrations (Ng et al., 2000). IRL assumes that demonstrations come from an optimal policy and the goal of IRL is to infer the most robust reward function that generates trajectories as given demonstrations. Ng et al. (2000) indicated that reward is ambiguous, because there is more than one reward function that can satisfy demonstrations. Furthermore, as demonstrators' policies may not be optimal (Abbeel and Ng, 2011), the assumption of demonstrations from optimal policies is too strong to be applied in a real applications. Abbeel and Ng (2004) established a max-margin formulation to define a new IRL, apprenticeship learning which can reduce the ambiguity of the reward function. Following this idea, Ratliff et al. (2006), relaxed the assumption of optimal demonstrations with a slack variable.

3.5 Research gaps

Based on the review above, several research gaps were identified. First, building on the research on similarity based shaping (SBS) (Brys et al., 2015), Introspective Q-learning is proposed in this thesis. It is the first algorithm that filters high-value self-demonstrations to reshape the reward function. SBS directly applies the Gaussian distance between demonstrations and the current state as a potential function to shape the reward function. However, Introspective Q-learning builds a filter to obtain higher-value samples from agent-environment interactions as 'self-demonstrations' but also could use external demonstrations to construct the potential function. Furthermore, Introspective Q-learning is able to utilise demonstrations from external policies to initialise the filter. Those external demonstrations can be combined with internal demonstrations from the agent itself to reshape the reward function and improve the sample efficiency of Q-learning.

Second, previous studies did not deal conflicts in demonstrations an explicit manner. For example, pre-training methods Yosinski et al. (2014); Cruz Jr et al. (2017) directly learn the model in the demonstration dataset without considering conflicts between demonstrations. To fill this gap, the Two-level Q-learning (TLQL) algorithm is proposed in this dissertation. Though the pseudo-action approach (Taylor and Stone, 2007) is similar to TLQL, it only considers one expert, which means that it only possesses one pseudo-action. In the TLQL, there are multiple pseudo-actions(experts) in a high Q-table. The high-Q table record cumulative rewards of each state following with every expert.

When the number of experts is less than the number of actions, TLQL will narrow the action space of candidates by following experts. Therefore, it can explore the state-action space in a biased manner and speed up the RL process.

In some states, demonstrations for all conflicted actions exist. Under this circumstance, TLQL is no longer suitable. Existing methods cannot leverage information from these conflicting demonstrations. The Radius Restrained Weighted Voting (RRWV) method is therefore proposed to address this issue. Wang et al. (2018) proposed an approach, transfer with kNN, which is the most similar one to the RRWV. It derive a policy from demonstrations with the k-nearest neighbour(kNN) and applied the derived policy with a probability. In the proposed RRWV, a hyper-parameter radius is used to retrain candidate demonstrations. Instead of using the kNN technique, the softmax function is applied in the RRWV to map the total number of weighted votes to a policy. The algorithm takes distance, frequency, and radius into account, which can better utilise information obtained from the demonstrations.

Table 3.5 shows summary of kinds of RLfD algorithm we reviewed above.

A table of summary of RL from demonstrations

Approach	Demo injection	Conflict resolution	Internal demo
CHAT (Taylor et al., 2011b)	Extra bonus	no	no
SAS (Brys et al., 2015)	Reward shaping	no	no
RLfID (Gao et al., 2018)	Pre-training	no	no
DQfD (Hester et al., 2018)	Mix loss	no	no
PPR (Fernández and Veloso, 2006)	Policy reuse	yes	no
LTQL (Li and Kudenko, 2018)	Policy reuse	yes	no
Introspective Q-learning (Li et al., 2018)	Reward shaping	no	yes
RRWV (Li and Kudenko, 2018)	Policy reuse	yes	no

3.6 Summary

In this chapter, we introduced how to speed up RL using external information. Vanilla reinforcement learning assumes learning solely from environment-agent interactions. However, in real applications, there exists domain knowledge from different sources, including heuristic rules, online advisers, the transfer policies of similar tasks, and human expert demonstrations. Such accumulated knowledge can contribute to biased exploration of the state-action space and approach an acceptable performance level at an earlier stage. Previous research on different sources of extensional information was reviewed.

As we know, human beings are not perfectly rational agents; transferred knowledge is not suitable for all states of the target task; demonstration data may be noisy. External knowledge of a domain may be sub-optimal, in conflict, and have noise. To keep the optimal policy learned from external information the same as the optimal policy of the original task, a novel algorithm needs to be developed to apply external information into bias exploration and to address constraints on external knowledge to converge in the optimal policy.

Additionally, different methods of injecting knowledge into RL loops, such as reward shaping, probabilistic reuse, pre-training and pseudo-action were reviewed in this chapter. For each technique, different perspectives were provided to integrate RL with supervised learning for better performance, with this chapter comparing the strengths and weaknesses of those methods.

Chapter 4

Introspective Reinforcement learning

Efficiency of sampling is a key challenge for successful reinforcement learning. In this chapter, we illustrate how to increase the efficiency of Q-learning by reusing high value samples to reshape the reward function. We apply Gaussian distance between states and the reuse samples (called self-demonstration in the thesis) to shape the reward function. The method allowed the RL agent to not only propagate a reward signal to state-action pairs in trajectories but also to neighbouring state-action for the reward function has been reshaped by neighbouring state-action.

4.1 Research Motivation

Model free reinforcement learning assumes that the agent does not have any knowledge about the task. The goal of the task has been designed by the reward function. However, in the real domain, and because of the lack of domain knowledge, the middle goal or sub-goal of a task is hard to recognise. This phenomenon has been called the sparsity of the reward function'. It means, in practical terms, that for most state-action pairs $< s_t, a_t >$, reward function $F(s_t, a_t) = 0$. A zero reward signal is not helpful for exploring the optimal policy. Therefore, in the spare reward space, the RL agent needs to have a large number of interactions with the environment in which it is being applied in order to learn an acceptable policy.

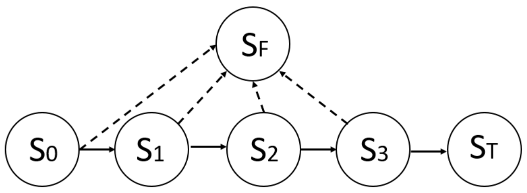

Figure 4.1: 5 states MDP

For example, as Figure 4.1 shows, there are 5 states in the Markov Decision Process (MDP). Each state has two actions: to move to failed state s_F and to move forward the target s_T. The environment is static (the transition probability for each state-action to be equal to 100%) and the reward $+1$ is given only on the target state s_T. If the agent learns the MDP via q-learning with all q-values being at zero, it will need explore 2^4 trajectories in average to approach the target state with an uniform and random exploration.

$Q(\lambda)$ contributes to the mitigation of this challenge by propagating a reward signal to all the state-action pairs within the whole trajectory; Sutton et al. (1998). We called method of $Q(\lambda)$ as the longitudinal propagating of the reward signal'.

In this chapter, we proposal a novel idea that propagates the reward signal horizontally, propagating it to a rearby state. This algorithm builds a priority queue as a filter to keep the performance sample high and then reuses it to shape similar state-action pairs. An external demonstration from a human expert was used to initialise the filter and contributed to the biased exploration of the space of the state-action.

4.2 Assumption

In this chapter, we follow an assumption which is similar to the based optimal action proposed by Brys et al. (2015). Their assumption is that nearby states have the same optimal actions. This is the foundation of propagating reward signals horizontally. Many domains such as classic control and games satisfy this assumption. In Brys et al. (2015), demonstrations are encoded as a reward shaping function by defining a Gaussian similarity measure over the state space:

$$\Phi(s,a) = e^{\left(-\frac{1}{2}(s-s^d)^T\Sigma^{-1}(s-s^d)\right)} \tag{4.1}$$

$s - s^d$ is the Euclidean distance between state s and state s^d. If the demonstration state is the same as the current state $s = s^d$, the Gaussian similarity is 1. The Gaussian similarity will be 0 if two states are sufficiently far apart that they could not influence each other. The co-variance matrix noted as Σ is defined as the sphere of influence for demonstrating the extent to which the state-actions are separated. In my research, the state space has been normalised to $[0, 1]$. Using Σ of the form $\Sigma = \sigma \cdot I$, an identity matrix time was established in which the co-variance matrix was a constant σ. It defines the sphere of influence of the demonstrated state-action and should be tailored according to the individual domain (Taylor et al., 2011a).

$$R'(s,a',s') = \gamma\Phi(s',a') - \Phi(s,a) + R(s',a') \tag{4.2}$$

Equation 4.2 shows that the reward has been reshaped by potential function Φ.

4.3 Methodology of Introspective RL

This section describes the methodology of how to use an agent's own experiences to shape the reward function in order to achieve bias exploration and speed up reinforcement learning. Typically, in complex environments, rewards must be observed many times before the agent acquires a significant behavioural bias towards pursuing those rewards. In the Introspective Reinforcement Learning approach proposed in the thesis, these experiences are leveraged to include a more explicit bias. This is done by shaping the reward function, and rewarding current behaviour that is similar to past behaviour that led to rewards. An alternative way to explain this is to suggest that the function works through those previous experiences that led to rewards being seen as task demonstrations (provided by the agent itself), and that these are then used to bias the agent's current exploration in the same fashion. If actual external expert demonstrations are available, these could even be used to initialise the introspective agent's bias.

4.3.1 Collecting the experiences

Introspective Reinforcement Learning extends RL by adding an experience filter module, and addresses the reward sparsity problem using a dynamic reward shaping approach which is based on the filtered experiences. The experience filter collects the agent's exploratory behaviour that led to positive outcomes into a priority queue. These experiences are then immediately used as an exploratory bias to speed up the learning process.

Specifically, during a learning episode, every state-action-next state-reward tuple (s_t, a_t, s_{t+1}, r_t) is stored in memory. At the end of the episode, the Q-value $\widehat{q}(s_t, a_t)$ for each state-action in this episode is estimated by Monte Carlo until the final time step T:

$$\hat{q}(s_t, a_t) = \sum_{k=t}^{T} \gamma^{k-t} r_t \tag{4.3}$$

The state-action pairs and their estimated Q-values are then stored in a priority queue with the estimated Q-values being the sort key for the queue. If the queue is full (defined by queue size parameter qs), only state-action pairs with a higher estimated Q-value than the smallest in the queue are added (with the smallest being consequently removed). If the queue is not full, all state-action pairs and their estimated Q-value are added.

Progressively, poorer Q-value elements in the queue will be removed, with only experiences with higher estimated performance levels remaining. Thus, the exploratory bias induced by these experiences will progressively increase in quality.

Introspective RL uses prior experience to compute a reward shaping function, rather than using it to replay past experience directly (Schaul et al., 2015). As such, our approach is closer to the automatic generation of a reward shaping function. Further, our approach does not depend on using poor performing actions in the priority queue, which are sorted by Q value (and not TD error as in prioritised experience replay). Furthermore, dynamic reward shaping as defined by Devlin and Kudenko (2012), is applied to experience reuse in our approach, and this has been proven to conserve the convergence guarantees of the algorithm.

4.3.2 Defining the Potential Function

In the manner of Brys et al. (2015), we encode state-action pairs as a potential function using a Gaussian similarity metric. [1] The assumption is that, if in the agent's past experience, certain state-action pairs led to high rewards, taking the same action in a similar state might lead to similarly high rewards.

When the potential function $\Phi(s, a)$ needs to be calculated in a certain state-action pair (s, a), the agent must first look in the priority queue for the recorded state-action pair that is most similar to the current, using Equation 4.1. Then the potential function is defined as follows:

$$\Phi(s, a) = \rho \max_{(s^d, a)} g(s, s^d, \Sigma)\widehat{q}(s^d, a) \tag{4.4}$$

which is a modification of Equation 4.1, incorporating the actual estimated quality of that state-action pair $\widehat{q}$, and a scaling factor ρ to control the strength of the exploratory bias induced. g is the distance between s and s^d defined as shown in Equation 4.5.

$$g(s, a) = e^{-\frac{1}{2}(s - s^d \Sigma^{-1}(s - s^d))} \tag{4.5}$$

Following Ng et al. (1999), we define a potential function $\Phi : S \to R$ over the state space, and take the reward shaping function F as the difference between the new and old states' potential. Doing that maintains the total order over policies, and preserves any convergence guarantees in the manner noted by Ng et al. (1999):

[1]As opposed to the external expert demonstrations used in their work, we encode the agent's own filtered experiences in the potential function.

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s)$$

Prior knowledge can be numerically encoded in the potential function Φ. The effect of applying such reward shaping, which is typically denser than the sparse environment reward, is that the agent is more likely to encounter non-zero rewards, and thus its exploration will move away from uniform random much earlier in learning. Instead, it will be biased towards states with high potentials. For example, using height as a potential function in Mountain car (Singh and Sutton, 1996) biased the agent to select actions that would increase height. Since the goal location in Mountain Car is on the top of a hill, shaping using this heuristic helps an agent solve that task faster.

The definition of F and Φ was extended by others (Wiewiora, 2003), to include actions and time-steps. This allowed for the incorporation of behavioural knowledge that reflects the quality of actions as well as states, and also enabled the shaping to change over time (Devlin and Kudenko, 2012; Harutyunyan et al., 2015):

$$F(s, a, t, s', a', t') = \gamma\Phi(s', a', t') - \Phi(s, a, t) \tag{4.6}$$

These extensions also preserve the total order over policies and therefore do not change the task, given the assumptions by Ng et al. (1999).

Since the agent progressively collects more and more experiences, which, in principle, must be of higher and higher quality, the potential function will change from episode to episode, making this potential function a dynamic one. At the same time, the theoretical guarantees, (i.e. the optimal policy does not change), hold for dynamic potential-based advice as per Harutyunyan et al. (2015), whilst no modification to the policy needs to be made given that the initial Q-function is all zeroes.

4.3.3 Pulling it together

The pseudo-code in Algorithms 6 and 7 describes a Q-learning agent employing the introspective technique to shape its exploration, and shows how all the components interact on an algorithmic scale.

Algorithm 6 Introspective Q-Learning

```
Require: discount factor $\gamma$, learning rate $\alpha$, queue size $qs$
 1: procedure INTROSPECTIVE Q-LEARNING
 2:     initialise value-function $Q$ to all 0.
 3:     initialise the priority queue $PQ$
 4:                                              (empty or with demonstrations)
 5:     for each step of episode do
 6:         $s_t$ is initialised as the starting state
 7:         choose $a_t$ in $s_t$ using $\pi$ derived from $Q$
 8:         SC = empty list     ▷ SC collects all experience tuples (s,a,s',r) in an episode
 9:         repeat
10:             perform action $a_t$
11:             observe reward $r_t$ and new state $s_{t+1}$
12:             choose $a_{t+1}$ in $s_{t+1}$ using $\pi$ derived from $Q$
13:
```

$$\begin{aligned} Q(s_t, a_t) \leftarrow & Q(s_t, a_t) + \alpha(r_t \\ & + F(s_t, a_t, s_{t+1}, a_{t+1}) \\ & + \gamma \max_b Q(s_{t+1}, b) \\ & - Q(s_t, a_t)) \end{aligned} \tag{4.7}$$

```
14:             insert $(s_t, a_t, s_{t+1}, r_t)$ to SC                ▷ Collect steps
15:             $s_t \leftarrow s_{t+1}$
16:             $a_t \leftarrow a_{t+1}$
17:         until $s_t$ is a terminal state
18:         for each $(s_t, a_t, s_{t+1}, r_t)$ in SC do
19:             for $i$ in $[0, 1, .., length(\text{SC}) - t]$ do
20:                 $\widehat{q_t} \leftarrow \widehat{q_t} + \gamma^i r_{t+i}$
21:             end for
22:             FILTER($< s_t, a_t, \widehat{q_t} >, PQ$)
23:         end for
24:     end for
25: end procedure
```

As algorithm 6 shows, line 13 shows that the reward function has been reshaped via demonstration in the filter. The shaping reward is defined in equation 4.6.

There are three key steps to introspective Q-learning. First, collect samples of each episode via

Algorithm 7 Filter High Q-value Experience

```
Require: queue size $qs$
 1: procedure FILTER($< s_t, a_t, \widehat{q_t} >, PQ$)
 2:     if Size of $PQ < qs$ then
 3:         Insert($PQ, < s_t, a_t, \widehat{q_t} >$)
 4:     else
 5:         $< s_e, a_e, \widehat{q_e} > \leftarrow$ the last element of PQ
 6:         if $\widehat{q_t} > \widehat{q_e}$ then
 7:             Remove $< s_e, a_e, \widehat{q_e} >$ from $PQ$
 8:             Insert($PQ, < s_t, a_t, \widehat{q_t} >$)
 9:         end if
10:     end if
11: end procedure
```

a sample collector as SC in line 14. Second, after the end of every episode, estimate the value of each state-action pair in the SC via Monte Carlo method as shown in line 18-21. Last, use a filter to selectively filter the samples. Algorithm 7 describes the inside structure of the filter. The filter is a priority queue that places the lowest value element at the top of queue. Samples will be directly inserted into the queue if the queue is not full (as shown lines 2-3). However, if the queue is full and the value of the current sample is higher than that at the top of queue, the filter will remove the existent top element and insert the new sample.

4.4 Speed up from demonstrations

Even though the introspective reinforcement learning idea is focused on using the agent's own experiences to bias its learning, it is also completely amenable to receiving a prior-bias from demonstrations provided by an external agent. Such demonstrations can easily be incorporated by putting them through the same process of estimating Q-values and storing them in the same priority queue as the experiences collected by the agent itself. When qualitatively good, these demonstrations can prevent the agent from initially filling the priority queue with whatever low-quality random trajectories it executes first; thus it can be positively guided from the first episode.

Specifically, demonstration data from external agents will be collected as a set of episodes, i.e. sequences of state-action pairs that terminate in an end-state. A Monte Carlo estimation of the Q-value $\widehat{q}(s_i, a_i)$ is performed, and the individual state-action pairs (augmented with the estimated Q-value), are inserted into the priority queue. If the demonstration data does not include the reward signal, a viable solution may be to set the reward to 1 for all (s_n, a_n). The Reinforcement Learning from Demonstration work we base our approach on successfully deals with this problem in the same way as Brys et al. (2015).

4.5 Experimental validation

Two domains, CartPole and Super Mario, were selected to demonstrate the strength of the approach proposed in this paper. $Q(\lambda)$-learning and Reinforcement Learning from Demonstration (Brys et al., 2015) were used as benchmarks against which we compared the learning curve of the proposed approach. In CartPole and Super Mario, ten and twenty trial demonstrations from human experts were used respectively to initialise the priority queue and shape the RLfD agent.

The results of all curves have passed a t-test, in which every 100 running results were averaged to be one sample; 30 samples were involved in the test. The t-test was conducted every hundredth step for the samples. All the presented empirical results are statistically different with $p < 0.05$.

4.5.1 CartPole

Cartpole control (Michie and Chambers, 1968) which is also known as Inverted Pendulum', is a pendulum with a centre of gravity above its pivot point as shown in Figure 4.2. The inverted pendulum is unstable. The goal for cartpole is to keep the inverted pendulum balanced by applying appropriate forces to it. Cartpole is considered to be a classical test bed for reinforcement learning (Lillicrap et al., 2015). In the CartPole domain, the RL agent learns to keep the pole balanced by pushing the cart to the left or the right. The observation consists of 4 features: the cart's velocity, its position, the angle of the pole, and its angular velocity.

In the experiments, Q(λ)-learning and reinforcement learning from demonstration (abbreviated

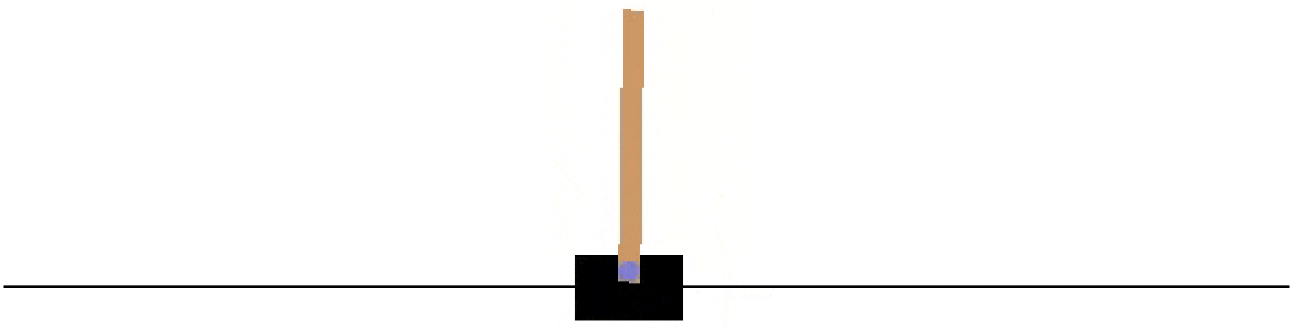

Figure 4.2: Cartpole domain is an inverted Pendulum (Michie and Chambers, 1968)

as RLfD), were used as the benchmarks to be compared to introspective RL with and without demonstrations, to show the advantages of introspection. We set all the parameters in accordance with the work of Brys et al. (2015) whose RLfD method we will compare: learning rate $\alpha = \frac{0.25}{16}$, discount rate $\gamma = 1.0$, $\lambda = 0.25$ and an ϵ-greedy exploration strategy with $\epsilon = 0.05$. Tile coding was used as the function approximation with 16 10×10 tilings. Additionally, the potential-function scaling parameter $\rho = 0.2$ was used.

Figure 4.3 shows the results of comparing plain Q(λ)-learning with introspective RL (without demonstrations). While both methods converge to optimal behaviour, the results show that introspection leads the agent to learn significantly faster than the regular Q(λ)-learning agent. Since the λ parameter chosen in Brys et al. (2015) and adopted by us is quite low, we show experiments with two higher $\lambda \in \{0.4, 0.8\}$ in Figure 4.4. For higher λ, convergence for both methods is significantly faster, but the introspection advantage remains.

In Figure 4.5, we show the effect of external demonstrations on the learning processes. We compare the baseline $Q(\lambda)$-learning without demonstrations, with the RLfD approach provided with 10 different demonstrations, and our Introspective $Q(\lambda)$ agent using the same 10 demonstrations to seed the experience queue. The external demonstrations have sub-optimal performance range of between 450 to 700 steps when keeping the pole up. The results show that while leveraging demonstrations can help, giving RLfD a jumpstart compared to the vanilla $Q(\lambda)$-learner

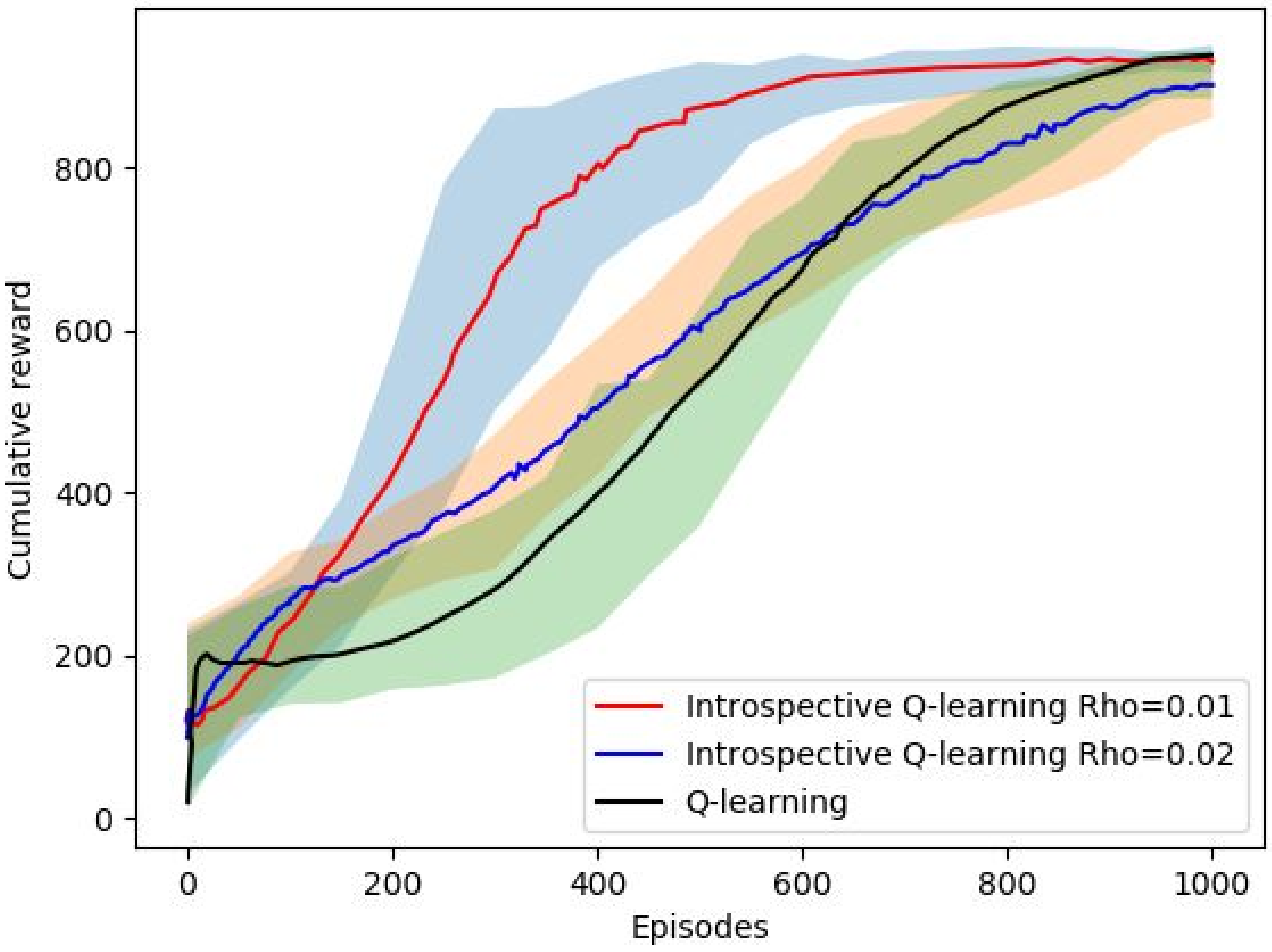

Figure 4.3: CartPole learning curves of Q(λ)-learning and Introspective RL without any external demonstrations. Learning rate $\alpha = \frac{0.25}{16}$, discount rate $\gamma = 1.0$, $\lambda = 0.25$ and an ϵ-greedy exploration strategy with $\epsilon = 0.05$. Tile coding was used as the function approximation with 16 10×10 tilings.

and the introspective agent, the introspective agent very quickly outperforms both other agents. Since the use of demonstrations and the introspective mechanism are orthogonal to each other, we see here that they can be combined to provide a more powerful learner. Furthermore, the introspective mechanism replaces the potentially sub-optimal external demonstrations' exploratory bias as the agent discovers better trajectories by itself. In the static RLfD case, the suboptimal bias remain throughout the learning process. Even though theoretical guarantees for convergence apply to both the dynamic introspective case as the static RLfD case, it is clearly better to dynamically change the bias to include higher and higher quality experiences.

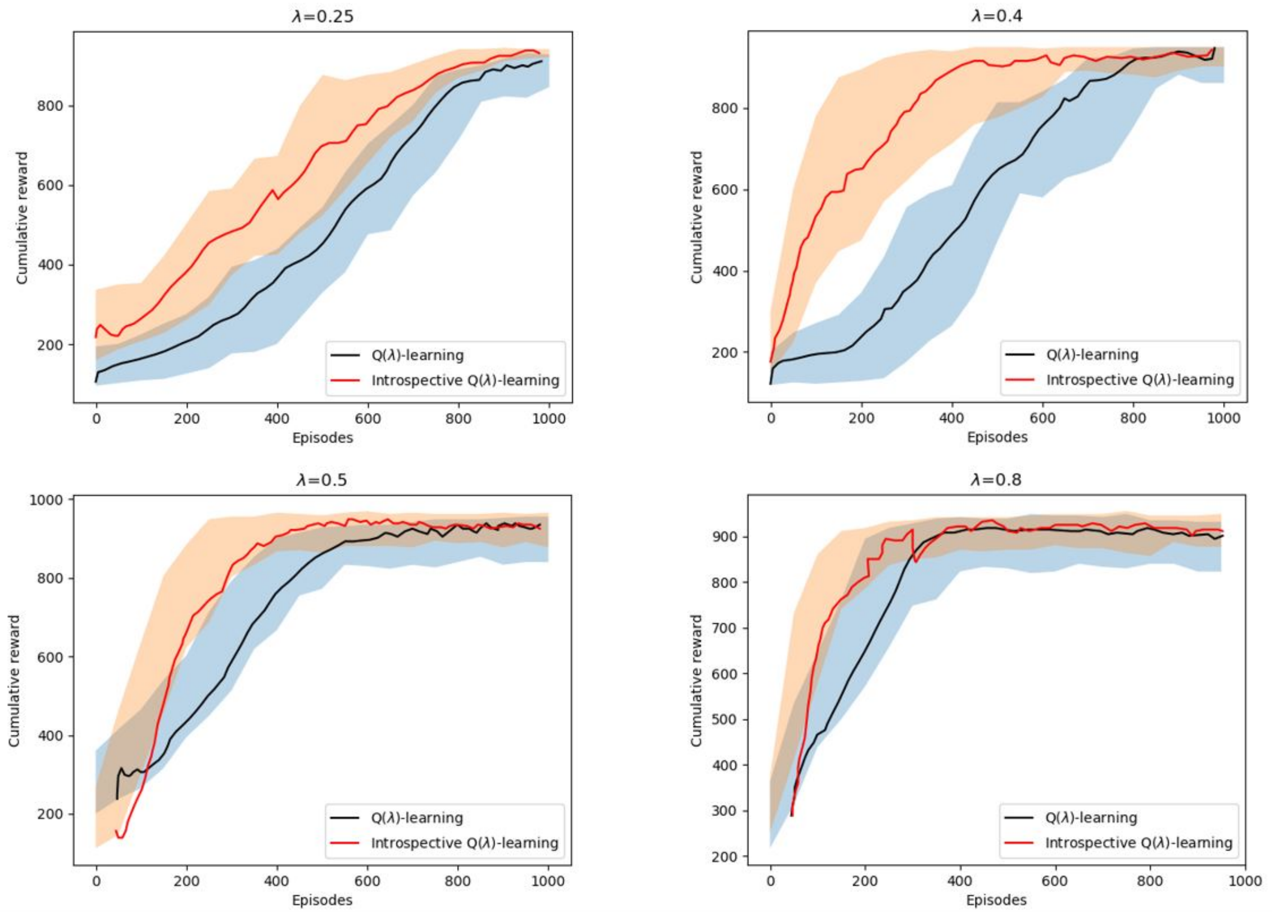

Figure 4.4: CartPole learning curves of Q(λ)-learning, and Introspective RL for different λ. Learning rate $\alpha = \frac{0.25}{16}$, discount rate $\gamma = 1.0$, $\lambda = 0.25$ and an ϵ-greedy exploration strategy with $\epsilon = 0.05$. Tile coding was used as the function approximation with 16 10×10 tilings.

4.5.2 Complex domain application: Super Mario

Super Mario Bros is a famous 2-D side-scrolling video game which was first released by Nintendo in 1985. Karakovskiy and Togelius (2012) converted this game into an AI algorithm competition benchmark. The goal of the game is for the agent to maximise its points. Points are earned for collecting coins, killing enemies, and finishing a game level, while points are subtracted for getting hurt and dying. The game ends when the agent is killed by an enemy, falls off a cliff, runs out of time, or finishes the level. The RL agent's reward corresponds to the points collected in the Super Mario game (e.g. for collecting coins or killing enemies) and to a large extent on being able to complete the level. Negative rewards are given for getting hurt by enemies or falling off a cliff.

The actions available to the Mario agent correspond to the buttons on the original game con-

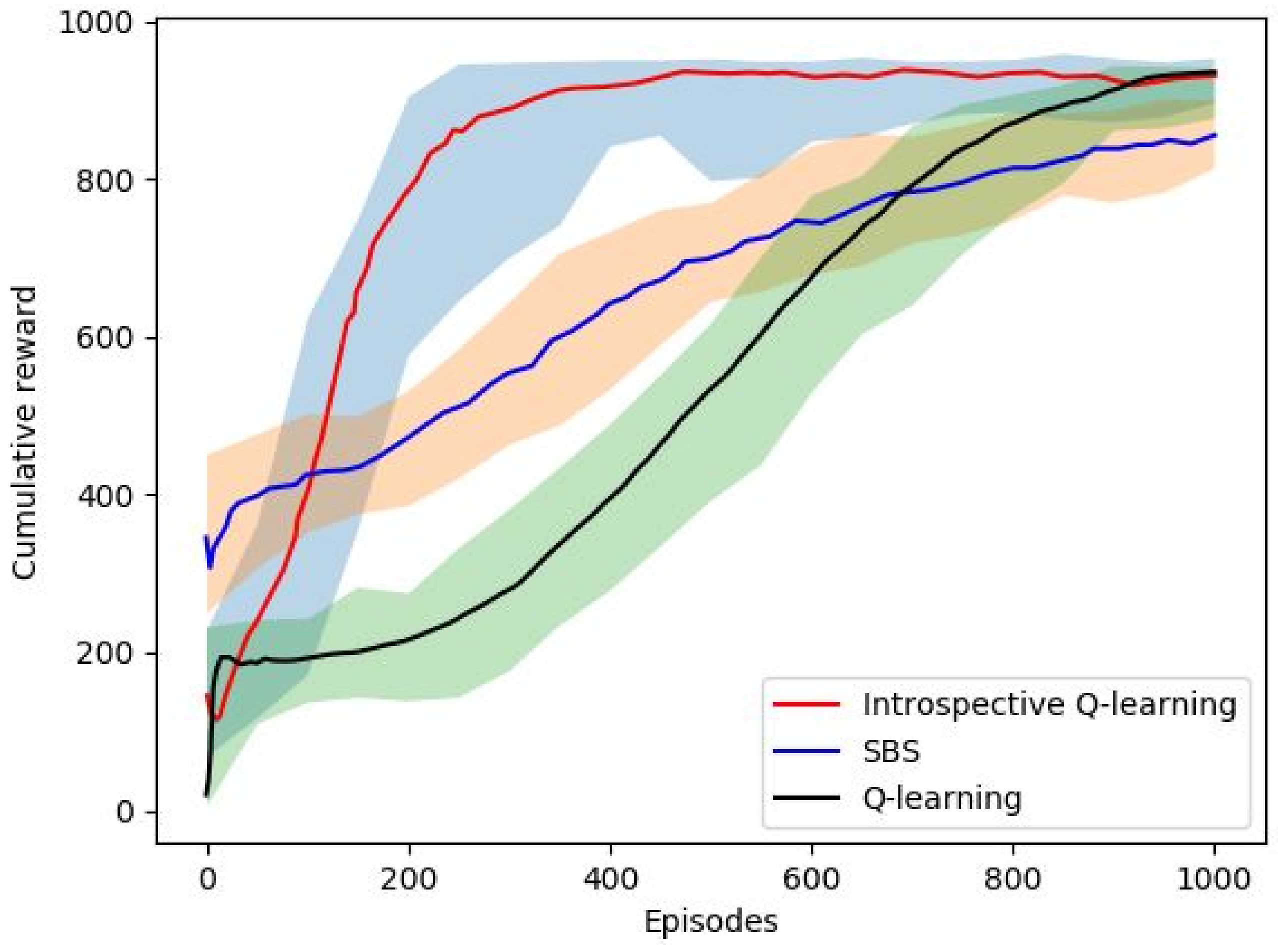

Figure 4.5: CartPole learning curves of Q(λ)-learning, RL from Demonstration, and Introspective RL with demonstration from a human player with a performance score between 400 to 650. Learning rate $\alpha = \frac{0.25}{16}$, discount rate $\gamma = 1.0$, $\lambda = 0.25$ and an ϵ-greedy exploration strategy with $\epsilon = 0.05$. Tile coding was used as the function approximation with 16 10×10 tilings.

troller, which are (left, right, no direction), (jump, don't jump), and (run/fire, don't run/fire). One action from each of these groups can be taken simultaneously, resulting in 12 distinct combined actions. We use the same state-space as described in Brys (2016), which involves 27 discrete state features:

1 is Mario able to jump? (Boolean)

2 is Mario on the ground? (Boolean)

3 is Mario able to shoot fireballs? (Boolean)

4-5 Mario's direction in the horizontal and vertical planes ($\{-1, 0, 1\}$)

6-9 is there an obstacle in one of the four vertical grid cells in front of Mario? (Boolean)

10-17 is there an enemy within one grid cell removed from Mario in one of eight different directions (left, up-left, up, up-right, etc.) (Boolean)

18-25 as the previous, but for enemies within two to three grid cells. (Boolean)

26-27 the relative horizontal and vertical positions of the closest enemy ($(-10, 10)$, measured in grid cells, plus one value indicating an absence of enemies)

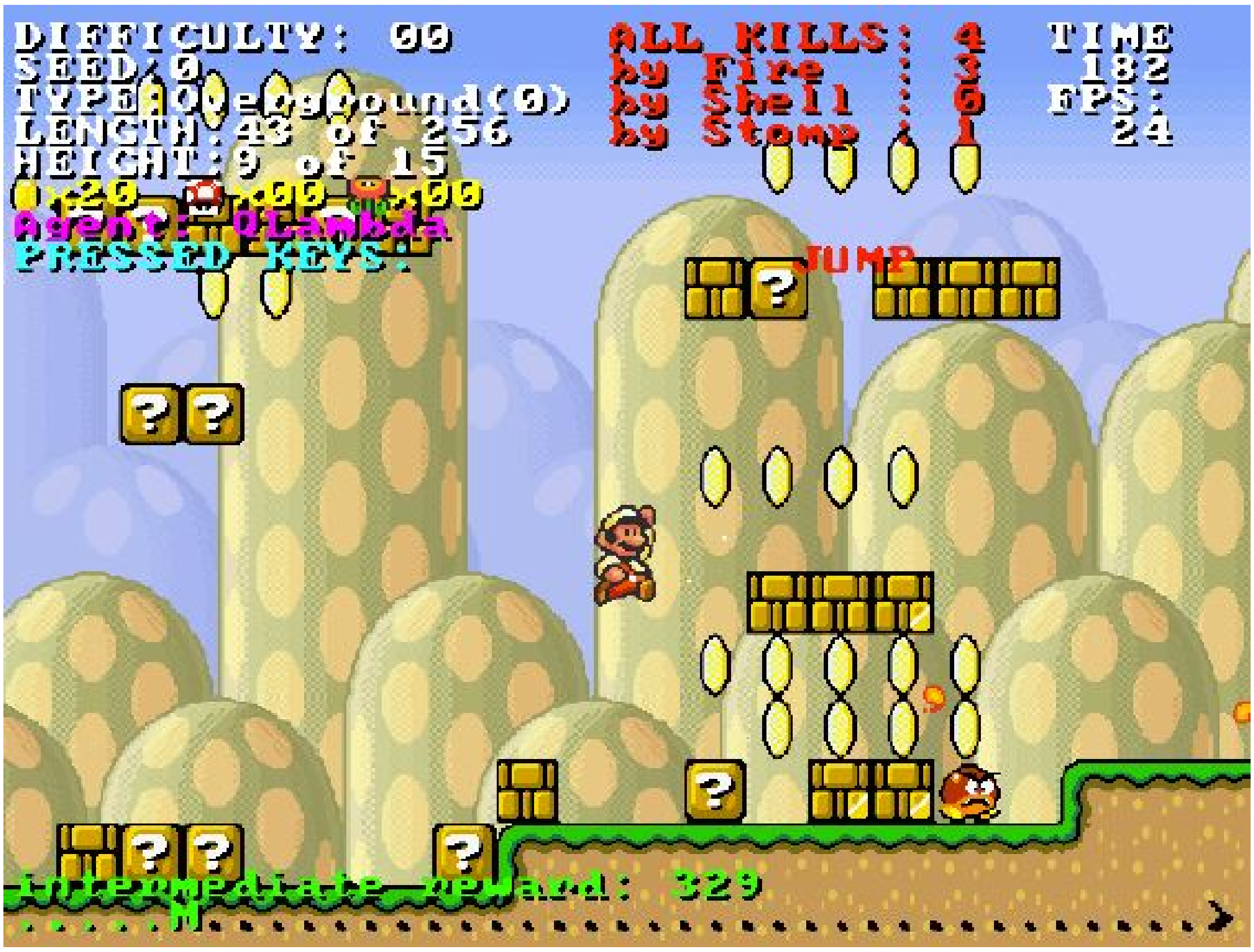

Figure 4.6: Screen shot of Super Mario benchmark (Karakovskiy and Togelius, 2012)

Similar to the CartPole domain, we used Q(λ)-learning and RLfD as benchmarks. The parameters were taken from Brys et al. (2015), with the learning rate $\alpha = 0.001$, discount factor $\gamma = 0.9$, ϵ-greedy exploration with $\epsilon = 0.05$, $\lambda = 0.5$ with an additional $\sigma = 0.5$ for RLfD.

Figure 4.7 shows the learning curves of Q(λ)-learning and the introspective reinforcement learning agent without demonstrations in the Super Mario domain. We show results for two different

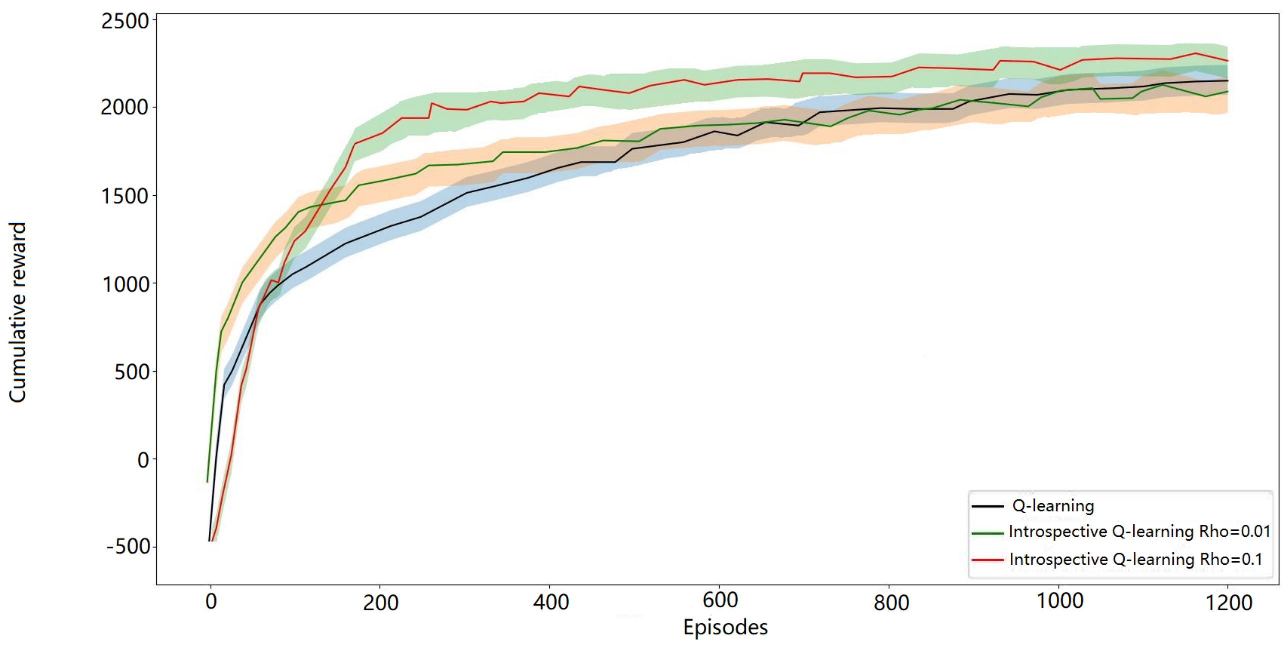

Figure 4.7: Super Mario Domain learning curves of Q(λ)-learning, and Introspective RL without demonstration. Learning rate $\alpha = 0.001$, discount factor $\gamma = 0.9$, ϵ-greedy exploration with $\epsilon = 0.05$, $\lambda = 0.5$ with an additional $\sigma = 0.5$

values of the potential function scaling factor ρ. The results are similar to the CartPole domain in that the learning performance of the introspective reinforcement learning agent without demonstrations significantly improves over that of $Q(\lambda)$-learning, and reaches the asymptotic performance level earlier.

In another set of experiments, shown in Figure 4.8, 20 demonstration episodes from a human player (all with a performance score between 400 to 650) were used to initialise the priority queue for introspective RL. The results show that in this highly complex domain, introspective RL with demonstrations once more outperforms both RLfD and regular $Q(\lambda)$-learning.

4.6 Summary

In this chapter, we presented the idea of introspective reinforcement learning taking inspiration from learning by demonstration. However, instead of using external expert demonstrations to guide the learning, the reinforcement learning agent uses its own experiences that have produced high rewards in the past. The agent keeps a priority queue to record the most

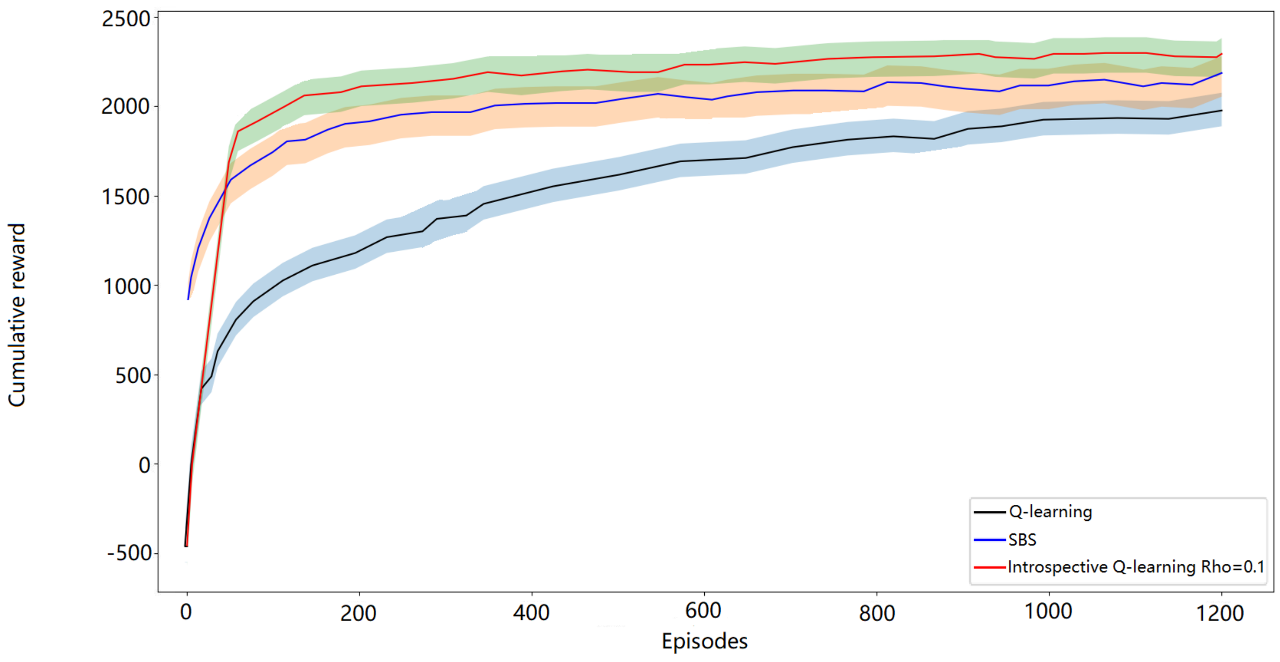

Figure 4.8: Super Mario Domain learning curves of Q(λ)-learning, RLfD, and Introspective RL with 20 demonstration episodes from a human player with a performance score between 400 to 650. Learning rate $\alpha = 0.001$, discount factor $\gamma = 0.9$, ϵ-greedy exploration with $\epsilon = 0.05$, $\lambda = 0.5$ with an additional $\sigma = 0.5$

successful experiences as tuples of the form$\langle s, a, \widehat{q} \rangle$, where the Q value is obtained through Monte Carlo estimation. When the agent executes an action that is the same as the action of the most similar state in the priority queue, a reward based on the product of ρ, $\widehat{q}$, and the degree of similarity is added, where ρ is the hyper parameter that controls the weight of the reward shaping signal. Human experts' demonstrations can be used by injecting the corresponding state-action-q triples into the priority queue. In this way, introspective RL can be complementary to reinforcement learning from demonstration.

We empirically evaluated our introspective RL approach on two domains, namely the Cart-Pole domain with 4 continuous features and the Super Mario domain with 27 discrete features. The results showed that the introspective reinforcement learning agent surpasses regular $Q(\lambda)$-learning performance significantly and reaches the asymptotic performance much earlier. Furthermore, introspective reinforcement learning with demonstrations significantly improves performance in both domains compared to state-of-the-art reinforcement learning from demonstration. The empirical results also show that while introspective RL can use up more computational time per learning step, this loss is counterbalanced by reduced sample complexity; the latter generally being more critical than computational complexity.

Chapter 5

Q-learning from multiple conflict demonstration

In this chapter, we present the challenges that arise from conflicts in demonstration and our approach to dealing with this issue.

5.1 Conflicting demonstrations

Reinforcement learning from Demonstration is an approach that employs experts' demonstrations of solving the target task to guide the reinforcement learning agent (e.g. by biasing the exploration), in order to speed up the learning process and to improve sample efficiency. Current RLfD techniques rely on the quality of the expert demonstrations. An accepted assumption of the above mentioned approaches is that the expert's policy on each state is consistent and beneficial. However, this assumption may be too strong. In most cases, demonstrations can be collected from multiple sources, such as multiple individuals' behaviour records using various heuristic rules. Moreover, the quality of these demonstrations is often imperfect. Facing a same state, demonstrations from different sources may give different action. This phenomenon called conflicting of demonstrations. It follows that there there may be conflicting advice suggested by different experts' demonstrations, and this may occur in a large number of situations.

Introspective Q-learning keeps a priority queue as a filter to record high-performance samples and reuse those samples to reshape the reward function. Demonstrations have been collected together to initialise the filter. However, conflicts among demonstrations have not been explicitly addressed.

In this chapter, we propose a two-level Q-learning (TLQL) approach to deal with the challenge of conflicting domain knowledge among multiple experts' demonstrations. TLQL includes two Q-tables: a high-level Q-table and low-level Q-table. Compared with traditional Q-learning, it uses an additional Q-table to record the performance of experts in each state. During the RL process, TLQL keeps track of both action quality and the reliability of experts in each state, and updates the two Q-tables simultaneously by using feedback signals (i.e. reward) from environment-agent interactions. As a result, TLQL overcomes the problems of learning from multiple demonstrations and thus performs better than state-of-the-art RLfD approaches.

5.2 Learn value of experts

This section first introduces the phenomenon of conflicts between demonstrations from different sources. Following that, our two-level Q-learning algorithm (TLQL) is proposed.

5.3 Multiple domain knowledge sources

In many applications, such as training an agent to play chess from human demonstrations, the knowledge comes from different individuals with different demonstration trajectories, heuristic rules, and so on. Each demonstration may, therefore, produce different actions as well as conflicting with other demonstrations in some states.

Despite this potential problem it is nevertheless beneficial, in early stages of learning, for the RL agent to follow suggestions from experts rather than randomly explore without extra information. The problem is to determine which suggested action from different experts can be

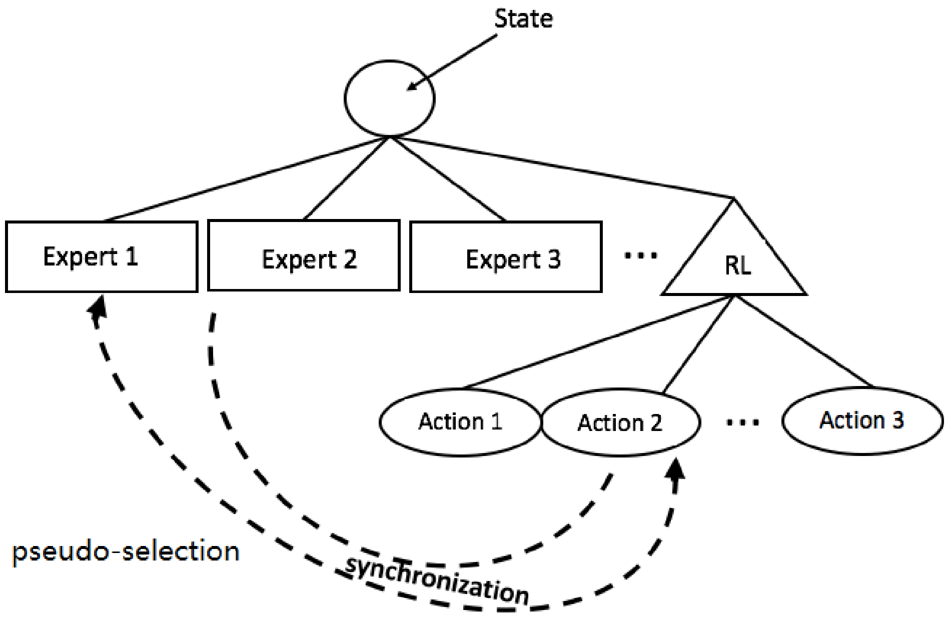

Figure 5.1: The structure of 2-level Q-learning, including two Q-tables: a high-level Q-table and low-level Q-table

trusted in the case of conflicting advice. One feasible idea is to investigate each $\langle state, expert \rangle$ pair by trial and error. The proposed novel method applies Q-learning to expert selection and learns a policy of assigning credit to experts. Combining both Q-learning of experts and Q-learning of actions simultaneously, the agent has the capability to effectively deal with conflicts and improve the sample efficiency of Q-learning even where demonstrations come from multiple conflicting sources.

5.4 Two-level structure of reinforcement learning

Figure 5.1 shows the structure of the proposed novel algorithm: two-level Q-learning (TLQL). This algorithm employs a low-level Q-table and a high-level Q-table. The high-level Q-table is used to store the value of $\langle state, expert \rangle$, representing the trust the RL agent has in the given expert, in the given state. The low-level Q-table is the same as in regular Q-learning, recording the Q-value of state-action pairs.

In the process of agent-environment interaction, the agent first observes the state of the environment and then selects an expert through by an ϵ-greedy policy according to the high-level

Q-table (the RL agent itself is also represented as an expert in this Q-table. If e=RL, execute normal Q-learning). Afterwards, the RL agent executes the action that the selected expert suggests, and finally receives the reward and next state feedback from the environment. The sample of the new algorithm is $\langle s, e, a, r, s' \rangle$, where e is the selected expert and s, a, r, and s', are the same as in regular Q-learning, denoting current state, action, reward and the next state, respectively.

5.5 Synchronised Q-table updating

The TLQL algorithm updates information by exploring both experts and actions with an experience sample $\langle s, e, a, r, s' \rangle$. First, the algorithm updates the low-level Q-value by $\langle s, a, r, s' \rangle$ in the same way as regular Q-learning does.

In order to synchronise information between the high-level Q-table and the low-level Q-table, as well as make full use of the information of every sample, it is also necessary to update both the low-level Q-table and the high-level Q-table. Checking all experts, if the experts have given the same action as the executed action, synchronise the value of lowQ(s,a) to High(s,e). In practical implementation, just update least update lowQ(s,a) to highQ(s,e).

5.6 Pulling it all together

Algorithm 8 shows the pseudo code of TLQL, indicating how all components noted in Figure 5.1 work on an algorithmic scale.

5.7 Empirical Study

In this section, we present the results of the TLQL proposed in three domains: maze navigation, coloured flags visiting and Atari Pong. To demonstrate that TLQL can significantly improve

Algorithm 8 Two-level-Q-learning

```
procedure TWO-LEVEL-Q-LEARNING($\langle s_t, a_t, \widehat{q_t}\rangle$, $PQ$)
    Let $E$ be the set of experts, including the RL agent
    for all s $\in \mathcal{S}, a \in \mathcal{A}$ : $lowQ(s,a) \leftarrow 0$
    for all s $\in \mathcal{S}, e \in E$ : $highQ(s,e) \leftarrow 0$
    for each episode do:
        Initialise $s_0$
        for each step in episode do:
            $\epsilon$-greedily choose $e$ from $highQ$
            action $a$ is the suggestion from expert $e$
            Take action $a$, observe $r$, $s'$
            predicted $\leftarrow r + \gamma \max_{a'} lowQ(s', a')$
            $\Delta_l = predicted - lowQ(s,a)$
            $lowQ(s,a) \leftarrow lowQ(s,a) + \alpha\Delta_l$
            $highQ(s, RL) \leftarrow \max_{a'} lowQ(s, a')$
            for each expert e in E do:
                expert e suggest action a
                $highQ(s,e) \leftarrow lowQ(s,a)$
```

reinforcement learning by utilising demonstrations from multiple conflicting sources, we define several domain experts which each possess different expertise in each domain.

In the domain of maze navigation, the maze is divided into three non-overlapping regions. Each expert is good at moving in a specific region and has no prior knowledge of other regions (note that this fact is known neither by to the experts nor the TLQL algorithm). In the domain of coloured flags visiting, the learning task and demonstration is more complicated. A training agent learns to finish a composite task in a grid world. Each expert is skilled in one sub-task and all the sub-tasks have the same state space. Due to the fact that each expert is only concerned with its own individual target, individual experts may make different decisions in a state. As a result, many conflicted but locally-optimal demonstrations will be recorded.

Unlike hierarchical RL, the partition of the domain is used to simulate the experts' different skills; the agent does not know the partition information in advance. The agent learns the strengths of each expert in high-level Q-learning.

In our experiments, we compared TLQL with two baselines: traditional reinforcement learning (RL) and confidence-based human-agent transfer (CHAT) as per Wang and Taylor (2017). The

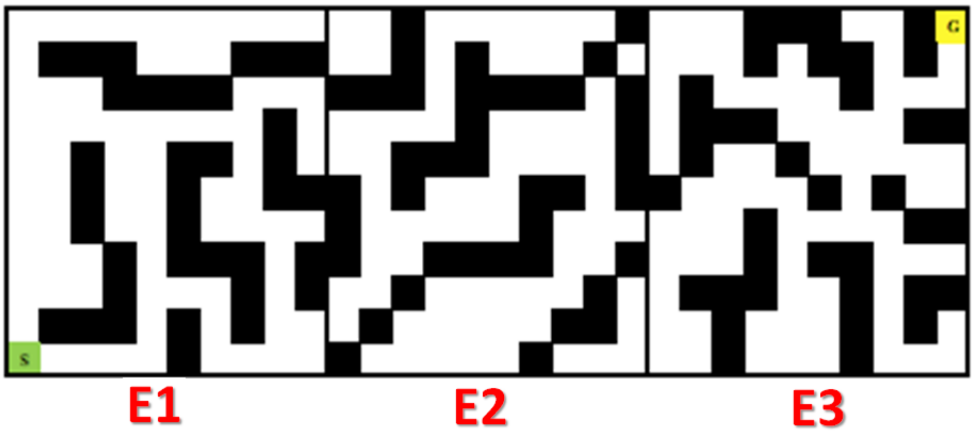

Figure 5.2: Maze divided into three non-overlapping regions

former uses Q-learning approach without any prior knowledge. In contrast, CHAT is the state-of-the-art approach for improving reinforcement learning through demonstration. As CHAT does not consider conflicting demonstrations caused by multiple domain knowledge sources, we adopted a weighted random policy to make choices for CHAT when a contradiction occurred. Specifically, when multiple experts made different decisions in a state, each action was given a weight based on how many experts had suggested this action. Then an action was chosen randomly based on these weight values: actions with higher weights had a probability of being chosen.

All reported results in our experiments were averaged over 100 trials. All result figures display a 99% confidence interval to show statistical significance.

5.8 Maze navigation

The first experiment was with a maze environment as shown in Figure 5.2. The maze consisted of 30×10 states. In each state, the agent has four available actions: up, down, left and right. It could move in one of the four directions as long as there was no obstacle in that particular direction. Furthermore, there is a probability of 0.1 that the agent would fail to move toward its desired direction. The agent's goal was to reach the upper right corner of the maze as soon as possible starting from the bottom left corner. The immediate reward for the agent was 0, unless the agent arrives at the goal state, where it was $+1$. Each episode starts from the initial state S and finished with the goal state G. The parameter settings of Q-learning are the same for all approaches: learning rate α=0.01, discount factor γ=0.99, ϵ-greedy were adopted as the exploration strategy, where ϵ=0.1.

The demonstrations were collected from multiple experts. In this experiment, there were three experts E1, E2 and E3; each claimed that they have enough experience to complete the maze. As Figure 5.2 shows, the maze was divided into 3 areas. Each expert was only a master in one are of the maze. However, the RL agent did not know this in advance. We assumed that E1, E2 and E3 knew the optimal policies for area 1, area 2 and area 3 respectively. They could give optimal actions as the demonstration with a probability of 0.9 in their corresponding areas. Furthermore, because each expert knew nothing about the maze except their area of expertise, they moved randomly in the other two areas. From the perspective of the learning agent, the experts' ability was unknown. Moreover, the learning agent did not know the confidence of experts' demonstrations. When conflicting actions were suggested by different experts, the learning agent was unable to know whose demonstration was right. Rather, the agent learned this, during the RL process.

For demonstration data collection, we generated 20 demonstrations from each expert via behaviour simulations of a complete episode, and removed any duplicates. When applying TLQL to the maze navigation game with conflicting demonstrations, high-level Q-learning was used to teach the agent which expert's demonstration was more reliable in each state. Low-level Q-learning taught the agent to move in the optimal direction through its interactions with the maze.

Figure 5.3 illustrates the learning curves of TLQL and two other baselines: RL without prior knowledge and CHAT. TLQL and CHAT use demonstrations from three experts. The figure clearly show that TLQL significantly outperforms RL and CHAT from several perspectives. Jumpstart was used to measure the average initial performance of the learning agent. A higher jumpstart performance means that the learning agent could benefit more from its prior knowledge in the early stages of the learning. As TLQL can make good use of conflicting demonstrations to train the agent, its jumpstart performance is significantly better than the

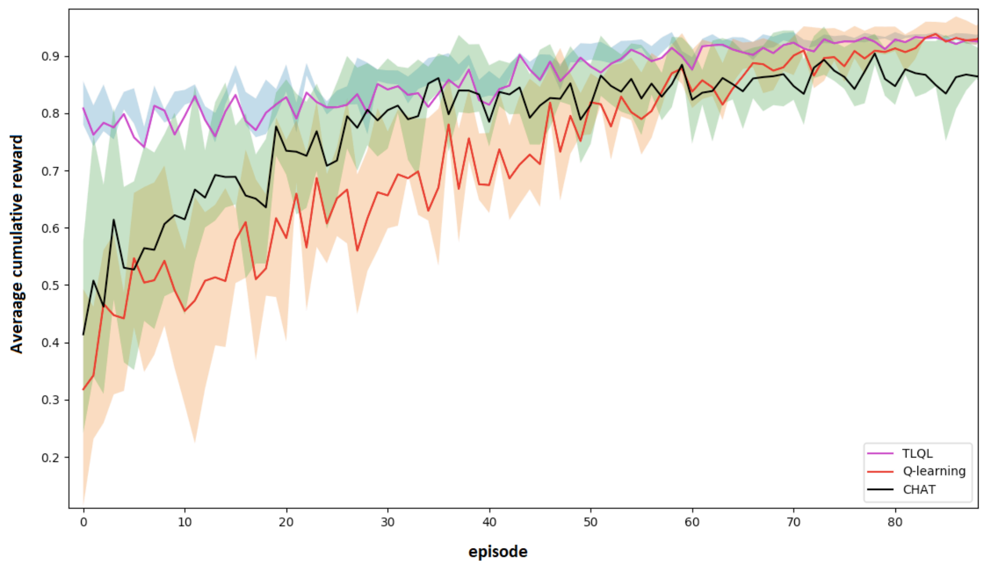

Figure 5.3: Performance comparison of TLQL, Q-learning and CHAT (Wang and Taylor, 2017) in the domain of maze navigation. Learning rate $\alpha = 0.01$, discount rate $\gamma = 0.99$, ϵ-greedy $\epsilon = 0.1$.

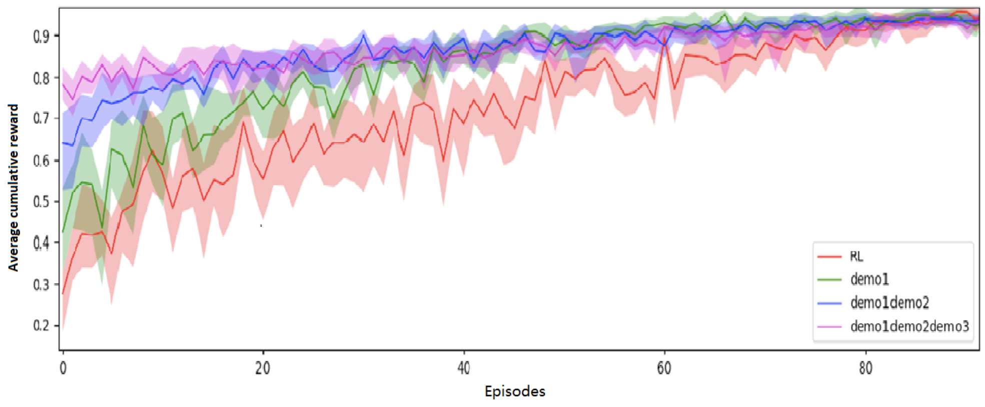

Figure 5.4: Comparison Q-learning of TLQL with 1 expert, 2 experts and 3 experts. Learning rate $\alpha = 0.01$, discount rate $\gamma = 0.99$, ϵ-greedy $\epsilon = 0.1$.

baselines. The overall performance was another metric which was measured by the area under the cumulative reward curve. Figure 5.3 indicates that no matter how many demonstrators are involved in a model, TLQL always performs better than RL and CHAT. In addition, as the agent trained by TLQL can achieve a relatively good performance very quickly, its asymptotic performance is superior to RL and CHAT.

Moreover, we also compared the learning performance of TLQL with different demonstrators. Figure 5.4 shows that the training agent can perform better by increasing the number of demonstrators. Although the inclusion of more demonstrators implies, ceteris paribus, that there will

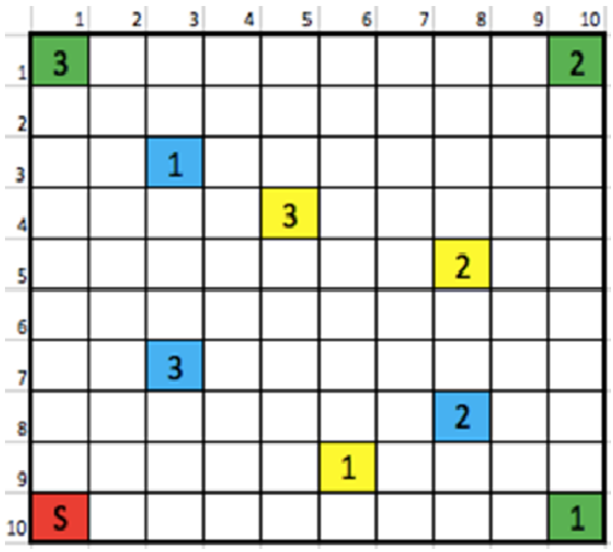

Figure 5.5: Coloured flags visiting problem domain

be more contradictions among demonstrations, TLQL can deal with conflicting demonstrations effectively. Therefore, the learning agent can gain more knowledge from the demonstrations involving more experts.

5.9 Coloured flags visiting

In the previous experiment, all experts shared the same goal (i.e. completing the maze) and they were skilled in different areas of the state space. However, domain knowledge conflicts may occur in more complex scenarios. For example, in the Super Mario game collecting coins, killing enemies and moving towards the goal are three different tasks. Players perform all of these tasks to obtain a higher score, which is the ultimate goal of the game. If we suppose that there are three experts it follows that each of them is only good at achieving one specific task while ignoring the other tasks of the goal. In this case, experts with different knowledge may suggest different optimal actions for the same state. Our two-level Q-learning algorithm can also handle this kind of conflicting demonstration problem.

In our experiment, we chose coloured flags visiting as the domain for the aforementioned scenario. In coloured flags visiting an agent's goal is to visit all flags in a given order in a discrete 10*10 grid world. A picture of this domain is shown in Figure 5.5. There are a total of 9 flags

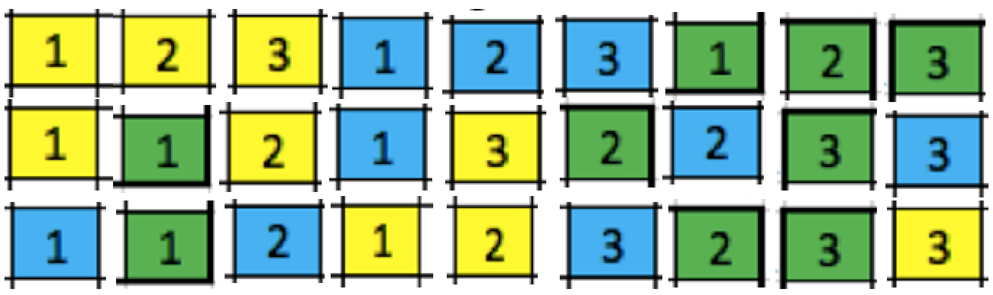

Figure 5.6: Three correct examples sequence of visiting flags

in three different colours, labelled with digits. An agent starts from the "S" position which is located at the bottom left corner of the maze. At each time step, the agent can move in 8 directions: up, down, left, right, left up, left down, right up and right down. There is a probability of 0.1 that the agent's move in the desired direction is unsuccessful and the agent remains in its original position. The agent's goal is to visit all flags as soon as possible and also obey some rules: (1) the agent needs to visit all flags of the same colour in ascending order; (2) for flags with different colours, there is no order requirement; (3) incorrect (i.e. out of order) visits to flag positions are ignored and not taken into account. Figure 5.6 shows three correct examples which satisfied all of the above rules. An entire episode is finished when the agent has visited all flags in the right order. At this juncture, the agent receives a reward equal to $+1$. No other rewards are given during the episode. The state information includes the position of the agent and how many flags of each colour were collected for each colour. As in the first experiment, we set learning rate $\alpha = 0.01$ and discount factor $\gamma = 0.99$ for all approaches. We also adopted ϵ-greedy as the exploration strategy, where $\epsilon = 0.1$.

We defined three experts (denoted by E1, E2 and E3) to provide demonstrations for playing this flag visiting game. E1, E2 and E3 were skilled in visiting yellow, blue and green flags respectively (each expert's goal and the learning agent's goal were not the same. The learning agent was required to visit all flags while each expert only needed to visit the flags of one of the three colours. Thus every expert only focused on how to complete its individual task as soon as possible rather than the ultimate goal of the game. Figure 5.7 depicts expert E1, E2 and E3 and the environment from their individual perspectives.

20 demonstrations, each completing an episode, were generated from each of the three experts,

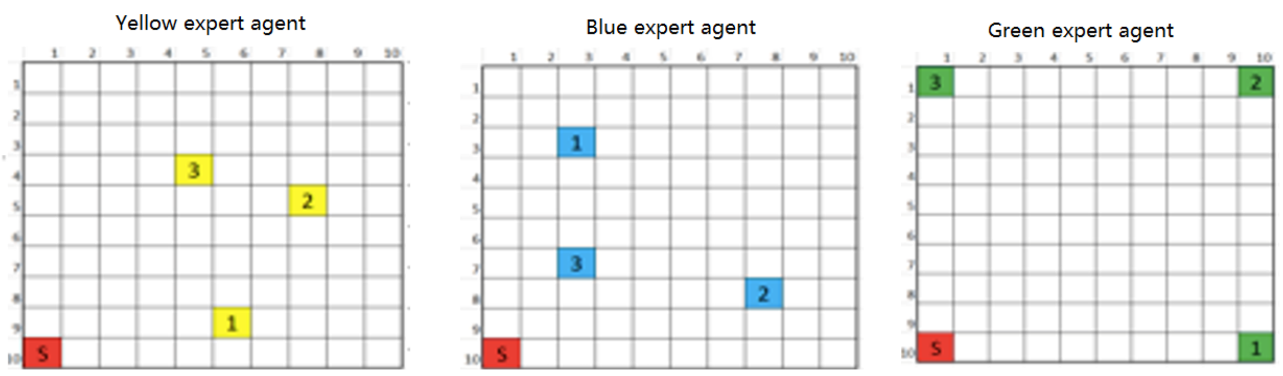

Figure 5.7: Three demonstrators and their view of the environment

with 10% random noise added to each. As E1, E2 and E3 had different goals, they were likely to take different actions for a given cell of the grid. From the perspective of the learning agent, all these suggested actions (i.e. demonstrations) could be helpful for achieving its goal. This was because each one of the suggested actions was an optimal choice for a specific task. Faced with the conflicting demonstrations from E1, E2 and E3, the learning agent needed to learn from these demonstrations and decide what action to take in each state. Incorporating noise into demonstrations, we assumed that the experts could not give the optimal actions with a probability of 0.1, and instead give a random action.

Performance comparisons regarding the cumulative reward and the number of steps of TLQL and the two baselines (i.e. regular Q-learning with no prior knowledge and CHAT) are shown in Figure 5.8. As per the results of experiment 1, the learning curves of TLQL still outperform the curves of regular Q-learning and CHAT. Furthermore, Figure 5.9 shows that the learning performance is bettered when there is an increased number of experts.

The reason for the much higher initial performance with three experts (rather than, for instance, with two experts) is that without having demonstrations of all three experts, there is a crucial part of the task information missing. Just using the advice of two experts is not sufficient to complete the overall task immediately. This is different from the maze navigation domain, where one expert alone can still help the agent to complete the overall task.

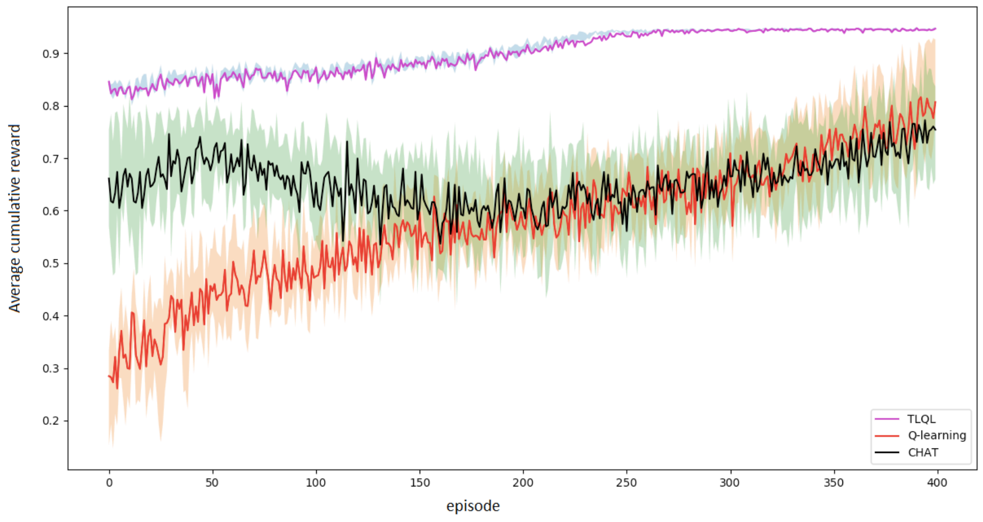

Figure 5.8: Performance comparison of TLQL,CHAT (Wang and Taylor, 2017) and Q-learning in the domain of coloured flags visiting. Learning rate $\alpha = 0.01$, discount rate $\gamma = 0.99$, ϵ-greedy $\epsilon = 0.1$.

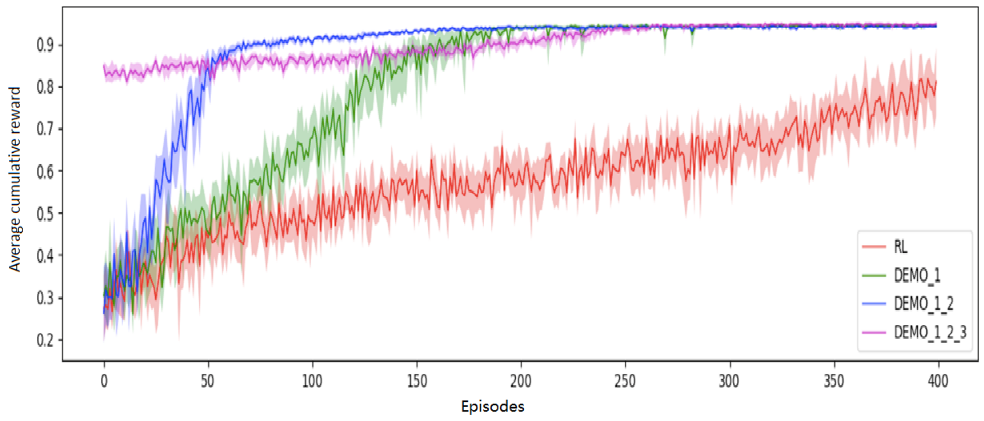

Figure 5.9: Comparison of TLQL with different number of experts and regular Q-learning in the domain of coloured flags visiting. Learning rate $\alpha = 0.01$, discount rate $\gamma = 0.99$, ϵ-greedy $\epsilon = 0.1$.

5.10 Application in complex domain

5.11 Pong

After the Deep Q-learning(DQN) (Mnih et al., 2015) paper was published, Atari games became a popular benchmark for reinforcement learning. We tested TLQL on the Pong 5.10 version from openAI gym (Brockman et al., 2016). Figure 5.11 shows the structure of the neural network used to train DQN.

We collected two demonstration sets from two imperfect agents: a human player and a rule-based agent. Each demonstration set contained 420 Pong games (i.e. complete episodes). The human and rule-based agent showed vastly different play behaviour. For each of these demonstration sets we used a convolutional neural networks (CNN) with the structure shown in Figure 5.11 to learn a state to action mapping. This resulted in a so-called human net and rule-based agent net. In addition, we trained another CNN with the same structure from the union of both demonstration sets as a baseline.

We then used the human net and the rule-based agent net in the TLQL approach as Expert 1 and Expert 2; the experts (i.e. the CNNs) were now treated as oracles, unlike in the other two evaluation domains. Figure 5.12 shows the performance of DQN without demonstration, DQN with TLQL, and DQN with CHAT. The results were similar to the ones in in the maze and the flag visiting domains. With TLQL, the agent can overcome the problem of conflicting demonstrations and speed up the learning process by learning when to trust each experts. Overall, TLQL significantly outperformed both CHAT and the original DQN.

5.12 Summary

In this chapter we proposed a novel algorithm, two-level Q-learning (TLQL), that incorporates demonstrations from multiple experts with varying expertise into reinforcement learning. The expert demonstrations were used to bias the exploration of the RL agent. The TLQL algorithm

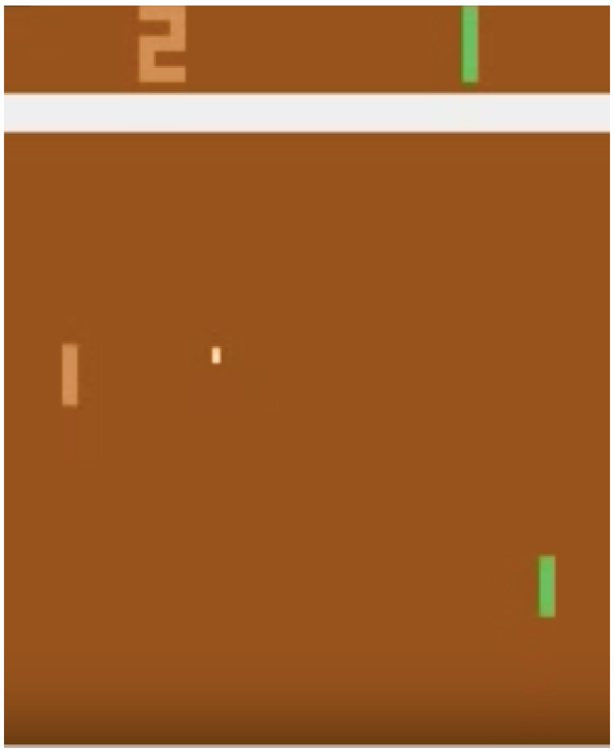

Figure 5.10: Pong: A game of Atari 2660

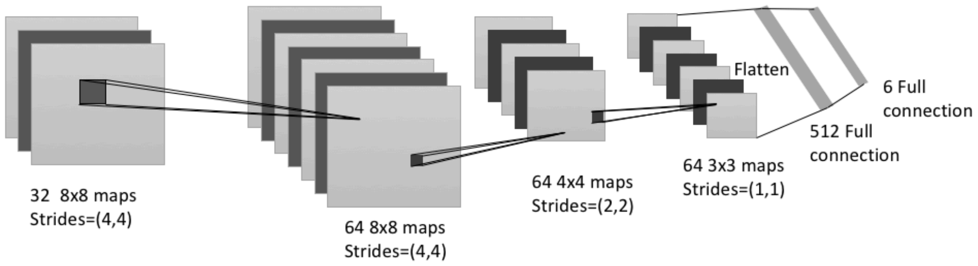

Figure 5.11: Structure of neural network of DQN and expert

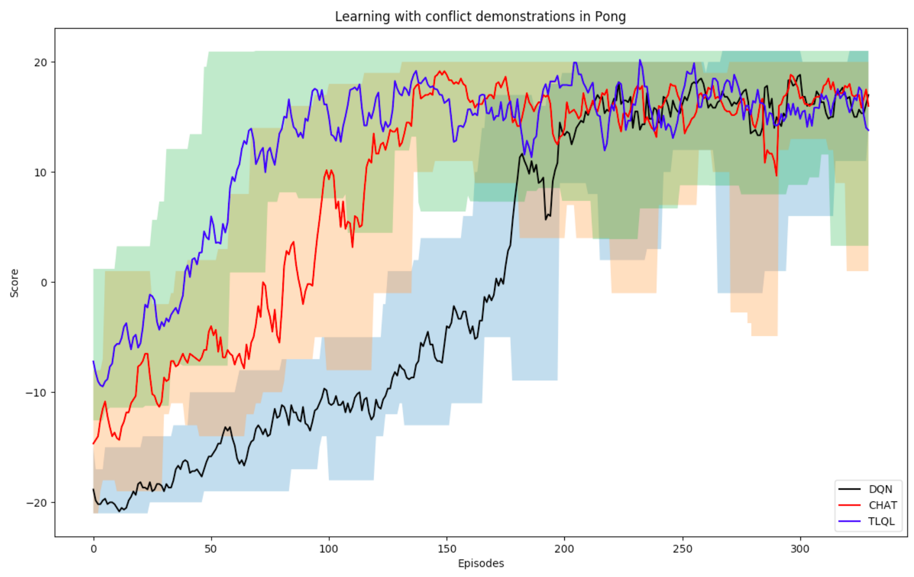

Figure 5.12: Comparison of TLQL with CHAT (Wang and Taylor, 2017) and regular Q-learning in the domain of Atari game Pong.

added a Q-table to record the degree of trust in an expert for different states. The RL agent learned a policy of selecting an expert simultaneously to learning the policy for maximising the cumulative reward. To keep the high-level and low-level Q-tables synchronised, we update both those to Q-tables in each agent-environment interaction.

Notably, the value of $\langle state, RL \rangle$ and values of $\langle state, otherexpert \rangle$ were updated for each sample. The algorithm always kept the value of $\langle state, RL \rangle$ equal to the low-level Q-table.

When using demonstrations from multiple experts, conflicting advice can often occur. In our experiments we focused on two common reasons for such conflicts: (1) individuals were experts in different parts of state space; (2) individuals were skilled in different sub-tasks of the goal.

We evaluated our proposed algorithm in a maze navigation domain, a coloured flags visiting domain, and the Atari game of Pong. The results of our experiments showed that TLQL significantly outperforms regular Q-learning without knowledge and the state-of-the-art CHAT algorithm in terms of jumpstart, overall performance, and asymptotic performance.

Future work includes constructing a distance measure for conflicts. In this way clusters of similar experts could be treated as one expert in the high-level Q-table. Applying clustering techniques could also provide a hyper-parameter, i.e. the number of clusters, which could thence be used to trade off the sample efficiency of RL and computational costs.

Chapter 6

Q-learning from a massive number of imbalanced demonstrations

As mentioned in previous chapters, the sample efficiency is very low specially in sparse reward space. Human experts' demonstrations can be applied to guide the biased exploration of an RL agent on state-action space and to approach a reasonable performance at an earlier stage. However, in some applications such as online games, the number of demonstrations is extraordinarily large and the distribution of demonstrations is imbalanced. Some states have a larger number of demonstrations for their actions. Under these situations, TLQL that can narrow down candidate action space through demonstrators is unsuitable. It is thus essential to propose a new algorithm that is able to consider frequencies of demonstrations on different states when guiding the an RL agent. Moreover, Brys et al. (2015) proposed the similarity-based optimal action hypothesis in which neighbouring states deliver same optimal actions. This hypothesis is not applicable for some domains such as maze because the transition function is unknown. Therefore, adjacent states may have different actions.

To address the two issues discussed above, a novel algorithm is proposed in this thesis: Radius-Restrained Weighted voting(RRWV). This algorithm introduces a hyperparameter, radius, to restrain the Euclidean distance between candidate demonstrations and the current state. Candidate demonstrations within the restricted radius are regarded as reference demonstrations that can vote for the action for the current state with weights. The weight of the vote from a candidate demonstration is defined as the Gaussian distance between the current state and the demonstration. Therefore, candidate demonstrations which are closer to the current state possess larger power to impact the derived policy. The softmax function is adopted to map frequencies of votes for each action into probabilities. These can then be used to produce a guidance policy that can be inserted into a RL loop to direct an agent's exploration.

6.1 Related research

Brys et al. (2015) developed the similarity-based shaping algorithm(SBS) which applies the Gaussian distance between the current state and the nearest demonstration as a potential function to reshape the reward function. As Equation 6.1 shows, d is the current state that an agent has observed; s^d is the state of a demonstration d; Σ is the covariance matrix; $P(s, a)$ is the potential function

$$P(s,a) = \max_{s^d,a} e^{-\frac{1}{2}(s-s^d)^T \Sigma^{-1} (s-s^d)} \tag{6.1}$$

According to the work by (Ng et al., 1999), the reshaped reward function has been defined as the origin reward plus the gap of potential between the current state and next state. Equation 6.2 exhibits the definition of potential-based reward shaping.

$$R'(s,a) = \gamma Potential(s', a') - P(s,a) + R(s,a) \tag{6.2}$$

In practice, it is not essential to encode the potential function into the reward function. Wiewiora (2003) proved that initialising the Q-value function as a potential function is equal to potential-based reward shaping.

The weakness of SBS is that it tends to have an over-generalisation problem under conditions with imbalanced demonstrations. Imbalanced demonstrations means that distribution of

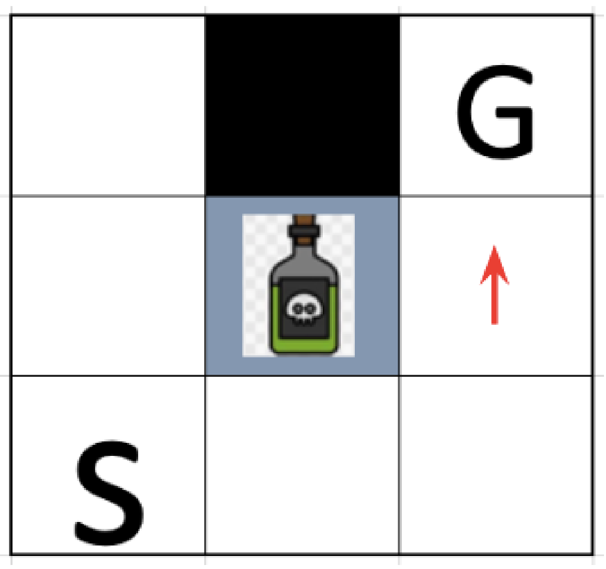

Figure 6.1: 3x3 grid domain with one correction demonstration

demonstration is not uniform. Demonstrations are concentrated in several area of state space. Over-generalisation is that the agent add too more bias into exploration and result in worse learning performance. To illuminate the over-generalisation of SBS clearly, a 3x3 maze domain is introduced. As the Figure 6.1 shows, there are 3x3 grids to represent 8 states and a wall(the black grid) that an agent cannot pass. The RL agent can execute 4 actions of going: left, right, up and down to move forward to corresponding states. The goal of the task is to move from state S to state G, the terminal state. The domain only has one non-zero reward, which is the state when the agent approaches the G state. Under this situation, a reward +1 is given. In the centre of the maze, there is a poison state. If the agent moves to the poison state, the game will terminate immediately without any reward being given. In the 3x3 maze domain, there is only one demonstration available, as shown by the red arrow in the figure. The demonstration suggests going up which is a correct demonstration.

Building on the method of SBS, an initialised Q-table was derived from the demonstrations as shown in the left side of the Figure 6.2. Numbers of each grid represent 4 values of potentials of 4 state-action pairs. According Wiewiora (2003), initialisation values of a Q-table equal values of potential on each state-action pair. As the right side figure shows, although there is only one optimal demonstration, the starting policy deriving from the initialised Q-value suggests that the initialised policy should move up at all states. Apparently, the starting policy tends

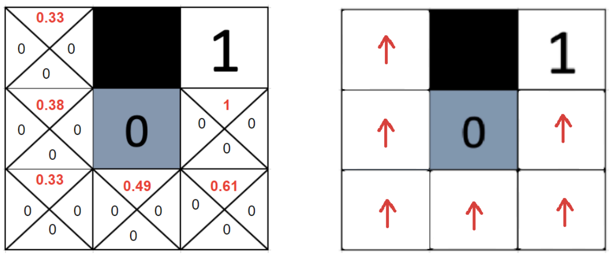

Figure 6.2: Initialised Q-value 3x3 grid domain via SBS (Brys et al., 2015)

to mislead the agent and cause a decrease in its learning speed.

Since there is only one demonstration in the domain, the agent will refer to the demonstration for every state, which will result in an over-generalisation problem. In fact, this situation is common in real applications as the distribution of demonstrations is usually imbalanced. If the agent only relies on the closest demonstration, the result tends to be over-generalisation as shown in the case: 3x3 domain with one demonstration. Wang et al. (2018) considered SBS as a special case of employing k-NN(k=1) classifier fitting on demonstrations. They proposed the online Least-Square Policy Iteration(LSPI) algorithm to transfer a policy from demonstrations via a k-NN classifier with different parameters k. LSPI integrates the transferred policy and RL via the probabilistic policy reuse proposed by Fernández and Veloso (2006).

6.2 Radius restrained weighted voting

To deal with the two issues noted above, two techniques are proposed in this dissertation: restraining reference demonstrations via a radius, and deriving the guidance policy on weighted voting via softmax.

6.2.1 Restrain radius

To avoid the over-generalisation problem from imbalanced demonstrations, a hyperparameter, radius, is introduced as a threshold value to discriminate whether a demonstration can be used as a reference demonstration for the current state. Only demonstrations with the Euclidean dis-

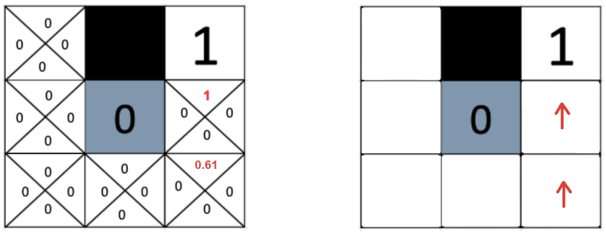

Figure 6.3: Initialised Q-value 3x3 grid domain

tance less than the radius can be considered as reference demonstrations. Afterwards, reference demonstrations are used to reshape the reward function or obtain the guidance policy.

For example, in a 3x3 maze domain, if the $radius = 1$, the Q-value will be initialised as the left of figure 6.3. It shows that reward function has been reshaped on only one state (state on lower right corner).

The initialised Q-value as in left of Figure 6.3 shows that at the beginning of the learning process, the agent only goes up on the right column as per the right of Figure 6.3 and keeps exploring other states uniformly and randomly. As a result, the initialised Q-value is able to improve the agent's performance.

6.2.2 Weighted voting

For each demonstration, two principles must be followed:

- Every reference demonstration has the right to influence the RL agent.
- Reference demonstrations closer to the current state have more impacts than others.

Following the two principles above, the weighted voting technique is proposed. The weight of a demonstration is defined as a Gaussian distance between a demonstration and the current state as Equation 6.3 shows.

$$W(s^d) = e^{-\frac{1}{2}(s-s^d)^T\Sigma^{-1}(s-s^d)} \tag{6.3}$$

As Equation 6.3 shows, if $s = s^d$, the current state and a demonstration are overlapping, and the weight will be equal to 1 ($W(s^d) = 1$). A demonstration with an infinite Euclidean distance to the state will have a weight of 0 ($W(s^d) = 0$).

The softmax function, as shown in the Equation 6.4, is applied to map the total number of weighted votes to probabilities for each action.

$$P(a_i|s) = \frac{exp(Vote(s, a_i))}{\sum_{k=1}^{K} exp(Vote(s, a_k))} \tag{6.4}$$

$Vote(s, a_i)$ is the sum of the weights of action a_i. $P(a_i|s)$ represents the guidance policy derived from the votes of the demonstrations.

6.2.3 Algorithm

Algorithm 9 shows the pseudo code of RRWV. Line 8 shows that only the demonstration satisfying the radius restriction can vote for its action in the current state. In lines 11 and 12, the softmax function derives a guidance policy π_D from the vote table, noted as VT. RRWV applies the technique of probabilistic policy reuse (Fernández and Veloso, 2006) to inject the guidance policy into a RL loop as presented in line 14. The policy π_D with the probability ψ will keep exploration, via sampling, of an action with an uniform distribution. It employs the $max_a Q(s, a)$ policy probability 1 - π_D - ψ. Line 17 describes the decay of ψ, and ρ is the decay rate.

6.3 Case study

Figure 6.4 demonstrates a two-dimensional two-action domain. There are 8 demonstrations in the domain represented by circles around by a state (the blue square). The colours of the circles represent two different actions. To make it simple, the red actions and black actions are used for the illustration. SBS (Brys et al., 2015) only refers to the nearest demonstration. Therefore, it will consider the derived policy for the current state as the red action. Transfer

Algorithm 9 Radius Restrained Weighted Vote

```
 1: procedure RADIUS RESTRAINED WEIGHTED VOTE($D$, radius)
 2:     Randomly initialised parameters of Deep Q-network $Q_\theta(s,a)$
 3:     Initialised vote table VT of all $(s,a)$ for demonstrations
 4:     Initialised vote table $\pi_D$ of all $(s,a)$ for demonstrations as 0.
 5:     for each episode do:
 6:         for each step t in episode do:
 7:             for each demonstration $< s_d, a_d >$ in demonstration set do:
 8:                 if $distance(s_t, s_d) < radius$ then $VT(s_t, a_d) = VT(s_t, a_d) + Gaussian(s_t, s_d)$
 9:                 end if
10:             end for
11:             for each action $a_i$ in VT do:
12:                 $\pi_D(s, a_i) = \frac{e^{VT(s,a_i)}}{\sum_j e^{VT(s,a_j)}}$
13:             end for
14:             $a_t = \leftarrow \begin{cases} \text{with probability } \psi & \text{select action via } \pi_D \\ \text{with probability } \epsilon & \text{select action uniformly} \\ \text{with probability } 1-\epsilon-\psi & \text{selection via } max_a Q(s,a) \end{cases}$
15:             $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
16:         end for
17:         $\psi \leftarrow \psi * \rho$
18:     end for
19: end procedure=0
```

k-NN (Wang et al., 2018), on the other hands, reaches the policy by a k-NN classifier. If k=7, the derived policy from transfer k-NN will select the black action for the current state. Our approach, RRWV, restrains demonstration candidates by a radius as the Figure 6.4 shows. All weights of three red candidate demonstrations is 0.0111 (their Euclidean distances are 3). Since the Euclidean distance between the state and the nearest black demonstration is 1, its weight will be 0.6065. Overall, according to the softmax function on weighted votes, the derived policy will select black and red actions with probabilities of 71.78% and 28.22% respectively. Table 6.1 summaries the results from SBS RRWV and k-NN transfer on the 8-demonstration domain.

Approach	classifier	Deriving policy
SBS	k-NN k=1	Black action
k-NN Transfer	k-NN k=7	Black action
RRWV	softmax	Red action

Table 6.1: Table to compare deriving policy from demonstrations

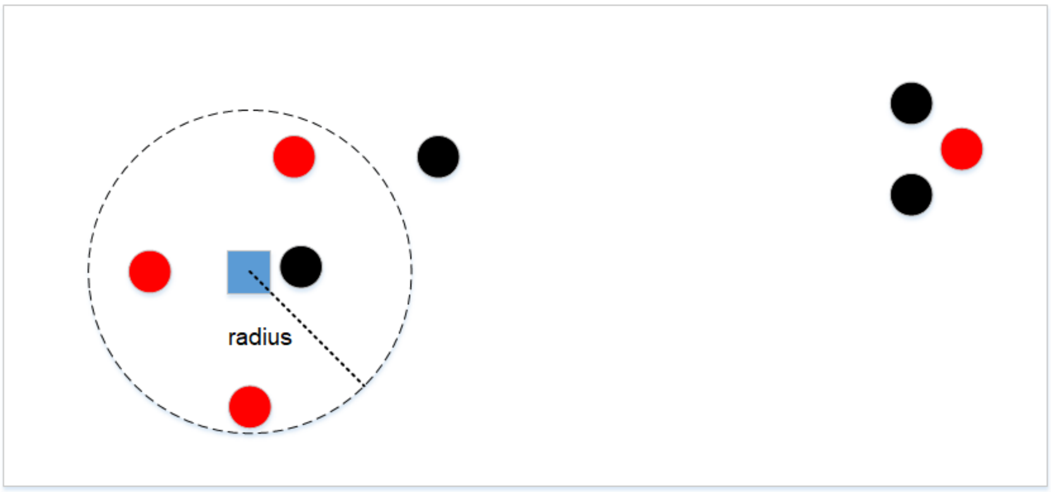

Figure 6.4: Compare RRWV, SBS and kNN Transfer

6.4 Experiments

6.4.1 Maze domain

Figure 6.5 shows a 10x9 maze domain with demonstrations. This domain is used to represent the benefits of selecting demonstrations by a radius. The goal of the 10x9 maze domain is moving from state S to state G. The agent can cross neither the boundary nor the walls (black lines). State G is the only one terminal state and non-zero reward state (reward +1). To reach the corresponding state, the agent can execute 4 actions: moving left, moving right, going up, and going down.

There are 4 optimal demonstrations in the domain shown as red arrows in Figure 6.5. The left picture in Figure 6.6 presents the nearest neighbouring states of demonstrations. According to the hypothesis of the similarity-based optimal action, the same colour of states and demonstrations means that they are in the same group with a same action as the demonstration shown. The figures shown in the states indicate the Manhattan distances between the individual states and their corresponding demonstrations. According to SBS, the derived policy from demonstrations, as shown in the right picture of Figure 6.6, will lead an agent to move forward walls in same states. Due to the presence of walls changes the transition function of the domain, the hypothesis of the similarity-based optimal action will be broken.

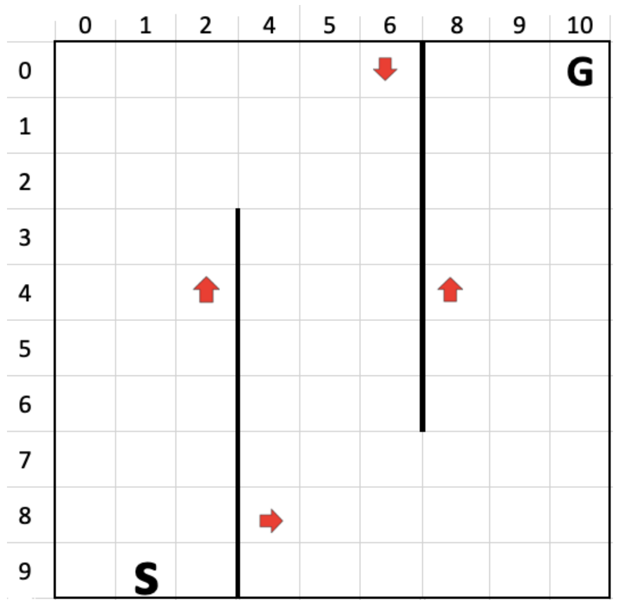

Figure 6.5: Maze domain with demonstration

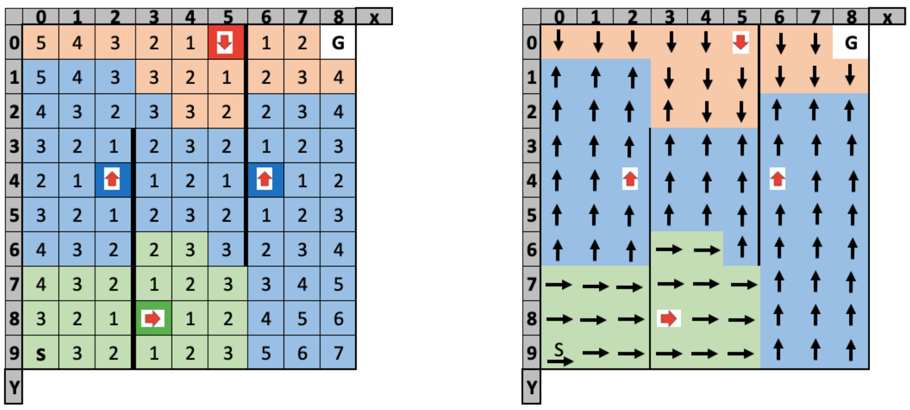

Figure 6.6: Derive a policy from demonstration in 9x10 maze via SBS

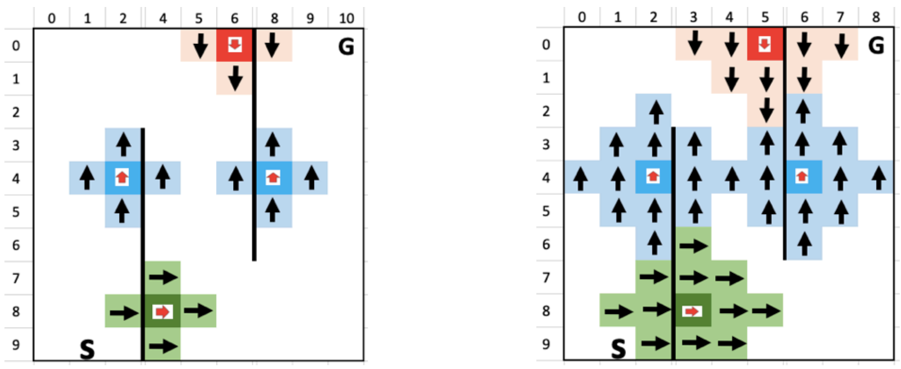

Figure 6.7: Derive a policy from the demonstration in 9x10 maze via RRWV radius=1 and 2

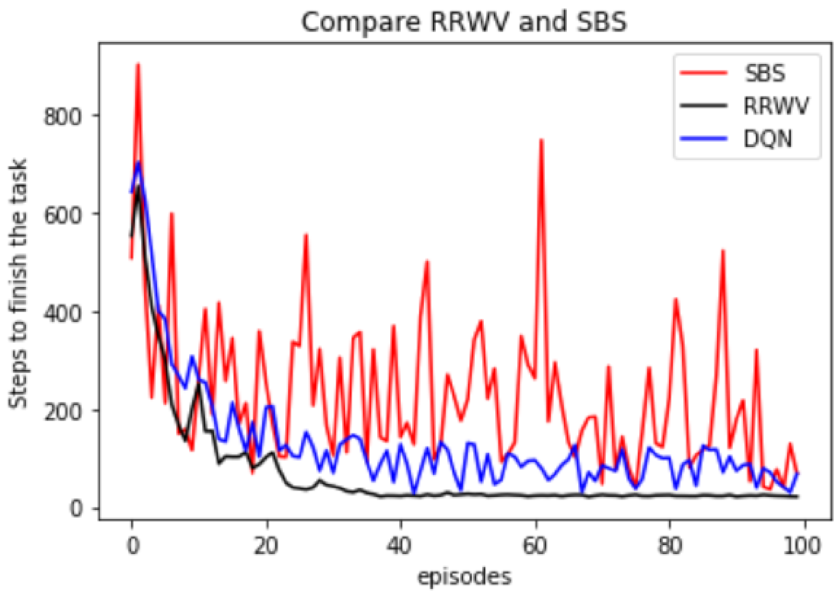

Figure 6.8: Compare the RRWV and SBS

When the radius restriction is introduced, only qualified demonstrations (called reference demonstrations) can be applied to encode the initialised Q-value function. A policy with high performance at the beginning can be achieved from the demonstrations. The left and right pictures in Figure 6.7 display guidance policies with radius=1 and radius=2 respectively.

Figure 6.8 presents the result from an comparison of SBS and RRWV (with radius=2). Due to the presence of walls breaking the hypothesis of the similarity-based optimal action, SBS cannot facilitate an agent to increase its performance. It may even hurt the agent's learning process. RRWV, on other hand, still benefits the RL processing in this setting, for it can derive a police that is close to the optimal policy. Compared with SBS and DQN, RRWV can perform more stable and approach at a high performance level at an early stage.

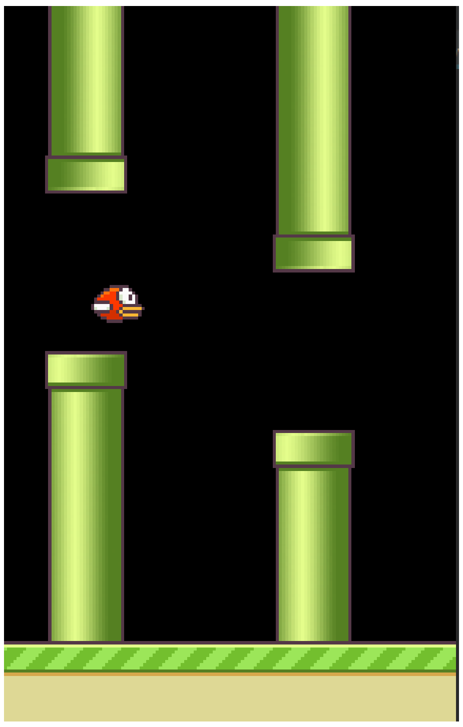

Figure 6.9: Domain of flappy bird game

6.4.2 Flappy bird domain

As shown in Figure6.9, Flappy bird is a video game in which the player controls the bird through gaps between pairs of pipes Chen (2015). The flappy bird in the game has two actions: the "up", which makes the bird fly upward and the "keep", which results in the bird falling off the pipe because of the gravity. For each step, if the bird does not touch the pipes, a reward of +0.1 is given. The game is terminated when a negative reward of -1 1 is given; this occurs when the bird hits hits the pipes.

In the Flappy bird experiment, the Deep Q-learning(DQN) algorithm was applied to learn a policy from raw pixels. The Q-network consisted of a 3-level Convolutional Neural Network(CNN) and a 2-level full connection neural network(CNN) as shown in figure 6.10.

As Figure6.11 shows, images of the game were clipped into 84x84 images and transformed into bitmaps. To capture the speed information of the bird, 4 abutting bitmaps were overlapped as a state to be input into the Q-network.

Ten episodes were collected, including five episodes coming from a human player. Another five

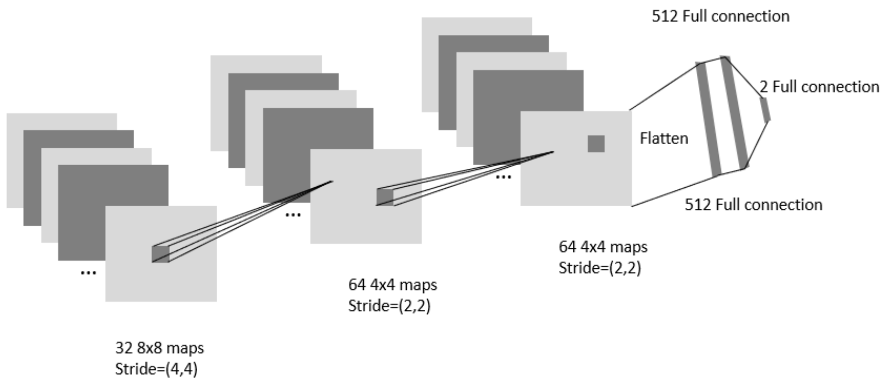

Figure 6.10: Neural network structure used in Flappy bird game

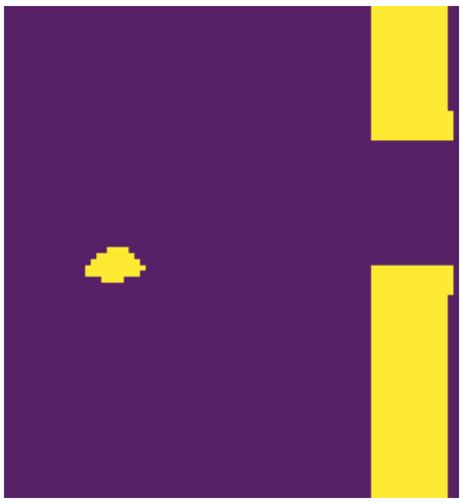

Figure 6.11: preprocessing state of flappy bird

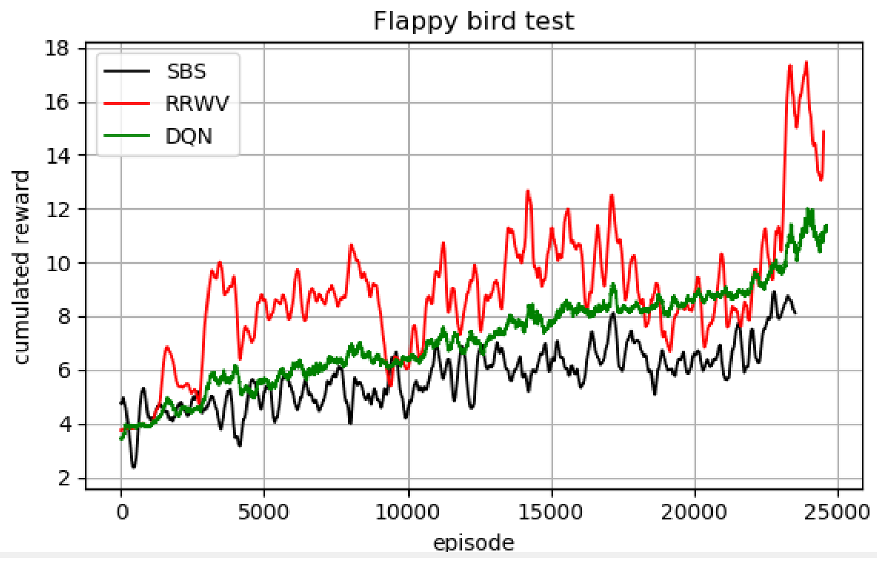

Figure 6.12: Compare RRWV, SBS and DQN in Flappy bird game

episodes coming from player agents. Each episode is composed of an observed image in the game. The performance of those episodes from the human player and rule-based agent is 5 to 7 score. Hamming distances between states were employed to measure the gaps between demonstrations and the current state. In information theory, the Hamming distance means the number of positions with different symbols. It is used to measure the minimum number of errors to transform one string into another. In this experiment, the hyper-parameter radius was set as 700, which refers to the distance between each bitmap of the candidate demonstration and the current state should be less than 700. RRWV and SBS use the same structure of CNN as shown in figure 6.10 to fit the Q-value network.

As shown in Figure 6.12, the result shows that the SBS had a lower performance level than that of the DQN, because the similarity-based shaping is not suitable for data of raw images. However, the proposed approach, RRWV, was able to obtain advantages from demonstrations.

6.5 Summary

This chapter examined RL with a massive number of sub-optimal demonstrations. Due to the distribution of demonstrations being sparse, some parts of state-action space may have sparse demonstrations. Under this circumstance, existing studies such as SBS (Brys et al., 2015) and kNN transfer (Wang et al., 2018) tends to refer to demonstrations that cannot satisfy the hypothesis of similarity-based optimal action. To address this issue, a hyper-parameter, radius, is proposed in this study to restrain demonstrations that will be used for deriving the guidance policy. It follows, that only demonstrations satisfying the restriction can be used as reference demonstrations.

In the setting with a massive number of demonstrations, a weighted voting method is proposed to derive a guidance policy based on frequencies of demonstration voting. The weight of a demonstration is defined as a Gaussian distance between the current state and the demonstration. The Softmax function maps the total voting numbers of actions into probabilities. RRWV employs the technique of probabilistic policy reuse to insert the guidance policy into the RL agent.

Chapter 7

Conclusion

7.1 Contributions

The Ph.D. research presented in this thesis focuses on the combined areas of reinforcement learning (RL) and supervised learning. Classic RL emphasises knowledge-free learning, which means that the agent learns the optimal policy by relying on samples from agent-environment interactions. However, knowledge-free RL, in a real application, is faced with sample inefficiency which means that it costs a lot of computation time and resources in order to get a reasonable performance. In some tasks, external information such as heuristic rules and human expert demonstrations are available. Therefore, a biased exploration, guided by external information rather than a uniform exploration at the beginning, could speed up the learning process and help the agent to achieve an acceptable performance earlier. This reduces the number of agent-environment interactions, saving time and resources.

There are two significant challenges in RL aided by external information. First, the external guidance may be sub-optimal as its performance is unknown. This is because the agent not only needs to rely on external information to guide biased exploration in the early-stages of learning, but also needs to learn to make an independent decision after it has learned knowledge about the environment. Secondly, external information such as demonstrations usually come from different sources. Therefore, these demonstrations may be in conflict, which means for a state s, demonstrations may have different actions. Conflicts of demonstrations exist widely, because humans are not pure rational agents. Rather, their behaviours are based on their own knowledge and life experiences and, thus imperfect knowledge. In practice, therefore, different people make different decisions when facing the same situation. Even when the same person observes the same state again, he or she may make a different decision to that which they made first time round.

In light of the research gaps mentioned above, this thesis contributes to the furtherance of state-of-art research in the following three ways. First, introspective RL reshapes the reward function via filtered samples from agents possessing experience. Secondly, Two-level Q-learning can deal with demonstrations from different resources that may be in conflict. Thirdly, RL from massive demonstrations constructs a policy from voting upon a partial demonstration set to address demonstration imbalance distribution and the problem of conflicting demonstrations experts.

7.2 Introspective RL

In Introspective RL, we proposed a hypothesis that samples from RL agent could be filtered and reused to shape the reward function. It could speed up the learning process and over the performance of state-of-art algorithm: similarity-based reward shaping (SBS) (Brys et al., 2015).

Introspective RL extends SBS and reshapes the reward function using Gaussian distance between the current state and the nearest demonstrations. The key idea of introspective is to use a filter to select high-value samples. Introspective RL can speed up the learning process by reusing its high-value experience in the filter. Demonstrations from human experts are used to initialise the filter and speed up the learning process further. Due to the fact that the filter weeds out low-value demonstrations and experiences, it is able to tolerate sub-optimal samples and converge into the optimal policy.

We tested our approach using a classic domain, cartpole. Cartpole is a domain with 4 continuous dimensions and 3 actions. We also introduced a video game, Super Mario Bro, with 27 discrete dimensions to test this algorithm.

The results show that our approach significantly improved the performance of Q(λ)-learning. By initialising human demonstrations, the performance of the algorithm surpasses those of the two baselines, Brys et al. (2015) and Taylor et al. (2011b).

7.3 RL from conflict demonstrations

The hypothesis of chapter 5 is that RL agent would able to deal conflicts in sub-optimal demonstrations with two Q-tables. The novel approach two-level Q-learning (TLQL) could over performance compared state-of-art algorithm Confidence Human Agent Transfer(CHAT) (Wang and Taylor, 2017) in conflicting demonstration setting.

The principle of the TLQL is to learn the value function of the state-expert using trial and error. Due to the fact that the demonstrations are in conflict, following a particular demonstrator, called an expert, as an action, could be considered. The agent not only learns the Q- value of state-action pairs, but also learns the value of state-expert pairs, noted as $< s, e >$. We define the Q-table for the state-expert as the high-Q-table, and the Q-table for the state-action as the low-Q-table. The agent selecting to follow the low-Q-table (trusting itself rather than the experts) is also an action in the high-Q-table. In TLQL, samples from environment interactions are a quintuple consisting of state, expert, action, reward, and next state, noted as $< s, e, a, r, s' >$. The agent can use each sample to update the low-Q-table and the high-Q-table simultaneously.

In the experiment setting, we simulated two common scenarios that cause policy conflicts. The first was the three area maze domain, which simulated human experts being knowledgeable in one particular area. In the three area maze domain, three rule-based agents were used to stimulate human experts giving demonstrations. Each simulated expert only generated "approximately correct policy (perfect demonstration with 10% noise)" in 1/3 of the maze. The second experiment, coloured flag visiting, simulated a complex task requiring multiple skills, but with the human expert only having one of the required skills. Three rule-based agents with their own single rule demonstrated their policies with 10% noise. Both the maze domain and the flag visiting domain showed that TLQL can efficiently combine conflicting demonstrations to speed up the RL process. Pong, one of the Atari 2600 games, was used to test TLQL in a deep RL setting. A neural network was trained to fit state-action mapping on demonstrations from a human player, noted as the Expert 1. A heuristic rule-based agent was regarded as Expert 2. Even though the performances of both experts were poor, the TLQL agent could dynamically learn to follow one experts suggestions or follow the low-Q-network (same as the low-Q-table) according to the state it observed. The results show that the performance of combining two conflicting experts is better than CHAT (Wang and Taylor, 2017) with demonstrations from both experts.

In conclusion, test in Three area maze, Flag visiting and Pong all support the hypothesis that Two-level Q-learning could deal conflict in sup-optimal demonstrations.

7.4 RL from massive demonstrations

In chapter 6, a hypothesis that restrained radius of state improve the performance in massive and imbalanced demonstrations setting compared to state-of-art approaches such as SBS and kNN-transfer (Wang et al., 2018) is proposed.

To avoid over-fitting demonstrations in the spare sub-spaces of state-action, a radius-restrained weighted voting algorithm was proposed. In this novel technique, the radius is a hyper parameter used to limit the distance of demonstrations that would be considered as candidate from which to derive an assistant policy to guide the RL agents explorations. Each candidate demonstration votes on its action weighted by the Gaussian distance between the demonstration and the current state. We employed softmax function to map the voting of actions to an assistant policy. The policy was inserted into the RL loop to guide RL agent biased exploration.

A 9x10 maze domain and a two-action domain were applied as case-studies to compare our novel technique and relative techniques. The analysis shows that compared with the SBS and kNN transfer algorithms, the RRWV is robust in imbalance demonstration. We also applied the Flappy bird, a video game to test RRWV. 5 trajectories game from human player and 5 trajectories from a player agent has been used to inject into RL. The results show that the performance of RRWV surpass the performance of DQN. However, for the game not satisfied similarity based assumption well, SBS got worst performance than the DQN.

The test result of Flappy bird and maze domain support the hypothesis significantly.

7.5 Limitations

The limitation of introspective RL is that it uses Gaussian distance to reshape the reward function. It introduces an assumption that close states have a high probability of taking the same optimal action (Gaussian distribution in this thesis). This assumption exists in many video games. However, some domains (e.g. a maze) may not support this assumption.

The principle of the Two-level Q-learning narrows the action space using experts. The limitation of this algorithm is that it only works when the degree of conflict is less than the number of actions. If the conflict covers all actions, the two-level Q-learning will retreat to normal Q-learning.

For RL from massive demonstrations, it introduces a hyper-parameter, the radius. The radius is dependent on domain knowledge and needs to be tune manually.

7.6 Combination

Unlike Introspective Q-learning that can shape the reward function, the Radius-Restrained Weighted voting (RRWV) technique uses the voting of demonstrations as a bias to be injected into the Q-learning loop. However, the radius restraint technique in RRWV has the potential to be integrated into Introspective Q-learning.

Introspective Q-learning applies demonstrations from the agent itself or the external demonstrator. It thus faces an over-generation problem. When it shapes the reward function, unqualified demonstrations may be involved and result in a badly biased exploration. By integrating with the radius restraint technique, Introspective Q-learning is able to deal with the over-generation problem as it selects demonstrations that satisfy the radius constriction.

7.7 Future work

Our future work will focus on addressing the limitations above. An algorithm suitable for massive conflicting demonstrations will be constructed. In the perspective of introspective RL, we need to refine the demonstration filter technique and explore how to update sample information to surrounding states efficiently. For conflicting issues, more methods to explore useful information from massive conflicting demonstrations need to be studied. A combination method of TLQL and Introspective Q-learning should be studied in the future.

In the RL massive algorithm, the radius is a hyper-parameter. In future research, a method of dynamically learning the radius is expected to be found. Unsupervised learning techniques have the possibility to facilitate the discovery of the space structure of state-actions. A combination of RL and unsupervised learning may be a novel way to deal with the sample inefficiency problem of RL.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

Abbeel, P. and Ng, A. Y. (2011). Inverse reinforcement learning. In *Encyclopedia of machine learning*, pages 554–558. Springer.

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer.

Barto, A. G. (1997). Reinforcement learning. In *Neural systems for control*, pages 7–30. Elsevier.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org.

Bellman, R. E. (1961). Dynamic programming treatment of the traveling salesman problem.

Bellman, R. E. (2015). *Adaptive control processes: a guided tour*, volume 2045. Princeton university press.

Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Bianchi, R. A., Ribeiro, C. H., and Costa, A. H. (2004). Heuristically accelerated q–learning: a new approach to speed up reinforcement learning. In *Brazilian Symposium on Artificial Intelligence*, pages 245–254. Springer.

Boularias, A., Kober, J., and Peters, J. (2011). Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brys, T. (2016). *Reinforcement Learning with Heuristic Information*. PhD thesis, PhD thesis, Vrije Universitet Brussel.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015). Reinforcement learning from demonstration through shaping. In *IJCAI*, pages 3352–3358.

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators*. CRC press.

Celiberto, L. A., Ribeiro, C. H., Costa, A. H., and Bianchi, R. A. (2007). Heuristic reinforcement learning applied to robocup simulation agents. In *Robot Soccer World Cup*, pages 220–227. Springer.

Chen, K. (2015). Deep reinforcement learning for flappy bird.

Cruz Jr, G. V., Du, Y., and Taylor, M. E. (2017). Pre-training neural networks with human demonstrations for deep reinforcement learning. *arXiv preprint arXiv:1709.04083*.

Devlin, S. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 433–440. International Foundation for Autonomous Agents and Multiagent Systems.

Devlin, S., Kudenko, D., and Grześ, M. (2011). An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278.

Efthymiadis, K. and Kudenko, D. (2013). Using plan-based reward shaping to learn strategies in starcraft: Broodwar. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE.

Fernández, F. and Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM.

Ferster, C. B. and Skinner, B. F. (1957). Schedules of reinforcement.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

Gao, Y., Lin, J., Yu, F., Levine, S., Darrell, T., et al. (2018). Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*.

Geva, S. and Sitte, J. (1993). A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine*, 13(5):40–51.

Ghahramani, Z. (2003). Unsupervised learning. In *Summer School on Machine Learning*, pages 72–112. Springer.

Golovin, N. and Rahm, E. (2004). Reinforcement learning architecture for web recommendations. In *null*, page 398. IEEE.

Harutyunyan, A., Devlin, S., Vrancx, P., and Nowé, A. (2015). Expressing arbitrary reward functions as potential-based advice. In *AAAI*, pages 2652–2658.

Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Karakovskiy, S. and Togelius, J. (2012). The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67.

Knox, W. B. and Stone, P. (2010). Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 5–12. International Foundation for Autonomous Agents and Multiagent Systems.

Kober, J. and Peters, J. (2012). Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lange, S., Riedmiller, M., and Voigtlander, A. (2012). Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE.

Lee, J. (2017). A survey of robot learning from demonstrations for human-robot collaboration. *arXiv preprint arXiv:1710.08789*.

Li, M., Brys, T., and Kudenko, D. (2018). Introspective reinforcement learning and learning from demonstration. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1992–1994. International Foundation for Autonomous Agents and Multiagent Systems.

Li, M. and Kudenko, D. (2018). Reinforcement learning from multiple experts demonstrations. In *Workshop on Adaptive Learning Agents (ALA) at the Federated AI Meeting*, volume 18.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Loftin, R., Peng, B., MacGlashan, J., Littman, M. L., Taylor, M. E., Huang, J., and Roberts, D. L. (2016). Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous agents and multi-agent systems*, 30(1):30–59.

Michie, D. and Chambers, R. A. (1968). Boxes: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*.

Nevmyvaka, Y., Feng, Y., and Kearns, M. (2006). Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Nilsson, N. J. (1998). Artificial intelligence: A new synthesis. morgan kauffmann publishers. *Inc. San Francisco, California.*

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

OpenAI (2018). Openai five. `https://blog.openai.com/openai-five/`.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034.

Osband, I., Van Roy, B., and Wen, Z. (2014). Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635.*

Poole, D. L., Mackworth, A. K., and Goebel, R. (1998). *Computational intelligence: a logical approach*, volume 1. Oxford University Press New York.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087.*

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952.*

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815.*

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158.

Skinner, B. F. (1990). *The behavior of organisms: An experimental analysis*. BF Skinner Foundation.

Suay, H. B., Brys, T., Taylor, M. E., and Chernova, S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 429–437. International Foundation for Autonomous Agents and Multiagent Systems.

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

Sutton, R. S., Barto, A. G., Bach, F., et al. (1998). Reinforcement learning: An introduction.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

Taylor, M. E., Kulis, B., and Sha, F. (2011a). Metric learning for reinforcement learning agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 777–784. International Foundation for Autonomous Agents and Multiagent Systems.

Taylor, M. E. and Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Taylor, M. E., Suay, H. B., and Chernova, S. (2011b). Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Turing, C. (1950). Computing machinery and intelligence. *Mind*, 59(236):433.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ.

Van Roy, B. (2001). *Temporal-difference learning and applications in finance*. Cambridge, MA: MIT Press.

Wang, G.-f., Fang, Z., Li, P., and Li, B. (2018). Transferring knowledge from human-demonstration trajectories to reinforcement learning. *Transactions of the Institute of Measurement and Control*, 40(1):94–101.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Wang, Z. and Taylor, M. E. (2017). Improving reinforcement learning with confidence-based demonstrations. In *IJCAI*, pages 3027–3033.

Whiteson, S., Taylor, M. E., Stone, P., et al. (2007). *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin.

Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wiskott, L. (2016). Lecture notes on reinforcement learning. *environment*, 1:2.

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. *arXiv preprint arXiv:1803.07246*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.