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: *Control*

+ *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 now ge to the control

Recap: *Online Control ϵ-Greedy*

We have seen a typical control loop via ϵ-greedy algorithm

```
X_Control():
```

1: *Initiate two random policies π and $\bar{\pi}$*
2: **while** $\pi \neq \bar{\pi}$ **do**
3: $\quad \hat{q}_\pi = \texttt{X\_QUpdate}(\pi)$ *and* $\pi \leftarrow \bar{\pi}$ via function approximation
4: $\quad \bar{\pi} = \epsilon\text{-}\texttt{Greedy}(\hat{q}_\pi)$
5: **end while**

We can realize this loop using function approximation

1. *Start with an initial approximator*
2. *Use the approximator to improve policy via ϵ-Greedy*
3. *Use SGD to update the approximation model from observation*

We need an action-value approximator in this case

let's formally define the Q-Network then

Q-Net $\equiv$ *Action-Value Approximator – Form II*

Q-Network

In the context of RL, the action-value approximation model that maps features to the vector of all action values is often referred to as Q-Net

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

Q-Net

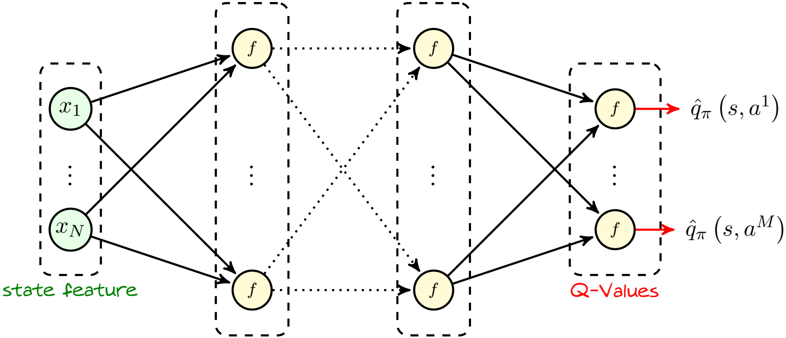
Deep Q-Network

If we set Q-Net to be a DNN; then, it is usually called

Deep Q-Network $\equiv$ DQN

Example: *MLP as DQN*

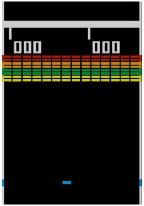
DQN can be simply an MLP



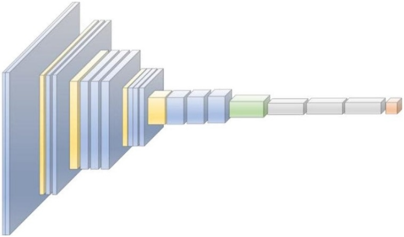
- *Give the feature as an input vector*
- *The MLP estimates all action-values*

Example: *CNN as DQN*

It can be a convolutional neural network $\equiv$ *CNN*



$\longrightarrow$



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

- *Give the feature as an input tensor, e.g., when feature is a frame*
- *The CNN estimates all action-values*

Building Control Loop with Q-Net

We can use Q-Net to build a control loop: *similar to tabular RL, we can have on-policy or off-policy approaches*

- *Deep on-policy RL*
  - ↳ *We can use a DQN to realize an on-policy control loop, e.g., SARSA*
  - ↳ *They are less in use as compared to off-policy versions*
- *Deep off-policy RL*
  - ↳ *We may use a DQN to realize an off-policy control loop, e.g., Q-learning*
  - ↳ *Due to some reasons deep off-policy RL is more popular*
    - ↳ *We can use experience replay in this case*
    - ↳ *They are therefore more sample-efficient*
    - ↳ *No worries! We will see these reasons* ☺

Let's start with on-policy algorithms

Remark: *Value Network*

+ *But, when I look at internet, I usually see the term value network! Don't we have this concept?!*
– Sure! We have already worked with *value networks*

Value Network

Any approximation model that gets features and returns values is a value function; this value could be a state value or an actio-value

- *Q-Net is a value network*
- $v_{\mathbf{w}}(\cdot)$ *that we used for prediction was also value network*

+ *Then, do we have any other sort of networks?!*
– Yes! We could have *policy networks*: *we will see them in the next chapter*

Recap: *Going On-Policy*

Let's look back at our TD-based prediction algorithm

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 $\Delta_t = R_{t+1} + \gamma v_{\mathbf{w}} \left(S_{t+1} \right) - Q_{\mathbf{w}} \left(S_t, A_t \right)$ `# forward propagation`

    Compute $\nabla_t = \nabla Q_{\mathbf{w}} \left(S_t, A_t \right)$ `# backpropagation`

    $E_{\mathbf{w}} \leftarrow \lambda \gamma E_{\mathbf{w}} + \nabla_t$

    Update weights as

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

  end for

end for

The key point in going on policy is to evaluate $v_{\mathbf{w}} \left(S_{t+1} \right)$ *using our actual policy*

SARSA: *Going On-Policy*

We can modify line *line 5*: *we compute* $v_{\mathbf{w}}\left(S_{t+1}\right)$ *as*

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

In on-policy approach, we act before we update

our policy leads us to next action A_{t+1}

So, we could move on our policy and write

$$\pi\left(a | S_{t+1}\right) = \begin{cases} 1 & a = A_{t+1} \\ 0 & a \neq A_{t+1} \end{cases} \rightsquigarrow v_{\mathbf{w}}\left(S_{t+1}\right) = Q_{\mathbf{w}}\left(S_{t+1}, A_{t+1}\right)$$

SARSA with *Q-Net*

```
SGD_SARSA():
```

1: *Initiate with* $\mathbf{w}$ *and learning rate* α
2: ***for*** *episode* $= 1 : K$ *or until* π *stops changing* ***do***
3: *Initiate with a random state-action pair* (S_0, A_0)
4: ***for*** $t = 0 : T - 1$ *where* S_T *is either terminal or terminated* ***do***
5: *Act* A_t *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
6: *Update policy to* $\pi \leftarrow \epsilon\text{-}\texttt{Greedy}(Q_{\mathbf{w}}(S_t, A_t))$
7: *Draw the new action* A_{t+1} *from* $\pi(\cdot|S_{t+1})$ *and move on policy*

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1}$$

8: *Set* $\Delta \leftarrow R_{t+1} + \gamma Q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - Q_{\mathbf{w}}(S_t, A_t)$ `# forward propagation`
9: *Update* $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$ `# backpropagation`
10: ***end for***
11: ***end for***

SARSA with *Q-Net and Eligibility Tracing*

We have seen that *with function approximation eligibility tracing reduces to*

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

Let's fit it into our SARSA control loop!

Eligibility Tracing $\propto$ Backpropagation

If we use a DQN, we should backpropagate to compute the eligibility tracing: this point is intuitive as both approaches naturally follow the same logic

SARSA(λ) with *Function Approximation*

```
SGD_SARSA(λ):
```

Initiate with $\mathbf{w}$ and learning rate α
for** episode $= 1 : K$ or until π stops changing **do
  Initiate with a random state-action pair (S_0, A_0)
  for** $t = 0 : T-1$ where S_T is either terminal or terminated **do
    Act A_t and observe $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
    Update policy to $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}\left(Q_{\mathbf{w}}\left(S_t, \mathbf{a}\right)\right)$
    Draw the new action A_{t+1} from $\pi\left(\cdot | S_{t+1}\right)$ and move on policy

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1}$$

    Set $\Delta \leftarrow R_{t+1} + \gamma Q_{\mathbf{w}}\left(S_{t+1}, A_{t+1}\right) - Q_{\mathbf{w}}\left(S_t, A_t\right)$ `# forward propagation`
    $E_{\mathbf{w}} \leftarrow \lambda\gamma E_{\mathbf{w}} + \nabla Q_{\mathbf{w}}\left(S_t, A_t\right)$ `# backpropagation`
    Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta E_{\mathbf{w}}$
  end for
end for

Recap: *Going Off-Policy*

In off-policy control: *we behave with a policy π but update by a target policy $\bar{\pi}$*

- *We could use importance sampling to evaluate target policy*
- *We mainly focused on Q-learning approach*

Q-Learning

*Q-learning is an off-*policy TD *control algorithm, where we sample with ϵ-greedy policy* but *update* with *greedy policy*

Key property of Q-learning is that we don't really need importance sampling: we could directly update as

$$\hat{q}_{\bar{\pi}}(S_t, A_t) \leftarrow \hat{q}_{\bar{\pi}}(S_t, A_t) + \alpha \Big(R_{t+1} + \gamma \max_m \hat{q}_{\bar{\pi}}(S_{t+1}, a^m) - \hat{q}_{\bar{\pi}}(S_t, A_t) \Big)$$

Q-Learning with Q-Net

We can therefore extend out Q-learning algorithm to

`SGD_Q-Learning():`

Initiate with $\mathbf{w}$ *and learning rate* α

for *episode* $= 1 : K$ *or until* π *stops changing* **do**

  Initiate with a random state S_0

  for $t = 0 : T-1$ *where* S_T *is either terminal or terminated* **do**

    Update policy to $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}\left(Q_{\mathbf{w}}\left(S_t, \mathbf{a}\right)\right)$

    Draw action A_t *from* $\pi\left(\cdot|S_t\right)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$

    $\Delta \leftarrow R_{t+1} + \gamma \max_m Q_{\mathbf{w}}\left(S_{t+1}, a^m\right) - Q_{\mathbf{w}}\left(S_t, A_t\right)$ `# forward propagation`

    Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}\left(S_t, A_t\right)$ `# backpropagation`

  end for

end for

We could potentially use eligibility tracing as well!

Incremental Algorithms: *Challenges*

What we have developed by now are the so-called *incremental approaches*

Incremental Algorithms

Incremental algorithms use the actual sample at each time to update the Q-Net

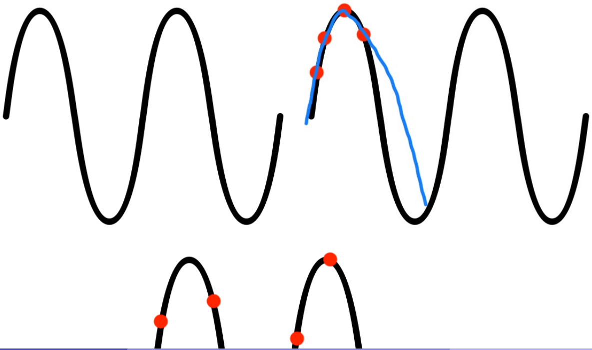
Though easy to implement, incremental approaches are not efficient

1. *They are extremely sample-inefficient*
   - ↳ *Once we use a sample, we are over with it*
   - ↳ *But, in deep learning we used to use the same samples several times*
     - ↳ *We make a training loop with multiple epochs*
     - ↳ *In each epoch, we go through the whole dataset*

Incremental Algorithms: *Challenges*

Though easy to implement, incremental approaches are not efficient

2. *They use samples that are strongly correlated*
   - ↳ *In sample* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1}$ *states* S_t *and* S_{t+1} *are closely related*
     - ↳ *If state is the location; then, locations in time* t *and* $t+1$ *are very close*
   - ↳ *But, we know that we need independent samples*
     - ↳ *With correlated samples we can easily stick to a local fitting*
     - ↳ *Our Q-Net does not generalize very well*



Solution: *Batch Training*

The solution to these challenges is to use *batch training*

1. *Save every sample in a database*
   - ↳ *We can only save the values we need, i.e., estimator values*
2. *In each iteration sample from database*
   - ↳ *We can reuse (reply) our sampled experiences*
   - ↳ *We can reduce the correlation between succeeding samples*

This is what we call experience reply

- *This needs us to control off-policy*
  - ↳ *We are using samples of previous non-improved policies*
  - ↳ *We want to update the value of a different target policy*
- *This is why deep off-policy algorithms are more sample-efficient*

We next study this in details ☺