



Moscow State University
Faculty of Computational Mathematics and Cybernetics
Department of Mathematical Methods of Forecasting

Modern Deep Reinforcement Learning Algorithms

Incomplete draft
March 23, 2019

Made by:
student of 517 group
Sergey Ivanov

Scientific advisor:
Alexander D'yakonov

Moscow, 2019

Contents

1	Introduction	4
2	Reinforcement Learning problem setup	5
2.1	Assumptions of RL setting	5
2.2	Environment model	6
2.3	Objective	6
2.4	Value functions	8
2.5	Classes of algorithms	8
2.6	Measurements of performance	9
3	Value-based algorithms	10
3.1	Temporal Difference learning	10
3.2	Deep Q-learning (DQN)	12
3.3	Double DQN	14
3.4	Dueling DQN	15
3.5	Noisy DQN	16
3.6	Prioritized experience replay	17
3.7	Multi-step DQN	18
4	Distributional approach for value-based methods	20
4.1	Theoretical foundations	20
4.2	Categorical DQN	22
4.3	Quantile Regression DQN (QR-DQN)	24
4.4	Rainbow DQN	26
5	Policy Gradient algorithms	29
5.1	REINFORCE	29
5.2	Advantage Actor-Critic (A2C)	29
5.3	Generalized Advantage Estimation (GAE)	31
5.4	Natural Policy Gradient (NPG)	31
5.5	Trust-Region Policy Optimization (TRPO)	31
5.6	Proximal Policy Optimization (PPO)	31
6	Overview of other approaches	31
7	Experiments	31
8	Conclusion	31

Abstract

Recent advances in Reinforcement Learning, grounded on combining classical theoretical results with Deep Learning paradigm, led to breakthrough results in a wide variety of tasks and gave birth to Deep Reinforcement Learning (DRL) as a field of research. In this work latest DRL algorithms are reviewed with a focus on their theoretical justification, practical limitations and observed empirical properties.

1. Introduction

In the last several years Deep Reinforcement Learning proved to be a fruitful approach to many artificial intelligence tasks of diverse domains. Breakthrough achievements include reaching human-level performance in such complex games as Go [13], multiplayer Dota [10] and real-time strategy StarCraft II [16]. At the same time application of DRL framework is not limited to discrete-action cases and can be utilized in continuous domain to solve tasks of optimal control in robotics and simulated environments [8].

Reinforcement Learning (RL) is usually viewed as general formalization of decision-making task and is deeply connected to dynamic programming, optimal control and game theory. [14] Yet its problem setting makes almost no assumptions about world model or its structure and usually supposes that environment is given to agent in a form of black-box. This allows RL to be applied practically in all settings and forces designed algorithms to be adaptive to many kinds of challenges. Latest RL algorithms are usually reported to be transferable from one task to another with no task-specific changes and little to no hyperparameters tuning.

As an object of desire is a strategy, its mathematical nature is a function mapping agent's observations to possible actions. That is why reinforcement learning is considered to be a subfield of machine learning, but instead of learning from data, as it is established in classical supervised and unsupervised learning problems, the agent learns from experience of interacting with environment. Being more "natural" model of learning, this setting causes new challenges, peculiar only to reinforcement learning, such as necessity of exploration integration and the problem of delayed and sparse reward. The full setup and essential notation are introduced in section 2.

Classical Reinforcement Learning research in the last third of previous century developed an extensive theoretical core for modern algorithms to ground on. Several algorithms were known ever since and were able to solve small-scale problems when either environment states can be enumerated (and stored in the memory) or optimal policy can be searched in the space of linear or quadratic functions of state representation features. Although these restrictions are extremely limiting, foundations of classical RL theory underlie modern approaches. These theoretical fundamentals are discussed in sections 3.1 and 5.1.

Combining this framework with Deep Learning [4] was popularized by Deep Q-Learning algorithm, introduced in [9], which was able to play any of 57 Atari console games without tweaking network architecture or algorithm hyperparameters. This novel approach was extensively researched and significantly improved in the following years. These principles of value-based direction in deep reinforcement learning are presented in section 3.

One of the key ideas in the recent value-based DRL research is distributional approach, proposed in [1]. Further extending classical theoretical foundations and coming with practical DRL algorithms, it gave birth to distributional reinforcement learning paradigm, which potential is now being actively investigated. Its ideas are described in section 4.

Second main direction of DRL research is policy gradients methods, which attempt to directly optimize the objective function, explicitly present in problem setup. Their application to neural nets involve a series of particular obstacles, which were being overcome with different rate of success during recent years. Today they represent a competitive and scalable approach in deep reinforcement learning due to their enormous parallelization potential and continuous domain applicability. Policy gradient methods are discussed in section 5.

Despite seemingly different nature, these two branches of RL algorithms are closely related and, in certain conditions, may be equivalent [12]. Substantially alternative approaches are discussed in section 6.

Despite the wide range of successes, current state-of-art DRL methods still face a number of significant drawbacks. As training of neural networks requires huge amounts of data, DRL demonstrates unsatisfying results in settings where data generation is expensive. In cases, where interaction is nearly free (i. e. in simulated environments), DRL algorithms tend to require excessive amounts of iterations, which raises their computational and wall-clock time cost. Furthermore, DRL suffers from random initialization and hyperparameters sensitivity, and its optimization process is known to be uncomfortably unstable [7]. Especially embarrassing consequence of these DRL features turned out to be low reproducibility of empirical observations from different research groups [5]. In section 7, we attempt to launch algorithms on several standard environments and discuss practical nuances of modern RL algorithms application.

2. Reinforcement Learning problem setup

2.1. Assumptions of RL setting

Informally, the process of sequential decision-making proceeds as follows. The **agent** is provided with some initial observation of environment and is required to choose some action from given set of possibilities. The **environment** responds by transitioning to another state and generating a **reward signal** (scalar number), which is considered to be a ground-truth estimation of agent performance. The process continues repeatedly with agent making choices of actions from observations and environment responding with next states and reward signals. The only goal of agent is to maximize cumulative reward.

This description of learning process model already introduces several key assumptions. Firstly, the time space is considered to be discrete, as agent interacts with environment sequentially. Secondly, it is assumed that provided environment incorporates some reward function as supervised indicator of goal achievement. This is an embodiment of the **reward hypothesis**, also referred to as **Reinforcement Learning hypothesis**:

Proposition 1. (Reward Hypothesis)

All of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).

Exploitation of this hypothesis draws a line between reinforcement learning and classical machine learning settings, supervised and unsupervised learning. Unlike unsupervised learning, RL assumes supervision, which, similar to labels in data for supervised learning, has a stochastic nature and represents a sole source of learning. At the same time, no data or «right answer» is provided to training procedure, which distinguishes RL from standard supervised learning. Moreover, RL is the only machine learning task providing explicit objective function (cumulative reward signal) to maximize, while in supervised and unsupervised setting optimized loss function is usually constructed by engineer and is not "included" in data. The fact that reward signal is incorporated in environment is considered to be one of the weakest points of RL paradigm, as for many real-life human goals introduction of this scalar reward signal is at the very least unobvious.

For practical applications it is also natural to assume that agent's observations can be represented by some feature vectors, i. e. elements of \mathbb{R}^d . The set of possible actions in most practical applications is usually uncomplicated and is either discrete (number of possible actions is finite) or can be represented as subset of \mathbb{R}^m (almost always $[-1, 1]^m$ or can be reduced to this case)¹. RL algorithms are usually restricted to these two cases, but the mix of two (agent is required to choose both discrete and continuous quantities) can also be considered.

The final assumption of RL paradigm is a **Markovian property**:

Proposition 2. (Markovian property)

Transition to some state depends solely on previous state and the last chosen action and is independent of all previous interaction history.

Although this assumption may seem overly strong, it actually formalizes the fact that the world modeled by considered environment obeys some general laws. Giving that the agent knows the current state of the world and the laws, it is assumed that it is able to predict the consequences of his actions up to the internal stochasticity of these laws. In practice, both laws and complete state representation is unavailable to agent, which limits its forecasting capability.

In the sequel we will work within the setting with one more assumption of **full observability**. This simplification supposes that agent can observe complete world state, while in many real-life tasks only a part of observations is actually available. This restriction of RL theory can be removed by considering **Partially observable Markov Decision Processes (PoMDP)**, which basically forces learning algorithms to have some kind of memory mechanism to store previously received observations. Further on we will stick to fully observable case.

¹this set is considered to be permanent for all states of environment without any loss of generality as if agent chooses invalid action the world may remain in the same state with zero reward signal or punish agent with negative reward or choosing valid action for him.

2.2. Environment model

Though the definition of **Markov Decision Process (MDP)** varies from source to source, its essential meaning remains the same. The definition below utilizes several simplifications without loss of generality.²

Definition 1. *Markov Decision Process (MDP)* is a tuple $(\mathcal{S}, \mathcal{A}, T, r, s_0)$, where:

- $\mathcal{S} \subseteq \mathbb{R}^d$ — arbitrary set, called the **state space**.
- \mathcal{A} — a set, called the **action space**, either
 - **discrete**: $|\mathcal{A}| < +\infty$, or
 - **continuous domain**: $\mathcal{A} = [-1, 1]^m$.
- \mathbb{T} — **transition probability** $p(s' | s, a)$, where $s, s' \in \mathcal{S}, a \in \mathcal{A}$.
- $r : \mathcal{S} \rightarrow \mathbb{R}$ — **reward function**.
- $s_0 \in \mathcal{S}$ — starting state.

It is important to notice that in the most general case the only things available for RL algorithm beforehand are d (dimension of state space) and action space \mathcal{A} . The only possible way of collecting more information for agent is to interact with provided environment and observe s_0 . It is obvious that the first choice of action a_0 will be probably random. While the environment responds by sampling $s_1 \sim p(s_1 | s_0, a_0)$, this distribution, defined in \mathbb{T} and considered to be a part of MDP, may be unavailable to agent's learning procedure. What agent does observe is s_1 and reward signal $r_1 = r(s_1)$ and it is the key information gathered by agent from interaction experience.

Definition 2. The tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is called **transition**. Several sequential transitions are usually referred to as **roll-out**. Full track of observed quantities

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3 \dots$$

is called a **trajectory**.

In general case, the trajectory is infinite which means that interaction process is neverending. However, in most practical cases the **episodic property** holds, which basically means that the interaction comes to some sort of an end³. Formally, it can be simulated by the environment sticking in the last state with zero probability of transitioning to any other state and zero reward signal. Then it is convenient to consider trajectory ending at some finite T -th timestep and reset the environment back to s_0 to initiate new interaction. One interaction cycle from s_0 till reset, spawning one trajectory, is called an **episode**. Without loss of generality, it can be considered that there exists a set of **terminal states** \mathcal{S}^+ , which mark the ends of interactions. By convention, transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ are accompanied with binary flag $\text{done}_{t+1} \in \{0, 1\}$, whether s_{t+1} belongs to \mathcal{S}^+ . As timestep t at which the transition was gathered is usually of no importance, transitions are often denoted as $(s, a, r', s', \text{done})$, marking the «next timestep» with a prime.

Note that the length of episode T may vary between different interactions, but the episodic property holds if interaction is guaranteed to end after some finite time T^{\max} . If this is not the case, the task is called **continuing**.

2.3. Objective

In reinforcement learning, the agent's goal is to maximize a cumulative reward. In episodic case, this reward can be expressed as summation of all received reward signals during one episode and is

²the reward function is often introduced as stochastic and dependent on action a , i. e. $R(r | s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R})$, while instead of fixed s_0 a distribution over \mathcal{S} is given. Both extensions can be taken into account in terms of presented definition by extending the state space and incorporating all uncertainty into transition probability \mathbb{T} .

³natural examples include end of the game or agent's failure/success in completing some task.

called **the return**:

$$R := \sum_{t=1}^T r_t \quad (1)$$

Note that this quantity is formally a random variable, which depends on agent's choices and the outcomes of environment transitions. As this stochasticity is an inevitable part of interaction process, the underlying distribution from which r_t is sampled must be properly introduced to set rigorously the task of return maximization.

Definition 3. Agent's algorithm for choosing a by given current state s , which in general can be viewed as distribution $\pi(a | s)$ on domain \mathcal{A} , is called **policy** (strategy).

Deterministic policy, when the policy is represented by deterministic function $\pi: S \rightarrow \mathcal{A}$, can be viewed as a particular case of **stochastic policy**, when policy $\pi(a | s)$ is degenerated. In this case, agent's output is still a distribution with zero probability to choose an action other than $\pi(s)$. In stochastic case, agent sends to environment a sample $a \sim \pi(a | s)$.

Note that given some policy $\pi(a | s)$ and transition probabilities \mathbb{T} , the complete interaction process becomes defined from probabilistic point of view:

Definition 4. For given MDP and policy π , the probability of observing

$$s_0, a_0, s_1, a_1, s_2, a_2 \dots$$

is called **trajectory distribution** and is denoted as \mathcal{T}_π :

$$\mathcal{T}_\pi := \prod_{t=0} p(s_{t+1} | s_t, a_t) \pi(a_t | s_t)$$

It is always substantial to keep track of what policy was used to collect certain transitions (roll-outs and episodes) during the learning procedure, as they are essentially samples from corresponding trajectory distribution.

Now when a policy induces a trajectory distribution, it is possible to formulate a task of **expected reward** maximization:

$$\mathbb{E}_{\mathcal{T}_\pi} \sum_{t=1}^T r_t \rightarrow \max_{\pi}$$

To ensure the finiteness of this expectation and avoid the case when agent is allowed to gather infinite reward, limit on absolute value of r_t can be assumed:

$$|r_t| \leq R^{\max}$$

Together with the limit on episode length T^{\max} this restriction guarantees finiteness of optimal expected reward.

To extend this intuition to continuing tasks, the reward for each next interaction step is multiplied on some discount coefficient $\gamma \in [0, 1)$, which is often introduced as part of MDP. This corresponds to logic that with probability $1 - \gamma$ agent "dies" and does not gain any additional reward, which models the paradigm "better now than later". In practice, this discount factor is set very close to 1.

Definition 5. For given MDP, policy π the **discounted expected reward** is defined as

$$J(\pi) := \mathbb{E}_{\mathcal{T}_\pi} \sum_{t=0} \gamma^t r_{t+1}$$

Reinforcement learning task is to find an **optimal policy** π^* , which maximizes the discounted expected reward:

$$J(\pi) \rightarrow \max_{\pi} \quad (2)$$

2.4. Value functions

Solving reinforcement learning task (2) usually leads to a policy, that maximizes the expected reward not only for starting state s_0 , but for any state $s \in \mathcal{S}$. That follows from Markov property: the reward which is yet to be collected from some step t does not depend on previous history and for agent staying at state s the task of behaving optimal is equivalent to maximization of expected reward with current state s as starting state. This is the particular reason why many reinforcement learning algorithms do not only find optimal policy, but additional information about usefulness of each state.

Definition 6. For given MDP and policy π the **value function under policy π** is defined as

$$V^\pi(s) := \mathbb{E}_{\mathcal{T}_\pi | s_0=s} \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

This value function estimates how good it is for agent utilizing strategy π to visit state s and generalizes the notion of discounted expected reward $J(\pi)$ that corresponds to $V^\pi(s_0)$.

As value function is induced by any policy, value function $V^{\pi^*}(s)$ under optimal policy π^* can also be considered. By convention, it is denoted as $V^*(s)$ and is called an **optimal value function**.

Obtaining optimal value function $V^*(s)$ doesn't provide enough information to reconstruct some optimal policy π^* due to unknown world dynamics, i. e. transition probabilities. In other words, being blind to what state s may be the environment's response on certain action in given state makes knowing optimal value function unhelpful. This intuition suggests to introduce a similar notion comprising more information:

Definition 7. For given MDP and policy π the **quality function (Q-function) under policy π** is defined as

$$Q^\pi(s, a) := \mathbb{E}_{\mathcal{T}_\pi | s_0=s, a_0=a} \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

It directly follows from the definitions that these two functions are deeply interconnected:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma V^\pi(s')] \quad (3)$$

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) \quad (4)$$

The notion of **optimal Q-function** $Q^*(s, a)$ can be introduced analogically. But, unlike value function, obtaining $Q^*(s, a)$ actually means solving a reinforcement learning task: indeed,

Proposition 3. If $Q^*(s, a)$ is a quality function under some optimal policy, then

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

is an optimal policy.

This result implies that instead of searching for optimal policy π^* , an agent can search for optimal Q-function and derive the policy from it.

Proposition 4. For any MDP existence of optimal policy leads to existence of deterministic optimal policy.

2.5. Classes of algorithms

Reinforcement learning algorithms are presented in a form of computational procedure specifying a strategy of collecting interaction experience and obtaining policy with as higher $J(\pi)$ as possible. They rarely include a stopping criterion like in classic optimization methods as the stochasticity of given setting prevents the verification of optimality; usually number of iterations is determined by the amount of computational resources.

All reinforcement learning algorithms can be roughly divided into four⁴ classes:

- **meta-heuristics**: this class of algorithms treats the task as black-box optimization with zeroth-order oracle. They usually generate a set of policies $\pi_1 \dots \pi_P$ and launch several episodes of interaction for each to determine best and worst policies according to average return. After that they try to construct more optimal policies using evolutionary or advanced random search techniques.
- **policy gradient**: these algorithms directly optimize (2), trying to obtain π^* and no additional information about MDP, using approximate estimations of gradient with respect to policy parameters. They consider RL task as optimization with stochastic first-order method and make use of interaction structure to lower the variance of gradient estimations. They will be discussed in sec. 5.
- **value-based** algorithms construct optimal policy implicitly by gaining an approximation of optimal Q-function $Q^*(s, a)$. This framework will be discussed in sec. 3 and 4.
- **model-based** algorithms exploit learned or given world dynamics, i. e. distributions $p(s' | s, a)$ from T . The class of algorithms to work with when the model is explicitly provided is represented by such algorithms as Monte-Carlo Tree Search; if not, it is possible to imitate the world dynamics by learning the outputs of black box from interaction experience.

2.6. Measurements of performance

Achieved performance (**score**) from the point of average cumulative reward is not the only one measure of RL algorithm quality. When speaking of real-life robots, the required number of simulated episodes is always the biggest concern. It is usually measured in terms of interaction steps (where step is one transition performed by environment) and is referred to as **sample efficiency**.

When the simulation is more or less cheap, RL algorithms can be viewed as a special kind of optimization procedures. In this case, the final performance of the found policy is opposed to required computational resources, measured by **wall-clock time**. In most cases RL algorithms can be expected to find better policy after more iterations, but the amount of these iterations tend to be unjustified.

The ratio between amount of interactions and required wall-clock time for one update of policy varies significantly for different algorithms. It is well-known that model-based algorithms tend to have the greatest sample-efficiency at the cost of expensive update iterations, while evolutionary algorithms require excessive amounts of interactions while providing massive resources for parallelization and reduction of wall-clock time. Value-based and policy gradient algorithms, which will be the focus of further discussion, are known to lie somewhere in between.

⁴in many sources evolutionary algorithms are bypassed in discussion as they do not utilize the structure of RL task in any way.

3. Value-based algorithms

3.1. Temporal Difference learning

In this section we will consider temporal difference (also named TD(0)) learning algorithm, which is a classical Reinforcement Learning method in the base of modern value-based approach in DRL.

The first idea behind this algorithm is to search for optimal Q-function $Q^*(s, a)$ by solving a system of recursive equations which can be derived by recalling interconnection between Q-function and value function (3):

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma V^\pi(s')] = \\ &= \{\text{using (4)}\} = \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma \mathbb{E}_{a' \sim \pi(a'|s')} Q^\pi(s', a')] \end{aligned}$$

This equation, named **Bellman equation**, remains true for value functions under any policies including optimal policy π^* :

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma \mathbb{E}_{a' \sim \pi^*(a'|s')} Q^*(s', a')] \quad (5)$$

Recalling proposition 3, optimal (deterministic) policy $\pi^*(s)$ has a form $\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a)$. Substituting this for $\pi^*(s)$ in (5), we obtain fundamental **Bellman optimality equation**:

Proposition 5. (Bellman optimality equation)

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma \max_{a'} Q^*(s', a')] \quad (6)$$

The straightforward utilization of this result is as follows. Consider **tabular case**, in which both state space \mathcal{S} and action space \mathcal{A} are finite (and small enough to be listed in computer memory). Let us also assume for now that transition probabilities are available to training procedure. Then $Q^*(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ can be represented as a finite table with $|\mathcal{S}||\mathcal{A}|$ numbers. In this case (6) just gives a set of $|\mathcal{S}||\mathcal{A}|$ equations for this table to satisfy.

Addressing the values of the table as unknown variables, this system of equations can be solved using basic **point iteration** method: let $Q_0^*(s, a)$ be initial arbitrary values of table (with the only exception that for terminal states $s \in \mathcal{S}^+$, if any, $Q(s, a) = 0$ for all actions a). On each iteration t the table is updated by substituting current values of the table to the right side of equation until the process converges:

$$Q_{t+1}^*(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma \max_{a'} Q_t^*(s', a')] \quad (7)$$

This straightforward approach of learning the optimal Q-function was named **Q-learning** and was extensively studied in classical Reinforcement Learning. One of the central results is presented in the following convergence theorem:

Proposition 6. Let by \mathcal{B} denote an operator $(\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}) \rightarrow (\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R})$, updating Q_t^* as in (7):

$$Q_{t+1}^* = \mathcal{B}Q_t^*$$

for all state-action pairs s, a .

Then \mathcal{B} is a **contraction mapping**, i. e. for any two tables $Q_1, Q_2 \in (\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R})$

$$\|\mathcal{B}Q_1 - \mathcal{B}Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

Therefore, there is a unique fixed point of system of equations (7) and the point iteration method converges to it.

The contraction mapping property is actually of high importance. It demonstrates that the point iteration algorithm converges with exponential speed and will require small amount of iterations. The trick is that each iteration demands full pass across all state-action pairs and exact computation of expectation over transition probabilities.

In general case, this expectation can't be explicitly computed. Instead, agent is restricted to samples from transition probabilities during some interaction experience. **Temporal Differences (TD)**⁵ algorithm proposes to collect this data using $\pi_t = \underset{a}{\operatorname{argmax}} Q_t^*(s, a) \approx \pi^*$ and after each gathered transition $(s_t, a_t, r_{t+1}, s_{t+1})$ update only one cell of the table:

$$Q_{t+1}^*(s, a) = \begin{cases} (1 - \alpha_t)Q_t^*(s, a) + \alpha_t \left[r_{t+1} + \gamma \max_{a'} Q_t^*(s_{t+1}, a') \right] & \text{if } s = s_t, a = a_t \\ Q_t^*(s, a) & \text{else} \end{cases} \quad (8)$$

where $\alpha_t \in (0, 1)$ plays the role of exponential smoothing parameter for estimating expectation $\mathbb{E}_{s' \sim p(s'|s_t, a_t)}(\cdot)$ from samples.

Two key ideas are introduced in the update formula (8): exponential smoothing instead of exact expectation computation and cell by cell updates instead of updating full table at once. Both are required to settle Q-learning algorithm for online application.

As the set \mathcal{S}^+ of terminal states in online setting is usually unknown, a slight modification of update (8) is used. If observed next state s' turns out to be terminal (recall the convention to denote this by flag `done`), its value function is known to be equal to zero:

$$V^*(s') = \max_{a'} Q^*(s', a') = 0$$

This knowledge is embedded in the update rule (8) by multiplying $\max_{a'} Q_t^*(s_{t+1}, a')$ on $(1 - \text{done}_{t+1})$. For the sake of shortness, this factor is often omitted but always present in implementations.

Second important note about formula (8) is that it can be rewritten in the following equivalent way:

$$Q_{t+1}^*(s, a) = \begin{cases} Q_t^*(s, a) + \alpha_t \left[r_{t+1} + \gamma \max_{a'} Q_t^*(s_{t+1}, a') - Q_t^*(s, a) \right] & \text{if } s = s_t, a = a_t \\ Q_t^*(s, a) & \text{else} \end{cases} \quad (9)$$

The expression in the brackets, referred to as **temporal difference**, represents a difference between state-action Q-value estimation $Q_t^*(s, a)$ and its one-step approximation $r_{t+1} + \gamma \max_{a'} Q_t^*(s_{t+1}, a')$. We will use this alternative representation for future narration.

They allow us to formulate first practical algorithm which can work in the tabular case with unknown world dynamics:

Algorithm 1: Temporal Differences algorithm

Hyperparameters: $\alpha_t \in (0, 1)$

Initialize $Q_*(s, a)$ arbitrary

On each interaction step:

1. select $a = \underset{a}{\operatorname{argmax}} Q^*(s, a)$
2. observe transition $(s, a, s', r', \text{done})$
3. update table:

$$Q^*(s, a) \leftarrow Q^*(s, a) + \alpha_t \left[r' + (1 - \text{done}) \gamma \max_{a'} Q^*(s', a') - Q^*(s, a) \right]$$

It turns out, that under several assumptions on state visitation during interaction process this procedure holds similar properties in terms of convergence guarantees, which are stated by the following theorem:

⁵also known as TD(0) due to theoretical generalizations

Proposition 7. *Let's define*

$$e_t(s, a) = \begin{cases} \alpha_t & (s, a) \text{ is updated on step } t \\ 0 & \text{otherwise} \end{cases}$$

Then, if for every state-action pair (s, a)

$$\sum_t^{+\infty} e_t(s, a) = \infty \quad \sum_t^{+\infty} e_t(s, a)^2 < \infty$$

the algorithm 1 converges to optimal Q^ .*

This theorem states that basic policy iteration method can be actually applied online in the way proposed by TD algorithm, but demands «enough exploration» from the strategy of interacting with MDP during training. Satisfying this demand remains a unique and common problem of reinforcement learning.

The widespread kludge is ε -**greedy strategy** which basically suggests to choose random action instead of $a = \underset{a}{\operatorname{argmax}} Q^*(s, a)$ with probability ε_t . The probability ε_t is usually set close to 1 during first interaction iterations and scheduled to decrease to a constant close to 0. This heuristic makes agent visit all states with non-zero probabilities independent of what current approximation $Q^*(s, a)$ suggests.

The main practical issue with Temporal Differences algorithm is that it requires table $Q^*(s, a)$ to be explicitly stored in memory, which is impossible for MDP with high state space complexity. This limitation substantially restricted its applicability until its combination with deep neural network was proposed.

3.2. Deep Q-learning (DQN)

Utilization of neural nets to model either a policy or a Q-function frees from constructing task-specific features and opens possibilities of applying RL algorithms to complex tasks, e. g. tasks with images as input. Video games like Atari 57 are classical example of such tasks where raw pixels of screen are provided as state representation and, correspondingly, as input to either policy or Q-function.

Main idea of Deep Q-learning [9] is to adapt Temporal Difference algorithm so that update formula (9) would be equivalent to gradient descent step for training a neural network to solve a certain regression task. Indeed, it can be noticed that the exponential smoothing parameter α_t resembles learning rate of first-order gradient optimization procedures, while the exploration conditions from theorem 7 look identical to restrictions on learning rate of stochastic gradient descent.

The key hint is that (9) is a gradient descent step in the parameter space of the table functions family:

$$Q(s, a, \theta) = \theta^{s, a}$$

where all $\theta^{s, a}$ form a vector of parameters $\theta \in \mathbb{R}^{|S| \times |A|}$. Note that derivative of $Q(s, a, \theta)$ by θ for given input s, a is one-hot encoding:

$$\frac{\partial Q^*(s, a, \theta)}{\partial \theta} = e^{s, a}, \quad \text{where} \quad e_{i, j}^{s, a} := \begin{cases} 1 & (i, j) = (s, a) \\ 0 & (i, j) \neq (s, a) \end{cases} \quad (10)$$

To unravel this fact, it is convenient to introduce some notation from regression tasks. First, let's denote by y the **target** of our regression task, i. e. the quantity that our model is trying to predict:

$$y(s, a) := r(s') + \gamma \max_{a'} Q^*(s', a', \theta) \quad (11)$$

where s' is a sample from $p(s' | s, a)$ and s, a is input data. In this notation (9) is equivalent to:

$$\theta_{t+1} = \theta_t + \alpha_t [y(s, a) - Q^*(s, a, \theta_t)] e^{s, a}$$

Note that we multiplied scalar value $\alpha_t [y(s, a) - Q^*(s, a, \theta_t)]$ on matrix $e^{s, a}$ to formulate an update of only one component of θ in vector form. By this we transitioned to update in parameter

space using $Q^*(s, a) = \theta^{s,a}$. The statement now is that this formula is a gradient descent update for regression with MSE loss function, input s, a and target $y(s, a)$:

$$\text{Loss}(y(s, a), Q^*(s, a, \theta_t)) = (Q^*(s, a, \theta_t) - y(s, a))^2 \quad (12)$$

Indeed:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha_t [y(s, a) - Q^*(s, a, \theta_t)] e^{s,a} = \\ \{12\} &= \theta_t - \alpha_t \frac{\partial \text{Loss}(y, Q^*(s, a, \theta_t))}{\partial Q} e^{s,a} \\ \{10\} &= \theta_t - \alpha_t \frac{\partial \text{Loss}(y, Q^*(s, a, \theta_t))}{\partial Q^*} \frac{\partial Q^*(s, a, \theta_t)}{\partial \theta} = \\ \{\text{chain rule}\} &= \theta_t - \alpha_t \frac{\partial \text{Loss}(y, Q^*(s, a, \theta_t))}{\partial \theta} \end{aligned}$$

The obtained result is evidently a gradient descent step formula to minimize MSE loss function with target (11):

$$\theta_{t+1} = \theta_t - \alpha_t \frac{\partial \text{Loss}(y, Q^*(s, a, \theta_t))}{\partial \theta} \quad (13)$$

It is important that dependence of y from θ is ignored during gradient computation (otherwise the chain rule application with y being dependent on θ is incorrect). On each step of temporal differences algorithm new target y is constructed using current Q-function approximation, and a new regression task with this target is set. For this fixed target one MSE optimization step is done according to (13), and on the next step a new regression task is defined. Though during each step the target is considered to represent some ground truth like it is in supervised learning, here it provides a direction of optimization and because of this reason is sometimes called a **guess**.

Notice that representation (13) is equivalent to standard TD update (9) with all theoretical results remaining until the parametric family Q_θ is table functions family. At the same time, (13) can be formally applied to any parametric function family including neural networks. Yet, this transition is not rigorous and all theoretical guarantees provided by theorem 7 are lost at this moment.

We will assume that optimal Q-function is approximated with neural network $Q_\theta^*(s, a)$ with parameters θ . Note that for discrete action space case this network may take only s as input and output $|\mathcal{A}|$ numbers representing $Q_\theta^*(s, a_1) \dots Q_\theta^*(s, a_{|\mathcal{A}|})$, which allows to find optimal action in given state s with one forward pass through the net. This allows for given transition $(s, a, r', s', \text{done})$ to compute target y with one forward pass and perform optimization step in one more forward⁶ and one backward pass.

Small issue with this straightforward approach is that it is impractical to train neural networks with batches of size 1. In [9] it is proposed to use **experience replay** to store all collected transitions $(s, a, r', s', \text{done})$ as data samples and on each iteration sample a batch of standard for neural network training size. As usual, loss function is assumed to be an average of losses for each transition from the batch. This utilization of previously experienced transitions is legit because TD algorithm is known to be an **off-policy** algorithm, which means it can work with arbitrary transitions gathered by any agent's interaction experience.

Though empirical results of described algorithm turned out to be promising, the behaviour of Q_θ^* values indicated on the instability of learning process. Reconstruction of target after each optimization step led to so-called **compound error** when approximation error propagated from the close-to-terminal states to the starting in avalanche manner and could lead to guess being 10^6 times bigger than true Q^* value. To address this problem, [9] introduced a kludge known as **target network**, which basic idea is to solve fixed regression problem for $K > 1$ steps, i. e. recompute target every K -th step instead of each.

To avoid target recomputation for the whole experience replay, the copy of neural network Q_θ^* is stored, called the target network. Its architecture is the same while weights θ^- are a copy of Q_θ^* from the moment of last target recomputation⁷ and its main purpose is to generate targets y for given current batch.

⁶in implementations it is possible to combine s and s' in one batch and perform these two forward passes at once.

⁷alternative, but more computationally expensive option, is to update target network weights on each step using exponential smoothing

Combining all things together and adding ε -greedy strategy to facilitate exploration, we obtain classic DQN algorithm:

Algorithm 2: Deep Q-learning (DQN)

Hyperparameters: N — batch size, K — target network update frequency, $\varepsilon(t) \in (0, 1]$ — greedy exploration parameter, Q_θ^* — neural network, SGD optimizer.

Initialize weights of θ arbitrary

Initialize $\theta^- \leftarrow \theta$

On each interaction step:

1. select a randomly with probability $\varepsilon(t)$, else $a = \underset{a}{\operatorname{argmax}} Q_\theta^*(s, a)$
2. observe transition $(s, a, s', r', \text{done})$
3. add observed transition to experience replay
4. sample batch of size N from experience replay
5. for each transition T from the batch compute target:

$$y(T) = r(s') + \gamma \max_{a'} Q^*(s', a', \theta^-)$$

6. compute loss:

$$\text{Loss} = \frac{1}{N} \sum_T (Q^*(s, a, \theta) - y(T))^2$$

7. make a step of gradient descent using $\frac{\partial \text{Loss}}{\partial \theta}$
8. if $t \bmod K = 0$: $\theta^- \leftarrow \theta$

3.3. Double DQN

Although target network successfully prevented Q_θ^* from unbounded growth and empirically stabilized learning process, the values of Q_θ^* on many domains were evident to tend to overestimation. The problem is presumed to reside in max operation in target construction formula (11):

$$y = r(s') + \gamma \max_{a'} Q^*(s', a', \theta^-)$$

During this estimation \max shifts Q-value estimation towards actions that led to high reward due to luck and actions with overestimating approximation error.

The solution proposed in [15] is based on idea of separating **action selection** and **action evaluation** to carry out each of these operations using peculiar approximation of Q^* :

$$\begin{aligned} \max_{a'} Q^*(s', a', \theta^-) &= Q^*(s', \underset{a'}{\operatorname{argmax}} Q^*(s', a', \theta^-), \theta^-) \approx \\ &\approx Q^*(s', \underset{a'}{\operatorname{argmax}} Q^*(s', a', \theta_1^-), \theta_2^-) \end{aligned}$$

The simplest, but expensive, implementation of this idea is to run two independent DQN («twin DQN») algorithms and use twin network to evaluate actions:

$$y_1 = r(s') + \gamma Q_1^*(s', \underset{a'}{\operatorname{argmax}} Q_2^*(s', a', \theta_2^-), \theta_1^-)$$

$$y_2 = r(s') + \gamma Q_2^*(s', \underset{a'}{\operatorname{argmax}} Q_1^*(s', a', \theta_1^-), \theta_2^-)$$

Intuitively, each Q-function here may prefer lucky or overestimated actions, but the other Q-function will judge them according to its own luck and approximation error, which may be as underestimating

as overestimating. Ideally these two DQNs should not share interaction experience to achieve that, which makes such algorithm twice as expensive both in terms of computational cost and sample efficiency.

Double DQN is more compromised option which suggests to use current weights of network θ for action selection and target network weights θ^- for action evaluation, assuming that when target network update frequency K is big enough these two networks are sufficiently different:

$$y = r(s') + \gamma Q^*(s', \underset{a'}{\operatorname{argmax}} Q^*(s', a', \theta), \theta^-)$$

3.4. Dueling DQN

Another issue with DQN algorithm 2 emerges when a huge part of considered MDP consists of states of low optimal value $V^*(s)$, which is an often case. The problem is that when the agent visits unpromising state instead of lowering its value $V^*(s)$ it remembers only low pay-off for performing some action a in it by updating $Q^*(s, a)$. This leads to continuous returns to this state during future interactions until all actions proof to be unpromising and all $Q^*(s, a)$ are updated. The problem gets worse when the cardinality of action space is high or there are many similar actions in action space.

One benefit of deep reinforcement learning is that we are able to facilitate generalization across actions by specifying the architecture of neural network. To do so, we need to encourage learning of $V^*(s)$ from updates of $Q^*(s, a)$. The idea of **dueling architecture** [17] is to incorporate approximation of $V^*(s)$ explicitly in computational graph. For that we will need the definition of advantage function:

Definition 8. For given MDP and policy π the **advantage function under policy π** is defined as

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

Advantage function is evidently interconnected with Q-function and value function and actually shows the relative advantage of selecting action a comparing to other actions. If $A^\pi(s, a) > 0$, then modifying π to select a more often will lead to better policy as from state s its average return will become bigger than initial $V^\pi(s)$. This follows from the following property of arbitrary advantage function:

$$\begin{aligned} \mathbb{E}_{a \sim \pi(a|s)} A^\pi(s, a) &= \mathbb{E}_{a \sim \pi(a|s)} [Q^\pi(s, a) - V^\pi(s)] = \\ &= \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) - V^\pi(s) = \\ &= V^\pi(s) - V^\pi(s) = 0 \end{aligned} \tag{14}$$

Definition of optimal advantage function $A^*(s, a)$ is analogous and allows us to reformulate $Q^*(s, a)$ in terms of $V^*(s)$ and $A^*(s, a)$:

$$Q^*(s, a) = V^*(s) + A^*(s, a) \tag{15}$$

Straightforward utilization of this result is following: after several feature extracting layers the network is joined with two heads, one outputting single scalar $V^*(s)$ and one outputting $|\mathcal{A}|$ numbers $A^*(s, a)$ like it was in DQN with Q-function. After that scalar value estimation is added to all components of $A^*(s, a)$ to obtain $Q^*(s, a)$. The problem with this naive approach is that due to (14) advantage function can not be arbitrary and must hold (14) for $Q^*(s, a)$ to be identifiable.

This restriction (14) on advantage function can be simplified for the case of optimal policy induced by optimal Q-function:

$$\begin{aligned} 0 &= \mathbb{E}_{a \sim \pi^*(a|s)} Q^*(s, a) - V^*(s) = Q^*(s, \underset{a}{\operatorname{argmax}} Q^*(s, a)) - V^*(s) = \\ &= \max_a Q^*(s, a) - V^*(s) = \max_a [Q^*(s, a) - V^*(s)] = \max_a A^*(s, a) \end{aligned}$$

This condition can be easily satisfied in computation graph by subtracting $\max_a A^*(s, a)$ from advantage head. This will be equivalent to the following formula of dueling DQN:

$$Q^*(s, a) = V^*(s) + A^*(s, a) - \max_a A^*(s, a) \tag{16}$$

The interesting nuance of this improvement is that after evaluation on Atari-57 authors discovered that substituting max operation in (16) with averaging across actions led to better results (while usage of unidentifiable formula (15) led to poor performance). Although gradients can be backpropagated through both operation and formula (16) seems theoretically justified, in practical implementations averaging instead of maximum is widespread.

3.5. Noisy DQN

By default, DQN algorithm does not concern the exploration problem and is always augmented with ε -greedy strategy to force agent discover new states. This baseline exploration strategy suffers from being extremely hyperparameter-sensitive as early decrease of $\varepsilon(t)$ to close to zero values may lead to sticking in local optima, unable to explore new options due to imperfect Q^* , while high values of $\varepsilon(t)$ forces agent to behave randomly for excessive amount of episodes, which slows down learning. In other words, ε -greedy strategy transfers responsibility to solve **exploration-exploitation** trade-off on engineer.

The key reason why ε -greedy exploration strategy is relatively primitive is that exploration priority does not depend on current state. Intuitively, the choice whether to exploit knowledge by selecting approximately optimal action or explore MDP by selecting some other depends on how explored the current state s is. If a new part of state space is reached after any amount of interactions probably indicates that random actions are good to try, and initial state will probably be sufficiently explored after several first episodes.

For ε -greedy strategy agent selects action using deterministic $Q^*(s, a, \theta)$ and only afterwards injects state-independent noise in a form of $\varepsilon(t)$ probability to choose random action. **Noisy networks** [3] were proposed as simple extension of DQN to provide state-dependent and parameter-free exploration by injecting noise of learnable volume to all (or most) nodes in computational graph.

Let a linear layer with m inputs and n outputs in q-network perform the following computation:

$$y(x) = Wx + b$$

where $x \in \mathbb{R}^m$ is input, $W \in \mathbb{R}^{n \times m}$ — weights matrix, $b \in \mathbb{R}^n$ — bias. In noisy layers it is proposed to substitute deterministic parameters with samples from $\mathcal{N}(\mu, \sigma)$ where μ, σ are trained with gradient descent⁸. On the forward pass through the noisy layer we sample $\varepsilon_W \sim \mathcal{N}(0, I_{nm \times nm})$, $\varepsilon_b \sim \mathcal{N}(0, I_{n \times n})$ and then compute

$$\begin{aligned} W &= (\mu_W + \sigma_W \odot \varepsilon_W) \\ b &= (\mu_b + \sigma_b \odot \varepsilon_b) \\ y(x) &= Wx + b \end{aligned}$$

where \odot denotes element-wise multiplication, $\mu_W, \sigma_W \in \mathbb{R}^{n \times m}$, $\mu_b, \sigma_b \in \mathbb{R}^n$ — trainable parameters of the layer.

As the output of q-network now becomes a random variable, loss value becomes a random variable too. Like in similar models for supervised learning, on each step an expectation of loss function over noise is minimized:

$$\mathbb{E}_\varepsilon \text{Loss}(\theta, \varepsilon) \rightarrow \min_\theta$$

The gradient in this setting can be estimated using Monte-Carlo:

$$\nabla_\theta \mathbb{E}_\varepsilon \text{Loss}(\theta, \varepsilon) = \mathbb{E}_\varepsilon \nabla_\theta \text{Loss}(\theta, \varepsilon) \approx \nabla_\theta \text{Loss}(\theta, \varepsilon) \quad \varepsilon \sim \mathcal{N}(0, I)$$

It can be seen that amount of noise actually inflicting output of network may vary for different inputs, i. e. for different states. There are no guarantees that this amount will reduce as the interaction proceeds. The behaviour of average variance of network with time is reported to be extremely sensitive to initialization of σ_W, σ_b and vary from MDP to MDP.

One technical issue with noisy layers it that on each pass an excessive amount (by the number of network parameters) of noise samples are required. This may substantially reduce computational efficiency of forward pass though the network. For optimization purposes it is proposed to obtain noise for weights matrices in the following way: sample just $n + m$ noise samples $\varepsilon_W^1 \sim \mathcal{N}(0, I_{m \times m})$, $\varepsilon_W^2 \sim \mathcal{N}(0, I_{n \times n})$ and acquire matrix noise in a factorized form:

$$\varepsilon_W = f(\varepsilon_W^1) f(\varepsilon_W^2)^T$$

⁸by using reparametrization trick

where f is a scaling function, e. g. $f(x) = \text{sign}(x) \sqrt{|x|}$. The benefit of this procedure is that it requires $m + n$ samples instead of mn , but sacrifices the interlayer independence of noise.

3.6. Prioritized experience replay

In DQN each batch of transitions is sampled from experience replay using uniform distribution, treating collected data as equally prioritized. In such scheme states for each update come from the same distribution as they come from interaction experience (except that they become decorrelated), which agrees with TD algorithm as the base for DQN.

Intuitively observed transitions vary in their importance. At the beginning of training most guesses tend to be more or less random as they rely on arbitrarily initialized Q_θ^* and the only source of trusted information are transitions with non-zero received reward, especially near terminal states where $V_\theta^*(s')$ is known to be equal to 0. In the midway of training, most of experience replay is filled with memory of interaction within well-learned part of MDP while the most crucial information is contained in transitions where agent explored new promising areas and gained emergent reward yet to be propagated through Bellman equation. All these significant transitions are drowned in collected data and rarely appear in sampled batches.

Central idea of prioritized experience replay [11] is that priority of transition $T = (s, a, r', s', \text{done})$ is proportional to temporal difference:

$$\rho(T) := y(T) - Q^*(s, a, \theta) = \sqrt{\text{Loss}(y(T), Q^*(s, a, \theta))} \quad (17)$$

Using these priorities as indication of transition importances, sampling from experience replay proceeds using following probabilities:

$$\mathcal{P}(T) \propto \rho(T)^\alpha$$

where hyperparameter $\alpha \in \mathbb{R}^+$ controls the degree to which the priorities sparsify data weights. The case $\alpha = 0$ corresponds to uniform sampling distribution while $\alpha = +\infty$ is equivalent to greedy sampling of transitions with highest priority.

The problem with (17) claim is that each transition's priority changes after each network update. As it is impractical to recalculate loss for the whole data after each step, some simplifications must be put up with. The straightforward option is to update priority only for sampled transitions in the current batch. New transitions can be added to experience replay with highest priority, i. e. $\max_T \rho(T)$ ⁹.

Second debatable issue of prioritized replay is that it actually substitutes loss function of DQN updates, which assumed uniform sampling of visited states to ensure they come from state visitation frequency distribution:

$$\mathbb{E}_{T \sim \text{Uniform}} \text{Loss}(T) \rightarrow \min_{\theta}$$

While it is not clear what distribution is better to sample from to ensure exploration restrictions of theorem 7, prioritized experienced replay changes this distribution in uncontrollable way. Despite its fruitfulness at the beginning and midway of training process, this distribution shift may destabilize learning close to the end and make algorithm stuck with locally optimal policy. As formally this issue is of estimating an expectation over one probability with preference to sample from another one, the standard technique of **importance sampling** can be used as countermeasure:

$$\mathbb{E}_{T \sim \text{Uniform}} \text{Loss}(T) = \sum_{i=0}^B \frac{1}{B} \text{Loss}(T_i) = \sum_{i=0}^B \mathcal{P}(T_i) \frac{1}{B\mathcal{P}(T_i)} \text{Loss}(T_i) = \mathbb{E}_{T \sim \mathcal{P}(T)} \frac{1}{B\mathcal{P}(T)} \text{Loss}(T)$$

where B is a number of transitions stored in experience replay. Importance sampling implies that we can avoid distribution shift that introduces undesired bias by making smaller gradient updates for significant transitions that appear in the batches with higher frequency. The price for bias elimination is that importance sampling weights lower prioritization effect by slowing down learning of selected new information.

This duality resembles trade-off between bias and variance, but important moment here is that distribution shift does not cause any seeming issues at the beginning of training when agent behaves close to random and do not produce valid state visitation distribution anyway. The idea proposed

⁹which can be computed online with $\mathcal{O}(1)$ complexity

in [11] based on this intuition is to anneal the importance sampling weights so they correct bias properly only towards the end of training procedure.

$$\text{Loss}^{\text{prioritizedER}} = \mathbb{E}_{T \sim \mathcal{P}(T)} \left(\frac{1}{B\mathcal{P}(T)} \right)^{\beta(t)} \text{Loss}(T)$$

where $\beta(t) \in [0, 1]$ and approaches 1¹⁰ as more interaction steps are executed. If $\beta(t)$ is set to 0, no bias correction is held, while $\beta(t) = 1$ corresponds to unbiased loss function, i. e. equivalent to sampling from uniform distribution.

The most significant and obvious drawback of prioritized experience replay approach is that it introduces additional hyperparameters. Although α represents one number, algorithm's behaviour may turn out to be sensitive to its choosing, and $\beta(t)$ must be designed by engineer as some scheduled motion from something near 0 to 1, and its well-turned selection may require inaccessible knowledge about how many steps it will take for algorithm to «warm up».

3.7. Multi-step DQN

One more widespread modification of Q-learning in RL community is substituting one-step approximation present in Bellman equation (6) with N -step:

Proposition 8. (N -step Bellman Equation)

$$Q^*(s_0, a_0) = \mathbb{E}_{\mathcal{T}_{\pi^*} | s_0, a_0} \left[\sum_{t=1}^N \gamma^{t-1} r(s_t) + \gamma^N \max_{a_N} Q^*(s_N, a_N) \right] \quad (18)$$

Indeed, definition of $Q^*(s, a)$ consists of average return and can be viewed as making T^{\max} steps from state s_0 after selecting action a_0 , while vanilla Bellman equation represents $Q^*(s, a)$ as reward from one next step in the environment and estimation of the rest of trajectory reward recursively. N -step Bellman equation (18) generalizes these two opposites.

All the same reasoning as for DQN can be applied to N -step Bellman equation to obtain N -step DQN algorithm, which only modification appears in target computation:

$$y(s_0, a_0) = \sum_{t=1}^N \gamma^{t-1} r(s_t) + \gamma^N \max_{a_N} Q^*(s_N, a_N, \theta) \quad (19)$$

To perform this computation, we are required to obtain for given state s and a not only one next step, but N steps. To do so, instead of transitions N -step roll-outs are stored, which can be done by precomputing following tuples:

$$T = \left(s, a, \sum_{n=1}^N \gamma^{n-1} r^{(n)}, s^{(N)}, \text{done} \right)$$

where $r^{(n)}$ is the reward received in n steps after visitation of considered state s , $s^{(N)}$ is state visited in N steps, and done is a flag whether the episode ended during N -step roll-out¹¹. All other aspects of algorithm remain the same in practical implementations, and the case $N = 1$ corresponds to standard DQN.

The goal of using $N > 1$ is to accelerate propagation of reward from terminal states backwards through visited states to s_0 as less update steps will be required to take into account freshly observed reward and optimize behaviour at the beginning of episodes. The price is that formula (19) includes an important trick: to calculate such target, we are required to sample $s' \sim p(s' | s, a)$ and for second (and following) step action a' must be sampled from π^* for Bellman equation (18) to remain true. In other words, application of N -step Q-learning is theoretically improper when behaviour policy differs from π^* . Note that we do not face this problem in case $N = 1$ as we are required to sample only from transition probability $p(s' | s, a)$ for given state-action pair s, a .

¹⁰often it is initialized by a constant close to 0 and is linearly increased until it reaches 1

¹¹all N -step roll-outs must be considered including those terminated at k -th step for $k < N$.

Even considering $\pi^* \approx \underset{a}{\operatorname{argmax}} Q^*(s, a, \theta)$, where Q^* is our current approximation of π^* , makes N -step DQN an **on-policy** algorithm when for every state-action pair s, a it is preferable to sample target using the closest approximation of π^* available. This questions usage of experience replay or at the very least encourages to limit its capacity to store only B^{\max} newest transitions.

To see the negative effect of N -step DQN, consider the following toy example. Suppose agent makes a mistake on the second step after s_0 and ends episode with negative reward. Then in the case of $N > 2$ each time the roll-out starting with this s_0 is sampled in the batch, the value of $Q^*(s, a, \theta)$ will be updated with this received negative reward even if $Q^*(s_1, \cdot, \theta)$ already learned not to repeat this mistake again.

Yet empirical results in many domains demonstrate that raising N from 1 to 2-3 may result in substantial acceleration of training and positively affect the final performance. On the contrary, the theoretical groundlessness of this approach explains its negative effects when N is set too big.

4. Distributional approach for value-based methods

4.1. Theoretical foundations

The setting of RL task inherently carries internal stochasticity of which agent has no substantial control. Sometimes intelligent behaviour implies taking risks with substantial chance of low episode return. All this information resides in distribution of return R (1) as random variable.

While value-based methods aim at learning expectation of this random variable as the quantity we actually care about, in distributional approach [1] it is proposed to learn the whole distribution of returns. It further extends the information gathered by algorithm about MDP towards model-based case in which the whole MDP is imitated by learning both reward function $r(s)$ and transitions \mathbb{T} .

In this section we discuss some theoretical extensions of temporal differences ideas in the case when expectations on both sides of Bellman equation (5) and Bellman optimality equation (6) are taken away.

The central object of study in Q-learning was Q-function, which for given state and action returned an expectation of reward. To rewrite Bellman equation not in terms of expectations, but in terms of the whole distributions, we require a corresponding notation.

Definition 9. For given MDP and policy π the **value distribution of policy** π is a random variable defined as

$$Z^\pi(s, a) := \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a$$

Note that Z^π just represent a random variable which is taken expectation of in definition of Q-function:

$$Q^\pi(s, a) = \mathbb{E}_{\mathcal{T}_\pi} Z^\pi(s, a)$$

Using definition of value distribution, Bellman equation can be rewritten to extend the recursive connection between adjacent states from expectations of returns to the whole distributions of returns:

Proposition 9. (Distributional Bellman Equation)

$$Z^\pi(s, a) \stackrel{\text{c.d.f.}}{=} r(s') + \gamma Z^\pi(s', a') \mid s' \sim p(s' \mid s, a), a' \sim \pi(a' \mid s') \quad (20)$$

Here we used some auxiliary notation: by $\stackrel{\text{c.d.f.}}{=}$ we mean that cumulative distribution functions of given two random variables are equal almost everywhere. Such equations are called **recursive distributional equations** and are well-known in theoretical probability theory¹². By using \mid we describe a sampling procedure for the random variable to the right side of equation: for given s, a next state s' is sampled from transition probability, then a' is sampled from given policy, then random variable $Z^\pi(s', a')$ is sampled to calculate a resulting sample $r(s') + \gamma Z^\pi(s', a')$.

Note that while $Q^\pi(s, a) \in \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, the space of value distributions is a space of mappings from state-action pair to continuous distributions:

$$Z^\pi(s, a) \in \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R})$$

It is important to notice that even in table-case when state and action spaces are finite, the space of Q-functions is finite while the space of value distributions is essentially infinite. Crucial moment for us will be that convergence properties now depend on chosen metric¹³.

The choice of metric in $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R})$ represents the same issue as in the space of continuous random variables $\mathcal{P}(\mathbb{R})$: if we choose a metric in the latter, we can construct one in the former:

¹²to get familiar with this notion, consider this basic example:

$$X_1 \stackrel{\text{c.d.f.}}{=} \frac{X_2}{\sqrt{2}} + \frac{X_3}{\sqrt{2}}$$

where X_1, X_2, X_3 are random variables coming from $\mathcal{N}(0, \sigma^2)$.

¹³in finite spaces it is true that convergence in one metric guarantees convergence to the same point for any other metric.

Proposition 10. If $d(X, Y)$ is a metric in the space $\mathcal{P}(\mathbb{R})$, then

$$\bar{d}(Z_1, Z_2) := \sup_{s \in \mathcal{S}, a \in \mathcal{A}} d(Z_1(s, a), Z_2(s, a))$$

is a metric in the space $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R})$.

The particularly interesting for us example of metric in $\mathcal{P}(\mathbb{R})$ will be Wasserstein metric, which concerns only random variables with bounded moments, so we will additionally assume that for all state-action pairs s, a

$$\mathbb{E} Z^\pi(s, a)^p \leq +\infty$$

are finite for $p \geq 1$.

Proposition 11. For $1 \leq p \leq +\infty$ for two random variables X, Y on continuous domain with p -th bounded moments and cumulative distribution functions F_X and F_Y correspondingly a **Wasserstein distance**

$$W_p(X, Y) := \left(\int_0^1 |F_X^{-1}(\omega) - F_Y^{-1}(\omega)|^p d\omega \right)^{\frac{1}{p}}$$

$$W_\infty(X, Y) := \sup_{\mathbb{R}} |F_X^{-1}(\omega) - F_Y^{-1}(\omega)|$$

is a metric in the space of random variables with p -th bounded moments.

Thus we can conclude from proposition 10 that maximal form of Wasserstein metric

$$\bar{W}_p(Z_1, Z_2) = \sup_{s \in \mathcal{S}, a \in \mathcal{A}} W_p(Z_1(s, a), Z_2(s, a)) \quad (21)$$

is a metric in the space of value distributions.

We now concern convergence properties of point iteration method to solve (20) in order to obtain Z^π for given policy π , i. e. solve the task of **policy evaluation**.

For that purpose we initialize $Z_0^\pi(s, a)$ arbitrarily¹⁴ and perform the following updates for all state-action pairs s, a :

$$Z_{t+1}^\pi(s, a) \stackrel{\text{c.d.f.}}{:=} r(s') + \gamma Z_t^\pi(s', a') \quad (22)$$

Here we assume that we are able to compute the distribution of random variable on the right side knowing Z_t^π , π , all transition probabilities \mathbb{T} and reward function. The question whether the sequence $\{Z_t^\pi\}$ converges to Z^π can be given a detailed answer:

Proposition 12. Denote by \mathcal{B} the following operator $(\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R})) \rightarrow (\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathbb{R}))$, updating Z_t^π as in (22):

$$Z_{t+1}^\pi = \mathcal{B} Z_t^\pi$$

for all state-action pairs s, a .

Then \mathcal{B} is a contraction mapping in \bar{W}_p (21) for $1 \leq p \leq +\infty$, i.e. for any two value distributions Z_1, Z_2

$$\bar{W}_p(Z_1, Z_2) \geq \gamma \bar{W}_p(\mathcal{B} Z_1, \mathcal{B} Z_2)$$

Hence there is a unique fixed point of system of equations (20) and the point iteration method converges to it.

One more curious theoretical result is that \mathcal{B} is in general not a contraction mapping for such distances as Kullback-Leibler divergence, Total Variation distance and Kolmogorov distance¹⁵. It shows

¹⁴here we consider value distributions from theoretical point of view, assuming that we are able to explicitly store a table of $|S||A|$ continuous distributions without any approximations.

¹⁵one more metric for which the contraction property was shown is Cramer metric:

$$l_2(X, Y) = \left(\int_{\mathbb{R}} (F_X(\omega) - F_Y(\omega))^2 d\omega \right)^{\frac{1}{2}}$$

where F_X, F_Y are c.d.f. of random variables X, Y correspondingly.

that metric selection indeed influences convergence rate.

Similar to traditional value functions, we can define **optimal value distribution** $Z^*(s, a)$. Substituting $\pi^* = \underset{a}{\operatorname{argmax}} \mathbb{E}_{\mathcal{T}_{\pi^*}} Z^*(s, a)$ into (4), we obtain distributional Bellman optimality equation:

Proposition 13. (Distributional Bellman optimality equation)

$$Z^*(s, a) \stackrel{\text{c.d.f.}}{=} r(s') + \gamma Z^*(s', \underset{a'}{\operatorname{argmax}} \mathbb{E}_{\mathcal{T}_{\pi^*}} Z^*(s', a')) \mid s' \sim p(s' \mid s, a) \quad (23)$$

Now we concern the same question whether point iteration method of solving (23) will lead to solution Z^* and whether it will be a contraction mapping for some metric. The answer turns out to be negative.

Proposition 14. *Point iteration for solving (23) may diverge.*

Level of impact of this result is not completely clear. Point iteration for (23) preserves means of distributions, i. e. it will eventually converge to $Q^*(s, a)$ with all theoretical guarantees from classical Q-learning. The reason behind divergence theorems hides in the rest of distributions like other moments and situations when equivalent (in terms of average return) actions may lead to different higher moments.

4.2. Categorical DQN

There are obvious obstacles for practical application of distributional Q-learning following from complication of working with arbitrary continuous distributions. Usually we are restricted to approximations inside some family of parametric distributions, so we have to perform a projection step on each iteration.

Second matter in combining distributional Q-learning with deep neural networks is that we lack theoretical results on distributional analog of temporal differences algorithm 9. It must be taken into account that only samples from $p(s' \mid s, a)$ are available for each update.

Categorical DQN [1] (also referred as c51) provides straightforward design of practical distributional algorithm. While DQN was resemblance of temporal differences algorithm, Categorical DQN attempts to follow the same logic as in DQN.

The concept is as following. The neural network with parameters θ in this setting takes as input $s \in \mathcal{S}$ and for each action a outputs parameters $\zeta_\theta(s, a)$ of distributions of random variable $Z_\theta^*(s, a)$. As in DQN, experience replay can be used to collect observed transitions to sample a batch for each update step. For each transition $T = (s, a, r', s', \text{done})$ in the batch a guess is computed:

$$y(T) \stackrel{\text{c.d.f.}}{:=} r' + (1 - \text{done})\gamma Z_\theta^*(s', \underset{a'}{\operatorname{argmax}} \mathbb{E}_{Z_\theta^*} Z_\theta^*(s', a')) \quad (24)$$

Note that expectation of $Z_\theta^*(s', a')$ is computed explicitly using the form of chosen parametric family of distributions and outputted parameters $\zeta_\theta(s', a')$, as is the distribution of random variable $r' + (1 - \text{done})\gamma Z_\theta^*(s', a')$. In other words, in this setting guess $y(T)$ is also a continuous random variable similar to $Z_\theta^*(s, a)$. Because of that, the loss function is reasonable to be designed in a form of some divergence \mathcal{D} between $y(T)$ and $Z_\theta^*(s, a)$:

$$\text{Loss}(\theta) = \mathbb{E}_T \mathcal{D}(y(T) \parallel Z_\theta^*(s, a)) \quad (25)$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \text{Loss}(\theta_t)}{\partial \theta}$$

The particular choice of this divergence must be made with concern that $y(T)$ is a «sample» from a full one-step approximation of Z_θ^* which includes transition probabilities:

$$y^{\text{full}}(s, a) \stackrel{\text{c.d.f.}}{:=} \sum_{s' \in \mathcal{S}} p(s' \mid s, a) y(s, a, r(s'), s', \text{done}(s')) \quad (26)$$

which is precisely the right side of distributional Bellman optimality equation. In other words, if transition probabilities \mathbb{T} were known, the update could be made using y^{full} as target.

$$\text{Loss}^{\text{full}}(\theta) = \mathbb{E}_{s,a} \mathcal{D}(y^{\text{full}}(s, a) \parallel Z_{\theta}^*(s, a))$$

This motivates to choose $\text{KL}(y(T) \parallel Z_{\theta}^*(s, a))$ (specifically with this order of arguments) as \mathcal{D} to exploit the following property (we denote by p_X a p.d.f. of random variable X):

$$\begin{aligned} \nabla_{\theta} \mathbb{E}_T \text{KL}(y^{\text{full}}(s, a) \parallel Z_{\theta}^*(s, a)) &= \nabla_{\theta} \left[\mathbb{E}_T \int_{\mathbb{R}} -p_{y^{\text{full}}(s, a)}(\omega) \log p_{Z_{\theta}^*(s, a)}(\omega) d\omega + \text{const}(\theta) \right] = \\ \{\text{using (26)}\} &= \nabla_{\theta} \mathbb{E}_T \int_{\mathbb{R}} \mathbb{E}_{s' \sim p(s'|s, a)} -p_{y(T)}(\omega) \log p_{Z_{\theta}^*(s, a)}(\omega) d\omega = \\ \{\text{taking expectation out}\} &= \mathbb{E}_{s' \sim p(s'|s, a)} \nabla_{\theta} \mathbb{E}_T \int_{\mathbb{R}} -p_{y(T)}(\omega) \log p_{Z_{\theta}^*(s, a)}(\omega) d\omega = \\ &= \mathbb{E}_{s' \sim p(s'|s, a)} \nabla_{\theta} \mathbb{E}_T \text{KL}(y(T) \parallel Z_{\theta}^*(s, a)) \end{aligned}$$

This property basically states that gradient of loss function (25) with KL as \mathcal{D} is an unbiased (Monte-Carlo) estimation of gradient of KL-divergence for «full» distribution (26), which resembles employment of exponential smoothing in temporal differences learning. For many other divergences, including Wasserstein metric, same statement is not true, so their utilization in described setting will lead to biased gradients and all theory-grounded intuition that algorithm moves in the right direction becomes distinctively lost. Moreover, KL-divergence is known to be one of the easiest divergences to work with due to its nice smoothness properties and wide prevalence in many deep learning pipelines.

Described above motivation to choose KL-divergence as actual objective for minimization is contradictory. Theoretical analysis of distributional Q-learning, specifically theorem 12, though concerning policy evaluation other than optimal Z^* search, explicitly hints that the process converges exponentially fast for Wasserstein metric, while even for precisely made updates in terms of KL-divergence we are not guaranteed to get any closer to true solution.

More «practical» defect of KL-divergence is that it demands two comparable distributions to share the same domain. This means that by choosing KL-divergence we pledge to guarantee that $y(T)$ and $Z_{\theta}^*(s, a)$ in (25) have coinciding support. This emerging restriction seems limiting even beforehand as for episodic MDP value distribution in terminal states is obviously degenerated (their support consists of one point $r(s)$ which is given all probability mass) which means that our value distribution approximation is basically ensured to never be precise.

In Categorical DQN, as follows from the name, the family of distributions is chosen to be categorical on fixed support $\{z_0, z_1 \dots z_{A-1}\}$ where A is number of **atoms**. As no prior information about MDP is given, the basic choice of this support is uniform grid from some $V_{\min} \in \mathbb{R}$ to $V_{\max} \in \mathbb{R}$:

$$z_i = V_{\min} + \frac{i}{A-1}(V_{\max} - V_{\min}), \quad i \in 0, 1, \dots, A-1$$

These bounds, though, must be chosen carefully as their choice implicitly assumes

$$V_{\min} \leq Z^*(s, a) \leq V_{\max}$$

and if these inequalities are not tight, approximation will obviously become poor.

Therefore the neural network outputs A numbers, summing into 1, to represent arbitrary distribution on this support:

$$\zeta_i(s, a, \theta) = \mathcal{P}(Z_{\theta}^*(s, a) = z_i)$$

Withing this family of distributions, computation of expectation, greedy action selection and KL-divergence between two distributions is trivial. One problem hides in target formula (24): while we can compute distribution $y(T)$, its support may in general differs from $\{z_0 \dots z_{A-1}\}$. To avoid the issue of disjoint supports, a **projection step** must be done to find the closest to target distribution within the chosen family¹⁶. Therefore the resulting target used in loss is

$$y(T) \stackrel{\text{c.d.f.}}{:=} \Pi_C \left[r' + (1 - \text{done}) \gamma Z_{\theta}^* \left(s', \underset{a'}{\text{argmax}} \mathbb{E} Z_{\theta}^*(s', a') \right) \right]$$

¹⁶to project categorical distribution on support $\{v_0, v_1 \dots v_{A-1}\}$ on categorical distributions with support $\{z_0, z_1 \dots z_{A-1}\}$ one can just find for each v_i the closest two atoms $z_j \leq v_i \leq z_{j+1}$ and split all probability mass for v_i between z_j and z_{j+1} proportional to closeness. If $v_i < z_0$, then all its probability mass is given to z_0 , same with upper bound.

where Π_C is projection operator.

The resulting practical algorithm, named c51 after categorical distributions with $A = 51$ atoms, inherits ideas of experience replay, ε -greedy exploration and target network from DQN. Empirically, though, usage of target network remains an open question as chosen family of distributions restricts value approximation from unbounded growth by «clipping» predictions at z_{A-1} and z_0 , yet it is still considered improving performance.

Algorithm 3: Categorical DQN (c51)

Hyperparameters: N — batch size, V_{\max}, V_{\min} , A — parameters of support, K — target network update frequency, $\varepsilon(t) \in (0, 1]$ — greedy exploration parameter, ζ^* — neural network, SGD optimizer.

Initialize weights θ of neural net ζ^* arbitrary

Initialize $\theta^- \leftarrow \theta$

Precompute support grid $z_i = V_{\min} + \frac{i}{A-1}(V_{\max} - V_{\min})$

On each interaction step:

1. select a randomly with probability $\varepsilon(t)$, else $a = \underset{a}{\operatorname{argmax}} \sum_i z_i \zeta_i^*(s, a, \theta)$
2. observe transition $(s, a, s', r', \text{done})$
3. add observed transition to experience replay
4. sample batch of size N from experience replay
5. for each transition T from the batch compute target:

$$\mathcal{P}(y(T) = r' + \gamma z_i) = \zeta_i^* \left(s', \underset{a'}{\operatorname{argmax}} \sum_i z_i \zeta_i^*(s', a', \theta^-), \theta^- \right)$$

6. project $y(T)$ on support $\{z_0, z_1 \dots z_{A-1}\}$
7. compute loss:

$$\text{Loss} = \frac{1}{N} \sum_T \text{KL}(y(T) \parallel Z^*(s, a, \theta))$$

8. make a step of gradient descent using $\frac{\partial \text{Loss}}{\partial \theta}$
9. if $t \bmod K = 0$: $\theta^- \leftarrow \theta$

4.3. Quantile Regression DQN (QR-DQN)

Categorical DQN discovered a gap between theory and practice where KL-divergence, used in practice, is theoretically unjustified. Theorem 12 hints that the true divergence we should care about is actually Wasserstein metric, but it remained unclear how it could be optimized using only samples from transition probabilities \mathbb{T} .

In [2] it was discovered that selecting another family of distributions to approximate $Z_\theta^*(s, a)$ will reduce Wasserstein minimization task to the search for quantiles of specific distributions. The latter can be done in online setting using **quantile regression** technique. This led to alternative distributional Q-learning algorithm named Quantile Regression DQN (QR-DQN).

The basic idea is to «swap» fixed support and learned probabilities of Categorical DQN. We will now consider the family with fixed probabilities for A -atomed categorical distribution with arbitrary support $\{\zeta_0^*(s, a, \theta), \zeta_1^*(s, a, \theta), \dots, \zeta_{A-1}^*(s, a, \theta)\}$. Again, we will assume all probabilities equal given the absence of any prior knowledge; namely, our distribution family is now

$$Z_\theta^*(s, a) \sim \text{Uniform}(\zeta_0^*(s, a, \theta), \dots, \zeta_{A-1}^*(s, a, \theta))$$

The only hyperparameter now to provide is number of atoms A . Our neural network now outputs A

arbitrary real numbers that represent the support of uniform categorical distribution.¹⁷

For table-case setting, on each step of point iteration we desire to update the cell for given state-action pair s, a with full distribution of random variable to the right side of (23). If we are limited to store only A atoms of the support, the true distribution must be projected on the space of A -atomed categorical distributions. Consider now this task of projecting some given random variable with c.d.f. $F(\omega)$ in terms of Wasserstein distance. Specifically, we will be interested in minimizing \mathcal{W}_1 -distance for $p = 1$ as theorem 12 states the contraction property for all $1 \leq p \leq +\infty$ and we are free to choose any:

$$\int_0^1 \left| F^{-1}(\omega) - U_{z_0, z_1 \dots z_{A-1}}^{-1}(\omega) \right| d\omega \rightarrow \min_{z_0, z_1 \dots z_{A-1}} \quad (27)$$

where $U_{z_0, z_1 \dots z_{A-1}}$ is c.d.f. for uniform categorical distribution on given support. Its inverse, also known as **quantile function**, has a following simple form:

$$U_{z_0, z_1 \dots z_{A-1}}^{-1}(\omega) = \begin{cases} z_0 & 0 \leq \omega < \frac{1}{A} \\ z_1 & \frac{1}{A} \leq \omega < \frac{2}{A} \\ \vdots & \\ z_{A-1} & \frac{A-1}{A} \leq \omega < 1 \end{cases}$$

Substituting this into (27):

$$\sum_{i=0}^{A-1} \int_{\frac{i}{A}}^{\frac{i+1}{A}} \left| F^{-1}(\omega) - z_i \right| d\omega \rightarrow \min_{z_0, z_1 \dots z_{A-1}}$$

Notice that obtained optimization task separates to A independent tasks that can be solved independently.

$$\int_{\frac{i}{A}}^{\frac{i+1}{A}} \left| F^{-1}(\omega) - z_i \right| d\omega \rightarrow \min_{z_i} \quad (28)$$

Proposition 15. *Let's denote*

$$\tau_i := \frac{\frac{i}{A} + \frac{i+1}{A}}{2}$$

Then every solution for (28) satisfies $F(z_i) = \tau_i$, i. e. it is τ_i -th quantile of projected random variable.

The result 15 states that we require only A specific quantiles of random variable to the right side of Bellman equation¹⁸. Hence the last thing to do to design a practical algorithm is to develop a procedure of unbiased estimation of quantiles for the random variable on the right side of distribution Bellman optimality equation (23).

Quantile regression is the standard technique to estimate the quantiles of **empirical distribution** (i. e. distribution that is represented by finite amount of i. i. d. samples from it). Recall from machine learning that the constant solution optimizing l1-loss is median, i. e. $\frac{1}{2}$ -th quantile. This fact can be generalized to arbitrary quantiles:

Proposition 16. (Quantile Regression) *Let's define loss as*

$$\text{Loss}(a, X) = \begin{cases} \tau(a - X) & a \geq X \\ (1 - \tau)(X - a) & a < X \end{cases}$$

Then solution for

$$\mathbb{E}_X \text{Loss}(a, X) \rightarrow \min_{a \in \mathbb{R}} \quad (29)$$

¹⁷Note that target distribution is now guaranteed to remain within this distribution family as multiplying on γ just shrinks the support and adding r' just shifts it. We assume that if some atoms of the support coincide, the distribution is still A -atomed categorical; for example, for degenerated distribution (like in case of terminal states) $\zeta_0^*(s, a, \theta) = \zeta_1^*(s, a, \theta) = \dots = \zeta_{A-1}^*(s, a, \theta)$. This shows that projection step heuristic is not needed for this particular choice of distribution family.

¹⁸It can be proved that for table-case policy evaluation algorithm which stores in each cell not expectations of reward (as in Q-learning) but A quantiles updated according to distributional Bellman equation (20) using theorem 15 converges to quantiles of $Z^*(s, a)$ in Wasserstein metric for $1 \leq p \leq +\infty$ and its update operator is a contraction mapping in \mathcal{W}_∞ .

is τ -th quantile of distribution of X .

As usually in case of neural networks, it is impractical to optimize (29) until convergence on each iteration for each of A desired quantiles τ_i . Instead just one step of gradient optimization is made and outputs of neural network $\zeta_i^*(s, a, \theta)$, which play the role of a in formula (29) are moved towards the quantile estimation via backpropagation. In other words, (29) sets a loss function for network outputs; the losses for different quantiles are summed up. The resulting loss is

$$\text{Loss}^{\text{QR}}(s, a, \theta) = \sum_{i=0}^{A-1} \mathbb{E}_{s' \sim p(s'|s, a)} \mathbb{E}_{y \sim y(T)} (\tau - \mathbb{I}[\zeta_i^*(s, a, \theta) < y]) (\zeta_i^*(s, a, \theta) - y) \quad (30)$$

where \mathbb{I} denotes an indicator function. The expectation over $y \sim y(T)$ for given transition can be computed in closed form: indeed, $y(T)$ is also a A -atomed categorical distribution with support $\{r' + \gamma \zeta_0^*(s', a'), \dots, r' + \gamma \zeta_{A-1}^*(s', a')\}$, where

$$a' = \underset{a'}{\operatorname{argmax}} \mathbb{E} Z^*(s', a', \theta) = \underset{a'}{\operatorname{argmax}} \frac{1}{A} \sum_i \zeta_i^*(s', a', \theta)$$

and expectation over transition probabilities, as always, is estimated using Monte-Carlo by sampling transitions from experience replay.

Algorithm 4: Quantile Regression DQN (QR-DQN)

Hyperparameters: N — batch size, A — number of atoms, K — target network update frequency, $\varepsilon(t) \in (0, 1]$ — greedy exploration parameter, ζ^* — neural network, SGD optimizer.

Initialize weights θ of neural net ζ^* arbitrary

Initialize $\theta^- \leftarrow \theta$

Precompute mid-quantiles $\tau_i = \frac{i + \frac{1}{A}}{2}$

On each interaction step:

1. select a randomly with probability $\varepsilon(t)$, else $a = \underset{a}{\operatorname{argmax}} \frac{1}{A} \sum_i \zeta_i^*(s, a, \theta)$
2. observe transition $(s, a, s', r', \text{done})$
3. add observed transition to experience replay
4. sample batch of size N from experience replay
5. for each transition T from the batch compute the support of target distribution:

$$y(T)_j = r' + \gamma \zeta_j^* \left(s', \underset{a'}{\operatorname{argmax}} \frac{1}{A} \sum_i \zeta_i^*(s', a', \theta^-), \theta^- \right)$$

6. compute loss:

$$\text{Loss} = \frac{1}{NA} \sum_T \sum_i \sum_j (\tau_i - \mathbb{I}[\zeta_i^*(s, a, \theta) < y(T)_j]) (\zeta_i^*(s, a, \theta) - y(T)_j)$$

7. make a step of gradient descent using $\frac{\partial \text{Loss}}{\partial \theta}$
8. if $t \bmod K = 0$: $\theta^- \leftarrow \theta$

4.4. Rainbow DQN

Success of Deep Q-learning encouraged an full-scale research of value-based deep reinforcement learning by studying various drawbacks of DQN and developing auxiliary extensions. In many articles some extensions from previous research were already considered and embedded in empirically compared algorithms.

In Rainbow DQN [6], seven Q-learning-based ideas are united in one procedure with ablation studies held whether all incorporated extensions are essentially necessary for resulted RL algorithm:

- DQN (sec. 3.2)
- Double DQN (sec. 3.3)
- Dueling DQN (sec. 3.4)
- Noisy DQN (sec. 3.5)
- Prioritized Experience Replay (sec. 3.6)
- Multi-step DQN (sec. 3.7)
- Categorical DQN (sec. 4.2)

There is little ambiguity on how these ideas can be combined; we will discuss several non-straightforward circumstances and provide the full algorithm description after.

To apply prioritized experience replay in distributional setting, the measure of transition importance must be provided. The main idea is inherited from ordinary DQN where priority is just loss for this transition:

$$\rho(T) := \text{Loss}(y(T), Z^*(s, a, \theta)) = \text{KL}(y(T) \parallel Z^*(s, a, \theta))$$

To combine noisy networks with double DQN heuristic, it is proposed to resample noise on each forward pass through the network or through its copy for target computation. This decision implies that action selection, action evaluation and network utilization must be stochastic (for exploration cultivation) and decorrelated.

The one snagging combination here is categorical DQN and dueling DQN. To merge these ideas, we need to model advantage $A^*(s, a, \theta)$ in distributional setting. In Rainbow this is done straightforwardly: the network has two heads, value stream $v(s, \theta)$ outputting A real values and advantage stream $a(s, a, \theta)$ outputting $A \times |\mathcal{A}|$ real values. Then these streams are integrated using the same formula (16) with the only exception being softmax applied across atoms dimension to guarantee that output is categorical distribution:

$$\zeta_i^*(s, a, \theta) \propto \exp \left(v(s, \theta)_i + a(s, a, \theta)_i - \frac{1}{|\mathcal{A}|} \sum_a a(s, a, \theta)_i \right)$$

Combining lack of intuition behind this integration formula with usage of mean instead of theoretically justified max makes this element of Rainbow the most questionable.

Algorithm 5: Rainbow DQN

Hyperparameters: M — batch size, V_{\max}, V_{\min}, A — parameters of support, K — target network update frequency, N — multi-step size, α — degree of prioritized experience replay, $\beta(t)$ — importance sampling bias correction for prioritized experience replay, ζ^* — neural network, SGD optimizer.

Initialize weights θ of neural net ζ^* arbitrary

Initialize $\theta^- \leftarrow \theta$

Precompute support grid $z_i = V_{\min} + \frac{i}{A-1}(V_{\max} - V_{\min})$

On each interaction step:

1. select $a = \underset{a}{\operatorname{argmax}} \sum_i z_i \zeta_i^*(s, a, \theta, \varepsilon), \varepsilon \sim \mathcal{N}(0, I)$
2. observe transition $(s, a, r', s', \text{done})$
3. construct N -step transition $T = (s, a, \sum_{n=0}^N \gamma^n r^{(n+1)}, s^{(N)}, \text{done})$ and add it to experience replay with weight $\max_T \rho(T)$
4. sample batch of size M from experience replay using probabilities $\mathcal{P}(T) \propto \rho^\alpha$
5. compute weights for the batch

$$w(T) = \left(\frac{1}{B\mathcal{P}(T)} \right)^{\beta(t)}$$

6. for each transition T from the batch compute target:

$$\mathcal{P}(y(T) = r' + \gamma^N z_i) = \zeta_i^* \left(s', \underset{a'}{\operatorname{argmax}} \sum_i z_i \zeta_i^*(s', a', \theta, \varepsilon_1), \theta^-, \varepsilon_2 \right), \varepsilon_1, \varepsilon_2 \sim \mathcal{N}(0, I)$$

7. project $y(T)$ on support $\{z_0, z_1 \dots z_{A-1}\}$
8. update transition priorities

$$\rho(T) \leftarrow \text{KL}(y(T) \parallel Z^*(s, a, \theta, \varepsilon)), \varepsilon \sim \mathcal{N}(0, I)$$

9. compute loss:

$$\text{Loss} = \frac{1}{M} \sum_T w(T) \rho(T)$$

10. make a step of gradient descent using $\frac{\partial \text{Loss}}{\partial \theta}$

11. if $t \bmod K = 0$: $\theta^- \leftarrow \theta$

5. Policy Gradient algorithms

...WORK IN PROGRESS...
...UNDER CONSTRUCTION...

5.1. REINFORCE

The policy may be searched in some parametric distribution family $\mathcal{P} = \{p_\theta(x) \mid \theta \in \Theta\}$. This changes the form of optimization task (2) to the following:

$$R(\pi(\theta)) \rightarrow \max_{\theta} \quad (31)$$

REINFORCE is a straightforward application of gradient optimization to the given task.

Proposition 17. *If $\pi(\theta)$ is differentiable, then:*

$$\nabla_{\theta} R(\pi(\theta)) = \mathbb{E}_{\mathcal{T}_{\pi}} \sum_{t=1} \left[\nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \sum_{t'=t} \gamma^{t-t'} r(s_{t'}) \right]$$

The expectation over trajectories can be estimated using Monte-Carlo estimation, i. e. playing one game and replacing infinite integral with one (or several) sampled trajectory. This estimation is unbiased, which allows reasonable usage of stochastic gradient descent or more advanced stochastic optimization techniques.

Algorithm 6: REINFORCE

Play one game $a_0, s_1, a_1 \dots$ with current policy π_{θ} . Update parameters using unbiased estimation of gradients:

$$\nabla_{\theta} R(\pi(\theta)) \approx \sum_{t=1} \left[\nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \sum_{t'=t} \gamma^{t-t'} r(s_{t'}) \right]$$

5.2. Advantage Actor-Critic (A2C)

Theoretically REINFORCE (sec. 5.1) can be applied straightforwardly in the case when $\pi_{\theta}(a \mid s)$ is represented as neural network. Moreover, theory states that stochastic optimization using unbiased estimations of gradients will converge to locally optimal solution. Yet the time required for convergence and the problem of sticking in local optimums make this naive approach completely impractical.

The main source of problems is believed to be *high variation* of gradient estimation in REINFORCE. The basic technique of variance reduction is an introduction of baseline.

Proposition 18. *If $\pi(\theta)$ is differentiable, then for any arbitrary function $b(s): S \rightarrow \mathbb{R}$, called baseline:*

$$\nabla_{\theta} R(\pi(\theta)) = \mathbb{E}_{\mathcal{T}_{\pi}} \sum_{t=1} \left[\nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \left(\sum_{t'=t} \gamma^{t-t'} r(s_{t'}) - b(s_t) \right) \right]$$

To reduce variance, the baseline must be chosen so that absolute values of expression inside the expectation is as lower as possible and gives zero on average. This leads to thought that a good choice of baseline is

$$b(s) = \mathbb{E}_{\mathcal{T}_{\pi} \mid s_t=s} \sum_{t'=t} \gamma^{t-t'} r(s_{t'})$$

The definition of value function being recalled, it can be recognized in the expression above:

$$b(s_t) = V^{\pi}(s_t)$$

Using the properties of expectation and recalling the definition of Q-function in the same manner, the gradient can now be rewritten as follows:

Proposition 19.

$$\nabla_{\theta} R(\pi(\theta)) = \mathbb{E}_{\mathcal{T}_{\pi}} \sum_{t=1} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (Q^{\pi}(s_t, a_t) - V^{\pi}(s_t))]$$

The expression in the brackets $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is often referred to as *advantage function under policy π* $A^{\pi}(s_t, a_t)$.

Lower variance of this reformulated gradient instigates estimation of A^{π} for current policy π . If A^{π} was known, utilization of proposition 19 would lead to better convergence.

It turns out that obtaining value function V^{π} is enough as knowing value function yields unbiased approximation of advantage¹⁹:

$$A^{\pi}(s_t, a_t) \approx r(s_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

This estimation would have been unbiased if the value function V^{π} was precise. Instead the only way to estimate value function is using another approximation. For these purposes a second neural network is introduced, $V_{\phi}^{\pi}(s)$.

The training procedure for this second neural network is built using similar reasoning as in Deep Q-learning: on each optimization step neural network makes a step towards value estimation:

$$y = r(s_{t+1}) + \gamma V_{\phi}^{\pi}(s_{t+1})$$

$$(y - V_{\phi}^{\pi}(s_t))^2 \rightarrow \min_{\phi}$$

Combining all together, classic Actor-Critic algorithm is obtained:

Algorithm 7: Advantage Actor-Critic

Play several steps using current policy $\pi_{\theta}(a | s)$ (*actor*).

For each transition (s, a, r, s') calculate advantage estimation using *critic* V_{ϕ}^{π}

$$A^{\pi}(s, a) \approx r + \gamma V_{\phi}^{\pi}(s') - V_{\phi}^{\pi}(s)$$

Update actor using gradient estimation:

$$\nabla_{\theta} R(\pi(\theta)) \approx \sum_{(s, a, r, s')} \nabla_{\theta} \log \pi_{\theta}(a | s) A^{\pi}(s, a)$$

For each transition (s, a, r, s') calculate target for critic update:

$$y(r, s') = r + \gamma V_{\phi}^{\pi}(s')$$

Update critic using loss function:

$$\text{Loss} = \sum_{(s, a, r, s')} (y(r, s') - V_{\phi}^{\pi}(s))^2 \rightarrow \min_{\phi}$$

¹⁹that follows from definitions

5.3. Generalized Advantage Estimation (GAE)

5.4. Natural Policy Gradient (NPG)

5.5. Trust-Region Policy Optimization (TRPO)

5.6. Proximal Policy Optimization (PPO)

6. Overview of other approaches

7. Experiments

That's one is a problem.

8. Conclusion



References

- [1] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.
- [2] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [3] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.
- [4] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [5] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [6] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [7] A. Irpan. Deep reinforcement learning doesn't work yet. *Online (Feb. 14): <https://www.alexirpan.com/2018/02/14/rl-hard.html>*, 2018.
- [8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [10] OpenAI. Openai five. <https://blog.openai.com/openai-five/>, 2018.
- [11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

- [12] J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.
- [13] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [14] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [15] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [16] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>, 2019.
- [17] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.