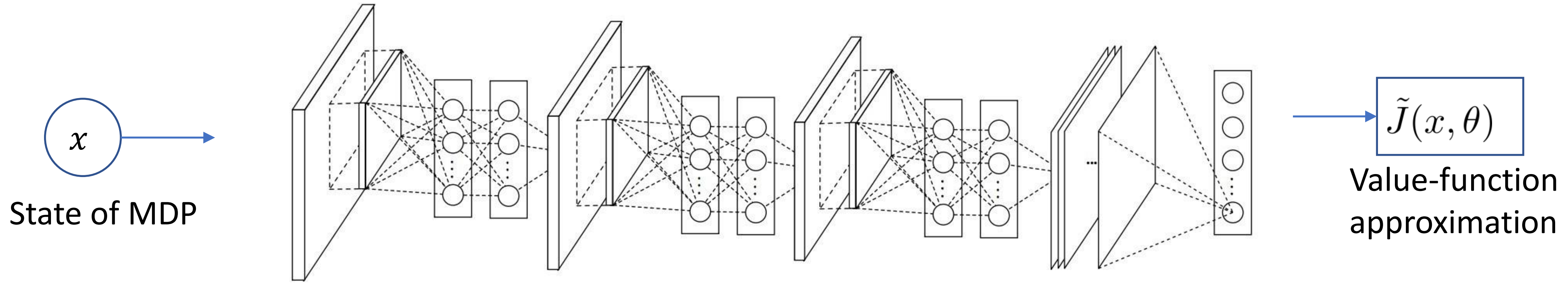
A recap of what we covered so far

- So far we covered approximation algorithms in the **Value Space:**
  - Essentially mapping states/controls to cost-to-go values ("cost values")

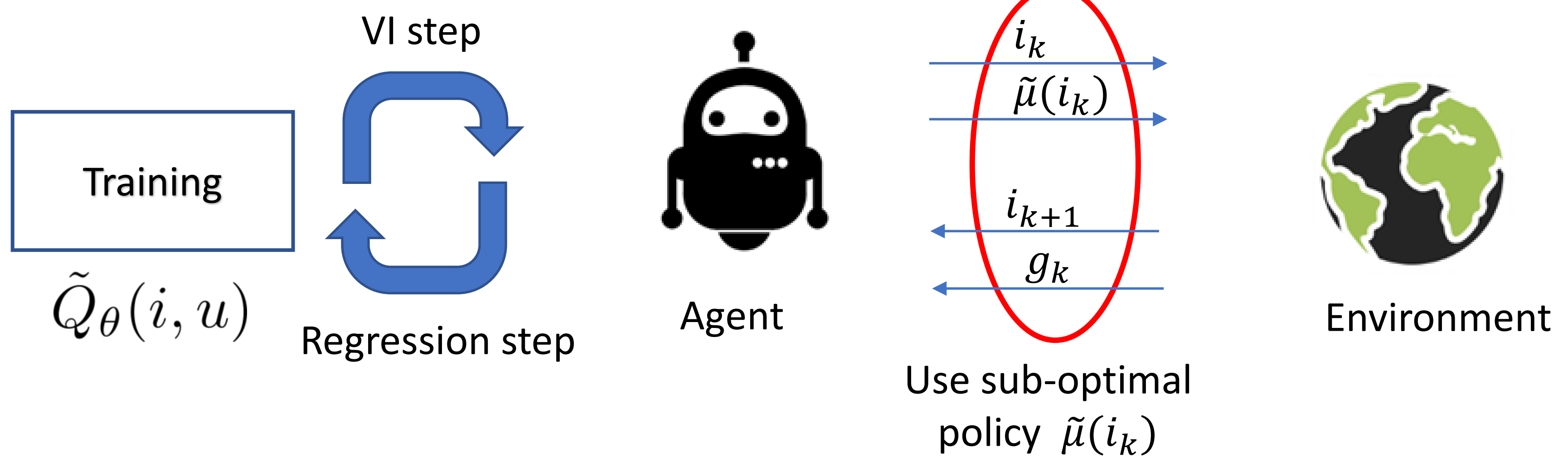


- And the policy was obtained, almost as an "afterthought", by using 1-step lookahead:

$$\tilde{\mu}(i) \in \arg \min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)(g(i^s, u, j) + \alpha \tilde{J}(j, \theta^{(t)})) \right\}$$

A recap of what we covered so far

- We covered the DQN algorithm which essentially apply the Value Iteration Algorithm using Deep Neural Networks to approximate the Q-factors.



- And the policy is given directly as:

$$\tilde{\mu}(i) = \arg \min_{u \in U(i)} \left\{ \tilde{Q}_\theta(i, u) \right\}$$

Improving Policies

- We will cover now the last remaining “block” in Approximation in Value Space, which is how to improve the policies that are being used to generate samples.

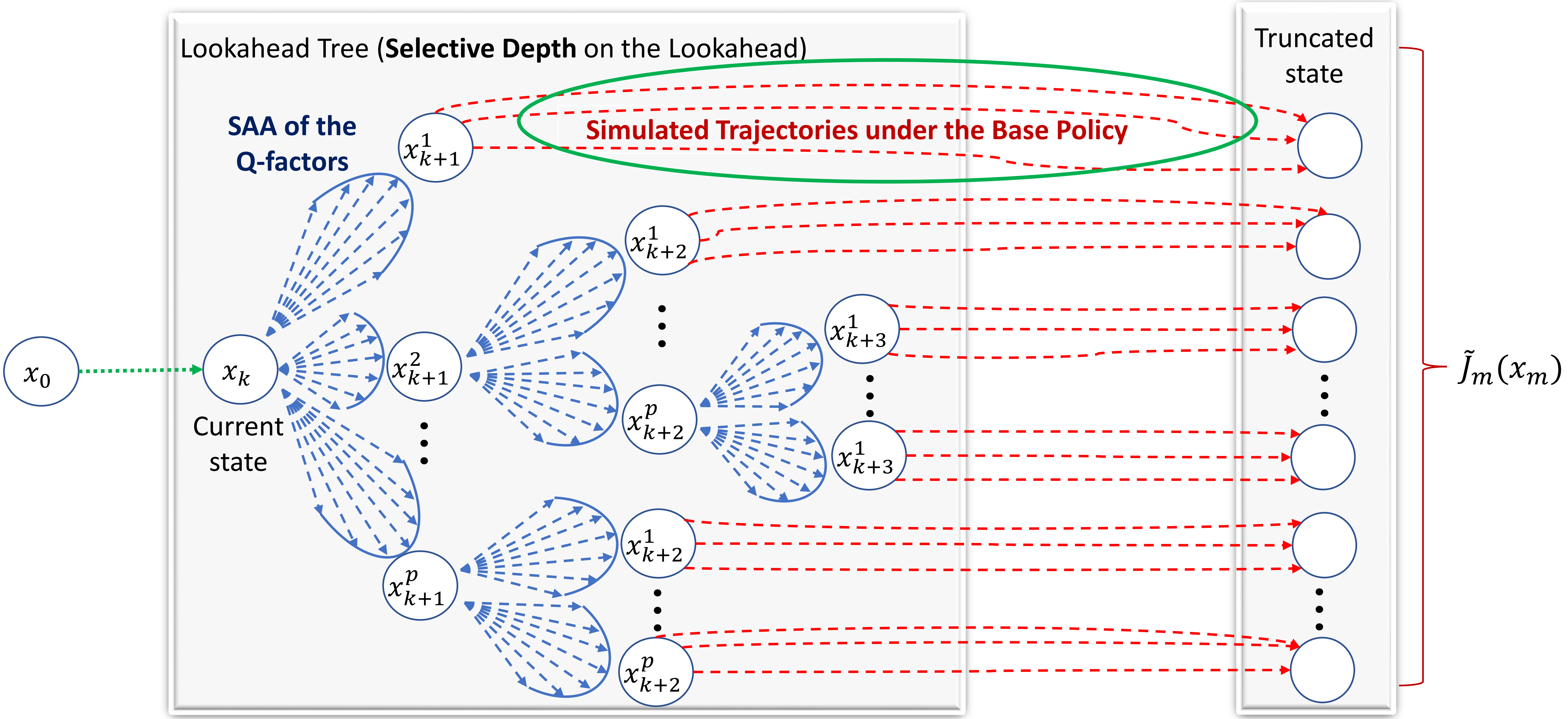
- We touched this subject on DQN, as we used the updated DNN parameter θ^{t+1} to define a sub-optimal policy with random exploration

- This had the potential of generating good samples after each gradient-step iteration:

$$\tilde{\mu}^{(t+1)}(i) = (1 - \epsilon^{(t+1)}) \arg\min_{u \in U(i)} \left\{ \tilde{Q}_{\theta^{(t+1)}}(i, u) \right\} + \epsilon^{(t+1)} \mathrm{A}(U(i))$$

- Is there are a way to generate a sequence of improving polices such that they converge to the optimal policy?

Revisiting the Monte-Carlo Tree Search



Recap: Rollout Algorithm

- We will start our analyzes of approximation in Policy Space, by first recording the Rollout Algorithm: the goal is to solve a 1-step lookahead minimization:

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \left\{ \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right] \right\}, \forall k \in \{0, ..., N-1\}$$

- Where the cost-to-go approximation $\tilde{J}_{k+1}(x_{k+1})$ is as the total cost of some *Base Policy* $\hat{\pi} = (\hat{\mu}_{k+1}, \dots, \hat{\mu}_{N-1})$:

$$x_{i+1} = f_i(x_i, \hat{\mu}_i(x_i), w_i),\ \forall i = k+1, ..., N-1$$

- For some simulated disturbances sequences $(w_k, \dots, w_{N-1})$.

Recap: Rollout Algorithm

- Then we use Sample-Average Approximation to compute the approximate Q-factors:

$$\tilde{Q}_k(x_k, u_k) = \mathbb{E}_{w_k}[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k))]$$

$$\tilde{Q}_k(x_k, u_k) \approx \sum_{s=1}^{S} r_s\big(g_k(x_k, u_k, w_k^s) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k^s))\big)$$

- And the Rollout Policy becomes:

$$\tilde{\mu}_k(x_k) \in \min_{u_k \in U_k(x_k)} \Big\{ \sum_{s=1}^{S} r_s\big(g_k(x_k, u_k, w_k^s) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k^s))\big) \Big\}$$

Policy Iteration

- Observe what the Rollout Algorithm does:
  - (1) We start with a base policy $\hat{\mu}_k$
  - (2) Then it generate a new rollout policy $\tilde{\mu}_k$

- This begs the question: What if we, now, use $\tilde{\mu}_k$ as the base policy in order to get a new policy?

- Can we generate a “perpetual” Rollout Algorithm that sequentially generates a sequence of policies:

$$\tilde{\mu}_k^{(0)}(x_k) \rightarrow \tilde{\mu}_k^{(1)}(x_k) \rightarrow \tilde{\mu}_k^{(2)}(x_k) \rightarrow \cdots$$

- This is the **Policy Iteration Algorithm** (or just PI Algorithm)

Policy Iteration and MDP

- As we did in the Value Iteration(VI) analyses, we will use the equivalence between finite-state DP and MDP:
  - $x = \{1,2, \dots, n\}$: “set of integers”. $u \in U(i)$: “actions/controls available at state i”.
  - $p_{ij}(u)$: “probability of moving from i to j, given control u”.
  - g(i, u, j): “cost of moving from i to j, given control u”.

- In addition we will use the infinite-horizon formulation. The goal, as before, is to solve the **Bellman’s Equation**:

$$J^*(i) = \min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha J^*(j)) \right\}$$

- In Exact VI, we have convergence: after finding the values $J^*(1), \dots, J^*(n)$, the optimal **stationary** policy can be read from the right-hand side of the Bellman’s Equation.

- In Exact PI, it is the opposite: First we seek to obtain the stationary policy, and then, show that it solves the Bellman’s Equation.

Policy Iteration and MDP

- The Policy Iteration begins with a (stationary) base policy $\mu^{(t)}$ and operates in two steps.

- <u>**Policy Evaluation step:**</u> We compute $J_{\mu^{(t)}}(1), \dots, J_{\mu^{(t)}}(n)$ which solves the system of equations:

$$J_{\mu^{(t)}}(i) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\left(g(i, \mu^{(t)}(i), j) + \alpha J_{\mu^{(t)}}(j)\right)$$

  - This step, solves a "version" of the Bellman's Equation where we stick to base policy $\mu^{(t)}$.

  - This is a **linear** system on the variables $J_{\mu^{(t)}}(1), \dots, J_{\mu^{(t)}}(n)$.

Policy Iteration and MDP

- **Policy Improvement step:** We compute a new policy $\mu^{(t+1)}$ as:

$$\mu^{(t+1)}(i) \in \arg\min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{\mu^{(t)}}(j)\big) \right\},\ \forall i \in \{1, \ldots, n\}$$

- Notice that this is similar to a 1-step lookahead minimization.

- So the Policy Improvement step, is essentially the Rollout Algorithm, where $\mu^{(t)}$ plays the role of the base policy and $\mu^{(t+1)}$ plays the role of the rollout policy.

- The PI Algorithm alternates between these two steps sequentially, until:

$$J_{\mu^{(t+1)}}(i) = J_{\mu^{(t)}}(i),\ \forall i \in \{1, \ldots, n\}$$

Policy Iteration and MDP

- It follows that PI algorithm converges to the optimal stationary policy, which solves the Bellman Equation:

(Convergence of PI): Given any initial stationary policy $\mu^{(0)}$, the sequence $\{\mu^{(t)}\}_{t\geq 0}$ generated by the PI Algorithm, have the policy improvement property:

$$J_{\mu^{(t+1)}}(i) \leq J_{\mu^{(t)}}(i), \; \forall i \in \{1, \ldots, n\} \text{ and } t \geq 0$$

And the algorithm terminates with the optimal stationary policy that solves the Bellman Equation:

$$J^*(i) = \min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha J^*(j)) \right\}$$

Policy Iteration: Proof of Convergence

- The proof of this result has a very nice intuition as it ties with the **policy improvement property** we saw in the Rollout Algorithm.

- Let μ be our starting policy and $\bar{\mu}$ be the policy generated by 1 iteration of the PI Algorithm. The goal is show that:

$$J_{\bar{\mu}}(i) \leq J_{\mu}(i),\ \forall i \in \{1, ..., n\}$$

- Consider the cost J_N of a policy that applies $\bar{\mu}$ for the first N stages, and then applies μ for every subsequent stage:

Policy Iteration: Proof of Convergence

- Now we can write the Bellman's Equation:

$$J_\mu(i) = \sum_{j=1}^{n} p_{ij}(\mu(i))\big(g(i,\mu(i),j) + \alpha J_\mu(j)\big)$$

- Now by the Policy Improvement step, it follows that:

$$\bar{\mu}(i) \in \arg\min_{u\in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu(j)\big) \right\}, \forall i \in \{1, ..., n\}$$

- Which gives us:

$$J_1(i) = \sum_{j=1}^{n} p_{ij}(\bar{\mu}(i))\big(g(i,\bar{\mu}(i),j) + \alpha J_\mu(j)\big) \leq J_\mu(i)$$

Policy Iteration: Proof of Convergence

- Now using $J_1(i)$ in place $J_\mu(i)$, we obtain:

$$J_2(i) = \sum_{j=1}^{n} p_{ij}(\bar{\mu}(i))\big(g(i, \bar{\mu}(i), j) + \alpha J_1(j)\big) \leq J_1(i)$$

- Then we have the inequality:

$$J_1(i) \leq J_2(i) \leq J_1(i) \leq J_\mu(i),\ \forall i \in \{1, ..., n\}$$

- Now, proceeding inductively, we can conclude:

$$J_{N+1}(i) \leq J_N(i) \leq J_\mu(i),\ \forall i \in \{1, ..., n\} \text{ and } \forall N \geq 1$$

- Now taking the limit of $N \to \infty$, it follows that: $J_N(i) \to J_{\bar{\mu}}(i)$ and we have:

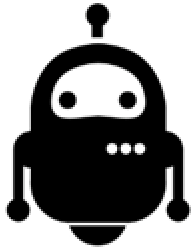
$$J_{\bar{\mu}}(i) \leq J_\mu(i),\ \forall i \in \{1, ..., n\}$$

Example: treasure hunting

- Consider the problem of an agent trying to maximize his profit:

Every day

1. Investigate the state
2. Go home



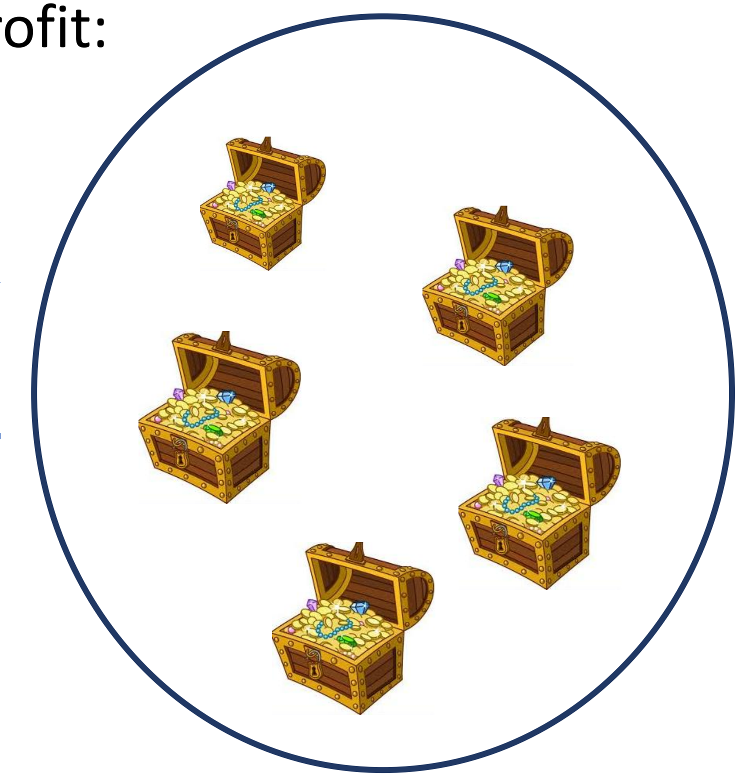
Agent

Explore

Cost: c

Obtain treasure

Value of a treasure: 1



Site with n treasures

- If the agent decides to explore the site when there are i treasures left, they find $m \in [0, i]$ treasures with probability $p(m|i)$. Assume that $p(0|i) < 1$, for all $i \geq 1$.

Example: treasure hunting

- Therefore, if there are i treasures remaining the expected reward to be found on the site is:

$$r(i) = \sum_{m=0}^{i} m p(m|i)$$

- And we assume that $r(i)$ is monotonically increasing with i (so the more treasures on the site, the more we expect to make by exploring it).

- Let the states of the MDP be the amount of treasures <u>not yet</u> found plus an additional termination state 0, that indicate that either the agent decided to go home or there are no more treasures to be found.

- The controls are either "explore" or "go home". If we explore, we pay the fixed cost c.

Example: treasure hunting

- So if we are on state $i \geq 1$, then the state moves to state $i - m$ with probability $p(m|i)$

- We start by writing the Bellman's Equation:

$$J^*(i) = \max\left\{0, r(i) - c + \sum_{m=0}^{i} p(m|i) J^*(i-m)\right\},\ i \in \{1, ..., n\}$$

- And $J^*(0) = 0$.

- Let's start the PI Algorithm using the first policy $\mu^{(0)}$ to <u>never</u> explore.

Example: treasure hunting

- Let's apply the policy evaluation step:

$$J_{\mu^{(t)}}(i) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\big(g(i, \mu^{(t)}(i), j) + \alpha J_{\mu^{(t)}}(j)\big)$$

- Which, in this case, reduces to:

$$J_{\mu^{(0)}}(i) = 0,\ i \in \{1, ..., n\}$$

- Now, we apply the policy improvement step:

$$\mu^{(t+1)}(i) \in \arg\min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha J_{\mu^{(t)}}(j)\big) \right\},\ \forall i \in \{1, ..., n\}$$

- Which, in this case, reduces to:

$$\mu^{(1)}(i) \in \arg\max \big\{0, r(i) - c\big\},\ \forall i \in \{1, ..., n\}$$

Example: treasure hunting

- This policy can be written equivalently as:

$$\mu^{(1)}(i) = \begin{cases} \text{“explore”, if } r(i) > c \\ \text{“go home”, otherwise} \end{cases}$$

- Now we perform the policy evaluation step again:

$$J_{\mu^{(1)}}(i) = \begin{cases} 0, & \text{if } r(i) \leq c \\ r(i) - c + \sum_{m=0}^{i} p(m|i) J_{\mu^{(1)}}(i-m), & \text{if } r(i) > c \end{cases}$$

- Note that as given by the convergence result, we have that:

$$J_{\mu^{(1)}}(i) \geq J_{\mu^{(0)}}(i) = 0, \forall i \in \{1, ..., n\}$$

Example: treasure hunting

- Now we apply the Policy Improvement step again:

$$\mu^{(2)}(i) \in \arg\max\left\{0, r(i) - c + \sum_{m=0}^{i} p(m|i) J_{\mu^{(1)}}(i-m)\right\}, \forall i \in \{1, ..., n\}$$

- Now observe if $r(i) \leq c$ then it follows that $r(j) \leq c$ for all $j < i$, since $r(i)$ monotonically increasing. Then it follows that for all i such that $r(i) \leq c$:

$$0 \geq r(i) - c + \sum_{m=0}^{i} p(m|i) J_{\mu^{(1)}}(i-m)$$

- So

$$\mu^{(2)}(i) = \text{ ``go home ''}, \forall i \text{ such that } r(i) \leq c$$

Example: treasure hunting

- Now if $r(i) > c$, since:

$$J_{\mu^{(1)}}(i) \geq J_{\mu^{(0)}}(i) = 0,\ \forall i \in \{1, ..., n\}$$

- It follows that

$$0 < r(i) - c + \sum_{m=0}^{i} p(m|i) J_{\mu^{(1)}}(i-m)$$

- So

$$\mu^{(2)}(i) = \text{ “explore ”},\ \forall i \text{ such that } r(i) > c$$

- Then we have that:

$$\mu^{(2)}(i) = \begin{cases} \text{“explore”, if } r(i) > c \\ \text{“go home”, otherwise} \end{cases} \Rightarrow \text{Same as } \mu^{(1)}(i) \Rightarrow \text{Done!}$$

Policy Iteration and Q-factors

- Similar to VI, we may also implement PI through the use of Q-factors, by writing:

$$Q_\mu(i,u) = \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha J_\mu(j)),\ i \in \{1,..,n\}, u \in U(i)$$

$$J_\mu(j) = Q_\mu(i, \mu(j))$$

- Where $Q_\mu(i,u)$ is the Q-factor of the state-action pair (i,u) associated with the policy μ. And the two steps become:

$$Q_{\mu^{(t)}}(i,u) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\big(g(i,u,j) + \alpha Q_{\mu^{(t)}}(j, \mu^{(t)}(j))\big)$$

Policy Evaluation Step

$$\mu^{(t+1)}(i) \in \arg\min_{u \in U(i)} \left\{ Q_{\mu^{(t)}}(i,u) \right\}, \forall i \in \{1, \ldots, n\}$$

Policy Improvement Step

Optimistic Policy Iteration

- Now let's address the fact that the Policy Evaluation step

$$J_{\mu^{(t)}}(i) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\big(g(i, \mu^{(t)}(i), j) + \alpha J_{\mu^{(t)}}(j)\big)$$

- Can potentially require a very large system of linear equations.

- When n is too large we can replace the Policy Evaluation step by the Value Iteration Algorithm.

- This can be a bit abstract, but the idea is to instead of solving a linear system, we iteratively attempt to solve it using VI (an idea very similar to Newton's method).

Optimistic Policy Iteration

- The Optimistic (or Generalized) PI Algorithm can be given as follows. Given some function $J^{(t)}(i)$:

- **<u>Policy Improvement step:</u>** We compute a new policy $\mu^{(t+1)}$ as:

$$\mu^{(t)}(i) \in \arg\min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J^{(t)}(j)\big) \right\}, \forall i \in \{1, \ldots, n\}$$

- **Policy Evaluation step:** Starting with $\hat{J}_0^{(t)} = J^{(0)}$ we apply m_t VI-steps for policy $\mu^{(t)}$ to compute $\hat{J}_1^{(t)}, \ldots, \hat{J}_{m_t}^{(t)}$ according to:

$$\hat{J}_{m+1}^{(t)}(i) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\big(g(i,\mu^{(t)}(i),j) + \alpha \hat{J}_m^{(t)}\big) \Big\}, \forall i \in \{1, \ldots, n\}$$

- For all $m \in \{0, \ldots, m_t - 1\}$ and sets $J^{(t+1)} = \hat{J}_{m_t}^{(t)}$.