Reinforcement Learning

Chapter 4: Function Approximation

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Back to Model-free RL

We now get back to model-free RL: *this time however*

we use a parameterized estimator

This means that

- *to estimate the value function, we use*

$$\hat{v}_{\mathbf{w}}(s)$$

- *to estimate the action-value function, we use*

$$\hat{q}_{\mathbf{w}}(s, a)$$

↳ *These functions are parameterized by some* $\mathbf{w}$

↳ *They could be as simple as linear approximators or advanced like DNNs*

Back to Model-free RL

+ *But, we have in general prediction and control problems. In which one are we going to use function approximation?*
– Well, we can use in both

Recall

We have two major problems in model-free RL

- *Prediction in which for a given policy π we evaluate values by sampling the environment*
- *Control in which after each interaction, we improve our policy aiming to converge to the optimal policy*

Let's start with the prediction

Basic Prediction via Approximator

Say we want to evaluate the value function of a policy π: *with function approximation we assume that the value approximator is*

$$s \longrightarrow \boxed{v_{\mathbf{w}}(\cdot)} \longrightarrow \hat{v}_{\pi}(s)$$

- *We sample the environment by policy π*
  - ↳ *We draw sample trajectories using policy π*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

- *We want to use these samples to train our approximation model*
  - ↳ *We want to find $\mathbf{w}$ that approximates the best $v_{\pi}(\cdot)$*

Building Simple Approximator

+ *Well you have talked a lot about approximators, but can you give us a concrete example?!*
– Sure! Let's look at *linear approximator: say our approximator is a linear function of the input*
+ *Does it mean the following diagram?!*

$$s \longrightarrow \boxed{v_w(\cdot)} \longrightarrow ws$$

+ *But it doesn't seem to work with only one parameter!*
– Oops! We haven't yet defined the *feature representation of states!*

Feature Representation

In general, we could represent a particular state in *various forms*

Say we have N states $s^1, \dots, s^N$: we can represent them by a scalar, e.g.

$$s^n \mapsto \mathrm{x}(s^n) = n$$

Or a one-sparse vector, i.e.,

$$s^n \mapsto \mathbf{x}(s^n) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{entry } n$$

Feature Representation

Feature Representation of States

Feature representation maps each state into a vector of features that corresponds to that state, i.e.,

$$\mathbf{x}(\cdot) : \mathbb{S} \mapsto \mathbb{R}^J$$

for some integer J that is the feature dimension

Example: *We can represents the feature by tokenization*

$$s^n \mapsto \mathbf{x}(s^n) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{entry } n$$

Feature Representation

Feature Representation of States

Feature representation maps each state into a vector of features that corresponds to that state, i.e.,

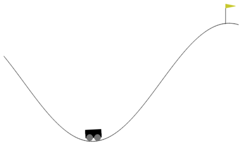
$$\mathbf{x}(\cdot) : \mathbb{S} \mapsto \mathbb{R}^J$$

for some integer J that is the feature dimension

In practice, we use more advanced problem-specific features

- *The state of a robot can be specified by its geometric features*
  - ↳ *its distance to the edges of the room, its direction, etc*
- *The state of a computer game is completely explained by its frames*
  - ↳ *we collect all frames from a state to another*

Example: *Mountain Car*



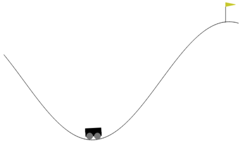
Let's consider the famous mountain car example

- *A car is stuck in a valley*
  - ↳ *it can accelerate to left, accelerate to right, or do nothing*
- *It observes its velocity v and location x on the horizontal axis*
  - ↳ *they are continuous variables*

The goal is to train the car to get out of this valley as quick as possible

- *The car is rewarded by -1 after each unsuccessful trial*

Example: *Mountain Car*



This car can be in infinite number of states; however, we can represent its state by its velocity and location

$$\mathbf{x}(s) = \begin{bmatrix} v \\ x \end{bmatrix}$$

This is much more feasible to work with!

Back to Simple Approximator

+ *How is this feature related to our discussion on function approximation?!*
  – Well! Approximation model *maps the features to value*

So, in our problem, we in fact assume that

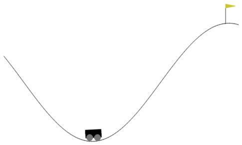
$$\mathbf{x}(s) \longrightarrow \boxed{v_{\mathbf{w}}(\cdot)} \longrightarrow \hat{v}_{\pi}(s)$$

Example: *Linear Approximator*

Linear approximation model maps the state features to its value as

$$v_{\mathbf{w}}(s) = \mathbf{w}^{\mathsf{T}} \mathbf{x}(s)$$

Example: *Mountain Car*



In mountain car example, we could have a two-dimensional linear approximator

$$v_{\mathbf{w}}(s) = \mathbf{w}^{\mathsf{T}} \mathbf{x}(s) = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v \\ x \end{bmatrix}$$

$$= w_1 v + w_2 x$$

Back to Simple Approximator

+ *How is this feature related to our discussion on function approximation?!*
– Well! Approximation model *maps the features to value*

So, in our problem, we in fact assume that

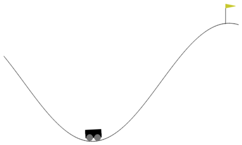
$$\mathbf{x}\left(s\right) \longrightarrow \boxed{v_{\mathbf{w}}\left(\cdot\right)} \longrightarrow \hat{v}_{\pi}\left(s\right)$$

Example: *Deep Approximator*

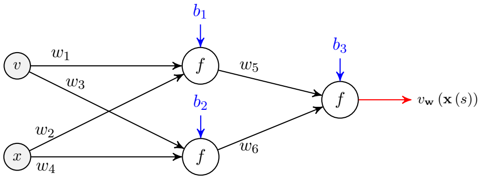
Deep approximation model maps the state features to its value via a DNN

$$v_{\mathbf{w}}\left(s\right) = \mathrm{DNN}\left(\mathbf{x}\left(s\right) | \mathbf{w}\right)$$

Example: *Mountain Car*



We may also use a NN to map the feature to its value



Here, the weights are $\mathbf{w} = \left[w_1, \dots, w_6, b_1, b_2, b_3\right]^\mathsf{T}$

Training Approximation Model

+ *How can we train a given approximation model?*
– Let's again get help from a genie

Assume that a genie could tell us the exact value of all states: in this case we want to find $\mathbf{w}$ such that for each s^n

$$v_{\mathbf{w}}\left(s^n\right) \overset{!}{=} v_\pi\left(s^n\right) \rightsquigarrow \left|v_{\mathbf{w}}\left(s^n\right) - v_\pi\left(s^n\right)\right| \overset{!}{=} 0$$

This is equivalent to say that

$$\frac{1}{N}\sum_{n=1}^{N}\left|v_{\mathbf{w}}\left(s^n\right) - v_\pi\left(s^n\right)\right|^2 \overset{!}{=} 0$$

Training Approximation Model

But, we cannot necessarily find such $\mathbf{w}$*: we could instead find*

$$\mathbf{w}^\star = \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \left| v_{\mathbf{w}}\left(s^n\right) - v_\pi\left(s^n\right)\right|^2$$

One may also think about a more general weighted average: we can assume that under policy π *each state* s^n *happens with a probability* $p_\pi\left(s^n\right)$ *and write*

$$\begin{aligned}\mathbf{w}^\star &= \min_{\mathbf{w}} \sum_{n=1}^{N} p_\pi\left(s^n\right) \left| v_{\mathbf{w}}\left(s^n\right) - v_\pi\left(s^n\right)\right|^2 \\ &= \min_{\mathbf{w}} \mathbb{E}_\pi \left\{ \left| v_{\mathbf{w}}\left(S\right) - v_\pi\left(S\right)\right|^2 \right\}\end{aligned}$$

We train by minimizing residual sum of squares $\equiv$ *least-squares (LS) method*

LS Training: *Gradient Descent*

We use gradient descent to solve this problem: *we are minimizing the risk*

$$\mathcal{L}\left(\mathbf{w}\right) = \mathbb{E}_\pi \left\{ \left| v_{\mathbf{w}}\left(S\right) - v_\pi\left(S\right) \right|^2 \right\}$$

```
GradientDescent():
```

1: *Initiate with some initial* $\mathbf{w}$ *and learning rate* η
2: ***while*** *weights not converged* ***do***
3: *Compute gradient* $\nabla \mathcal{L}\left(\mathbf{w}\right)$
4: *Update weights as* $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathcal{L}\left(\mathbf{w}^{t-1}\right)$
5: ***end while***

Let's compute the gradient: we can use the chain rule

$$\begin{aligned}\nabla \mathcal{L}\left(\mathbf{w}\right) &= \frac{\partial \mathcal{L}}{\partial v_{\mathbf{w}}\left(S\right)} \nabla v_{\mathbf{w}}\left(S\right) \\ &= 2\mathbb{E}_\pi \left\{ \left(v_{\mathbf{w}}\left(S\right) - v_\pi\left(S\right)\right) \nabla v_{\mathbf{w}}\left(S\right) \right\}\end{aligned}$$

LS Training: *Gradient Descent*

Now we set the learning rate to $\eta = 0.5\alpha$ for some α; then, we have

$$\begin{aligned}\mathbf{w}^{(t)} &\leftarrow \mathbf{w}^{(t-1)} - \alpha \mathbb{E}_\pi \left\{ \left(v_{\mathbf{w}}(S) - v_\pi(S) \right) \nabla v_{\mathbf{w}}(S) \right\} \\ &\leftarrow \mathbf{w}^{(t-1)} + \alpha \mathbb{E}_\pi \left\{ \left(v_\pi(S) - v_{\mathbf{w}}(S) \right) \nabla v_{\mathbf{w}}(S) \right\}\end{aligned}$$

So, we the gradient descent based evaluation reduces to

`GD_Eval():`

1: *Initiate with some initial $\mathbf{w}^{(0)}$ and learning rate η*
2: ***while*** *weights not converged* ***do***
3: *Update weights as $\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}_\pi \left\{ \left(v_\pi(S) - v_{\mathbf{w}}(S) \right) \nabla v_{\mathbf{w}}(S) \right\}$*
4: ***end while***

+ *That's nice! But, how in earth we know $v_\pi(S)$?!*
– Well! We may use what we learned in model-free RL again!

LS Training via Monte Carlo

The key challenge is to find out *an estimator for*

$$\mathbb{E}_\pi \left\{ \left(v_\pi \left(S \right) - v_{\mathbf{w}} \left(S \right) \right) \nabla v_{\mathbf{w}} \left(S \right) \right\}$$

We can do it by Monte Carlo: *say we have an episodic environment with terminal state and sampled an episode*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We can convert this trajectory into

$$\left(S_0, G_0 \right) \to \left(S_1, G_1 \right) \to \ldots \to \left(S_{T-1}, G_{T-1} \right)$$

with G_t being the sample return, i.e.,

$$G_t = \sum_{i=0}^{T-t-1} \gamma^i R_{t+1+i}$$

LS Training via Monte Carlo

The key challenge is to find out *an estimator for*

$$\mathbb{E}_\pi \left\{ \left(v_\pi \left(S \right) - v_{\mathbf{w}} \left(S \right) \right) \nabla v_{\mathbf{w}} \left(S \right) \right\}$$

We do know that G_t is an estimator of $v_\pi \left(S_t \right)$: *so we could say*

$$\mathbb{E}_\pi \left\{ \left(v_\pi \left(S \right) - v_{\mathbf{w}} \left(S \right) \right) \nabla v_{\mathbf{w}} \left(S \right) \right\} \approx \frac{1}{T} \sum_{t=0}^{T-1} \left(G_t - v_{\mathbf{w}} \left(S_t \right) \right) \nabla v_{\mathbf{w}} \left(S_t \right)$$

We can compute this one from our observations!

LS Training via Monte Carlo

We can then write this approximate gradient descent approach as

GD_MC_Eval():

1: *Initiate with some initial* $\mathbf{w}$ *and learning rate* α
2: ***while*** *weights not converged* ***do***
3: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: *Compute the sequence*

$$(S_0, G_0) \to (S_1, G_1) \to \ldots \to (S_{T-1}, G_{T-1})$$

5: *Update weights as*

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{T} \sum_{t=0}^{T-1} \left(G_t - v_{\mathbf{w}}(S_t) \right) \nabla v_{\mathbf{w}}(S_t)$$

6: ***end while***

LS Training via Monte Carlo

This is not practical to wait for exact convergence: we try couple of episodes

`GD_MC_Eval():`

1: *Initiate with some initial* $\mathbf{w}$ *and learning rate* α
2: ***for*** *episode* $k = 1 : K$ ***do***
3: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: *Compute the sequence*

$$(S_0, G_0) \to (S_1, G_1) \to \ldots \to (S_{T-1}, G_{T-1})$$

5: *Update weights as*

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{T} \sum_{t=0}^{T-1} \left(G_t - v_{\mathbf{w}} \left(S_t \right) \right) \nabla v_{\mathbf{w}} \left(S_t \right)$$

6: ***end for***

LS Training via Monte Carlo

+ *But, should we really update once an episode?!*
- Not really!

We can look at this algorithm as batch training with the batch being

$$(S_0, G_0), (S_1, G_1), \ldots, (S_{T-1}, G_{T-1})$$

We can also use mini-batches, and we can reduce the mini-batch size to 1: in this case, the estimator of the gradient will be a single sample, i.e.,

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(G_t - v_{\mathbf{w}} \left(S_t \right) \right) \nabla v_{\mathbf{w}} \left(S_t \right)$$

LS Training via Monte Carlo

We can train the approximator via SGD and Monte Carlo

`SGD_MC_Eval():`

1: *Initiate with some initial* $\mathbf{w}$ *and learning rate* α

2: **for** episode $k = 1 : K$ **do**

3: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4: *Compute the sequence*

$$(S_0, G_0) \to (S_1, G_1) \to \ldots \to (S_{T-1}, G_{T-1})$$

5: **for** $t = 0 : T-1$ **do**

6: *Update weights as*

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(G_t - v_{\mathbf{w}} \left(S_t \right) \right) \nabla v_{\mathbf{w}} \left(S_t \right)$$

7: **end for**

8: **end for**

LS Training via Temporal Difference

+ *So can't we use also TD to acquire an estimate?*
– Sure!

We can evaluate an estimator by TD: say we sample

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We can then compute an estimator of $v_\pi(S_t)$ *via TD as*

$$G_t^0 = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$$

LS Training via Temporal Difference

We can alternatively train the approximator via temporal difference

`SGD_TD_Eval():`

1: *Initiate with some initial* $\mathbf{w}$ *and learning rate* α
2: **for** episode $k = 1 : K$ **do**
3:     *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

4:     **for** $t = 0 : T-1$ **do**
5:         *Update weights as*

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R_{t+1} + \gamma v_{\mathbf{w}}\left(S_{t+1}\right) - v_{\mathbf{w}}\left(S_t\right)\right) \nabla v_{\mathbf{w}}\left(S_t\right)$$

6:     **end for**
7: **end for**

You can extend it to TD-n or TD$_\lambda$ as well

Back to Tabular RL

+ *These algorithm look like what we had before!*
– Well! This is actually an extended version of it!

Let's consider a special case in which

1. *We use tokenization for feature representation*

Tokenization

We represent the feature vector as

$$\mathbf{x}\left(s^n\right) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{entry } n = \mathbf{1}\left\{s = s^n\right\}$$

Back to Tabular RL

Let's consider a special case in which

1. *We use tokenization for feature representation*
2. *We use a linear model for approximation*

Linear Model

We approximate the value function via a linear transform

$$v_{\mathbf{w}}(s) = \mathbf{w}^{\mathsf{T}} \mathbf{x}(s)$$

With linear model and tokenization, our estimate of value at state s^n is

$$\hat{v}_\pi(s^n) = v_{\mathbf{w}}(s^n) = \mathbf{w}^{\mathsf{T}} \mathbf{1}\{s = s^n\} = w_n$$

Also we have

$$\nabla v_{\mathbf{w}}(s^n) = \mathbf{x}(s^n) = \mathbf{1}\{s = s^n\}$$

Back to Tabular RL

Let's consider a special case in which

1. *We use tokenization for feature representation*
2. *We use a linear model for approximation*

Let's now look at the SGD update with TD

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \alpha \left(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t) \right) \nabla v_{\mathbf{w}}(S_t) \\ &\leftarrow \mathbf{w} + \alpha \left(R_{t+1} + \gamma \underbrace{v_{\mathbf{w}}(S_{t+1})}_{\hat{v}_\pi(S_{t+1})} - \underbrace{v_{\mathbf{w}}(S_t)}_{\hat{v}_\pi(S_t)} \right) \underbrace{\mathbf{x}(S_t)}_{\mathbf{1}\{s=S_t\}}\end{aligned}$$

Say $S_t = s^n$: then, it is only entry n which gets update

$$w_n \leftarrow w_n + \alpha \left(R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}) - \hat{v}_\pi(S_t) \right)$$

Back to Tabular RL

Let's consider a special case in which

1. *We use tokenization for feature representation*
2. *We use a linear model for approximation*

Say $S_t = s^n$: then, it is only entry n which gets updated

$$w_n \leftarrow w_n + \alpha \left(R_{t+1} + \gamma \hat{v}_\pi \left(S_{t+1} \right) - \hat{v}_\pi \left(S_t \right) \right)$$

and we know that $w_n = \hat{v}_\pi \left(s^n \right) = \hat{v}_\pi \left(S_t \right)$, so we can write

$$\hat{v}_\pi \left(S_t \right) \leftarrow \hat{v}_\pi \left(S_t \right) + \alpha \left(R_{t+1} + \gamma \hat{v}_\pi \left(S_{t+1} \right) - \hat{v}_\pi \left(S_t \right) \right)$$

Bingo! This is the tabular TD update!

Back to Tabular RL

Moral of Story

Tabular RL is a special case of RL with function approximation when

1. *We use tokenization for feature representation*
2. *We use a linear model for approximation*

So, we expect learning better when we go for other feature representation and approximation models

Training Action-Value Approximator

+ *In practice however we always need action-values! Right?*
– Well! We can simply extend everything to state-action pairs

Feature Representation of State-Actions

Feature representation maps each state-action pair into a vector of features that correspond to that state and action, i.e.,

$$\mathbf{x}\left(\cdot\right) : \mathbb{S} \times \mathbb{A} \mapsto \mathbb{R}^J$$

for some integer J that is the feature dimension

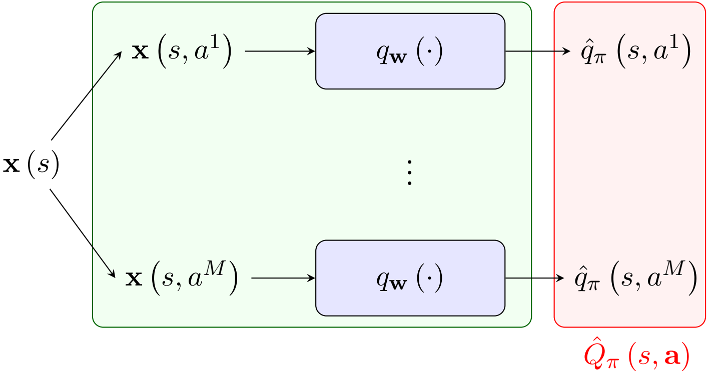
Action-Value Approximator: *Form I*

We can further consider an approximation model: *it maps the feature vector of each state-action pair* (s, a) *into its action-value*

$$\mathbf{x}(s, a) \longrightarrow \boxed{q_{\mathbf{w}}(\cdot)} \longrightarrow \hat{q}_{\pi}(s, a)$$

Action-Value Approximator: *Form II*

We may stack this approximator for various actions: *we can look at the end-to-end setting as a general approximator*



Action-Value Approximator: *Form II*

We may stack this approximator for various actions: *maybe, we can consider a general approximation model in this case*

$$\mathbf{x}(s) \longrightarrow \boxed{Q_{\mathbf{w}}(\cdot)} \longrightarrow \hat{Q}_\pi(s, \mathbf{a})$$

Let's make an agreement

> $\hat{Q}_\pi(s, \mathbf{a})$ *and* $Q_{\mathbf{w}}(s, \mathbf{a})$ *represent the complete vector of action-values and* $\hat{Q}_\pi(s, a)$ *and* $Q_{\mathbf{w}}(s, a)$ *denote the entry corresponding to* a

Training Approximation Model

The LS training can further be applied here: *with the help of genie, we train the action-value approximator as*

$$\mathbf{w}^{\star} = \min_{\mathbf{w}} \mathbb{E}_{\pi} \left\{ \left| Q_{\mathbf{w}} \left(S, A \right) - Q_{\pi} \left(S, A \right) \right|^2 \right\}$$

which is iteratively solved via gradient descent using update rule

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}_{\pi} \left\{ \left(Q_{\pi} \left(S, A \right) - Q_{\mathbf{w}} \left(S, A \right) \right) \nabla Q_{\mathbf{w}} \left(S, A \right) \right\}$$

Again, we could use Monte Carlo or TD to find an estimator of $Q_{\pi} \left(S, A \right)$

LS Training of Action-Value Approximator

Say we have sampled a trajectory

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

We can convert this trajectory into a sequence of (S_t, A_t, G_t)

- *Using Monte Carlo we can update by*

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(G_t - Q_{\mathbf{w}} \left(S_t, A_t \right) \right) \nabla Q_{\mathbf{w}} \left(S_t, A_t \right)$$

- *Using TD we can update as*

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right) - Q_{\mathbf{w}} \left(S_t, A_t \right) \right) \nabla Q_{\mathbf{w}} \left(S_t, A_t \right)$$

  and we can find $v_{\mathbf{w}} \left(S_{t+1} \right)$ *as*

$$v_{\mathbf{w}} \left(s \right) = \sum_{m=1}^{M} \pi \left(a^m | s \right) Q_{\mathbf{w}} \left(s, a^m \right)$$

LS Training of Action-Value Approximator

We can use all approaches we developed before

- *only difference is that we replace the simple update by gradient descent*
- *it gets back to tabular RL if we use tokenization and linear approximation*

Let's see a simple example: remember the backward view of TD$_\lambda$

- *we trace the eligibility of each state-action*
- *we propagate the update to previous state-action pairs*

```
ElgTrace(S_t, A_t, E(·) |λ):
```

Eligibility tracing function has NM components, i.e., $E(s, a)$ for all state-action pairs
for all state-action pairs (s, a) **do**
  Update $E(s, a) \leftarrow \gamma\lambda E(s, a)$
end for
Update $E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$

LS Training of Action-Value Approximator

ElgTrace($S_t, A_t, E(\cdot) \mid \lambda$):

1: *Eligibility tracing function has NM components, i.e., $E(s, a)$ for all state-action pairs*
2: **for** *all state-action pairs* (s, a) **do**
3: *Update* $E(s, a) \leftarrow \gamma\lambda E(s, a)$
4: **end for**
5: *Update* $E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$

This was with tabular RL which uses tokenization and linear approximation: let's see if we can represent it in terms of approximation model components

- *with tokenization, we have NM-dimensional feature*

$$\mathbf{x}(s^n, a^m) = \mathbf{1}\{s = s^n, a = a^m\}$$

- *with linear model, we have $\mathbf{w}$ which is of the same dimension*
- *with linear model, we have*

$$\nabla Q_{\mathbf{w}}(s, a) = \mathbf{x}(s, a)$$

LS Training of Action-Value Approximator

`ElgTrace`$(S_t, A_t, E(\cdot) \ |\lambda)$:

1: *Eligibility tracing function has NM components, i.e., $E(s, a)$ for all state-action pairs*
2: ***for** all state-action pairs (s, a) **do***
3: *Update $E(s, a) \leftarrow \gamma\lambda E(s, a)$*
4: ***end for***
5: *Update $E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$*

Let's define a vector $E_{\mathbf{w}}$ which has the NM entries of $E(S_t, A_t)$

- *we can write the update rule in line 4 as $E_{\mathbf{w}} \leftarrow \gamma\lambda E_{\mathbf{w}}$*
- *and the update rule in line 5 as*

$$\begin{aligned} E_{\mathbf{w}} &\leftarrow E_{\mathbf{w}} + \mathbf{1}\{s = S_t, a = A_t\} \\ &\leftarrow E_{\mathbf{w}} + \mathbf{x}(S_t, A_t) \\ &\leftarrow E_{\mathbf{w}} + \nabla Q_{\mathbf{w}}(S_t, A_t) \end{aligned}$$

LS Training of Action-Value Approximator

`ElgTrace(`$S_t, A_t, E(\cdot) \mid \lambda$`):`

1: *Eligibility tracing function has* NM *components, i.e.,* $E(s, a)$ *for all state-action pairs*
2: ***for*** *all state-action pairs* (s, a) ***do***
3:     *Update* $E(s, a) \leftarrow \gamma\lambda E(s, a)$
4: ***end for***
5: *Update* $E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$

Merging the two lines, we get into

$$E_{\mathbf{w}} \leftarrow \gamma\lambda E_{\mathbf{w}} + \nabla Q_{\mathbf{w}}(S_t, A_t)$$

This is the more general form of eligibility tracing

- *we can use it for any approximation*
- *our trace is of the size of the feature vector*

LS Training via TD$_\lambda$

So, we could evaluate via training as

`SGD_TD_QEval(`λ`):`

Initiate with some initial $\mathbf{w}$ and learning rate α
for episode $k = 1 : K$ **do**
  Sample a trajectory

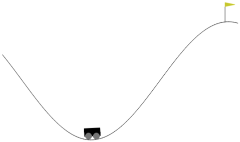
$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

  for $t = 0 : T-1$ **do**
    Compute $\Delta = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - Q_{\mathbf{w}}(S_t, A_t)$ `# forward propagation`
    Compute $\nabla = \nabla Q_{\mathbf{w}}(S_t, A_t)$ `# backpropagation`
    $E_{\mathbf{w}} \leftarrow \lambda\gamma E_{\mathbf{w}} + \nabla$
    Update weights as

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta E_{\mathbf{w}}$$

  end for
end for

Example: *Mountain Car*



We can compare tabular RL against the one with function approximation