Recap: 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 $J_{\mu^{(t)}}(1), \ldots, J_{\mu^{(t)}}(n)$ which solves the system of equations:

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

  - 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 $J_{\mu^{(t)}}(1), \ldots, J_{\mu^{(t)}}(n)$.

Recap Policy Iteration and MDP

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

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

- 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:

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

Recap: Optimistic Policy Iteration

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

- **Policy Improvement step:** 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 $J^{(t+1)} = \hat{J}_{m_t}^{(t)}$.

Approximate Policy Iteration

- As always let us introduce approximation architectures. Approximate PI can be framed in terms of each of the two steps:

- **Critic Step**: Given a current policy $\mu^{(t)}$ we use an approximation architecture to perform the policy evaluation, namely to compute the cost-to-go values $\tilde{J}_{\mu^{(t)}} \approx J_{\mu^{(t+1)}}$.

- **Actor Step:** Given the approximate cost-to-go values $\tilde{J}_{\mu^{(t)}}$, we solve a lookahead minimization to generate the improved policy$\mu^{(t+1)}$**.** This minimization can also be approximated by using an architecture to generate $\mu^{(t+1)} \approx \tilde{\mu}^{(t+1)}$.

- The Reinforcement Learning (RL) Algorithms that implement both steps are called the **Actor-Critic Algorithm**.

Critic-only Algorithms

- We begin our analyses with Critic-only algorithms:
  - Only the critic step (policy evaluation) is done with approximations
  - The actor step (policy improvement) is done exactly

- Suppose we start with a some base policy $\mu^{(t)}$. Consider an approximation architecture, say a DNN, is used to generate:

$$\tilde{J}_{\mu^{(t)}}(i,\theta) \approx J_{\mu^{(t)}}(i)$$

- Where θ represents DNN parameters.

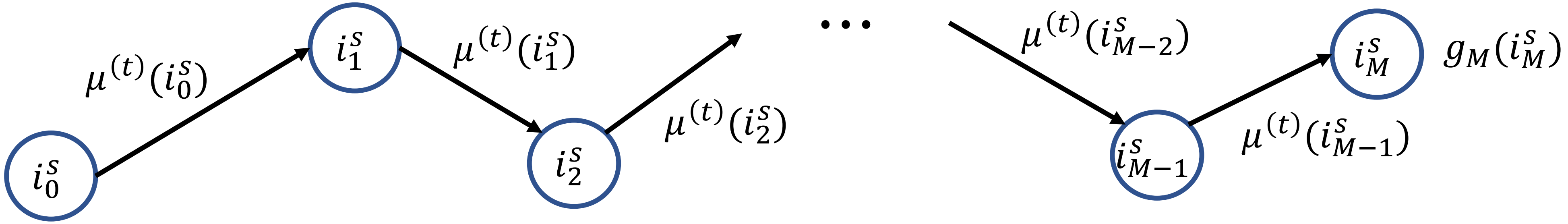
- Note that, differently from fitted-VI, the cost-to-go approximation is associated to a policy $\mu^{(t)}$.

Critic-only Algorithms

- As always, suppose that we can generate sample state/cost-to-go pairs:

$$(i_0^1, \beta^1), (i_0^2, \beta^2), \ldots, (i_0^S, \beta^S)$$

- Where S is number of samples. These costs can be generated by applying the policy $\mu^{(t)}$:



- Then we have:

$$\beta^s = \sum_{k=0}^{M-1} \alpha^k g(i_k^s, \mu^{(t)}(i_k^s), i_{k+1}^s) + \alpha^M \hat{J}(i_M^s)$$

Terminal cost function approximation

Critic-only Algorithms

- Notice that $\mu^{(t)}$ plays the role of the base policy, in the Rollout Algorithm. We can summarize the simulation process as:

- Then the critic-step reduced to the usual regression (training) problem:

$$\theta^{(t)} = \arg\min_{\theta} \left\{ \sum_{s=1}^{S} \left(\tilde{J}_{\mu^{(t)}}(i_0^s, \theta) - \beta^s \right)^2 \right\}$$

- Solved by gradient-type methods (e.g.: SGD), like always.

Critic-only Algorithms

- In critic-only algorithms, the actor-step (policy improvement) is obtained exactly:

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

- And then we repeat:
  - Use $\mu^{(t+1)}$ to simulate more rollout trajectories
  - Collecting more samples associated with policy $\mu^{(t+1)}$
  - Update $\theta^{(t)}$ to $\theta^{(t+1)}$; and so forth

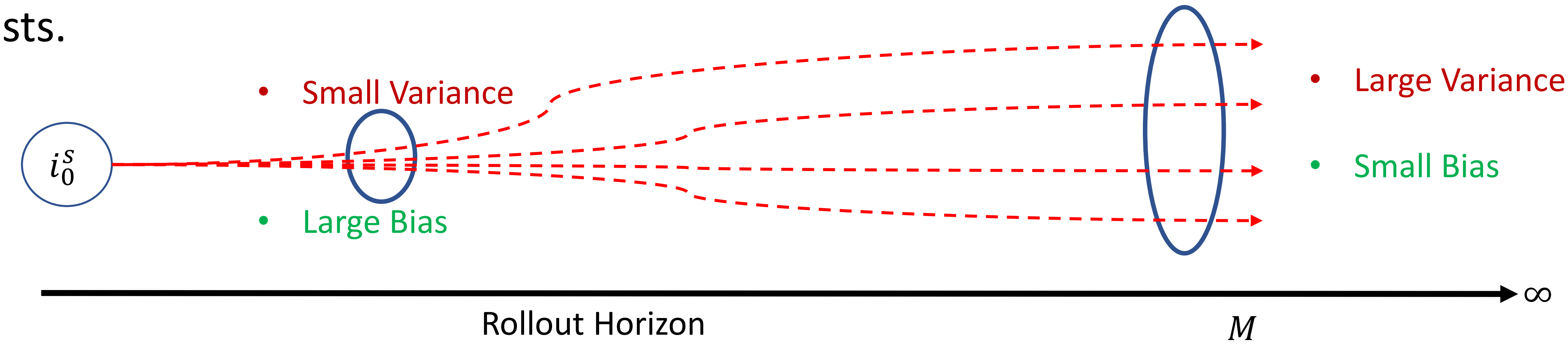
- Are there any problems with this approach?

Bias-Variance Tradeoff

- The central problem here is that the samples we are collecting are always linked to some policy:
  - Samples generated while under $\mu^{(t)}$ **cannot** be treated to be the same as sampled generated under $\mu^{(t+1)}$

- In addition, if the rollout horizon M is too long, there will be a lot of a variance in the simulated costs.

- However, if the rollout horizon M is too short, there will be a lot of bias in the simulated costs.



Critic-only Algorithms with Q-factors

- We can make the algorithm model-free (i.e.: without knowledge of the transition probabilities) by using Q-factors.

- As always with Q-factors, given a state-control pair (i, u) we want to generate the approximate Q-factor:

$$\tilde{Q}_{\mu^{(t)}}(i, u, \theta) \approx Q_{\mu^{(t)}}(i, u)$$

- The critic step becomes, after obtaining samples $\{(i_0^s, u^s, \beta^s)\}_{s=1}^{S}$:

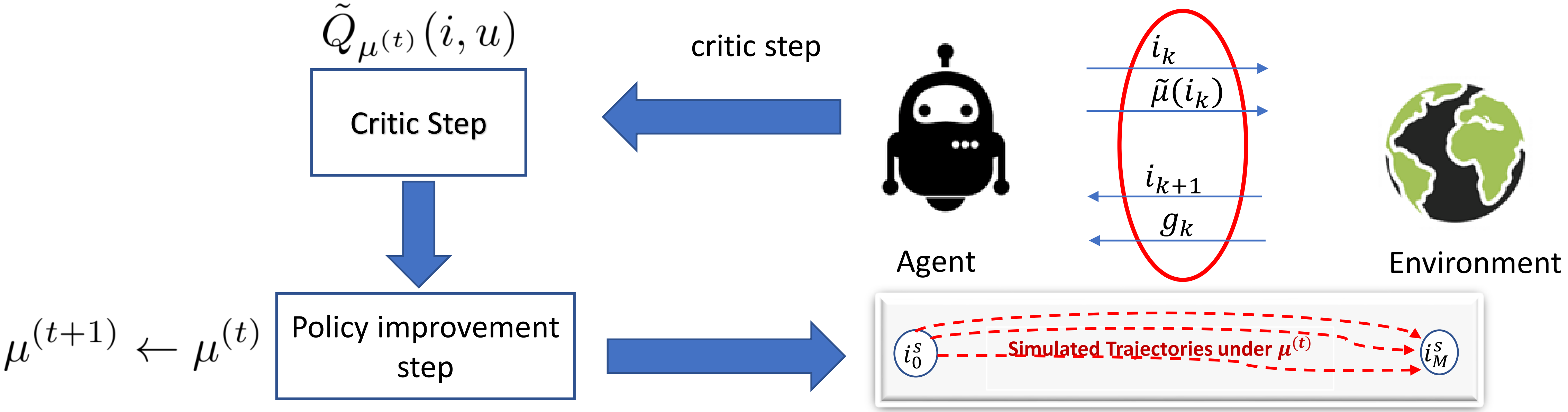
$$\theta^{(t)} = \arg\min_{\theta} \left\{ \sum_{s=1}^{S} \left(\tilde{Q}_{\mu^{(t)}}(i_0^s, u^s, \theta) - \beta^s \right)^2 \right\}$$

- And $$\beta^s = g(i_0^s, u^s, i_{k+1}^s) + \sum_{k=1}^{M-1} \alpha^k g(i_k^s, \mu^{(t)}(i_k^s), i_{k+1}^s) + \alpha^M \hat{J}(i_M^s)$$

Critic-only Algorithms with Q-factors

- Then the policy improvement step becomes:

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



- This variant with Q-factors, share the same issues as the one with value function approximations.

Actor-only Algorithms

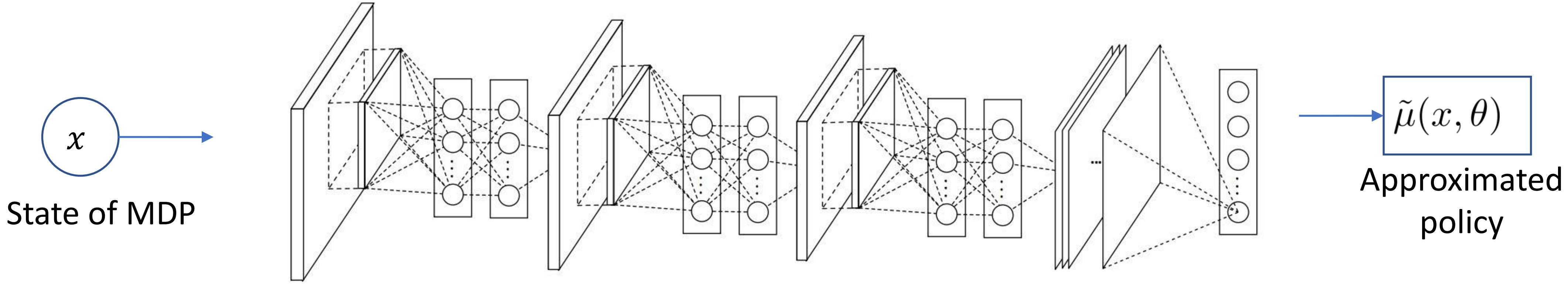
- Now let’s focus on the actor component step:

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

- We will introduce approximation in the **Policy Space**, by using say a DNN, to generate:

$$\tilde{\mu}(i, \theta) \approx \mu(i)$$

- Now, what we seek is a mapping from <u>states to actions</u>.



Actor-only Algorithms

- Notice, that we can define a policy parametrization through cost parametrization:

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

- In this case, the approximated policy is given as the solution of the 1-step lookahead minimization problem.

- The issue here is how can we update θ:
  - By using approximate VI?
  - By minimizing some other objective? For example $J_{\tilde{\mu}(\cdot,\theta)}$?

- We have to be careful because the policy given by an “ArgMin” may not be differentiable w.r.t. θ.

Training by Cost Optimization

- We will start by properly defining what we want to optimize when we generate:

$$\tilde{\mu}(i, \theta) \approx \mu(i)$$

- Ideally we want to perform the following optimization:

$$\min_{\theta} \mathbb{E}_{p_0} \left[J_{\tilde{\mu}(\theta)}(i_0) \right]$$

- Where $\tilde{\mu}(\theta)$ specifies an action for every possible state (we omit the state argument)
- p_0 is some initial distribution of the initial states.
- $J_{\tilde{\mu}(\theta)}(\cdot)$ is the cost-to-go associated with the policy $\tilde{\mu}(\theta)$.

Policy Gradient

- The framework of Cost (or Reward) Optimization leads to the very first algorithm in the RL community, developed separately from the DP framework: the **Policy Gradient Algorithm.**

- If we assume $J_{\widetilde{\mu}(\theta)}(\cdot)$ is differentiable w.r.t. θ and if we know the initial state i_0, we can write the usual gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \nabla J_{\tilde{\mu}(\theta^{(t)})}(i_0), \quad \forall t \geq 0$$

- The issue is that the gradient may not be explicitly given and we may not know p_0 , the initial state distribution.

- However we will leverage simulation and sampling to perform the gradient step.

Policy Gradient

- We will perform a “trick”, where we will allow our approximated policy to be randomized:

$$\tilde{\mu}(i,\theta) = u, \quad w.p. \quad p(u|i,\theta) \qquad \forall u \in U(i)$$

- Then, the optimization becomes:

$$\min_{\theta} \mathbb{E}_{p(z|\theta)} \big[\sum_{k=0}^{\infty} \alpha^k g(i_k, u_k) \big]$$

- Where $p_{(z|\theta)}$ is the condition distribution of $z = (i_0, u_0, i_1, u_1, \dots)$ given θ.

- For simplicity we assumed the stage cost only depend on the initial state and control.

Policy Gradient

- Let's unpack $p_{(z|\theta)}$:

$$p(z|\theta) = p(i_0, u_0, i_1, u_1 ..., |\theta)$$

- Applying the Markov Property:

$$p(z|\theta) = p(i_0, u_0, i_1, u_1 ..., |\theta) = p(i_0) \prod_{k=0}^{\infty} p_{i_k, i_{k+1}}(u_k) p(u_k | i_k, \theta)$$

- Now let $F(z)$ be:

$$F(z) = \sum_{k=0}^{\infty} \alpha^k g(i_k, u_k)$$

Policy Gradient

- Then the optimization problem becomes:

$$\min_{\theta} \mathbb{E}_{p_{z|\theta}} \big[F(z) \big]$$

- Now we apply the popular "log-trick" : $\nabla \ln(p) = \frac{\nabla p}{p}$ as follows:

$$\nabla_\theta \Big(\mathbb{E}_{p(z|\theta)} \big[F(z) \big] \Big) = \nabla_\theta \Big(\sum_{z \in Z} p(z|\theta) F(z) \Big) = \sum_{z \in Z} \nabla_\theta \big(p(z|\theta) \big) F(z) =$$

$$\sum_{z \in Z} p(z|\theta) \frac{\nabla_\theta (p(z|\theta))}{p(z|\theta)} F(z) = \sum_{z \in Z} p(z|\theta) \nabla_\theta \big(\ln(p(z|\theta)) \big) F(z) =$$

$$\mathbb{E}_{p(z|\theta)} \Big[\nabla_\theta \big(\ln(p(z|\theta)) \big) F(z) \Big]$$

Policy Gradient

- Now let's use the un-packed $p(z|\theta)$ to expand the gradient:

$$\nabla_\theta\big(\ln(p(z|\theta))\big) = \nabla_\theta\big(\ln(p(i_0)\prod_{k=0}^{\infty} p_{i_k,i_{k+1}}(u_k)p(u_k|i_k,\theta))\big) =$$

$$\nabla_\theta\left(\ln(p(i_0)) + \sum_{k=0}^{\infty}\ln(p_{i_k,i_{k+1}}(u_k)) + \sum_{k=0}^{\infty}\ln(p(u_k|i_k,\theta))\right) =$$

$$\sum_{k=0}^{\infty}\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big)$$

Policy Gradient

- Now, substituting back in the gradient expression:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}\big[F(z)\big]\big) = \mathbb{E}_{p(z|\theta)}\Big[\nabla_\theta\big(\ln(p(z|\theta))\big)F(z)\Big] =$$

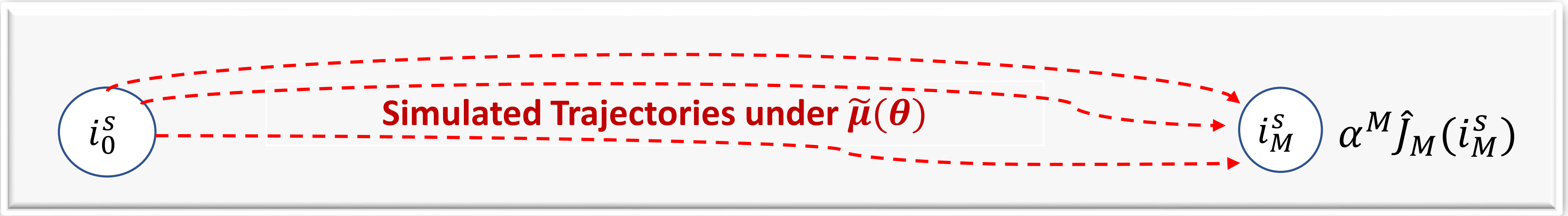
$$\mathbb{E}_{p(z|\theta)}\Bigg[\bigg(\sum_{k=0}^{\infty}\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big)\bigg)\bigg(\sum_{k=0}^{\infty}\alpha^k g(i_k,u_k)\bigg)\Bigg]$$

- Now we remove the infinite summations, by truncating the trajectories at some stage M and using a terminal cost approximation (obtained, for example by approx. VI):

$$\mathbb{E}_{p(z|\theta)}\Bigg[\bigg(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big)\bigg)\bigg(\sum_{k=0}^{M-1}\alpha^k g(i_k,u_k)+\alpha^M\hat{J}_M(i_m)\bigg)\Bigg]$$

Policy Gradient

- Now, we proceed like always: We replace expectations by sample average approximation (SAA) using simulation:



- and we obtain:

$$\nabla_\theta\left(\mathbb{E}_{p(z|\theta)}\left[F(z)\right]\right) \approx \frac{1}{S}\sum_{s=1}^{S}\left(\sum_{k=0}^{M-1}\nabla_\theta\left(\ln(p(u_k^s|i_k^s,\theta))\right)\right)\left(\sum_{k=0}^{M-1}\alpha^k g(i_k^s,u_k^s)+\alpha^M\hat{J}_M(i_m^s)\right)$$

REINFORCE: Policy Gradient Algorithm

- At last, the algorithm known as the REINFORCE algorithm (or simple the Policy Gradient) is given as follows:

Algorithm 1 REINFORCE Algorithm (Policy Gradient)

Input: Initial DNN parameters $\theta^{(0)}$ and randomized policy $\tilde{\mu}(\theta^{(0)})$.

1: **for** $t = 0, ..., T$ **do** (obtaining new samples)

2: Collect S sample trajectories $z^s = (i_0^s, u_0^s, ..., i_M^s)$ using the policy $\tilde{\mu}(\theta^{(t)})$

3: Compute the policy gradient:

$$\nabla_\theta \big(\mathbb{E}_{p(z|\theta^{(t)})} \big[F(z) \big] \big) \approx \frac{1}{S} \sum_{s=1}^{S} \left(\sum_{k=0}^{M-1} \nabla_\theta \big(\ln(p(u_k^s | i_k^s, \theta^{(t)})) \big) \right) \left(\sum_{k=0}^{M-1} \alpha^k g(i_k^s, u_k^s) + \alpha^M \hat{J}_M(i_m^s) \right)$$

4: Perform the gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \nabla_\theta \big(\mathbb{E}_{p(z|\theta^{(t)})} \big[F(z) \big] \big)$$

5: **end for**

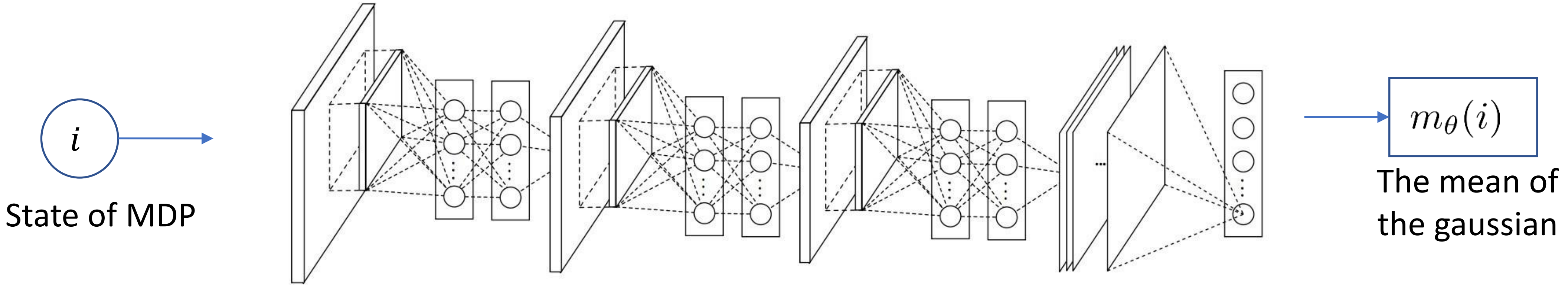
Output: The last DNN configuration $\theta^{(T)}$. A suboptimal policy $\tilde{\mu}(\theta^{(T)})$

Example: Gaussian Policies

- As an example of a randomized policy, Gaussian Networks are often used in the policy gradient framework, when we have continuous actions.

- In this case we have:

$$\tilde{\mu}(i, \theta) \sim \mathcal{N}(m_\theta(i), \Sigma)$$



- For discrete action space, we could use Logistic functions ("soft-max" policies).

Example: Gaussian Policies

- For the gaussian case, then we can write:

$$\ln(p(u_k|i_k,\theta)) = -\frac{1}{2}(m_\theta(i_k) - u_k)^\top \Sigma (m_\theta(i_k) - u_k) + \text{constant}$$

- Taking the gradient w.r.t. θ yields:

$$\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big) = -\frac{1}{2}\Sigma^{-1}(m(i_k) - u_k)\nabla_\theta(m_\theta(i_k))$$

Backpropagation

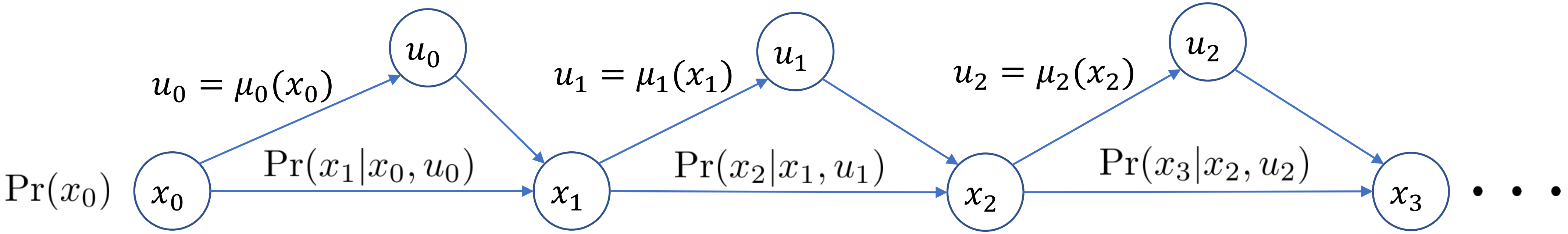
- The training becomes very efficient, if we use Gaussian Policies.

Policy Gradient as weighted likelihood

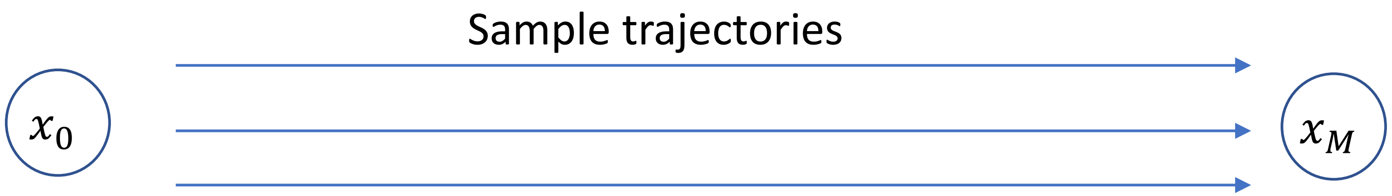
- Let us restate the main equation of the policy gradient:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}[F(z)]\big) \approx \frac{1}{S}\sum_{s=1}^{S}\left(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k|i_k^s,\theta))\big)\right)\left(\sum_{k=0}^{M-1}\alpha^k g(i_k^s,u_k^s)+\alpha^M \hat{J}_M(i_m^s)\right)$$

- Let's recall our MDP model:



- Let's ignore the costs for the moment. Suppose we collect trajectory samples:



Policy Gradient as weighted likelihood

- With a randomized policy we can ask the following the question: Find the parameter θ that makes the samples sequence most likely to occur:

- That is the Maximum Likelihood (ML) problem (Which we saw for the HMM problem):

$$\max_{\theta}\Big\{\prod_{s=1}^{S} p(z^s|\theta)\Big\} = \max_{\theta}\Big\{\prod_{s=1}^{S} p(i_0^s)\prod_{k=0}^{\infty} p_{i_k^s, i_{k+1}^s}(u_k^s)p(u_k^s|i_k^s,\theta)\Big\}$$

- We can take the long and then the gradient obtaining:

$$\frac{1}{S}\sum_{s=1}^{S}\Big(\sum_{k=0}^{M-1}\nabla_{\theta}\big(\ln(p(u_k^s|i_k^s,\theta))\big)\Big)$$

- This is exactly equal the first component of the Policy Gradient!

Policy Gradient as weighted likelihood

- Again:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}[F(z)]\big) \approx \frac{1}{S}\sum_{s=1}^{S} \underbrace{\left(\sum_{k=0}^{M-1} \nabla_\theta\big(\ln(p(u_k|i_k^s,\theta))\big)\right)}_{\text{Maximum-Likelihood}} \underbrace{\left(\sum_{k=0}^{M-1} \alpha^k g(i_k^s,u_k^s) + \alpha^M \hat{J}_M(i_m^s)\right)}_{\text{Costs “weights”}}$$

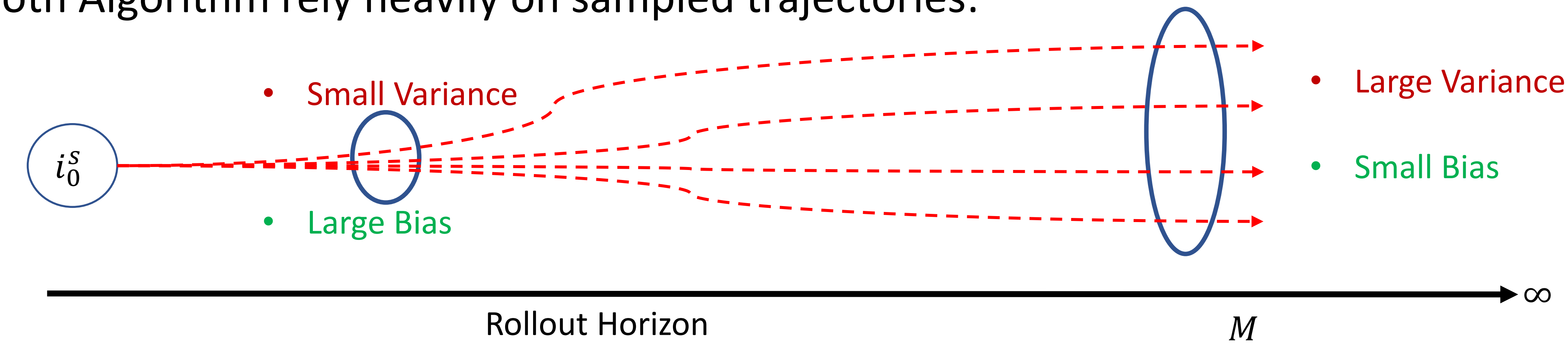
- So the Policy Gradient places a “weight” on each sampled trajectory equal to the total cost associated with that trajectory.

- So we “tilt” our search for the configuration θ that make the states with higher costs **less** likely (remember we go on the direction of the negative gradient!)

Issues of Policy Gradient

- We end the lecture some question regarding the Policy Gradient and the Critic-only Algorithm presented so far.

- Both Algorithm rely heavily on sampled trajectories:



- We need to address the Bias-Variance Trade-off.

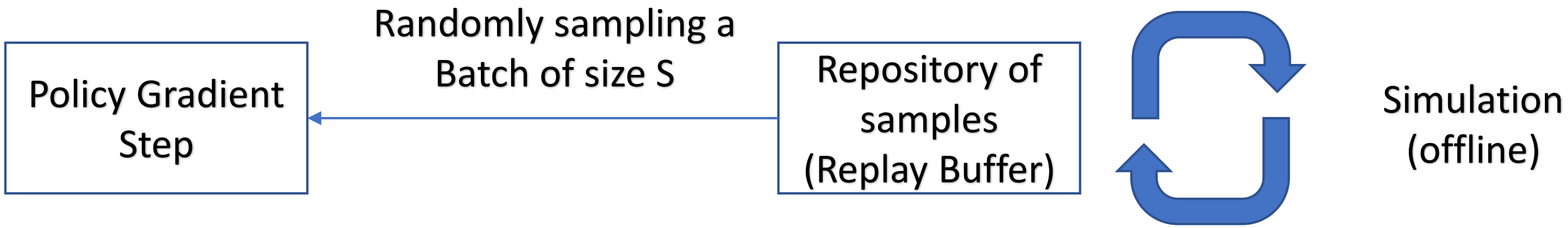
- We will study how to address this issue and how to combine both algorithm into the Actor-Critic Algorithm (which is also a framework).

Issues of Policy Gradient

- In addition the Policy Gradient (and the Critic-only) Algorithms are what is known as *on-policy algorithms:*
  - After every gradient-step we need to collect more samples with the updated policy.

- This fact can be very costly in practical problems, since between training steps we need to perform a lot of sampling.

- Ideally, we would like to do something like we did in the DQN, using a *Replay Buffer:*



- We will study how to do so next time.