10-708: Probabilistic Graphical Models, Spring 2020

Lecture 20: Reinforcement Learning and Control Through Inference in GM

Lecturer: Eric P. Xing *Scribe: Ankur Deka, Zuxin Liu, Yash Belhe, Tejus Gupta, Rohit Jena*

1 Control as Inference Recap

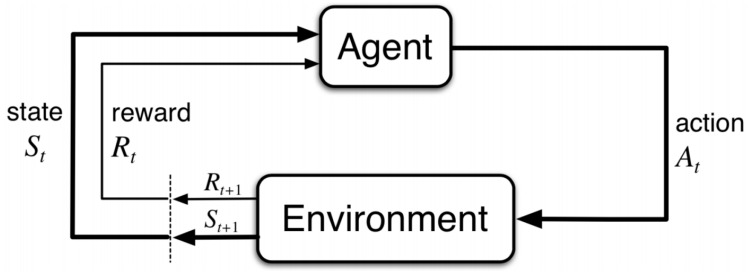

Figure 1: Reinforcement learning framework

Reinforcement learning is usually modeled as a Markov Decision Process(MDP), just as Fig. 1 shows. A typical MDP has 4 major components:

- The initial state distribution: $s_0 \sim p_0(s)$
- Transition probability: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
- Policy: $a_t \sim \pi(a_t|s_t)$
- Reward: $r_t = r(s_t, a_t)$

To represent the MDP with graphical model, we introduce an auxiliary variable $\mathcal{O}$ to define the distribution over optimal trajectories. The graphical model representation is shown in Fig. 2. We have:

- The initial state distribution: $s_0 \sim p_0(s)$
- Transition probability: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
- Policy: $a_t \sim \pi(a_t|s_t)$
- Reward: $r_t = r(s_t, a_t)$
- Optimality: $p(\mathcal{O}_t = 1|s_t, a_t) = \exp(r(s_t, a_t))$

The introduced auxiliary variable $\mathcal{O}$ allows us to model sub-optimal behavior and can be used to solve inverse reinforcement learning problem. The graphical model representation of RL provides us an alternative to solve control and planning problems via inference algorithms.

In the classical RL setup, we have the following equations for value function and Q-function:

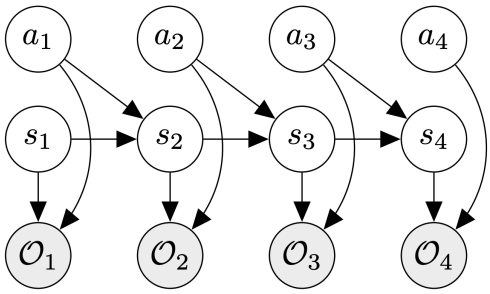

Figure 2: Graphical model representation of RL

$$V_\pi(s) := \mathbb{E}_\pi \sum_{k=0}^{T} \left[\gamma^k r_{t+k+1} \mid s_t = s\right] \tag{1}$$

$$Q_\pi(s,a) := \mathbb{E}_\pi \sum_{k=0}^{T} \left[\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{2}$$

where γ is the discount factor, which represents the importance of rewards in future states.

The optimal value function (Bellman optimality) are:

$$V_*(s) := \max_\pi V_\pi(s) = \max_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_*(s')] \tag{3}$$

$$Q_*(s,a) := \max_\pi Q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \max_{a'} Q_*(s',a')] \tag{4}$$

The above Bellman update procedure can also be illustrated by Fig. 3.

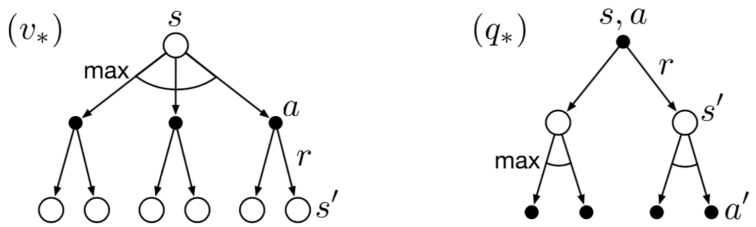

Figure 3: Backup diagrams for v_* and q_*

Given the optimal value function $Q_*(s,a)$, the optimal policy can be represented by:

$$\pi_*(a \mid s) := \delta\left(a = \underset{a}{\text{argmax}} Q_*(s,a)\right) \tag{5}$$

Let $V_t(s_t) = \log \beta_t(s_t), Q_t(s_t,a_t) = \log \beta_t(s_t,a_t)$. Denote $\tau = (s_1, a_1, \ldots, s_T, a_T)$ as the full trajectory. Denote $p(\tau) = p(\tau \mid \mathcal{O}_{1:T})$. Running inference in the graphical model as the Fig. 2 shows, we can compute:

$$p(\tau \mid \mathcal{O}_{1:T}) \propto p(s_t) \prod_{t=2}^{T} p(s_{t+1} \mid s_t, a_t) \times \exp(\sum_{t=1}^{T} r(s_t, a_t)) \tag{6}$$

Also, we know the following softmax relation:

$$V(s_t) = \log \int \exp(Q(s_t, a_t) + \log p(a_t \mid s_t)) da_t \tag{7}$$

We can also get:

$$p(a_t \mid s_t, \mathcal{O}_{1:T}) = exp(Q_t(s_t, a_t) - V_t(s_t)) \tag{8}$$

where we usually call $A_t(s_t, a_t) = Q_t(s_t, a_t) - V_t(s_t)$ as the advantage function.

The objective that the inference procedure want to optimize is the following KL divergence:

$$-D_{KL}(\hat{p}(\tau) \mid\mid p(\tau)) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\hat{p}(s_t,a_t)} \left[r(s_t, a_t)\right] + \mathbb{E}_{s_t\sim\hat{p}(s_t)} \left[\mathcal{H}(\pi(a_t \mid s_t))\right] \tag{9}$$

where $\mathcal{H}(\pi(a_t \mid s_t))$ is the entropy of the policy. The first term is just the standard RL objective, while the second entropy term is used for regularization.

For deterministic dynamics, we can get this objective directly. For stochastic dynamics, we obtain it from the evidence lower bound.

2 Policy Gradients

In this section, we'll be looking at directly optimizing the standard RL objective function $E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$. Here, θ are the parameters of our policy function i.e $\pi_\theta\left(\mathbf{a}|\mathbf{s}\right)$, so this amounts to finding the optimal policy function.

Let's first start off by defining the probability distribution over trajectories. The probability of any trajectory τ is given by:

$$p_\theta(\tau) = p_\theta\left(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\right) = p\left(\mathbf{s}_1\right) \prod_{t=1}^{T} \pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right) p\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) \tag{10}$$

Now, the optimal value of θ is that which maximizes our expected reward, i.e:

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{11}$$

Now, we define the objective function $J(\theta)$ as

$$J(\theta) = E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{12}$$

Hence, the optimal θ is just that θ which maximizes the objective function

$$\theta^\star = \arg\max_\theta J(\theta) \tag{13}$$

We can estimate this objective function by drawing trajectories $\tau \sim p_\theta(\tau)$ and computing a Monte-Carlo estimate of this expectation

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \approx \frac{1}{N} \sum_i \sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \tag{14}$$

Now, we simply perform gradient ascent on the objective function $J(\theta)$ to optimize it.

In the approximate form of the objective function, there is no explicit dependence of $J(\theta)$ on the parameters θ. This may naively lead us to believe that the $\nabla_\theta J(\theta) = 0$.

This of course is not true, dependence on θ has instead been absorbed into the Monte-Carlo approximation of the estimation. To make this dependence explicit, we may write $\nabla_\theta J(\theta)$ as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] && (15)\\
&= \nabla_\theta \int p_\theta(\tau) \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (16)\\
&= \int \nabla_\theta p_\theta(\tau) \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (17)\\
&= \int \left[\nabla_\theta p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (18)\\
& && (19)
\end{aligned}$$

Where the second step follows from the definition of the expectation, the third one is due to the linearity of the integration and the gradient operator and the fourth one from the fact that $[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)]$ does not depend on θ.

In its current form, $\nabla_\theta J(\theta)$ seems hard to compute since there is no obvious Monte-Carlo estimate of this integral and $\nabla_\theta p_\theta(\tau)$ depends on the dynamics of the environment $p\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right)$ which we may not know.

However, we can use the log-gradient trick to easily estimate $J(\theta)$. More specifically:

$$\nabla_\theta \log p_\theta(\tau) = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} \tag{20}$$

Hence, we can write $\nabla_\theta p_\theta(\tau)$ as:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \tag{21}$$

Now, we can substitute this expansion of $\nabla_\theta p_\theta(\tau)$ into our expression for $\nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \nabla_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \tag{22}$$

$$= \int \left[\nabla_\theta p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau \tag{23}$$

$$= \int \left[p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau \tag{24}$$

$$\tag{25}$$

Hence, we have written $\nabla_\theta J(\theta)$ in terms of an expectation over $p_\theta(\tau)$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) \sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \tag{26}$$

$$\tag{27}$$

Let's try to evaluate $\nabla_\theta \log p_\theta(\tau)$ by first writing out $\log p_\theta(\tau)$

$$\log p_\theta(\tau) = \log p\left(\mathbf{s}_1\right) + \sum_t \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) + \log p\left(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t\right) \tag{28}$$

Which gives us $\nabla_\theta \log p_\theta(\tau)$ as:

$$\nabla_\theta \log p_\theta(\tau) = \sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) \tag{29}$$

Finally, substituting the expression for $\nabla_\theta \log p_\theta(\tau)$ into the expression for $\nabla_\theta J(\theta)$, we have:

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) \right) \left(\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right) \right] \tag{30}$$

$$\tag{31}$$

Once again, we can estimate this expectation using a Monte-Carlo average by drawing sample trajectories $\tau \sim p_\theta(\tau)$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}\right) \right) \left(\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \right) \tag{32}$$

We can now update our estimate of θ by performing gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{33}$$

Looking at this expression, it is evident that the update rule is trying to up-weight those trajectories with higher total rewards (as $\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$ is higher for them) and suppress those trajectories with lower total rewards (as $\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$ is lower for them).

Putting all of these steps together, we have the REINFORCE algorithm:

Algorithm 1 REINFORCE Algorithm

1. sample $\left\{\tau^i\right\}$ from $\pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right)$
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t^i | \mathbf{s}_t^i\right)\right) \left(\sum_t r\left(\mathbf{s}_t^i, \mathbf{a}_t^i\right)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

3 Value Based Reinforcement Learning

Instead of explicitly learning the policy of a reinforcement learning agent, we can learn the optimum value function and retrieve the optimal policy from it. If we the value of $Q^*(s,a)\forall s,a$, we can obtain the opitmal policy as $\pi(a|s) = \delta(a = \text{argmax}_{a'}Q(s,a'))$.

3.1 Policy Iteration

Policy iteration iterates over two steps as shown in Algotihm 4. The first step, policy evaluation, iteratively evaluates the Q_π function of the policy using bellman update.

$$Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a)V(s')$$
$$\equiv Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a) \sum_{a'} \pi(a'|s')Q_\pi(s',a')$$

The second step greedily updates the policy by taking the action with the highest Q_π value.

Algorithm 2 Policy Iteration

Initialize random $\pi(a|s), Q(s,a), V(s)\forall s,a$
Repeat until convergence:

1. Policy Evaluation
   Repeat until convergence: $Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a) \sum_{a'} \pi(a'|s')Q_\pi(s',a')$
2. Policy Improvement
   $\pi(a|s) \leftarrow \delta\left(a = \text{argmax}_{a'}Q(s,a')\right)$

Policy improvement step is guaranteed to be at least as good as the current policy. This can be intuitively understood as follows. Let's say we take action $a_1 = \text{argmax}_a(Q_\pi(s,a))$ in the first step and follow policy π from there on. Doing so is better than or at least as good following policy π from the beginning because we took the action with highest Q_π value. We can then extend this argument when we transition to state s_2 from s_1. That is, we choose action $a_2\text{argmax}_a(Q_\pi(s_2,a))$ from state s_2 and follow policy π from there on. Likewise, following the updated policy at every step is guaranteed to be at least as good as the current policy.

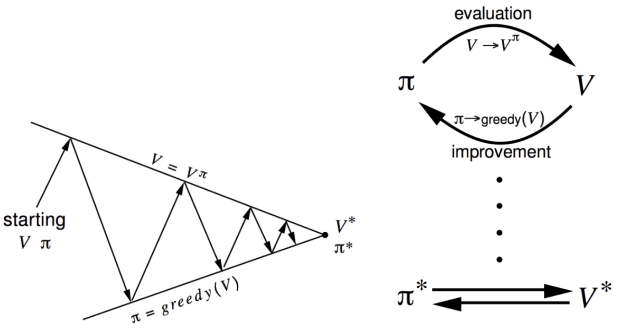

Figure 4: Left: Value function and policy are iteratively updated and at the end they converge to the optimal values. *Source: UCL Course on RL* [Link]

Fig. 4 shows policy iteration pictorially. Policy evaluation step evaluates the correct value function. A greedy update on the policy improves the policy but the value function is no longer correct. After multiple iterative steps both of these converge to the true values.

3.2 Value Iteration

Is there a way we could avoid explicitly representing the policy and perform reinforcement learning based only on value functions. The answer is a simple extension to the policy iteration algorithm we have just seen. We can combine the policy evaluation and policy improvement steps into a single step.

In the bellman update, we substitute $V(s')$ with $\max_{a'} Q(s', a')$

Algorithm 3 Value Iteration

Initialize random $Q(s, a) \forall s, a$
Repeat until convergence:
$Q(s, a) \leftarrow r(s, a) + \sum_{s'} \gamma p(s'|s, a) \max_{a'} Q(s', a')$

Due to the substitution, we can represent the update equation purely using value function. There is one key difference between policy iteration and value iteration. Policy iteration updates the Q value for multiples steps until convergence and then performs one single greedy update of the policy. Value iteration does a single update on the value function for every greedy update of the policy.

3.3 Fitted Q Iteration

Policy iteration and value iteration are applicable only for small sized discrete state-spaces. For an environment with S states and A actions per state, we need to store and update $S \times A$ Q values.

For large or continuous state spaces, we can approximate the value function Q with a function approximator with parameters θ. We can minimize the error:

$$\min_{\theta} \mathbb{E}_{a \sim \pi} ||Q^{\theta}(s, a) - y||$$
$$\text{where, } y = (r(s, a) + \gamma \max_{a'} Q^{\theta}(s', a')), \text{ is called the target}$$

Here $\pi(a|s) = \delta(a = \text{argmax}_{a'} Q(s, a'))$. We can minimize this objective with stochastic gradient descent. While updating the parameters we don't take the gradient of the target with respect to the parameters θ. We can them perform a greedy update on the policy as in policy iteration.

Algorithm 4 Fitted Q-Iteration

Initialize random $\pi(a|s), Q(s, a), V(s) \forall s, a$
Repeat until convergence:
1. Policy Evaluation
   $\min_{\theta} \mathbb{E}_{a \sim \pi} ||Q^{\theta}(s, a) - (r(s, a) + \gamma \max_{a'} Q(s', a'))||$
2. Policy Improvement
   $\pi(a|s) \leftarrow \delta\left(a = \text{argmax}_{a'} Q(s, a')\right)$

In practice, fitted Q-learning is very unstable and there are numerous tricks used to stabilize it. In particular, Deep Q Network (DQN, Mnih et al.) introduces a replay buffer and delayed parameters for the target network.

4 Soft Policy Gradient and Soft Q-learning

The soft-policy gradient is written as:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p(s_t,a_t)} \left[r(s_t,a_t)\right] + \mathbb{E}_{s_t\sim p(s_t)} \left[\mathcal{H}(\pi_\theta(a_t|s_t))\right] \tag{34}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[r(s_t,a_t) - \log(\pi_\theta(a_t|s_t))\right] \tag{35}$$

To calculate the gradient of the second term, we compute the following (using an expectation over trajectories):

$$\begin{aligned}
&\nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[\log(\pi(a_t|s_t))\right] \\
&= \int \nabla_\theta \left[p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t))\right] d\tau \\
&= \int \left[\nabla_\theta p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t)) + p(\tau)\nabla_\theta \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t))\right] d\tau \quad \text{(Using chain rule)} \\
&= \int \left[p(\tau)\nabla_\theta \log p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t)) + p(\tau) \sum_{t=1}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right] d\tau \\
&= \int p(\tau)\nabla_\theta \log(p(\tau)) \left[\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + 1\right] d\tau
\end{aligned}$$

From the backward messages in RL (previous lecture), recall that

$$Q_t(s_t,a_t) = \log(\beta_t(s_t,a_t)) = r(s_t,a_t) + \log(\mathbb{E}_{s_{t+1}\sim p(s_{t+1},a_{t+1})} \left[\exp(V_{t+1}(s_{t+1}))\right])$$

$$V_t(s_t) = \log(\beta_t(s_t)) = \log\left(\int \exp(Q_t(s_t,a_t))da_t\right)$$

From these two equations, it is easy to see that

$$\pi(a_t|s_t) = \frac{\beta_t(s_t,a_t)}{\beta_t(s_t)} = \exp\left(Q_t(s_t,a_t) - V(s_t)\right)$$

To get rid of the $p(\tau)$ term over trajectories, we approximate it using sampling N trajectories. Expanding Equation (35) and taking derivatives with respect to the parameters θ we get:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[r(s_t,a_t) - \log \pi_\theta(a_t|s_t)\right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left[r(s_t,a_t) + \left(\sum_{t'=t+1}^{T} r(s_{t'},a_{t'}) - \log(\pi_\theta(a_{t'}|s_{t'}))\right) - \log(\pi_\theta(a_t|s_t)) - 1\right] \\
&= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\nabla_\theta Q_\theta(s_t,a_t) - \nabla_\theta V_\theta(s_t)\right) \left[r(s_t,a_t) + Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) + V(s_t)\right]
\end{aligned}$$

Since the term inside the inner brackets containing the terms of t' becomes $Q_\theta(s_{t+1}, a_{t+1})$. Expanding the terms related to V_θ we get:

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\left(\nabla_\theta Q_\theta(s_t,a_t) - \nabla_\theta V_\theta(s_t)\right)\left[r(s_t,a_t) + Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) + V(s_t)\right]$$
$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta Q_\theta(s_t,a_t)\left[r(s_t,a_t) + \text{softmax}_{a_{t+1}} Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t)\right]$$

This equation results in a very similar update to Q-learning:

$$\theta \leftarrow \theta + \alpha\nabla_\theta Q_\theta(s,a)\left(r(s,a) + \gamma V(s') - Q_\theta(s,a)\right)$$

where the value function $V(s')$ is defined as:

$$V(s') = \log\int \exp(Q_\theta(s',a'))da'$$

We can technically add a temperature parameter β inside the exp of the Q-function such as $\exp\left(\frac{Q(s,a)}{\beta}\right)$. Higher values of β correspond to more 'random' policies and β close to 0 means a less stocastic policy.

Soft optimality has many benefits. Some of these benefits are listed below:

- Improves exploration and prevents entropy collapse.
- Easier to finetune policies for specific tasks. This is an empirical observation.
- Better robustness (due to wider coverage of states).
- Reduced to hard optimalty by increasing magnitude of rewards.
- This is a good model of human behavior.