1 | Preliminaries

This chapter starts introducing what a Markov Decision Process is and how it is used to formally define a sequential decision problem. Next it describes what a policy is and its associated utility functions used to evaluate its performance.

1.1. Markov Decision Process

In a Sequential Decision problem an agent interacts with an environment through actions and receives a feedback signal from it. The environment is usually modeled as a Markov Decision Process.

Definition 1. *A Finite Markov Decision Process (MDP) is a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, \mu)$ *specified by:*

- *a finite set* $\mathcal{S}$ *of possible states*
- *a finite set* $\mathcal{A}$ *of actions*
- *a transition probability function* $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$
- *a reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- *a discount factor* $\gamma \in (0, 1)$
- *a probability distribution over the initial states* $\mu : \mathcal{S} \to [0, 1]$

The agent interacts with it at discrete time steps $t = 0, 1, 2, 3, \ldots$. At each time step the agent receives some representation of the environment's *state*, $S_t \in \mathcal{S}$, and on that basis selects an *action*, $A_t \in \mathcal{A}(s)$. One time step later, the agent receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

The transition probability depends only on the previous state and action thanks to the Markov assumption:

$$p(s'|s, a) \doteq \mathbb{P}(S_t = s'|S_{t-1} = s, A_{t-1} = a)$$

The reward function is the expected reward the state action pair:

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a]$$

The agent choses actions according to a policy. A policy π is a distribution over actions given the state:

$$\pi(a|s) = \mathbb{P}(a|s)$$

Therefore the policy can be stochastic or deterministic and because it considers only the previous state it is also Markovian. The goal of the agent is to learn a policy π that maximizes the expected return, where the return is defined as the cumulative discounted reward:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Given a policy π, it is possible to define the utility of each state through a value function.

Definition 2. *The state value function $v^\pi : \mathcal{S} \to \mathbb{R}$ of an MDP is the expected return starting from state s and then following policy π*

$$v^\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$$

$\mathbb{E}_\pi[.]$ denotes the expected value of a random variable given that the agent follows policy π. For control purposes, rather than the value of each state we consider the value of each action in each state.

Definition 3. *The action-value function $q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of an MDP is the expected return starting from state s, taking action a and then following policy π*

$$q^\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

A fundamental property of the value functions is that they satisfy the following recursive relationships, called *Bellman expectation equations*:

$$v^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v^\pi(s') \right) \tag{1.1}$$

$$q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q^\pi(s', a') \tag{1.2}$$

Definition 4. *The Markov Reward Process induced by a policy π on a MDP is the tuple $(\mathcal{S}, p^\pi, r^\pi)$ specified by:*

- *the same set $\mathcal{S}$ from the MDP*
- *the transition probability function $p^\pi : \mathcal{S} \times \mathcal{S} \to [0, 1]$ obtained from following policy π*
- *the reward function $r : \mathcal{S} \to \mathbb{R}$ obtained fom following policy π*

$$p^\pi(s'|s) = \mathbb{P}_\pi(S_t = s'|S_{t-1} = s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s)p(s'|s,a)$$
$$r^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s)r(s,a)$$

The bellman expectation equation for v^π can be written in matrix form using the induced MRP and a closed form solution can be computed:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi$$
$$V^\pi = (I - P^\pi)^{-1} R^\pi$$

where $V^\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the matrix equivalent of v^π.

Definition 5. *The Bellman operator for V^π is defined as $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$*

$$(T^\pi v^\pi)(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^\pi(s') \right)$$

Using the Bellman operator for $v\pi$, the Bellman expectation equation can be compactly written as:

$$T^\pi V^\pi = V^\pi$$

Definition 6. *The Bellman operator for q^π is defined as $T^\pi : \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$*

$$(T^\pi q^\pi)(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q^\pi(s,a)$$

Using the Bellman operator for q^π, the Bellman expectation equations can be compactly written as:

$$T^\pi Q^\pi = Q^\pi$$

Definition 7. *The optimal state-value function $v^*(s)$ is the maximum state-value function over all policies*

$$v^*(s) = \max_\pi v^\pi(s)$$

Definition 8. *The optimal action-value function $q^*(s)$ is the maximum action-value function over all policies*

$$q^*(s,a) = \max_\pi q^\pi(s,a)$$

Also the optimal value functions satisfy a recursive relationship, called *Bellman Optimality Equations*

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}(s)} q^*(s,a) \\ &= \max_{a \in \mathcal{A}(s)} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^*(s) \right) \end{aligned} \tag{1.3}$$

$$\begin{aligned} q^*(s,a) &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^*(s') \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}(s)} q^*(s',a') \end{aligned} \tag{1.4}$$

Value functions define a partial ordering over policies

$$\pi > \pi' \text{ if } v^{\pi}(s) > v^{\pi'}(s), \quad \forall s \in \mathcal{S}$$

Theorem 1.1. *For any Markov Decision Process*

1. *There exists an optimal policy $\pi*$ that is better than or equal to all other policies $\pi^* \geq \pi, \quad \forall \pi$*
2. *All optimal policies achieve the optimal value function $v^{\pi^*}(s) = v*(s)$*
3. *All optimal policies achieve the optimal state-value function $q^{\pi^*}(s,a) = q^*(s,a)$*
4. *There is always a deterministic optimal policy for an MDP*

A determinisitc optimal policy can be found by maximizing over $q^*(s,a)$ (greedy policy)

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Definition 9. *The Bellman optimality operator for v^* is defined as $T^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$*

$$(T^* v^*)(s) = \max_{a \in \mathcal{A}(s)} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^*(s') \right)$$

Definition 10. *The Bellman optimality operator for q^* is defined as $T^* : \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|} \to \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$*

$$(T^* q^*)(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}(s')} q^*(s',a')$$

The Bellman operators exhibit the following properties:

- **Monotonicity**: if $f_1 \leq f_2$ component-wise

$$T^{\pi} f_1 \leq T^{\pi} f_2 \quad , \quad T^* f_1 \leq T^* f_2$$

- **Max-Norm Contraction**: for two vectors f_1 and f_2

$$\|T^\pi f_1 - T^\pi f_2\|_\infty \leq \gamma \|f_1 - f_2\|_\infty$$
$$\|T^* f_1 - T^* f_2\|_\infty \leq \gamma \|f_1 - f_2\|_\infty$$

- v^π is the **unique fixed point** of T^π
- v^* is the **unique fixed point** of T^*
- For any vector $f \in \mathbb{R}^{|\mathcal{S}|}$ and any policy π, we have

$$\lim_{k\to\infty} (T^\pi)^k f = v^\pi \quad , \quad \lim_{k\to\infty} (T^*)^k f = v^*$$

Differently from (1.1), the Bellman optimality equation is non linear due to the presence of a max operation and no closed form solution exists for the general case.
The bellman optimality equation of an MDP is solved by different iterative methods like Dynamic programming, Linear programming and Reinforcement Learning.

1.2. Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov Decision Process. Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their computational expense.

1.2.1. Policy Evaluation (Prediction)

In Dynamic Programming, policy evaluation is done iteratively using the bellman equation for $v^\pi(s)$ (1.1) as an update rule. This update rule is called Bellman Expectation Backup and applying it to each state is called a *sweep*.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k^{S_{t+1}} | S_t = s] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s)[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v_k(s')] \end{aligned}$$

Algorithm 1.1 Iterative Policy Evaluation

Input: A Markov Decision Process (MDP) defined by (S, A, P, R, γ) and a policy π
Output: State-value function V for policy π
Initialize $V(s) \leftarrow 0$ for all $s \in S$
Initialize $\theta \leftarrow$ a small positive number (convergence threshold)
repeat
  $\Delta \leftarrow 0$
  for each $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s)[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s', r|s, a)V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
return V

1.2.2. Policy Improvement

Considering a deterministic policy π we can improve it by acting greedily:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} q^{\pi}(s, a)$$

Theorem 1.2. *Let π and π' be any pair of deterministic policies such that*

$$q^{\pi}(s, \pi'(s)) \geq v^{\pi}(s) \quad , \quad \forall s \in \mathcal{S}$$

Then the policy π' must be as good as, or better than π

$$v^{\pi'}(s) \geq v^{\pi}(s) \quad , \quad \forall s \in \mathcal{S} \tag{1.5}$$

If the improvements stops ($v^{\pi'}(s) = v^{\pi}(s) \, \forall s \in \mathcal{S}$) then the Bellman Optimality Equation (1.3) holds and $\pi' = \pi$ is an optimal policy π^*.

The policy iteration algorithm 1.2 alternates between policy evaluation and policy improvements until the policy is stable and thus optimal.

Algorithm 1.2 Policy Iteration

Input: A Markov Decision Process (MDP) defined by $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and a policy π
Output: Optimal policy π^* and Optimal state-value function $V^*(s)$
Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
Initialize $\theta \leftarrow$ a small positive number (convergence threshold)
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s)[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s', r|s, a)V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$
policy-stable $\leftarrow$ *true*
for each $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)}[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) * V(s)]$
    if *old-action* $\neq \pi(s)$ **then**
        policy-stable $\leftarrow$ *false*
    end if
end for
if *policy-stable* **then**
    return $V \approx v^*$ and $\pi \approx \pi^*$
else
    go to 5
end if

A small modification to the policy iteration algorithms lead to the Value iteration algorithm 1.3 in which the evaluation is stopped after only one sweep and followed by the policy improvement. It basically implements the bellman optimality equation (1.3) as an update rule:

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_k(s) \right)$$

Once $v_k \approx v^*$ we can build a greedy policy π^*

Algorithm 1.3 Value Iteration

Initialize $V(s) = 0$ for all states $s \in S$
repeat
  $\Delta \leftarrow 0$
  for $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Output: V and greedy policy $\pi(s) = \arg\max_{a \in \mathcal{A}(s)} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')\right)$

1.3. Reinforcement Learning

In section 1.2 a perfect model of the environment was used in order to compute the backup updates. Model-Free Reinforcement Learning aims to solve the prediction and control problem only through experience borrowing ideas from Dynamic Programming.

1.3.1. Temporal-Difference Learning

Temporal Difference Learning (TD) is a **bootstrapping** method: it learns directly from transitions of incomplete episodes using previous estimates.

Prediction

For the prediction problem TD updates its estimate of $v^\pi(s_t)$ only using the next step reward r_{t+1} and its current estimate of $v^\pi(s_{t+1})$.

$$V(S_t) \leftarrow V(S_{t+1}) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{1.6}$$

$R_{t+1} + \gamma V(S_{t+1})$ is called TD *target* and the quantity $\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called TD *error*.

Control

For the Control problem TD algorithms can be divided into two categories:

1. On-Policy Learning
2. Off-Policy Learning

In On Policy Learning the algorithmy directly improves the policy π used to collect the samples from the environment while Off Policy Learning allows to improve a policy π' different from the one that interacts with the environment.
One major problem in Model-Free reinforcement learning is the *Exploration-Exploitation* tradeoff: Should the agent try out new actions to discover their potential rewards or

exploit its current knowledge in order to maximize the expected reward?

Definition 11. *A soft policy π is a policy with $\pi(a|s) > 0$, $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}(s)$.*

Definition 12. *An ϵ-soft policy π is a policy with $\pi(s,a) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$, $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}(s)$ and some $\epsilon > 0$.*

Definition 13. *An ϵ-greedy policy π is a policy with:*

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

The ϵ-greedy policy is a specific example of ϵ-soft policy that is the closest to a greedy policy. It is the simplest idea for ensuring continual exploration while acting greedily.

Theorem 1.3. *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q^π is an improvement.*

$$\begin{aligned} q^\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q^\pi(s,a) \\ &= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} q^\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q^\pi(s,a) \\ &\geq \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} q^\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1-\epsilon} q^\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q^\pi(s,a) = v^\pi(s) \end{aligned}$$

with $m = |\mathcal{A}|$.
Therefore from the policy improvement theorem 1.2, $v^{\pi'}(s) \geq v^\pi(s)$.

Sarsa: On-policy TD Control

Applying the same TD method (1.6) used to estimate v^π to the action-value function q^π leads to the following update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

.

This rule uses every element of a quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state-action pair to the next. This quintuple gives rise to thename *Sarsa* for the algorihtm.

It is straightforward to design an on-policy control algorithm based on the Sarsa prediction method. As in all on-policy methods, we continually estimate q^π for the behavior policy π, and at the same time change π towards greediness with respect to q^π.

Algorithm 1.4 Sarsa

Initialize $Q(s, a)$ arbitrarily
loop
  Initialize S
  Choose A from S using policy derived from Q (e.g. ϵ-greedy)
  repeat
    Take action A, observe R,S'
    Choose A' from S' using policy derived from Q (e.g. ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal
end loop

Q-Learning: Off-policy TD Control

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as *Q-Learning* [1].

In this algorithm the learned action-value function Q directly approximates q^*, the optimal action-value function, independent of the policy being followed:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Algorithm 1.5 Q-Learning

Initialize $Q(s, a)$ arbitrarily
loop
  Initialize S
  Choose A from S using policy derived from Q (e.g. ϵ-greedy)
  repeat
    Take action A, observe R,S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a \in \mathcal{A}} Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until S is terminal
end loop

Convergence of Sarsa and Q-Learning

1.3.2. Reinforcement Learning Taxonomy

This subsection includes a brief overview of the whole Reinforcement Learning taxonomy. 1.3.1 presented two well known TD tabular algorithms but often real world scenarios have to be modeled with complex state and action spaces with more dimensions resulting in intractable problems. Moreover these spaces could be continuous and tabular methods cannot be applied anymore.

Function Approximation

When the state space is too big for tabular reinforcement learning approximate solution methods are used instead. Typically the value function (or action-value function) is parameterized with a weight vector $\mathbf{w} \in \mathbb{R}^d$ and $d \ll |\mathcal{S}|$:

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

For example $\hat{v}$ might be a linear function in features of the state, with $\mathbf{w}$ the vector of feature weights. More generally, $\hat{v}$ might be the function computed by a multi-layer artificial neural network or even the function computed by a decision tree.

In addition, function approximation allow continous state spaces to be managed without the need for discretization techniques.

Value-Based Algorithms

All the algorithms that seek to learn the action values and then select actions based on their estimated values are called Value-Based Algorithms. The temporal difference algorithms presented in 1.3.1 are all value-based algorithms.

Policy-Based Algorithms

Differently from Value-Based Algorithms Policy-Based algorithms seek to learn a *parameterized policy* that can select actions without consulting a value function:

$$\pi(a|s, \boldsymbol{\theta}) = \mathbb{P}(A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$$

One important consequence of the use of parameterized policies is that they offer practical ways of dealing with large action spaces, even continuous spaces with an inifinite number of actions. Instead of computing the learned probability for each of themany actions, we instead learn statistics of the probability distribution. For example, the action set might be the real numbers, with actions chosen from a normal distribution:

$$\pi(a|s, \boldsymbol{\theta}) = \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right)$$

Actor-Critic Algorithms

A combination of value-based methods and policy-based methods gives rise to Actor-Critic algorithms. Actor critic algorithms learn a parameterized policy (Actor) and a value function (Critic). The actor is updated in the direction that maximizes the expected reward using gradients derived from the critic's feedback. The critic estimates the actions taken by the actor.

1.3.3. Deep Reinforcement Learning

In recent years Deep Learning techniques applied to Reinforcement Learning techniques have shown to yield very good results. The first notable example came from Mnih et al. [**?**] modifying Q-Learning to work with Deep Neural Networks. At that time, their approach achieved state of the art on multiple Atari Games with no prior knowledge and same hyperparameters for each game. This algorithm, called *Deep Q-Networks* (DQN) uses an *experience replay buffer* to store played transitions for sampling minibatches. These minibatches are used to train the Q-Network using Stochastic Gradient Descent with backpropagation on the neural network.

Unfortunately nonlinear function approximators like neural networks come with a theoretical cost: convergence to an optimum is only guaranteed locally. Despite this limitation, DQN and other deep reinforcement learning approaches remain the most widely used methods in the field due to their empirical success and ability to handle complex, high-dimensional state spaces.

Below, a state of the art algorithm called Soft Actor Critic (SAC) [**? ?**] is presented.

Soft Actor Critic

Bibliography

[] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

[] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL `http://arxiv.org/abs/1812.05905`.

[] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

[1] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989. URL `http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf`.