

Let

- $t \in \mathbb{N}$ be the current time step,
- $\mathbf{x}_t \in \mathbb{R}^d$ be the input at time t ,
- $a_t \in A$ be the action taken at time t based on input \mathbf{x}_t , and
- $r_t \in \mathbb{R}$ be the reward received at time t based on action a_t .

Then, we'll define

$$s'_t(x) = [x_1 \quad a_1 \quad x_2 \quad a_2 \quad \dots \quad x_{t-1} \quad a_{t-1} \quad \mathbf{x}_t \quad a_t \quad x]$$

and once x_{t+1} is observed, we'll write

$$\mathbf{s}_{t+1} = s'_t(x_{t+1})$$

then, our goal is to use \mathbf{s}_{t+1} to predict the best a_{t+1} .

First, we'll define what we mean by best. Let $T \in \mathbb{N}$ be the total number of time steps, and γ some discount rate on future rewards. Then, we'll define the future discounted return as

$$R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$$

which is what we'll be trying to optimize for. To do this, we want to find a function $Q(\mathbf{s}_t, a_t)$ which will tell us the future discounted return for taking action a_t given history \mathbf{s}_t . Specifically, we'll define the optimal Q^* , which we'll be trying to approximate, as

$$Q^*(\mathbf{s}_t, a_t) = \max_{\pi} \mathbb{E}_{\mathbf{s}_t, a_t, \pi} [R_t]$$

where $\pi : \{s \mid \forall s\} \rightarrow A$ is some policy for selecting actions based on states. Then, given Q^* , we can find the optimal action a_t^* as

$$a_t^* = \operatorname{argmax}_{a'} (r_t + \gamma Q^*(\mathbf{s}_t, a'))$$

therefore, if we can find an approximation for Q^* , we can also find an approximation for a_t^* , which is our actual goal. To do this, we want to find an iterative update rule for Q . Thus, we can rewrite the above formula for Q^* recursively as

$$Q^*(\mathbf{s}_t, a_t) = \mathbb{E}_{\mathbf{x}_{t+1}} \left[r_t + \gamma \max_{a'} Q^*(\mathbf{s}_{t+1}, a') \right]$$

which gives us

$$Q_{i+1}(\mathbf{s}_t, a_t) = \mathbb{E}_{\mathbf{x}_{t+1}} \left[r_t + \gamma \max_{a'} Q_i(\mathbf{s}_{t+1}, a') \right]$$

which is the standard Q -learning update rule.

Next, we want to adopt this formulation to allow us to approximate Q with a neural network. Thus, for weights $\boldsymbol{\theta}$ and feature map ϕ , we'll let

$$Q(\phi(\mathbf{s}_t), a_{t+1} | \boldsymbol{\theta})$$

be our new Q . Then, we'll define the target value at iteration i as

$$y_i(\mathbf{s}_t, a_t) = \mathbb{E}_{\mathbf{x}_{t+1}} \left[r_t + \gamma \max_{a'} Q(\phi(\mathbf{s}_{t+1}), a' | \boldsymbol{\theta}_i) \right]$$

which gives us the squared-loss loss function

$$L_i(\boldsymbol{\theta}_i) = \frac{1}{2} \mathbb{E}_{\mathbf{s}_t, a_t} [y_i(\mathbf{s}_t, a_t) - Q(\phi(\mathbf{s}_t), a_t | \boldsymbol{\theta}_i)]^2$$

which says that, for all states and actions, we want Q to be as close to the estimated future discounted reward as possible. Interestingly, note that we are using Q both as our predicted value and in constructing our target.

Finally, using our loss function, we can construct an update rule for $\boldsymbol{\theta}$ which will actually let us train our deep Q -learning neural net. Taking the gradient of L_i , we get

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_i} L_i(\boldsymbol{\theta}_i) = & - \mathbb{E}_{\mathbf{s}_t, a_t, \mathbf{x}_{t+1}} \left[\left(r_t + \gamma \max_{a'} Q(\phi(\mathbf{s}_{t+1}), a' | \boldsymbol{\theta}_i) - Q(\phi(\mathbf{s}_t), a_t | \boldsymbol{\theta}_i) \right) \nabla_{\boldsymbol{\theta}_i} Q(\phi(\mathbf{s}_t), a_t | \boldsymbol{\theta}_i) \right] \end{aligned}$$

which gives the gradient descent update rule for $\boldsymbol{\theta}_i$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta}_i} L_i(\boldsymbol{\theta}_i)$$

where η is the learning rate, which we can use to train our network.