Deep Reinforcement Learning with Double Q-learning

Lilia BEN BACCAR

Sarah LAUZERAL

ENSAE - Mastère Spécialisé

Contents

1 Main contribution of each member

We both choose the article we worked on for this project and of course we both read it. Then we splited the work as follow:

- **Lilia BEN BACCAR:** Summary of the article (writing parts 3 to 7) and implementation of the algorithm[1]
- **Sarah LAUZERAL:** Writing of the introduction and the literature review

2 Introduction

Reinforcement Learning is the field for solving problems we can present with Markov Decision Processes (MDPs). It relies on the interaction of a learning agent, that tries to achieve a goal, with an environment that has certain states. To do that, agent performs multiple actions that change the environment state and result in feedbacks from it. Feedbacks can be rewards or punishments through which the agent learns which actions are appropriate and then builds a policy.

Algorithm 1: Reinforcement Learning procedure

```
1. Observe the current state $s$ from the environment
2. Perform an action $a$ under a policy $\pi$ according to $s$
3. Observe the reward $r$ from the environment
4. Update the policy $\pi$
5. After taking the action, the environment updates and goes to the next state $s'$
6. Repeat steps 1 to 5
```

In the RL set-up, an autonomous agent, controlled by a machine learning algorithm, observes a state s_t from its environment at timestep t. The agent interacts with the environment by taking an action a_t in state s_t. When the agent takes an action, the environment and the agent transition to a new state s_{t+1} based on the current state and the chosen action.

The best sequence of actions is determined by the rewards provided by the environment. Every time the environment transitions to a new state, it also provides a scalar reward r_{t+1} to the agent as feedback. The goal of the agent is to learn a policy (control strategy) π that maximises the expected return. Given a state, a policy returns an action to perform ; an optimal policy is any policy that maximises the expected return in the environment. However, the challenge in RL is that the agent needs to learn about the consequences of actions in the environment by trial and error.

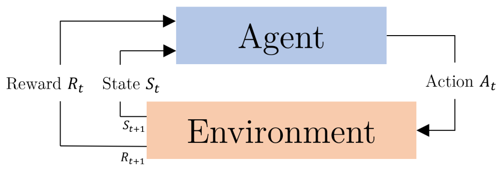


Figure 1: Reinforcement learning process

We provided a brief reminder of what RL is about, but the main topic of this project consists in studying the article entitled Deep Reinforcement Learning with Double Q-learning written by Hado van Hasselt, Arthur Guez and David Silver[13]. This article demonstrate how frequent Q-learning algorithms overestimate action values and how to prevent it and more especially this paper prooves that the DQN algorithm also suffers from substantial overestimations in some games in the Atari 2600 domain. That's why the authors introduce the Double Q-learning algorithm and propose an adaptation of the DQN algorithm architecture to reduce the overestimations.

[1]https://colab.research.google.com/drive/1FaFHzbRjtwB7cAjtcki9NKDL00yQkIUw?usp=sharing

3 Q-learning

3.1 Definition

Q-Learning was introduced by Watkins ([54], [55]) and is a well known algorithm in reinforcement learning whose goal is to estimate the value of taking an action a when the agent is in the state s with the policy π : the Q-value. It can be interpreted as the quality of an action a in a state s that means that when the Q-value is high, the reward is high. The learning agent uses these Q-values to know how it will achieve the defined goal and it is the policy that will determine which (S_t, A_t) couples are visited and updated.

During the training, for each couple (S_t, A_t) the agent will update Q-values:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

The target Y_t^Q is defined as:

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t)$$

All these values are stored in a Q-table. $\max_a Q(S_{t+1}, a)$ means that the Q-value at the current time step is computed thanks the Q-value at the future time step. That's why we have to set random values at the beginning for S_t and S_{t+1} Q-values. On the first iteration, we will update S_t Q-value based on the reward and S_{t+1} Q-Value. The process is led by the reward so it will converge to the best result.

Q-Table		**Actions**				
		Action 1	**Action 2**	**...**	**Action N-1**	**Action N**
States	**State 1**	$Q(s_1, a_1)$	$Q(s_1, a_2)$	...	$Q(s_1, a_{N-1})$	$Q(s_1, a_N)$
	State 2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	...	$Q(s_2, a_{N-1})$	$Q(s_2, a_N)$
	...	...	...	...	...	...
	State M-1	$Q(s_{M-1}, a_1)$	$Q(s_{M-1}, a_2)$	...	$Q(s_{M-1}, a_{N-1})$	$Q(s_{M-1}, a_N)$
	State M	$Q(s_M, a_1)$	$Q(s_M, a_2)$	...	$Q(s_M, a_{N-1})$	$Q(s_M, a_N)$

Table 1: Q-Table example

The target policy $\pi(a|s)$ is a greedy with respect to $Q(s, a)$. This means that our strategy is to take actions which result in highest values of Q. So $Q*(s, a)$ is optimal because it is $Q(s, a)$ that follows the policy that maximizes the action-values.

$$\pi(s, a) = \begin{cases} 1 & \text{if } a = \arg\max_a Q(s, a) \\ 0 & \text{otherwise} \end{cases}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[r + \gamma Q_\pi(s_{t+1}, a_{t+1}) | S_t = s, A_t = a]$$

Bellman Optimality Equation :

$$Q_*(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[r + \gamma \max_{a'} Q_*(s_{t+1}, a') | S_t = s, A_t = a]$$

Algorithm 2: Q-learning algorithm

```
1. **Initialize** all Q-values randomly : $Q(s, a) = n, \forall s \in S, \forall a \in A(s)$
2. **Initialize** the terminal state Q-value to 0 : $Q(\text{terminal state}, \cdot) = 0$
3. **Select** the action $a$ from $A(s)$ the set of action in $s$ under $\pi$
4. **Perform** the action $a$
5. **Observe** the reward $R$ and the next state $s'$
6. **Select** the action $a'$ with the highest (max) Q-value among the possible actions from $s'$
7. **Update** Q-values for $s$ : $Q(s, a) = Q(s, a) + \alpha[R + \gamma Q(s', a') - Q(s, a)]$
8. **Repeat** steps 3 to 6 for each time step until reaching the terminal state
9. **Repeat** steps 3 to 7 for each episode
```

Unfortunately, $\max Q(S_{t+1}, a)$ causes many problems in such a way that the Q-learning algorithm performs badly in stochastic environments. Indeed, the *max* operator causes the fact that the algorithm sometimes overestimate some action Q-values. This may lead him to believe that certain actions are worthwhile, even though they result in less reward in the end. Indeed, this can be shown by the following Markov Decision Process :

- We set an environment with four states which are : X, Y, Z, W.
- We set Z and W as terminal states.
- We set X as starting state.
- In X, there are two possible actions : UP ($R = 0$) and DOWN ($R = 0$)
- In Y, there are three possible actions (for all of them $R \sim \mathcal{N}(-1, 2)$) that led to W.

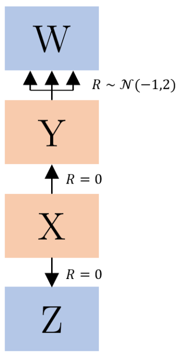


Figure 2: Markov Decision Process example

In this Markov Decision Process, in X, if we decide to take action UP, after a certain number of iterations, the reward will be negative so we should never take the UP action but rather the DOWN action. But we update the Q-value using the max operator and for this action the Q-value could be positive so that the agent would consider the UP action as a good move. Indeed, the estimator consider the maximum Q-value instead of the mean value.

3.2 Illustration

We will study the following usecase[2]. The agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water.

[2]https://gym.openai.com/envs/FrozenLake-v0/

Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.

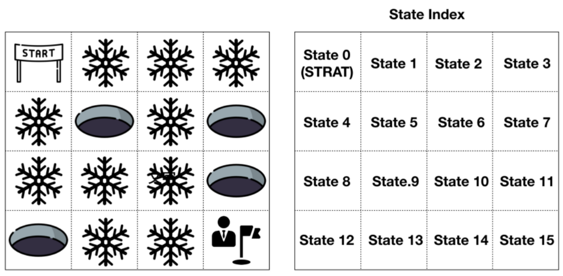


Figure 3: FrozenLake-v0

At the beginning of the game, our agent has an empty $Q(s, a)$ since it knows nothing about the environment. We can visualize our $Q(s, a)$ as a matrix. Each element in this matrix is the Q-value of $Q(s, a)$ in given state s and action a. In the training procedure, the agent needs to explore the environment to know more about it.

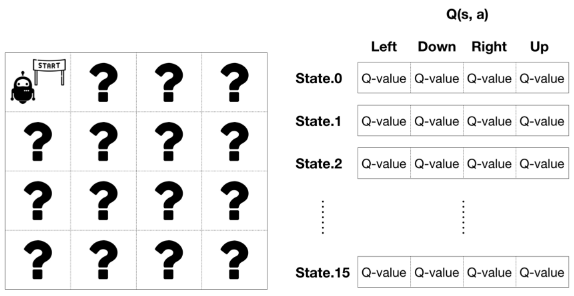


Figure 4: FrozenLake-v0

Here, we suppose that the agent is taking a random action *right*. So, the agent arrives at state 1 and gets the current rewards, the environment returns the maximum expected reward and then the agent updates $Q(s, a)$ based on current and expected rewards.

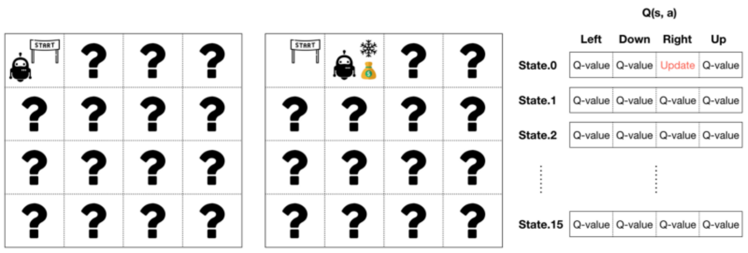
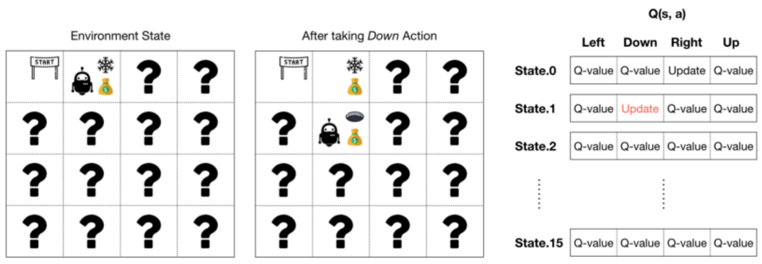


Figure 5: FrozenLake-v0

Here the agent takes action *down* and arrives state 5 : the agent gets rewards and updates $Q(s, a)$. But this training episode is terminated because the agent falls into the hole. So, we have to reset the environment for the next training episode.

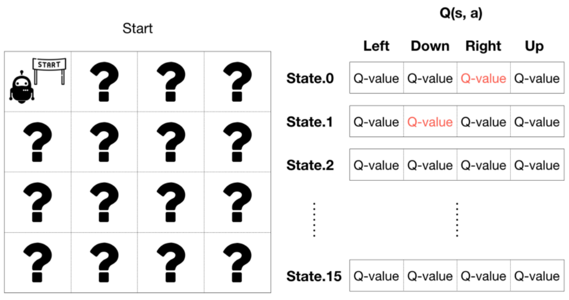


Q(s, a)

	Left	Down	Right	Up
State.0	Q-value	Q-value	Update	Q-value
State.1	Q-value	Update	Q-value	Q-value
State.2	Q-value	Q-value	Q-value	Q-value
⋮		⋮		
State.15	Q-value	Q-value	Q-value	Q-value

Figure 6: FrozenLake-v0

Now, the agent starts at state 0 and according to pass experience, it has learned the policy $Q(s, a)$. In this training episode, the agent keeps learning and takes action by considering current $Q(s, a)$.



Q(s, a)

	Left	Down	Right	Up
State.0	Q-value	Q-value	Q-value	Q-value
State.1	Q-value	Q-value	Q-value	Q-value
State.2	Q-value	Q-value	Q-value	Q-value
⋮		⋮		
State.15	Q-value	Q-value	Q-value	Q-value

Figure 7: FrozenLake-v0

Now, suppose the agent takes *right* action again and *right* again instead of *down* after considering $Q(s, a)$. At each step, the agent gets rewards and updates $Q(s, a)$.

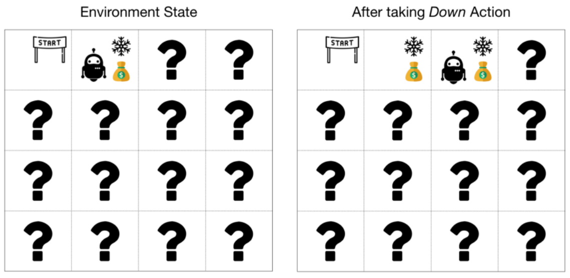


Figure 8: FrozenLake-v0

Little by little, the agent learned $Q(s, a)$ well so now we can reset the environment to test our process and now the agent can reach the goal easily : it has learned the best policy.

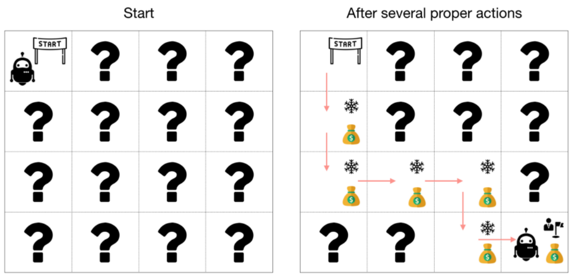


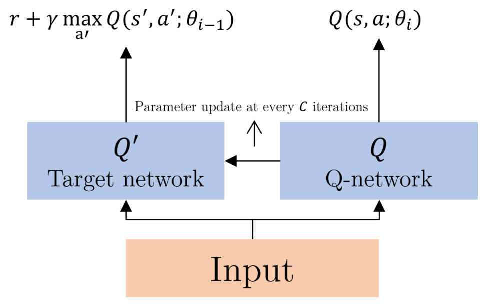
Figure 9: FrozenLake-v0

4 Deep Q-learning

Deep Q-Learning is the method for approximating $Q(s,a)$ with Deep Neural Networks, called Deep Q Network (DQN) and to learn policies from high dimensional sensory input presented in [27]. A DQN is a multi-layered neural network that for a given state s outputs a vector of action values $Q(s,\cdot;\theta)$ where θ are the parameters of the network. For an n-dimensional state space and an action space contraining m actions, the neural network is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.

In the simple version of Q-learning, we don't get any updates if the Temporal Difference Target y_i and $Q(s,a)$ have the same values. In this case $Q(s,a)$ converged to the true action-values and the goal is achieved. We recall that the Temporal Difference of $Q(s,a)$ is the difference between two "versions" of $Q(s,a)$ separated by time once before it takes an action a in state s and once after that.

Now, in Deep Q-learning, y_i and $Q(s,a)$ are estimated separately by two neural networks : the Target-Network and the Q-Network.



$r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$
$Q(s, a; \theta_i)$
Parameter update at every C iterations
Q'
Target network
Q
Q-network
Input

Figure 10: Target-Network and Q-Network

The loss function of Deep Q-Networks is :

$$L_i(\theta_i) = \mathbb{E}_{a\sim\mu}[(y_i - Q(s,a;\theta_i))^2]$$

where

$$y_i := \mathbb{E}_{a'\sim\mu}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | S_t = s, A_t = a]$$

Algorithm 3: Deep Q-learning algorithm in a nutshell

1. **Gather** and **store** samples in a replay buffer with current policy
2. **Experience Replay** Random sample batches of experiences from the replay buffer.
3. **Use** the sampled experiences to update the Q network.
4. **Repeat** steps 1 to 3.

Here, we will explain the 2nd step of the process. Usually, we use past sequential experiences but here we should random sample experiences. Indeed, sequential experiences are highly correlated with each other temporally and we know that, in statistical learning and optimization, data should be independently distributed and not correlated with each other. By using experiences random sampling, it will break the temporal correlation of behavior and distributes it over many of its previous states. Thanks to that, there's no significant oscillations or divergence in the model.

The aim of the 3rd step is to update the Q-network. To do that, we have to minimize the error between the target Q-value and the current Q-output:

$$L(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$$

Where

$$y_i = Y_t^{DQN} = \begin{cases} R_T & \text{for terminal state } s_T \\ R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_t^-) & \text{for non-terminal state } s^t \end{cases}$$

Both the target-network and the experience replay improve the performance of the algorithm.

To decrease the error, the current policy's outputs tends to be equal to the true Q-values. We have to perform a gradient step on the previous loss function :

$$\nabla_{theta_i} L(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot),s'\sim\varepsilon}[(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)]$$

Unfortunately, Deep Q-learning tends to learn unrealistically high $Q(s, a)$. As simple Q-learning, the *max* operator causes the fact that the algorithm sometimes overestimate some action Q-values. This may lead him to believe that certain actions are worthwhile, even though they result in less reward in the end.

In [13], Hasselt illustrates this problem with some experiments:

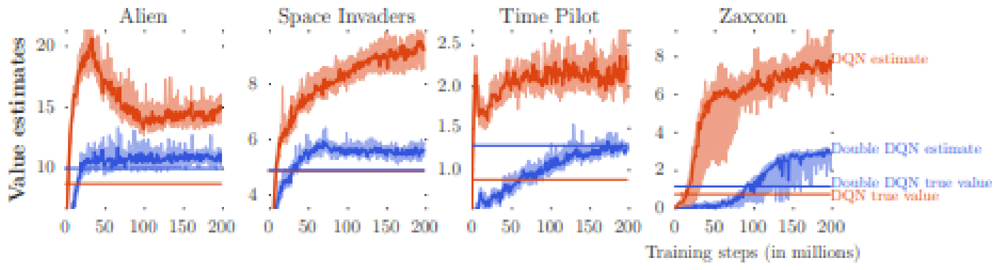


Figure 11: Experiments

5 Double Q-learning

In [12], Hasselt proposes a solution to this problem : Double Q-learning where he used two estimators instead of using one estimator.

Algorithm 4: Double Q-learning algorithm

```
1. Initialize all Q-values $Q_A$, $Q_B$ and the starting state $s$
2. Repeat
    (a) Select the action $a$
    (b) Get the reward $r$ and $s'$ depending on $Q_A(s,\cdot)$ and $Q_B(s,\cdot)$ Q-values
    (c) Update $A$ OR Update $B$ (randomly)
    (d) If $A$ was updated
        i. Select $a*$ : $a* = \arg\max Q_A(s', a)$
        ii. Update $Q_A$ : $Q_A(s, a) = Q_A(s, a) + \alpha[R + \gamma Q_B(s', a*) - Q_A(s, a)]$
    (e) If $B$ was updated
        i. Select $b*$ : $b* = \arg\max Q_B(s', b)$
        ii. Update $Q_B$ : $Q_B(s, a) = Q_B(s, a) + \alpha[R + \gamma Q_A(s', b*) - Q_B(s, a)]$
    (f) $s = s'$
3. Until End
```

For each (S_t, A_t) couple, we have to compute two Q-values : Q_A and Q_B.

$$Q_A(S_t, A_t) = Q_A(S_t, A_t) + \alpha[R + \gamma Q_B(S_{t+1}, \arg\max_a Q_A(S_{t+1}, a)) - Q_A(S_t, a)]$$

$$Q_B(S_t, A_t) = Q_B(S_t, A_t) + \alpha[R + \gamma Q_A(S_{t+1}, \arg\max_a Q_B(S_{t+1}, a)) - Q_B(S_t, A_t)]$$

That means that to update Q_A for example, we have to find an action $a*$ which maximises Q_A in the next state S_{t+1}, use it to compute Q_B in S_{t+1} and then update Q_A in the current state S_t.

These updates means that Double Q-learning decouples the selection from the evaluation. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. The Double Q-learning error can then be written as:

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta'_t)$$

With Double Q-learning, $\mathbb{E}[Q_B(s', a*)] \leqslant \max Q_A(s', a*)$ so after a certain number of iterations the expected value of $Q_B(s', a*)$ will be smaller than the maximal value of $Q_A(s', a*)$. In the same way, $Q_A(s, a)$ will never be updated with the max operator and so it will never be overestimated.

6 Double Deep Q-Learning

6.1 Definition

Double Deep Q-learning combines Double Q-learning and Deep Q-learning. Double Q-learning allows us to avoid overestimations by decomposing the max operator in the target in two distinct steps : action selection and action evaluation. Deep Q-learning estimates y_i and $Q(s, a)$ separately by two neural networks : the Target-Network and the Q-Network.

The Target-Network will compute $Q(s, a_i)$ for each action a_i in the set of possible actions at state s. The greedy policy π will select an action a_i using the highest values $Q(s, a_i)$. So, the Target-Network selects the action a_i and evaluates its quality by calculating $Q(s, a_i)$ at the same time. Recall that the Double Q-learning temporal difference target then be written as:

$$y_i = \mathbb{E}_{a' \sim \mu}[r + \gamma Q(s', \arg\max_a Q(s', a; \theta_i); \theta_{i-1}) | S_t = s, A_t = a]$$

So, while the Target-Network evaluates the action quality, the action is determined by the Q-Network. The calculation of new temporal difference target *y_i* can be put in a nutshell following these steps:

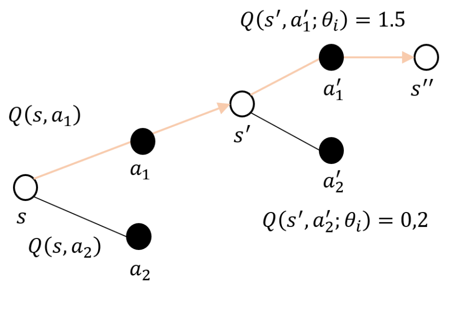
1. The Q-Network will use next state s' to calculate $Q(s', a)$ for each possible action a in s'
2. **Action selection** $\arg\max$ operation applied on $Q(s', a)$ chooses the action $a*$ that belongs to the highest quality
3. **Action evaluation** $Q(s', a*)$ (determined by the Target-Network) belongs to the action $a*$ (determined by the Q-Network) and is selected for the calculation of the target.

Double Deep Q-learning update is the same as for DQN but replacing the target Y_t^{DQN} with :

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

6.2 Illustration

We can visualize the Double Q-Learning process with the following graph:



$$Q(s, a_1) = r + \gamma Q(s', a'_1; \theta_{i-1})$$

Figure 12: Double Deep Q-Learning illustration

The agent is at the starting state s. Based on some previous experiences and calculations the Q-values $Q(s, a_1)$ and $Q(s, a_2)$ are known which respectively correspond to the quality of action

a_1 taken in state s and the quality of action a_2 taken in state s. The agent decides to take action a_1 and, thus, moves in state s'.

The Q-Network will calculate the qualities $Q(s', a'_1)$ and $Q(s, a'_2)$ which respectively correspond to the quality of action a'_1 taken in state s' and the quality of action a'_2 taken in state s'. Then, the agent decides to take action a'_1 because this will result in the highest quality according to the Q-Network.

Now, $Q(s, a_1)$ that corresponds to the quality of action a_1 taken in state s can be calculated knowing that $Q(s', a'_1)$ is the evaluation of a'_1 that is determined by the Target-Network.

7 Experimentation and results

Hasselt [13] experimentation consists of Atari 2600 games, using the Arcade Learning Environment. The goal was for a single algorithm, with a fixed set of hyperparameters, to learn to play each of the games separately from interaction given only the screen pixels as input. The results are the following :

7.1 Results on overoptimism

DQN and Double DQN were both trained under the exact conditions described by [26]. It seemed that DQN is consistently overoptimistic about the value of the current greedy policy.

They obtained the ground truth averaged values by running the best learned policies for several episodes and computing the actual cumulative rewards. DQN consistently results in much higher values than the true values and Double DQN values are much closer to the true values of the final policy. So Double DQN produces more accurate value estimates and better policies. They also saw that learning is much more stable with Double DQN. That means that Q-learning's overoptimism may cause these instabilities.

7.2 Quality of the learned policies

They showed that overoptimism doesn't always affect the learned policy quality. Indeed, despite overestimating the policy value, DQN can achieve optimal behavior But reducing overestimations can significantly benefit the learning stability.

They used same hyperparameters for Double DQN and DQN, to allow for a controlled experiment focused just on reducing overestimations.

In several games, Double DQN greatly improves upon DQN.

7.3 Robustness to Human starts

In deterministic games with a unique starting point, the learner could potentially learn to remember sequences of actions without needing to generalize. Even if it's successful, the solution would not be robust. They decided to test the agents from various starting points to test whether the found solutions generalize well, and as such provide a challenging testbed for the learned polices. Double DQN got higher scores. In several games, Double DQN greatly improves upon DQN and in some cases the scores are much closer to human, or even surpassing these. It seemed that Double DQN is more robust. It suggests that appropriate generalizations occur and that the found solutions do not exploit the determinism of the environments.

8 Literature review

We have witnessed in recent years the rise of deep learning, relying on powerful function approximation and representation learning properties of deep neural networks.

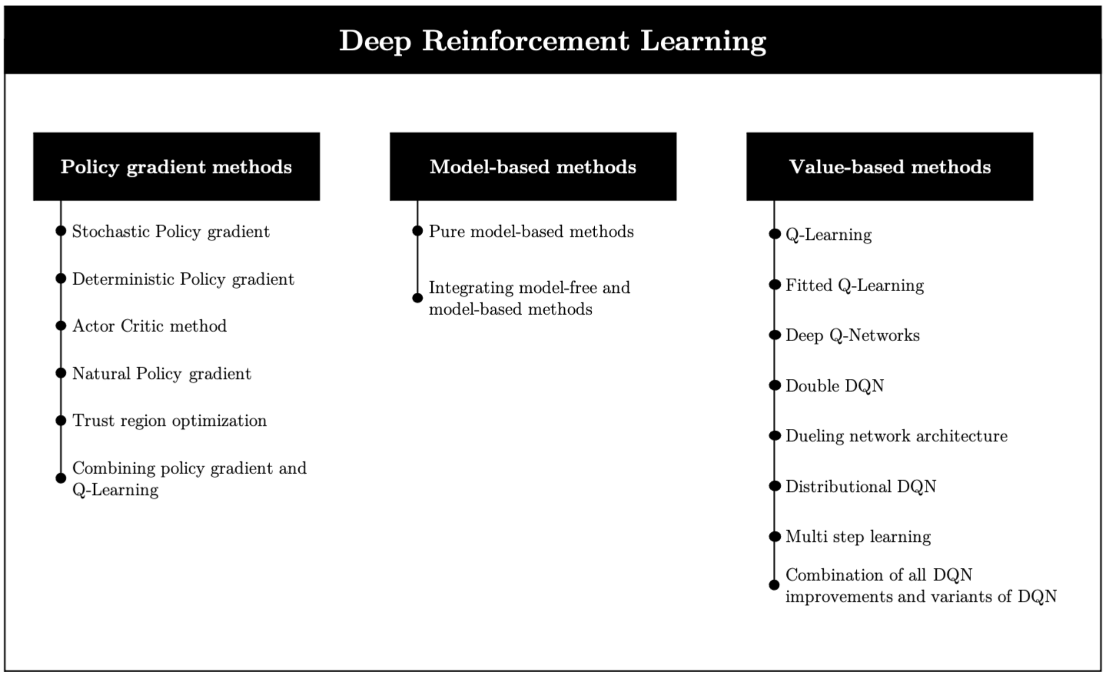

Deep Reinforcement Learning
Policy gradient methods
Stochastic Policy gradient
Deterministic Policy gradient
Actor Critic method
Natural Policy gradient
Trust region optimization
Combining policy gradient and Q-Learning
Model-based methods
Pure model-based methods
Integrating model-free and model-based methods
Value-based methods
Q-Learning
Fitted Q-Learning
Deep Q-Networks
Double DQN
Dueling network architecture
Distributional DQN
Multi step learning
Combination of all DQN improvements and variants of DQN

Figure 13: Reinforcement learning process

8.1 Value-based methods

Value function methods are based on estimating the value (expected return) of being in a given state. The state-value function $V^{\pi}(s)$ is the expected return when starting in state s and following π henceforth:

$$V^{\pi}(s) = \mathbb{E}[R|s, \pi]$$

The optimal policy, $\pi*$, has a corresponding state-value function $V^{(s)}$, and vice-versa, the optimal state-value function can be defined as

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

If we had $V^{*}(s)$ available, the optimal policy could be retrieved by choosing among all actions available at st and picking the action a that maximises $\mathbb{E}_{s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t,a)}$. In the RL setting, the transition dynamics $\mathcal{T}$ are unavailable. Therefore, we construct another function, the state-action value or quality function $Q^{\pi}(s, a)$, which is similar to V^{π}, except that the initial action a is provided, and π is only followed from the succeeding state onwards:

$$Q^{\pi}(s, a) = \mathbb{E}[R|s, a, \pi]$$

The best policy, given $Q^{\pi}(s, a)$, can be found by choosing a greedily at every state: $\underset{a}{\text{argmax}} Q^{\pi}(s, a)$. Under this policy, we can also define $V^{\pi}(s)$ by maximising $Q^{\pi}(s, a) : V^{\pi}(s) = \max_{a} Q^{\pi}(s, a)$.

The function approximation properties of neural networks have naturally led to the use of deep learning to regress functions to be used in RL agents. Indeed, one of the earliest successes in the field of RL was TD-Gammon, a neural network that achieved expert-level performance in backgammon in the early 1990s [47]. Advances in RL research have subsequently promoted the explicit use of value functions to capture the underlying structure of the environment. Since the first value function methods in DRL, which took simple states as input [37], current methods are now able to tackle visually and conceptually complex environments [26], [44], [25], [32], [60].

We will first study value-function based DRL algorithms by the deep Q-networks (DQN) [26], which has achieved scores on a wide range of classic Atari 2600 video games [3] comparable to a professional video game tester. In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output.

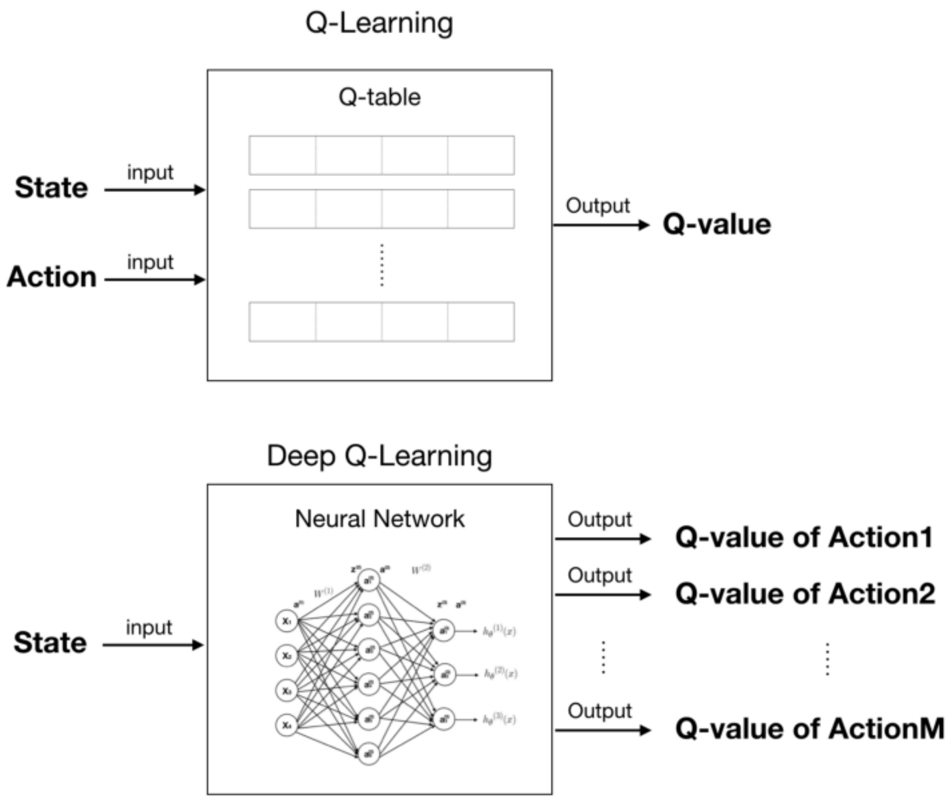


Figure 14: Q-Learning Vs Deep Q-Learning

A precursor to DQN (NFQ) was to train a neural network to return the Q-value based on a state-action pair [37]. NFQ was then extended to train a network to drive a sub-machine car from raw visual inputs from a camera above the racetrack, combining a deep autoencoder to reduce the dimensionality of the inputs with a separate branch to predict the Q-values [21].

This algorithm was the first RL algorithm shown to work directly from raw visual inputs and on a wide variety of environments. DQN learns to extract salient visual features, jointly encoding objects, their movements, and, most importantly, their interactions.

DQN addressed the fundamental instability problem of using function approximation in RL [48] using two techniques: experiment replay [23] and target networks.

1. While the original DQN algorithm used uniform sampling [26], later work showed that prioritizing samples based on TD errors was more effective for learning [39]. We note that although experiment replay is generally considered a model-free technique, it could actually be considered a simple model [49].

2. The second stabilization method, introduced by Mnih et al [26], involves using a target network that initially contains the weights of the network applying the policy, but is kept frozen for an extended period of time. Rather than having to compute the TD error based on its own estimates of rapidly fluctuating Q-values, the policy network uses the fixed target network. During training, the target network weights are updated to match the policy network after a fixed number of steps. Experience replay networks and target networks have been used in subsequent work on DRL [9], [22], [53], [29].)

Van Hasselt [12] showed that the single estimator used in the Q-learning update rule overestimates the expected performance due to the use of the maximum action value as an approximation to the maximum expected action value. Dual Q-learning provides a better estimate through the use of a dual estimator [12]. While dual Q-learning requires learning an additional function, the paper we chose to study proposed using the already available target network of the DQN algorithm, which achieved much better results with only a small modification to the update step.

A more radical proposal by Bellemare et al [4] was to learn the full distribution of values, rather than just the expectation, which provides additional information, such as whether the potential rewards come from a skewed or multimodal distribution. Although the resulting algorithm, based on learning categorical distributions, was used to construct the categorical DQN, the benefits can potentially be applied to any RL algorithm that uses learned value functions.

Other modifications have been suggested to the DQN architecture such as decomposing the Q-function into meaningful functions, such as constructing Q^{π} by adding together separate layers that compute the state-value function V^{π} and advantage function A^{π} [52]
Another implementation is the duelling DQN [52] which has a single baseline for the state in the form of V^{π}, and more learnable relative values in the form of A^{π}. The combination of duelling DQN with prioritized experience replay [39] is one of the leading techniques in discrete action settings.
A better understanding of the properties of A^{π} by Gu et al [9] led them to modify DQN and proposed yet another new DQN architecture that allows the algorithm to be generalized to continuous action sets, creating the normalized advantage function (NAF) algorithm that is one of the benchmark techniques in DRL.
Finally Metz et al [24] constructed an autoregressive policy, where each action in a single time step is predicted conditionally on the state and previously chosen actions in the same time step to construct the sequential DQN, which allowed them to discretize a large action space and outperform NAF in continuous control problems.

8.2 Policy gradient methods

Policy search methods do not need to maintain a value function model, but directly search for an optimal policy $\pi*$. We choose a parameterized policy π_{θ} and update the parameters to maximize the expected return $\mathbb{E}[R|\theta]$ using gradient-based optimization. However, to compute the expected return V^{π}, we need to average the plausible paths induced by the current policy parameterization. This averaging requires either deterministic approximations (e.g., linearization) or stochastic sampling approximations [6]. In the more common model-free RL framework, a Monte Carlo estimate of the expected performance is determined. For gradient-based learning, this Monte Carlo approximation is problematic since gradients cannot pass through these samples of a stochastic function. Therefore, we turn to a gradient estimator, known in LR as the REIN- FORCE rule [58].

The workhorse of DRL is backpropagation [57], [38]. The REINFORCE rule [58], discussed earlier, allows neural networks to learn stochastic policies based on the task at hand, such as deciding where to look in an image to track [40], classify [28], or caption objects [59]. In these cases, the stochastic variable determines the coordinates of a small portion of the image, reducing the amount of computation required. This use of RL to make discrete, stochastic decisions about inputs is known in the deep learning literature as hard attention, and is one of the most compelling uses of basic policy search methods in recent years. More generally, the ability to backpropagate through stochastic functions, using techniques such as REINFORCE [58] or the "reparametrization trick" [20], [36], allows neural networks to be treated as stochastic computational graphs on which to optimize [41], which is a key concept in algorithms such as stochastic value gradients (SVGs) [15].

Directly finding a policy represented by a neural network with very many parameters can be difficult and may suffer from large local minima. One way to circumvent this problem is to use a confidence region, in which the optimization steps are restricted to a region in which the approximation to the true cost function still holds. By preventing updated policies from deviating too much from previous policies, the risk of error is reduced. The idea of constraining each policy gradient update, as measured by the Kullback-Leibler (KL) divergence between the current and proposed policy, is not new to LR [17], [2], [19], [34]. One of the most recent algorithms in this area, Trusted Region Policy Optimization (TRPO), has been shown to be relatively robust and applicable to domains with high-dimensional inputs [44]. While TRPO can be used as a pure policy gradient method with a simple baseline, later work by Schulman et al [42] introduced generalized advantage estimation (GAE), which offers several more advanced variance reduction baselines. The combination of TRPO and GAE remains one of the most advanced LR techniques in the continuous control domain. However, the constrained optimization of TRPO requires the computation of second-order gradients, which limits its applicability. In contrast, the more recent proximal policy optimization (PPO) algorithm performs unconstrained optimization, requiring only first-order gradient information [14], [43].

Instead of using the average of multiple Monte Carlo returns as a baseline for policy gradient methods, critical actor approaches have gained popularity as an effective way to combine the benefits of policy search methods with learned value functions, which are capable of learning from full returns and/or TD errors. They can benefit from improvements in policy gradient methods, such as GAE [42], and value function methods, such as target networks [26].

A recent development in the context of actor-critic algorithms are deterministic policy gradients (DPGs) [45], which extend the standard policy gradient theorems for stochastic policies [58] to deterministic policies. One of the main advantages of DPGs is that, while stochastic policy gradients incorporate both state and action spaces, DPGs incorporate only the state space, requiring fewer samples in problems with large action spaces. In early work on DPGs, Silver et al [45] presented and demonstrated an actor-critic off-policy algorithm that significantly improved an equivalent stochastic policy gradient in high-dimensional continuous control problems. Later work introduced deep DPG (DDPG), which uses neural networks to operate on high-dimensional visual state spaces [22]. In the same vein as DPGs, Heess et al [15] devised a gradient method for optimizing stochastic policies, by "reparameterizing" [20], [36] the stochasticity out of the network, allowing the use of standard gradients (instead of the high variance REINFORCE estimator [58]) which gives rise to the Stochastic Value Gradient (SVG) methods.

Value functions introduce an advantage that is widely applicable in actor-critical methods: the possibility to use non-policy data. Methods with policy can be more stable, while non-policy methods can be more data efficient, so there have been several attempts to merge the two [53], [30], [10], [8], [11]. Previous work has either used a mixture of on-policy and off-policy gradient updates [53], [30], [8], or used the off-policy data to train a value function to reduce the variance of the on-policy gradient updates [10]. More recent work by Gu et al [11] unified these methods into the interpolated policy gradient (IPG) framework, resulting in one of the newest continuous DRL algorithms in the state of the art, and also provided guidance for future research in this area.

8.3 Model based-method

The main idea of model-based RL is to understand the environment and create a model to represent it. Model-based RL does not assume specific prior knowledge, however, to speed up learning, one can incorporate prior knowledge about the environment (e.g., physics-based models [18]). Model-based learning plays an important role in reducing the number of required interactions with the real environment, which may be limited in practice.

These models are used in many fields. DRL algorithms based on deep dynamic models [51], in which high-dimensional observations are embedded in a lower-dimensional space using autoencoders have been proposed for learning models and policies from pixel information [31], [56], [50]. Some algorithms learn sufficiently accurate models of the environment that even simple controllers can control a robot directly from camera images [7]. Deep models allow these techniques to be extended to highly dimensional visual domains [46].

Indeed, by embedding the activations and predictions of these models in a vector, a DRL agent can, not only obtain more information than just the final result, but it can also learn to minimize this information.

This approach is particularly attractive when the agent thinks the model is inaccurate [35]. There are other methods to reduce inaccuracy such as Bayesian methods of uncertainty propagation [16].

Another way to use the flexibility of neural network-based models is to let them decide when to plan, given a finite amount of computation, whether it is worthwhile to model one long path, several short paths, anything in between, or simply take an action in the real environment [33].

Although deep neural networks can make reasonable predictions in simulated environments over hundreds of time steps [5], they typically require a large number of observations to tune all parameters. For this reason, Gu et al [9] developed locally linear models for use with the NAF algorithm (the continuous equivalent of DQN [26]) to improve the sampling complexity of the algorithm in the robotics domain where samples are expensive.

References

[1] Kai Arulkumaran et al. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017), pp. 26–38. ISSN: 1053-5888. DOI: 10.1109/msp.2017.2743240. URL: http://dx.doi.org/10.1109/MSP.2017.2743240.

[2] J. Andrew Bagnell and Jeff G. Schneider. "Covariant Policy Search". In: *IJCAI*. 2003, pp. 1019–1024. URL: http://ijcai.org/Proceedings/03/Papers/146.pdf.

[3] M. G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279. ISSN: 1076-9757. DOI: 10.1613/jair.3912. URL: http://dx.doi.org/10.1613/jair.3912.

[4] Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG].

[5] Silvia Chiappa et al. *Recurrent Environment Simulators*. 2017. arXiv: 1704.02254 [cs.AI].

[6] Marc Deisenroth, Gerhard Neumann, and Jan Peters. *A Survey on Policy Search for Robotics*. Vol. 2. Aug. 2013.

[7] Chelsea Finn et al. *Deep Spatial Autoencoders for Visuomotor Learning*. 2016. arXiv: 1509.06113 [cs.LG].

[8] Audrunas Gruslys et al. *The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning*. 2018. arXiv: 1704.04651 [cs.AI].

[9] Shixiang Gu et al. *Continuous Deep Q-Learning with Model-based Acceleration*. 2016. arXiv: 1603.00748 [cs.LG].

[10] Shixiang Gu et al. *Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic*. 2017. arXiv: 1611.02247 [cs.LG].

[11] Shixiang (Shane) Gu et al. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/a1d7311f2a312426d710e1c617fcbc8c-Paper.pdf.

[12] Hado Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[13] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[14] Nicolas Heess et al. *Emergence of Locomotion Behaviours in Rich Environments*. 2017. arXiv: 1707.02286 [cs.AI].

[15] Nicolas Heess et al. *Learning Continuous Control Policies by Stochastic Value Gradients*. 2015. arXiv: 1510.09142 [cs.LG].

[16] Rein Houthooft et al. *VIME: Variational Information Maximizing Exploration*. 2017. arXiv: 1605.09674 [cs.LG].

[17] Sham M Kakade. "A Natural Policy Gradient". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[18] Ken Kansky et al. *Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics*. 2017. arXiv: 1706.04317 [cs.AI].

[19] H J Kappen. "Path integrals and symmetry breaking for optimal control theory". In: *Journal of Statistical Mechanics: Theory and Experiment* 2005.11 (Nov. 2005), P11011–P11011. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2005/11/p11011. URL: http://dx.doi.org/10.1088/1742-5468/2005/11/P11011.

[20] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[21] Sascha Lange, Martin Riedmiller, and Arne Voigtländer. "Autonomous reinforcement learning on raw visual input data in a real world application". In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 2012, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252823.

[22] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[23] Long-ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning*. 1992, pp. 293–321.

[24] Luke Metz et al. *Discrete Sequential Prediction of Continuous Actions for Deep RL*. 2019. arXiv: 1705.05035 [cs.LG].

[25] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[26] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[27] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[28] Volodymyr Mnih et al. "Recurrent Models of Visual Attention". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/09c6c3783b4a70054da74f2538ed47c6-Paper.pdf.

[29] Ofir Nachum et al. *Bridging the Gap Between Value and Policy Based Reinforcement Learning*. 2017. arXiv: 1702.08892 [cs.AI].

[30] Brendan O'Donoghue et al. *Combining policy gradient and Q-learning*. 2017. arXiv: 1611.01626 [cs.LG].

[31] Junhyuk Oh et al. *Action-Conditional Video Prediction using Deep Networks in Atari Games*. 2015. arXiv: 1507.08750 [cs.LG].

[32] Junhyuk Oh et al. *Control of Memory, Active Perception, and Action in Minecraft*. 2016. arXiv: 1605.09128 [cs.AI].

[33] Razvan Pascanu et al. *Learning model-based planning from scratch*. 2017. arXiv: 1707.06170 [cs.AI].

[34] Jan Peters, Katharina Mülling, and Yasemin Altun. "Relative Entropy Policy Search". In: *In Proceedings of the Conference on Artificial Intelligence*. 2010.

[35] Sébastien Racanière et al. "Imagination-Augmented Agents for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/9e82757e9a1c12cb710ad680db11f6f1-Paper.pdf.

[36] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. arXiv: 1401.4082 [stat.ML].

[37] Martin Riedmiller. "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method". In: *Machine Learning: ECML 2005*. Ed. by João Gama et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328. ISBN: 978-3-540-31692-3.

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[39] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG].

[40] Jürgen Schmidhuber and Rudolf Huber. *Learning To Generate Artificial Fovea Trajectories For Target Detection*. 1991.

[41] John Schulman et al. *Gradient Estimation Using Stochastic Computation Graphs*. 2016. arXiv: 1506.05254 [cs.LG].

[42] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation.* 2018. arXiv: `1506.02438 [cs.LG]`.

[43] John Schulman et al. *Proximal Policy Optimization Algorithms.* 2017. arXiv: `1707.06347 [cs.LG]`.

[44] John Schulman et al. *Trust Region Policy Optimization.* 2017. arXiv: `1502.05477 [cs.LG]`.

[45] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *ICML*. Beijing, China, June 2014. URL: `https://hal.inria.fr/hal-00938992`.

[46] Bradly Stadie, Sergey Levine, and Pieter Abbeel. "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models". In: (July 2015).

[47] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon." In: *J. Int. Comput. Games Assoc.* 18.2 (1995), p. 88. URL: `http://dblp.uni-trier.de/db/journals/icga/icga18.html#Tesauro95`.

[48] J.N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690. DOI: `10.1109/9.580874`.

[49] Harm Vanseijen and Rich Sutton. "A Deeper Look at Planning as Learning from Replay". In: *Proceedings of the 32nd International Conference on Machine Learning.* Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2314–2322. URL: `http://proceedings.mlr.press/v37/vanseijen15.html`.

[50] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. *From Pixels to Torques: Policy Learning with Deep Dynamical Models.* 2015. arXiv: `1502.02251 [stat.ML]`.

[51] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. *Learning deep dynamical models from image pixels.* 2014. arXiv: `1410.7550 [stat.ML]`.

[52] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning.* 2016. arXiv: `1511.06581 [cs.LG]`.

[53] Ziyu Wang et al. *Sample Efficient Actor-Critic with Experience Replay.* 2017. arXiv: `1611.01224 [cs.LG]`.

[54] C. J. C. H. Watkins. "Learning from Delayed Rewards". PhD thesis. King's College, Oxford, 1989.

[55] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: `10.1007/BF00992698`. URL: `https://doi.org/10.1007/BF00992698`.

[56] Manuel Watter et al. *Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images.* 2015. arXiv: `1506.07365 [cs.LG]`.

[57] P. J. Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences". PhD thesis. Harvard University, 1974.

[58] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8 (1992), pp. 229–256.

[59] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.* 2016. arXiv: `1502.03044 [cs.LG]`.

[60] Yuke Zhu et al. *Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning.* 2016. arXiv: `1609.05143 [cs.CV]`.