

Value Iteration in Infinite Horizon

- Most problems in Reinforcement Learning have a very large horizon:
 - Like a game of Go: the number of turns can be large, but it is eventually finite.
 - Driving from one point to another in the city autonomously
- However, it is often easier to treat the problem as having “infinite horizon”.

- Formally, we will study the following discounted infinite-horizon problem:

$$J_0^*(x_0) = \min_{\pi \in \Pi} \lim_{N \rightarrow \infty} \left\{ \mathbb{E}_{w_{k \geq 0}} \left[\sum_{k=0}^{N-1} \alpha^k g_k(x_k, \mu(x_k), w_k) \right] \right\}$$

- Where $0 < \alpha \leq 1$ is some discounting factor.
- There are many types of infinite-horizon problems, but we will focus on discounted problems.

Value Iteration in Infinite Horizon

- A central cornerstone of Infinite-Horizon DP is the so-called **Value Iteration (VI)**:

$$J^{t+1}(x) = \min_{u \in U(x)} \left\{ \mathbb{E}_w [g(x, u, w) + \alpha J^t(f(x, u, w))] \right\}, \forall t = \{0, 1, 2, \dots\}$$

- Several assumptions are made here:
 - (1) the disturbance vectors follow a *stationary distribution*, i.e.: do not change over time.
 - (2) the state space does not change over time
 - (3) the dynamics are *stationary*, i.e.: do not change over time.
- In summary, we are making the assumption of **stationarity**.
- So the question is: If we keep iterating the above equation again and again and again, does it converge? To what?

Value Iteration in Infinite Horizon

- Ideally we want to claim that:

$$J^*(x) = \lim_{t \rightarrow \infty} J^t(x)$$

- And that the following equation holds for all states x :

$$J^*(x) = \min_{u \in U(x)} \left\{ \mathbb{E}_w [g(x, u, w) + \alpha J^*(f(x, u, w))] \right\}$$

- This is the **Bellman Equation**
- If $\mu(x)$ attains the minimum of the Bellman Equation, then the policy:
$$\pi = \{\mu, \mu, \mu, \dots\}$$
- Is *optimal* and *stationary* => This allows us to only “search” for stationary policies

Value Iteration and MDP

- As we covered, before there is a one-to-one relation between finite-state DP and MDP. We make use of this relation to re-write the Value Iteration in the MDP form:

- $x = \{1, 2, \dots, n\}$: “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 ”.

- Then the VI algorithm becomes:

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

- For all states $i = 1, \dots, n$ and any initial conditions $J_0(1), \dots, J_0(n)$.
- The question is: As $t \rightarrow \infty$, what does the value function converges to?

Value Iteration and MDP

- It follows that VI algorithm converges to the optimal value function, which solves the Bellman Equation:

(Convergence of VI): Given any initial conditions $J^{(0)}(1), \dots, J^{(0)}(n)$, the sequence $\{J^{(t)}(i)\}_{t \geq 0}$ generated by the VI algorithm:

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

Converges to $J^*(i)$ for each state i , where $J^*(i)$ is the finite optimal value function and is the unique solution of the Bellman Equation:

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

In addition, a stationary policy $\pi = \{\mu, \mu, \dots\}$ is optimal if and only if it solves the Bellman Equation

Example: mean-passage time

- Suppose that we wish to know how long to a “goal” state i as fast as possible on average. Then we can write:

$$\alpha = 1 \quad g(i, u, j) = 1, \quad i \neq 0, j, u \in U(i)$$

- Then the Bellman Equation becomes:

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

- Suppose there is a single control u at every state. Then we recover the mean-passage time of the Markov chain (with no control options, the MPD boils down to a Markov Chain):

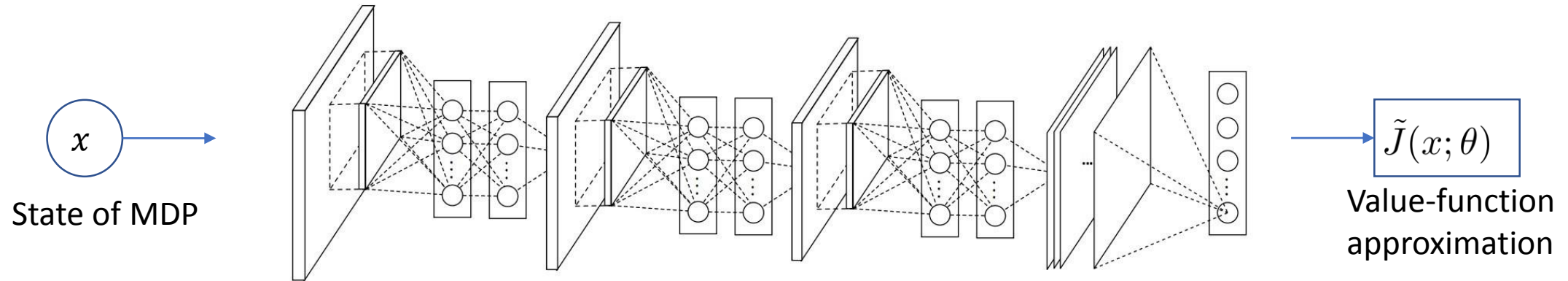
$$J^*(i) = 1 + \sum_{j=1}^n p_{ij} J^*(j), \quad \forall i = \{1, \dots, n\}$$

Fitted Value Iteration

- Now let's return to approximations. Let's state gain the VI algorithm:

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

- Note that at each iteration we need to compute $J(1), \dots, J(n)$.
 - In AlphaGo, the states are the board position: So n is extremely large.
- Hence, we need approximations. Let's say we use some Deep Neural Network to provide the first approximation:



Fitted Value Iteration

- With $\tilde{J}(i; \theta^0)$, suppose we are able to obtain some samples state transitions:

$$(i^0, i^1, \dots, i^S)$$

- Where S is the number of samples. Then for each sample we can apply the VI algorithm:

$$\beta(i^s) = \min_{u \in U(i)} \left\{ \sum_{j=1}^n p_{i^s j}(u) (g(i^s, u, j) + \alpha \tilde{J}^0(j; \theta^0)) \right\}, \forall s = \{1, \dots, S\}$$

- Now we perform the regression problem:

$$\theta^1 \in \arg \min_{\theta} \left\{ \sum_{s=1}^S \left(\tilde{J}(i^s; \theta) - \beta(i^s) \right)^2 \right\}$$

- And we repeat!

Fitted Value Iteration

Algorithm 1 Fitted Value Iteration

Input: Initial DNN parameters θ^0 and architecture $J(i, \theta^{(0)})$.

- 1: **for** $p = 1, \dots, P$: **do** (outer-iterations)
- 2: Collect a batch of S samples (i^1, \dots, i^S) .
- 3: **for** $t = 0, \dots, T$ **do** (VI-iterations)
- 4: **for** $i = i^1, \dots, i^S$: **do** (so, for each sample)
- 5: Perform the VI step:

$$\beta(i^s) = \min_{u \in U(i)} \left\{ \sum_{j=1}^n p_{i^s j}(u) (g(i^s, u, j) + \alpha \tilde{J}(j; \theta^{(t)})) \right\}, \forall s = \{1, \dots, S\}$$

- 6: **end for**
- 7: Solve the Regression:

$$\theta^{(t+1)} \in \arg \min_{\theta} \left\{ \sum_{s=1}^S \left(\tilde{J}(i^s; \theta) - \beta(i^s) \right)^2 \right\}$$

- 8: **end for**
- 9: **end for**

Output: A suboptimal policy is obtained via 1-step lookahead minimization:

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

Fitted Value Iteration

- This algorithm raises more questions than it answers:

- (1) Does this converge?

$$J^*(x) \stackrel{?}{=} \lim_{t \rightarrow \infty} \tilde{J}(x, \theta^{(t)})$$

- (2) What if we cannot perform the summation over all stages?

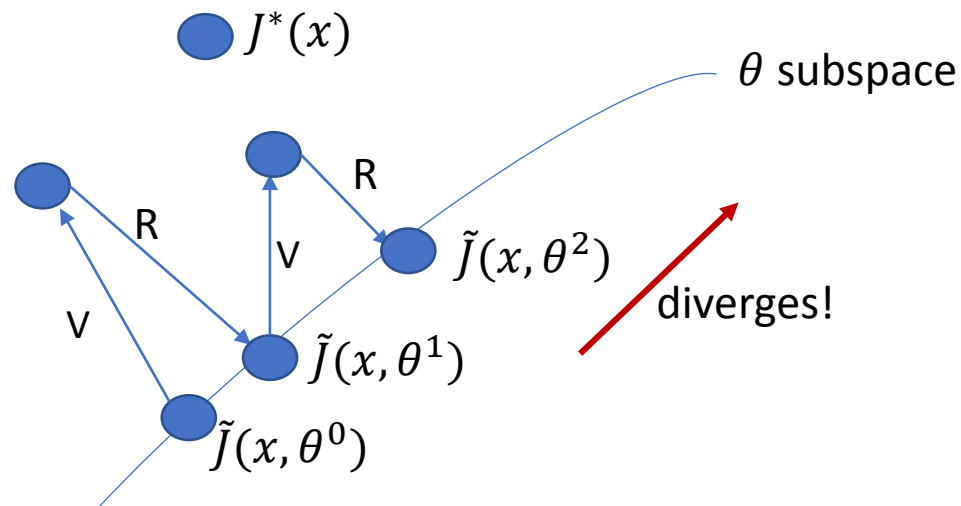
$$\beta(i^s) = \min_{u \in U(i)} \left\{ \sum_{j=1}^n p_{i^s j}(u) (g(i^s, u, j) + \alpha \tilde{J}^t(j; \theta^{(t)})) \right\}, \forall s = \{1, \dots, S\}$$

- (3) How do we actually generate samples?

$$(i^0, i^1, \dots, i^S) \quad ?$$

A brief insight on contractions

- While exact VI converges, the Fitted VI algorithm **does not** converge! Pathological...
- The VI-step (V) is a contraction w.r.t. to the $|| \cdot ||_{\infty}$ norm.
- The Regression-step (R) is a Projection: a contraction w.r.t. to the $|| \cdot ||_2$ norm.
- It turns out that the composition of two contraction mappings w.r.t. to different norms is **not** a contraction.



Value Iteration for Q-factors

- While in theory, it is somewhat hopeless to obtain convergence when combining Value Iteration + Approximation Architectures, all is not lost.

- Let's recall the Bellman Equation:

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

- And recall the Q-factors:

$$Q^*(i, u) = \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J^*(j))$$

- And the equivalent, Bellman Equation, written as a function of the Q-factors:

$$J^*(i) = \min_{u \in U(i)} \left\{ Q^*(i, u) \right\}$$

Value Iteration for Q-factors

- Then we can re-write the Bellman Equation, solely with the Q-factors:

$$Q^*(i, u) = \sum_{j=1}^n p_{ij}(u)(g(i, u, j) + \alpha \min_{v \in U(j)} Q^*(j, v)), \forall i = \{1, \dots, n\}, \forall u \in U(i)$$

- And the exact Value Iteration (which converges) for the Q-factors:

$$Q^{(t+1)}(i, u) = \sum_{j=1}^n p_{ij}(u)(g(i, u, j) + \alpha \min_{v \in U(j)} Q^{(t)}(j, v)), \forall i = \{1, \dots, n\}, \forall u \in U(i)$$

- Starting with some Q-factors $Q^{(0)}(i, u)$ and proceeding over the iterations $t = 1, 2, \dots$
- We will change the Fitted Value Iteration to use the Q-factors.

Fitted Q-iteration

- The first issue we will address is the problem regarding the potential large sums. Namely how can we compute this:

$$Q^{(t+1)}(i, u) = \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha \min_{v \in U(j)} Q^{(t)}(j, v)), \forall i = \{1, \dots, n\}, \forall u \in U(i)$$

- As before, let's use a parametric architecture (say a DNN) to provide the following approximation:

$$\tilde{Q}_{\theta}(i, u) \approx Q^*(i, u), \forall i = \{1, \dots, n\}, \forall u \in U(i)$$

- Where θ are the architecture parameters.
- Note that the r.h.s. is an *expectation*. As we did before, we will replace the expectation by a sample average approximation (SAA).

Fitted Q-iteration

- Let's say we have some *base policy* π that starting from initial state i_0 can generate a sequence of states, actions and costs:

$$(i^0, u^0, i^1, g^0), (i^1, u^1, i^2, g^1), \dots, (i^{S-1}, u^{S-1}, i^S, g^{S-1})$$

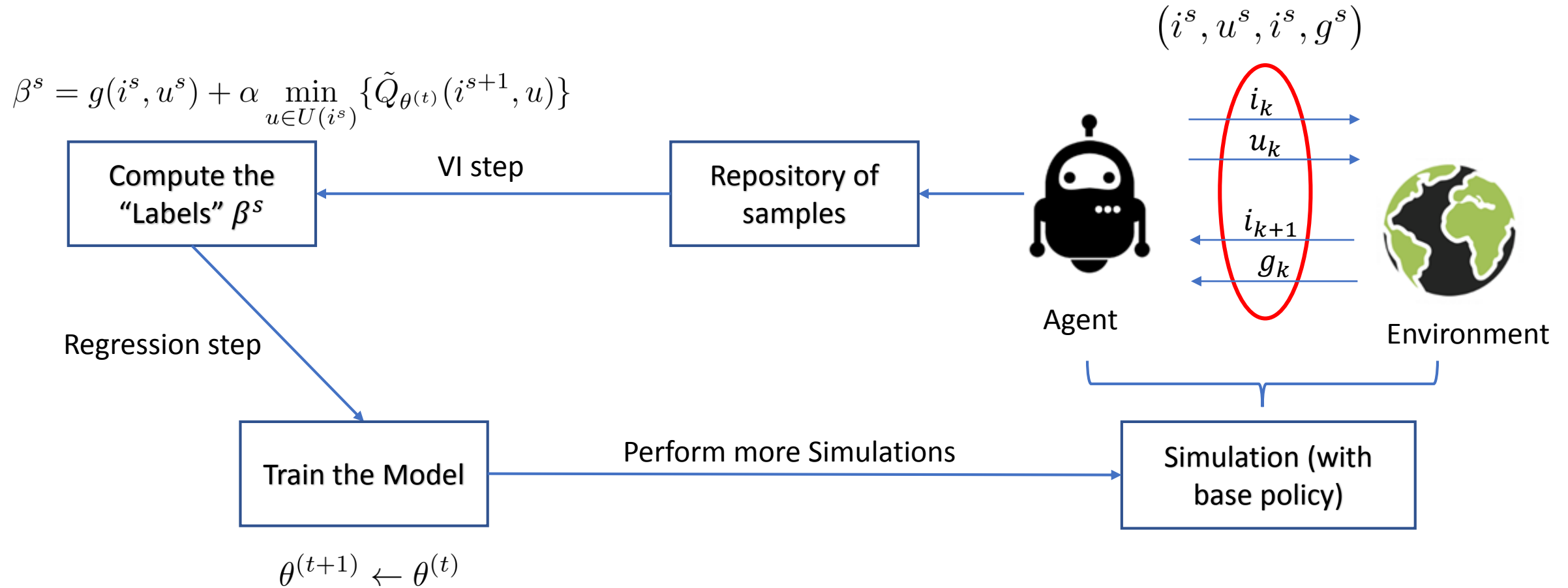
- Note that this policy can be, for example, randomly selecting any available control and storing the transition and the associated cost.
- Now we can compute the following quantities:

$$\beta^s = g(i^s, u^s) + \alpha \min_{u \in U(i^s)} \{\tilde{Q}_\theta(i^{s+1}, u)\}$$

- Key observation:** Note that we do not need to know the dynamics (that is the transition probabilities). We use the DNN **directly** to compute the “labels” β^s .

Fitted Q-iteration

- This is essentially a model-free approach:



Fitted Q-iteration: training

- With the labels:

$$\beta^s = g(i^s, u^s) + \alpha \min_{u \in U(i^s)} \{ \tilde{Q}_{\theta^{(t)}}(i^{s+1}, u) \}$$

- We formulate the regression problem:

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

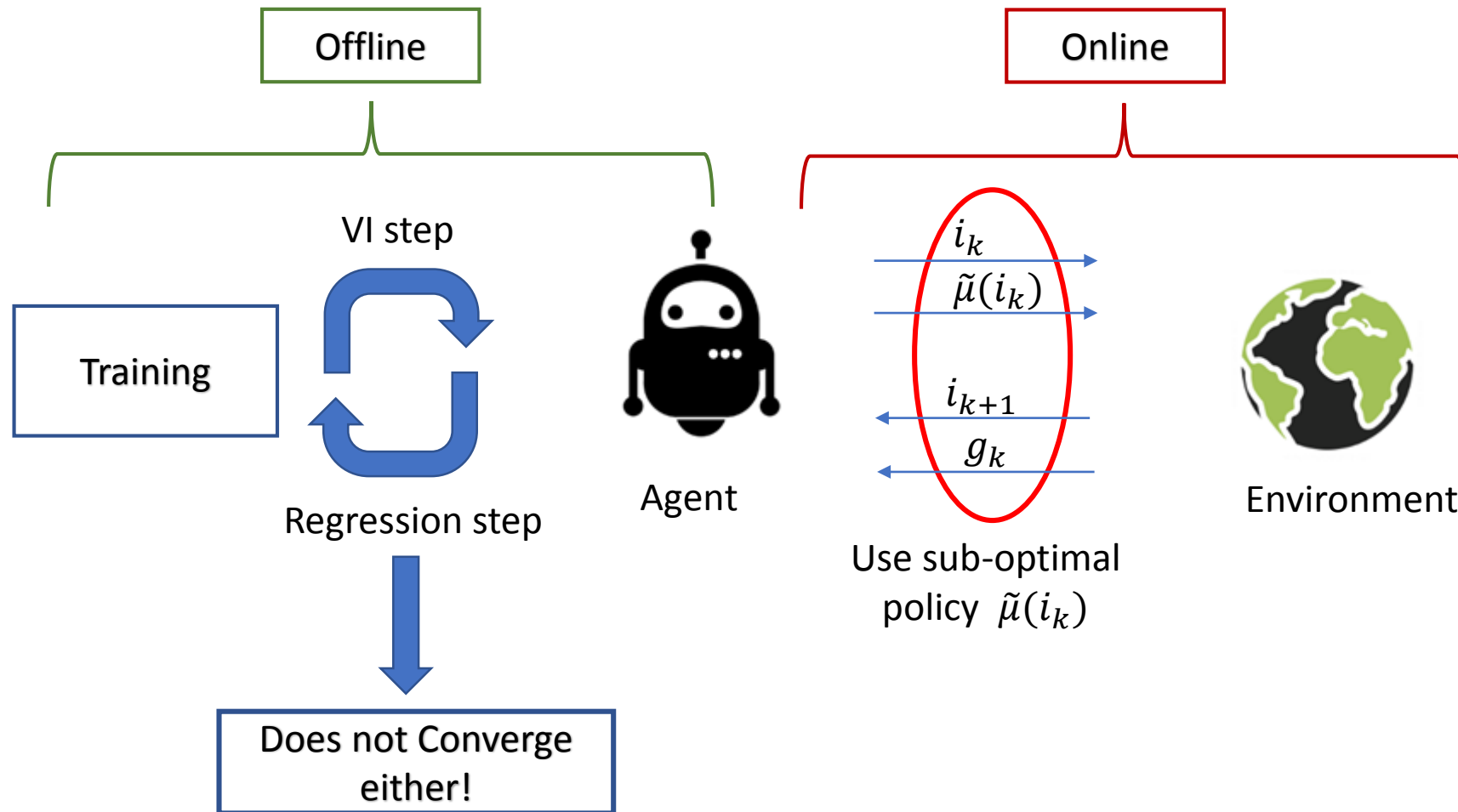
- The above two steps are essentially the VI step and the Regression step. So, the suboptimal policy is readily available as:

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

- This is an offline training method. The only task needed to be performed online is to evaluate the DNN to obtain the Q-factors in order to compute the policy $\tilde{\mu}(i)$.

Fitted Q-iteration

- The fact that we do not need to know the dynamics highlight the usefulness of using Q-factors in comparison to value functions.



Fitted Q Iteration

- Let's recall the three main issues of the Fitted Value Iteration

- (1) Does this converge? **No**

$$J^*(x) \stackrel{?}{=} \lim_{t \rightarrow \infty} \tilde{J}(x, \theta^t)$$



Can be mitigated by
obtaining “good”
samples

- (2) What if we cannot perform the summation over all stages? **Use Q-factors**

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



By using SAA as a
replacement for
expectations

- (3) How do we actually generate samples? That is the big question!

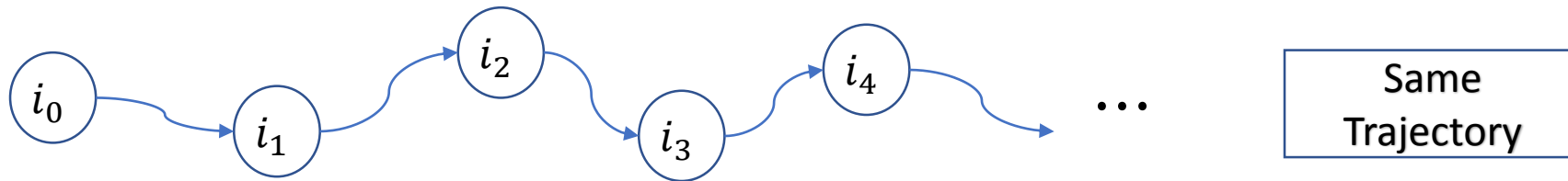
$$(i^0, i^1, \dots, i^S) \quad ?$$


Handling the sampling issue

- First, we started by using some base policy $\bar{\pi}$ to generate the samples:

$$(i^0, u^0, i^1, g^0), (i^1, u^1, i^2, g^1), \dots, (i^{S-1}, u^{S-1}, i^S, g^{S-1})$$

- These samples are correlated, since they belong to the same trajectory:



- There are many ways to try to overcome this issue. For example:
 - Running multiple threads (in parallel)
 - Run multiple threads in asynchronized fashion
 - Using experience replay (replay buffer) 

Experience Replay

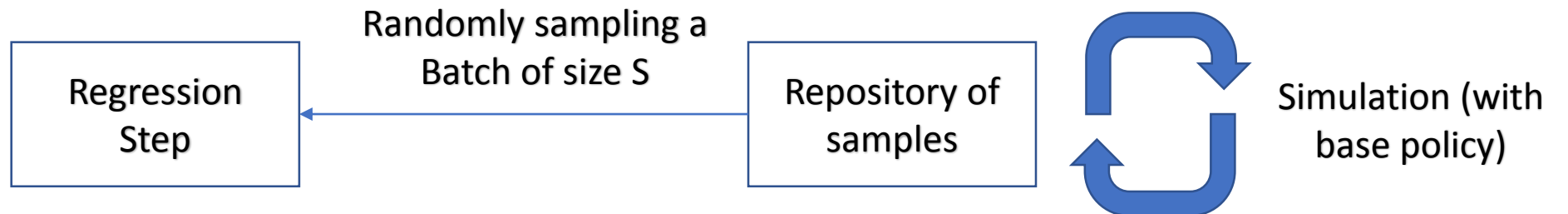
- Note that the regression step:

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

- Does not distinguish between samples. All it matters is that the samples come from some “offline experience”. Namely as long as we collect the tuples of transitions:

$$(i^s, u^s, i^s, g^s)$$

- And store then in the repository, we can then simply sample then uniformly at random to be used in the regression step:



Improving the base policy

- Now we touch the subtle point of the base policy. As we said before, the base policy can be as simple as a policy that picks a control/action at random from the available options.
- This approach can be enough for the first few iterations, however the “quality” of the samples quickly becomes bad, as they are obtained using a “bad” base policy.
- The intuition is that we need to somehow incorporate in the VI-step / Regression-step loop a way to improve the policy that generates our samples.
- This ties directly to policy iteration (which we shall see later). For now we will present one such instance of policy iteration, which leads to the **Q-Learning Algorithm**.

Revisiting the fitted Q-iterations

- Consider again the fitted Q-iterations:

$$\beta^s = g(i^s, u^s) + \alpha \min_{u \in U(i^s)} \{\tilde{Q}_{\theta^{(t)}}(i^{s+1}, u)\}$$

VI Step

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

Regression
Step

- Let's substitute the VI step into the Regression Step, obtaining the following:

$$\theta^{(t+1)} = \arg \min_{\theta} \left\{ \sum_{i=1}^S \left(\tilde{Q}_{\theta}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{\tilde{Q}_{\theta^{(t)}}(i^{s+1}, u)\} \right)^2 \right\}$$

- The idea now, is to instead of solving this problem to (local) optimality, we perform a single gradient steps, and then we proceed to obtain more samples.

Revisiting the fitted Q-iterations

- Taking the gradient, we obtain:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \left\{ \sum_{i=1}^S \nabla_{\theta} \tilde{Q}_{\theta^{(t)}}(i^s, u^s) (\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{\tilde{Q}_{\theta^{(t)}}(i^{s+1}, u)\}) \right\}$$

- There is a very nice intuition associated with this expression. This gradient step is trying to minimize what is called the sum of the Bellman Errors:

$$\sum_{i=1}^S \underbrace{(\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{\tilde{Q}_{\theta^{(t)}}(i^{s+1}, u)\})}_{\text{Bellman Error of sample } s}$$

- If the Bellman Errors are all zero, that would imply that the DNN architecture captures the Q-factors exactly for all the samples considered.

Revisiting the fitted Q-iterations

- After performing the gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \left\{ \sum_{i=1}^S \nabla_{\theta} \tilde{Q}_{\theta^{(t)}}(i^s, u^s) (\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{ \tilde{Q}_{\theta^{(t)}}(i^{s+1}, u) \}) \right\}$$

- We can obtain a “fresh” batch of samples using the suboptimal policy we obtained thus far. That is we generate new:

$$(i^s, u^s, i^s, g^s)$$

- Using the suboptimal-policy:

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

- And add those new samples to the Replay Buffer.

Using the suboptimal policy

- In practice, using *just* the suboptimal may not yield good results:
 - Since we are essentially exploiting (that is, using it over and over again) the policy to generate samples.
 - We need to incorporate some notion of exploration as well.

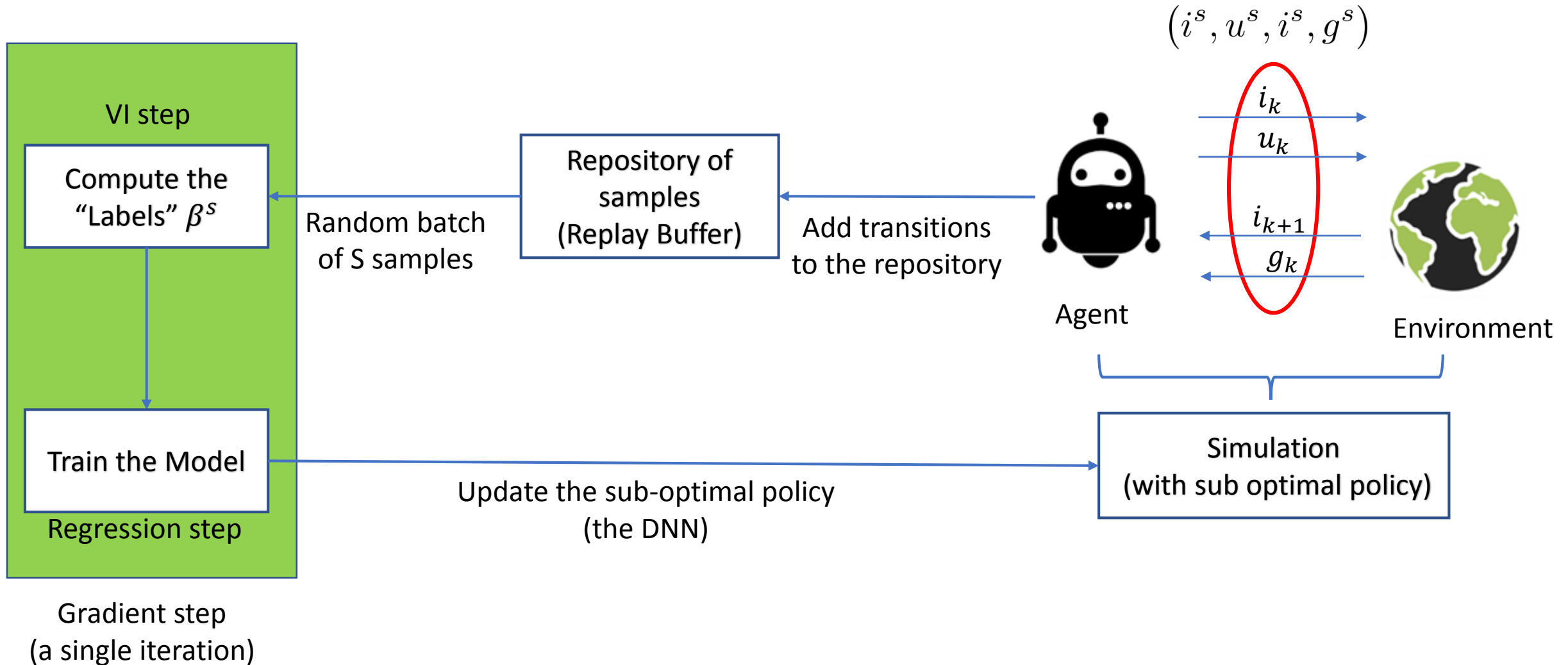
- So a variation of the sub-optimal to be used to obtain new samples can be:

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

- Where $1 > \epsilon^{(t)} > 0$ is some scalar, and $A(U(i))$ is a random variable that picks any element of the set $U(i)$ uniformly at random.
- This is called the ϵ -greedy exploration.

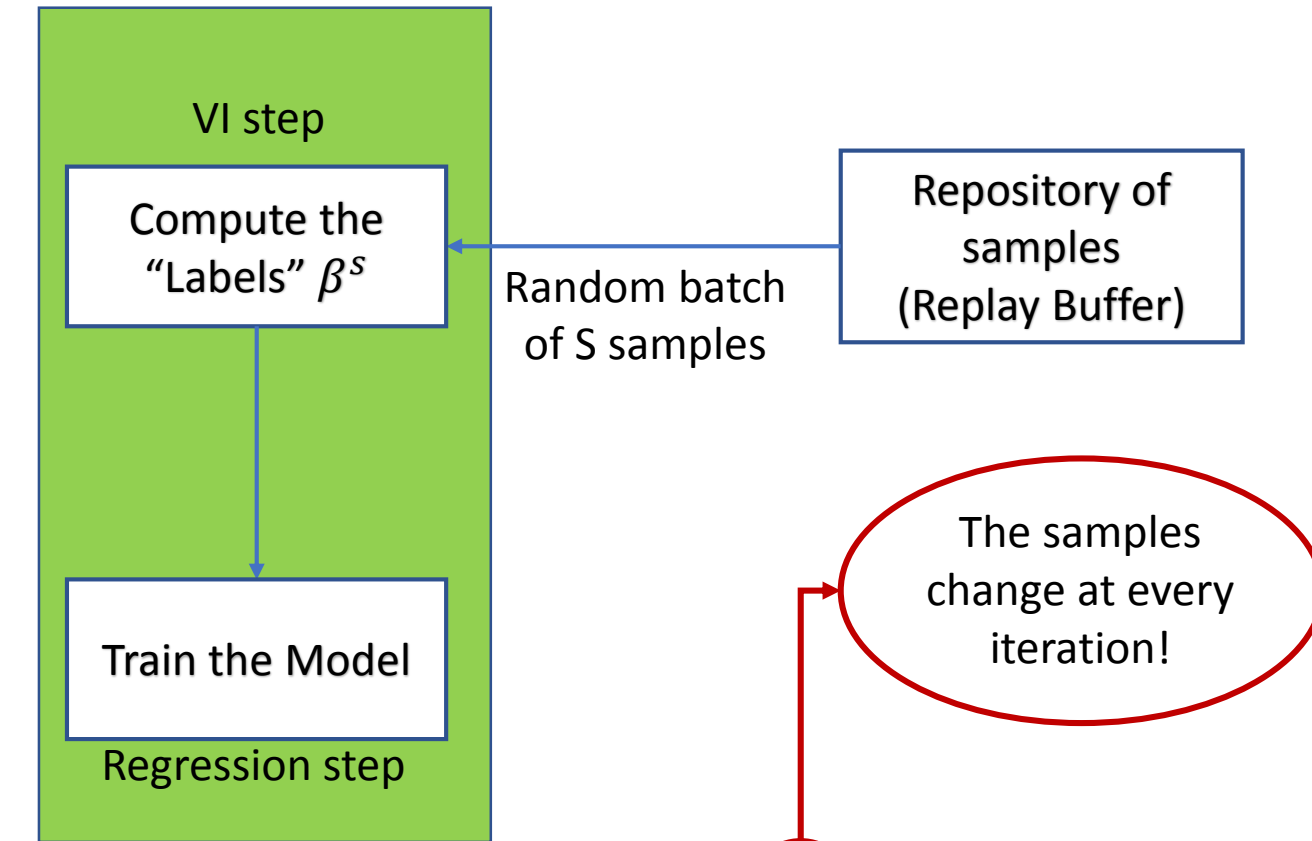
Deep Q-Learning Algorithm

- So we can summarize the Deep Q-learning algorithm by the following diagram:



Deep Q-Learning Algorithm

- There is approach, leads to a subtle issue:



- At every loop, the “labels” (or targets) are changing!

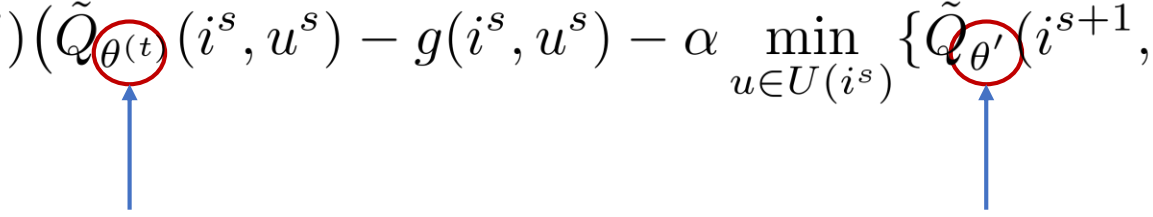
Gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \left\{ \sum_{i=1}^S \nabla_{\theta} \tilde{Q}_{\theta^{(t)}}(i^s, u^s) \left(\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - \overbrace{g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{ \tilde{Q}_{\theta^{(t)}}(i^{s+1}, u) \}}^{\text{Labels } \beta^s} \right) \right\}$$

Deep Q Networks Algorithm (DQN)

- We are almost there: The last step tries to solve the fact that the “target” of the gradient steps change every time we obtain new samples.
- The idea is to “freeze” a DNN configuration θ' and use that to compute the targets. This would help the Bellman Error to decrease for “more” samples than before.
- So the gradient step in the DQN algorithm becomes:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \left\{ \sum_{i=1}^S \nabla_{\theta} \tilde{Q}_{\theta^{(t)}}(i^s, u^s) \left(\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{ \tilde{Q}_{\theta'}(i^{s+1}, u) \} \right) \right\}$$



Current DNN parameters

“frozen” previous DNN parameters

Deep Q Networks Algorithm (DQN)

Algorithm 1 DQN Algorithm (Minh et al, 2015)

Input: Initial DNN parameters θ^0 and architecture $Q_{\theta^0}(i, u)$. Replay Buffer \mathcal{B} .

- 1: **for** $p = 1, \dots, P$: **do** (updates on the target network θ')
- 2: Save the network parameters $\theta' \leftarrow \theta$.
- 3: **for** $k = 0, \dots, K$ **do** (obtaining new samples)
- 4: Collect M sample transitions $\{(i^m, u^m, i^{m+1}, g^m)\}_{m=1}^M$ using the sub-optimal policy:

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

and add those sample transitions to the Replay Buffer \mathcal{B} .

- 5: **for** $t = 1, \dots, T$: **do** (gradient step)
- 6: Randomly sample a batch of size S from the Replay Buffer \mathcal{B} .
- 7: Perform one gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \left\{ \sum_{i=1}^S \nabla_{\theta} \tilde{Q}_{\theta^{(t)}}(i^s, u^s) (\tilde{Q}_{\theta^{(t)}}(i^s, u^s) - g(i^s, u^s) - \alpha \min_{u \in U(i^s)} \{\tilde{Q}_{\theta'}(i^{s+1}, u)\}) \right\}$$

- 8: **end for**
- 9: **end for**
- 10: **end for**

Output: The last DNN configuration $\bar{\theta}$. A suboptimal policy:

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