Reinforcement Learning

Chapter 4: Function Approximation

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Vanilla Deep Q-Learning

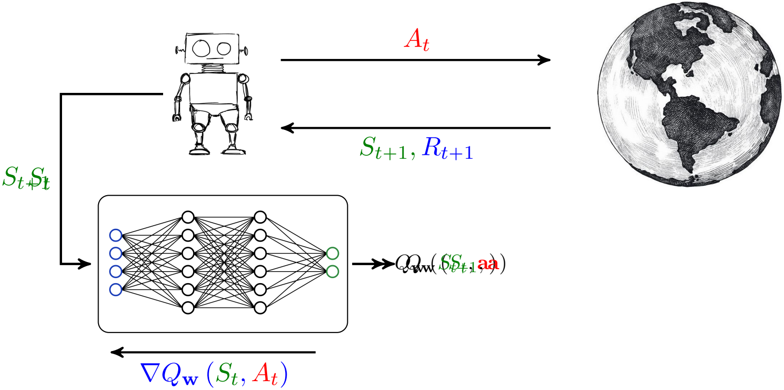
```
SGD_Q-Learning():
```

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 $\pi \leftarrow \epsilon\text{-}\texttt{Greedy}(Q_{\mathbf{w}}(S_t, \mathbf{a}))$
        Draw action A_t *from* $\pi(\cdot|S_t)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
        $\Delta \leftarrow R_{t+1} + \gamma \max_m Q_{\mathbf{w}}(S_{t+1}, a^m) - Q_{\mathbf{w}}(S_t, A_t)$ `# forward propagation`
        Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$ `# backpropagation`
    end for
end for

In deep Q-learning, we use a DQN to perform offline control via Q-learning

deep Q-learning $\equiv$ *DQL*

Vanilla DQL: *Visualization*



We update the weights on the DQN as $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$

$$\Delta \leftarrow \boxed{R_{t+1} + \gamma \max_m Q_{\mathbf{w}}(S_{t+1}, a^m)} - Q_{\mathbf{w}}(S_t, A_t)$$

Vanilla Deep Q-Learning: *Challenges*

Vanilla DQL does not perform *impressive*: *it suffers from two major challenges*

1. *We dear with strongly correlated samples*
   ↳ *We handle this issue via experience reply*
2. *The labels in the sample data-points change in each iteration*

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

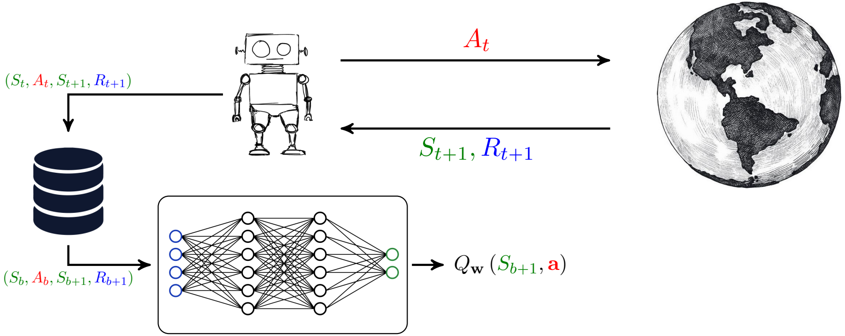
↳ *Here, we can look at each sampled state as a data sample with label*

$$y_t = R_{t+1} + \gamma \max_m Q_{\mathbf{w}} \left(S_{t+1}, a^m \right)$$

↳ *After each update of* $\mathbf{w}$ *this label changes*
↳ *This results in divergence or high error variance*
↳ *We are going to over-come this issue by using a target network*

Let's first get to experience replay

Experience Replay



We update DQN by mini-batches $\mathbf{w} \leftarrow \mathbf{w} + \sum_b \alpha \Delta_b \nabla Q_{\mathbf{w}}(S_b, A_b)$

$$\Delta_b \leftarrow \boxed{R_{b+1} + \gamma \max_m Q_{\mathbf{w}}(S_{b+1}, a^m)} - Q_{\mathbf{w}}(S_b, A_b)$$

DQL with Experience Replay

DQL_v1():

Initiate with $\mathbf{w}$, *empty replay buffer* $\mathbb{D}$ *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 $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(Q_{\mathbf{w}}(S_t, \mathbf{a}))$
        Draw action A_t *from* $\pi(\cdot|S_t)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
        Add sample $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$ *to the replay buffer* $\mathbb{D}$
        for *iteration* $\ell = 1 : L$ **do**
            Sample mini-batch $\mathbb{B} = \{S_b, A_b \xrightarrow{R_{b+1}} S_{b+1}$ *for* $b = 1 : B\}$ *from* $\mathbb{D}$
            $\Delta_b \leftarrow R_{b+1} + \gamma \max_m Q_{\mathbf{w}}(S_{b+1}, a^m) - Q_{\mathbf{w}}(S_b, A_b)$
            Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{b=1}^{B} \Delta_b \nabla Q_{\mathbf{w}}(S_b, A_b)$
        end for
    end for
end for

DQL with Experience Replay

In general, we can iterate multiple mini-batches, i.e., $L > 1$

- *This can improve the convergence speed*
- *The trained DQN may however stick to a bad local minima*

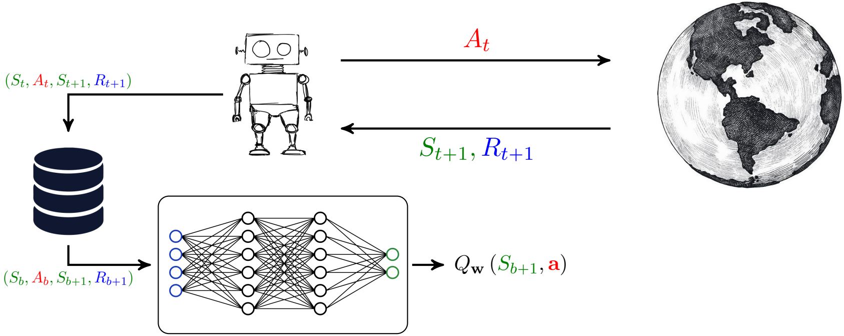
In practice *we typically set* $L = 1$

- *with a relatively large batch-size we can see good convergence results*

\+ *What about the second challenge? I didn't get really what was the issue at the first place!*

– Let's break it down!

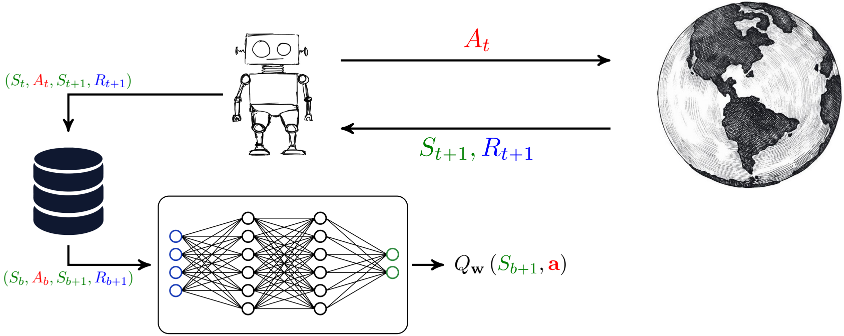
Varying Labels



We can look at the training procedure as supervised learning with label

$$y_b = R_{b+1} + \gamma \max_m Q_{\mathbf{w}}(S_{b+1}, a^m)$$

Varying Labels

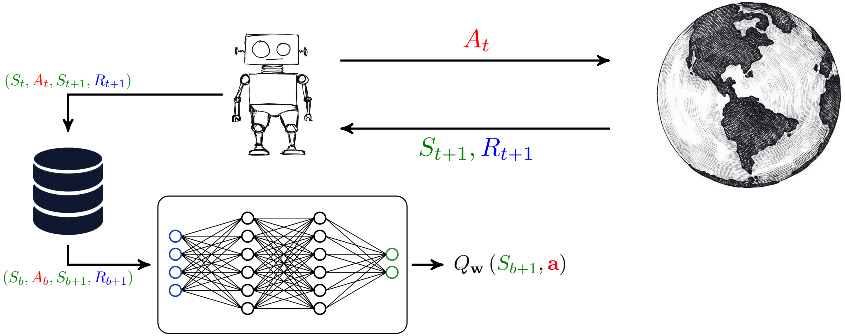


We are then updating $\mathbf{w}$ *gradually, such that*

$$\Delta_b = y_b - Q_{\mathbf{w}}(S_b, A_b)$$

shrinks: but, each time we update $\mathbf{w}$*, the label* y_b *also changes!*

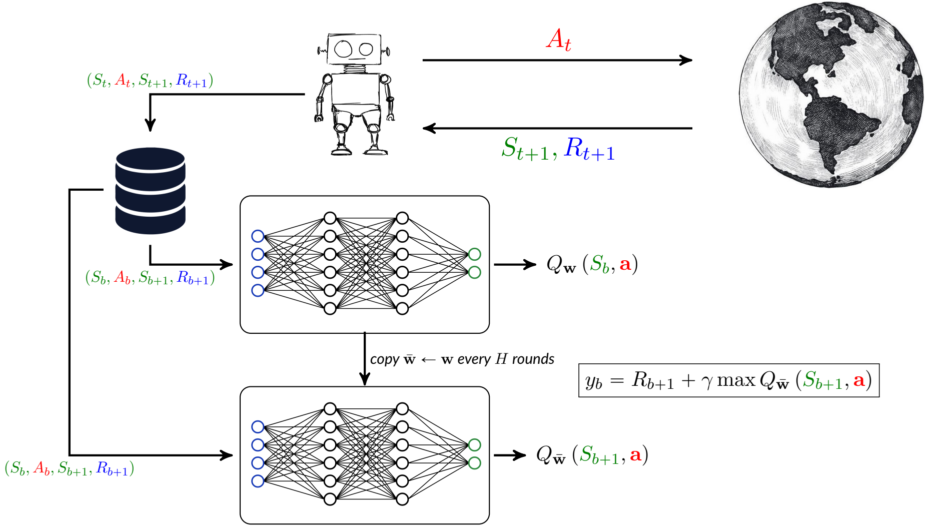
Varying Labels



This is an issue: in standard SGD, we have fixed labels and therefore

by multiple iterations the NN gradually converges to a good approximator

Target Network: *Simple Remedy*



DQL: *Classic Algorithm*

DQL():

Initiate with $\mathbf{w}$, *empty replay buffer* $\mathbb{D}$, *learning rate* α, *and a random state* S_t

while *interating* **do**

  Update $\bar{\mathbf{w}} \leftarrow \mathbf{w}$

  for $h = 1 : H$ **do**

    $S_t \leftarrow S_{t+1}$ **if** S_t *is a terminal state* **then** *replace* S_t *with a random state*

    Update policy to $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(Q_{\mathbf{w}})$ *and draw* A_t *from* $\pi(\cdot|S_t)$

    Add $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$ *to the replay buffer* $\mathbb{D}$

    for *iteration* $\ell = 1 : L$ **do**

      Sample mini-batch $\mathbb{B} = \{S_b, A_b \xrightarrow{R_{b+1}} S_{b+1} \text{ for } b = 1 : B\}$ *from* $\mathbb{D}$

      $\Delta_b \leftarrow \boxed{R_{b+1} + \gamma \max_m Q_{\bar{\mathbf{w}}}(S_{b+1}, a^m)} - Q_{\mathbf{w}}(S_b, A_b)$

      Update $\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{b=1}^{B} \Delta_b \nabla Q_{\mathbf{w}}(S_b, A_b)$

    end for

  end for

end while

A Revolution: *Google DeepMind*

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Just to have a clue about how revolutionary it had been

Human-level control through deep reinforcement learning

V Mnih, K Kavukcuoglu, D Silver, AA Rusu, J Veness, MG Bellemare, A Graves, M Riedmiller…

nature, 2015 • nature.com

Abstract

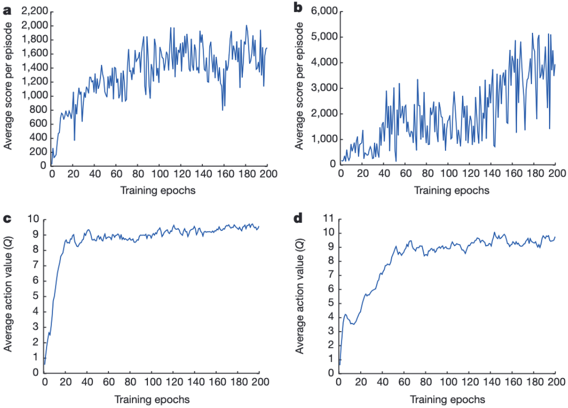
The theory of reinforcement learning provides a normative account, deeply rooted in psychological and neuroscientific perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new

SHOW MORE

☆ Save Cite Cited by 30437 Related articles All 57 versions Web of Science: 14719

A Revolution: *Google DeepMind*

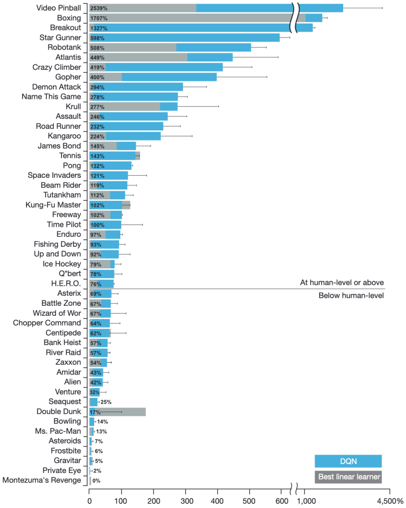
DQL could show extraordinary performance



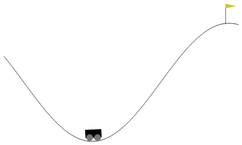
You may also watch the demo of the Breakout game on Youtube

A Revolution: *Google DeepMind*

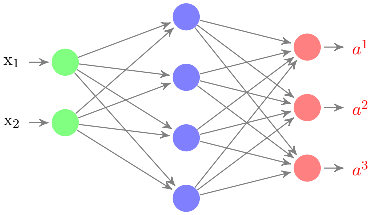
DQL was the first universal algorithm that could be applied to any environment



Example: *Mountain Car*



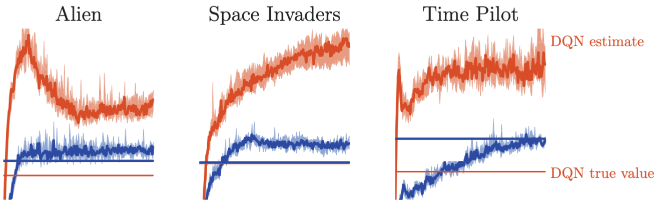
Let's try to imagine DQL in the mountain car example with a simple DQN



A more concrete example will be solved in the next Tutorial

DQL: *Bias Problem*

Q-learning is known to estimate action-values with bias: *take a look at few examples of DQL algorithm playing Atari games*



+ *Why is it happening? Is it simply because of TD approach?*
– It's severer than only TD: *it comes from the* $\max$ *operator in the update!*
+ *How does it come?*
– Well, It's best understood through an example

Bias Problem: *Basic Example*

Let's consider a simple example: *assume we are dealing with two computers, namely A and B. These computers are used to run the same program*

- *Computer A runs the program in exactly* $X_A = 8$ *seconds*
- *Computer B runs the program in exactly* $X_B = 5$ *seconds*

We however do not know these values: we can only run sample runs

- *Our time measurement is noisy, i.e., we compute on Computer* i

$$\hat{X}_i = X_i + \varepsilon$$

↳ *This error is random and can increase or decrease* X_i

Ultimate Goal

We want to compute the maximum runtime between the two computers

Bias Problem: *Basic Example*

We have access to noisy samples

$$\hat{X}_A^{(1)}, \ldots, \hat{X}_A^{(K)}, \hat{X}_B^{(1)}, \ldots, \hat{X}_B^{(K)}$$

If K is only moderately large, we could almost surely say that

$$\max \left\{ \hat{X}_A^{(1)}, \ldots, \hat{X}_A^{(K)}, \hat{X}_B^{(1)}, \ldots, \hat{X}_B^{(K)} \right\} > \max \left\{ X_A, X_B \right\} = 8$$

- *Within enough number of samples there is for sure a positive error sample*
- *This error sample renders an over-estimation*

A crucial point is that if we repeat this experiment several times, we always get an over-estimate; therefore,

after averaging over multiple instances, we are still biased!

Solution to Max-Bias: *Double Measurements*

There is a very simple and intuitive solution to this problem: *we can collect two sequences of samples*

$$\textit{Sequence 1: } \hat{X}_{A,1}^{(1)}, \ldots, \hat{X}_{A,1}^{(K)}, \hat{X}_{B,1}^{(1)}, \ldots, \hat{X}_{B,1}^{(K)}$$
$$\textit{Sequence 2: } \hat{X}_{A,2}^{(1)}, \ldots, \hat{X}_{A,2}^{(K)}, \hat{X}_{B,2}^{(1)}, \ldots, \hat{X}_{B,2}^{(K)}$$

We find the index of the maximizer in Sequence 1, i.e.,

$$(i, k) = \operatorname{argmax} \left\{ \hat{X}_{A,1}^{(1)}, \ldots, \hat{X}_{A,1}^{(K)}, \hat{X}_{B,1}^{(1)}, \ldots, \hat{X}_{B,1}^{(K)} \right\}$$

But we take the sample from Sequence 2, i.e.,

$$\hat{X}_{\max} = \hat{X}_{i,2}^{(k)}$$

This is an unbiased estimator: if we repeat this experiment several times

$$\boxed{\textit{after averaging over multiple instances, we get close to } X_A}$$

DQL with *Double Measurements*

We can apply this idea to DQL: *we train two DQNs simultaneously*

- *We find out the action with maximum value using one DQN*
- *We evaluate the action-value of this action by the other DQN*

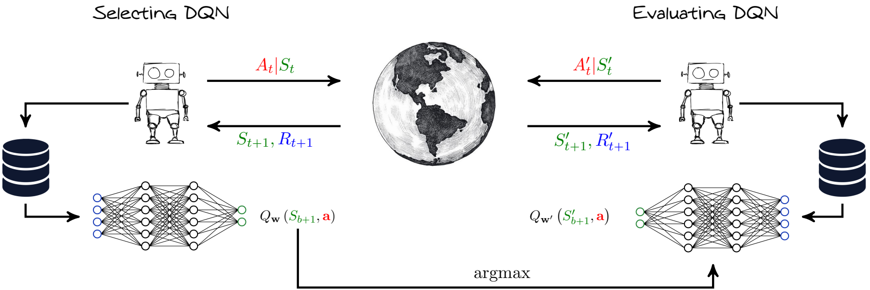
This way we get an unbiased estimator of the optimal action-value

we refer to this approach as double DQL

Attention

In general this does not mean that we are necessarily reaching to a better policy: we could have biased estimator and still play optimally!

Double DQL



We train two DQNs on two independent experience sets

- *With the online DQL we find*

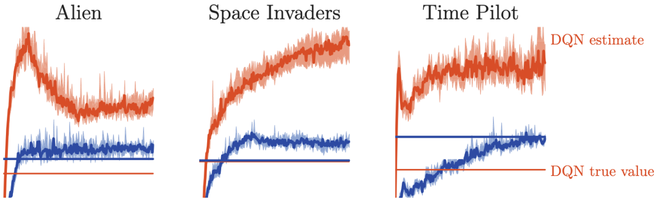
$$A^\star_{t+1} = \operatorname{argmax} Q_{\mathbf{w}}\left(S_{b+1}, \mathbf{a}\right)$$

- *With the double DQL we evaluate*

$$y_b = R_{b+1} + \gamma Q_{\mathbf{w}'}\left(S_{b+1}, A^\star_{t+1}\right)$$

Double DQL: *Sample Results*

Let's take a look back to the earlier examples



Here, blue curves show double DQL

- *In all examples we are getting less bias as compared to DQL*
- *In two example it helps converging to better policy*
  ↳ *Alien and Time Pilot*
- *In one example, it reduces bias but does not impact converging policy*

Other Variants

There are various extensions to classic DQL: *some famous ones are*

- *Double DQL*
  - ↳ *It tries to reduce the bias of value estimation*
- *Dueling DQL*
  - ↳ *It uses notion of advantage ≡ difference between action-value and value*
  - ↳ *It helps finding non-valuable actions*
- *Prioritized DQL*
  - ↳ *It gives priority to samples in the experience buffer*
- *Distributed DQL*
  - ↳ *It enables training DQN through pipelining*
  - ↳ *This let training of DQNs for massive problems*

Distributed DQL: *Gorila*

General Reinforcement Learning Architecture $\equiv$ *Gorila*

Gorila is implemented through four main generalization

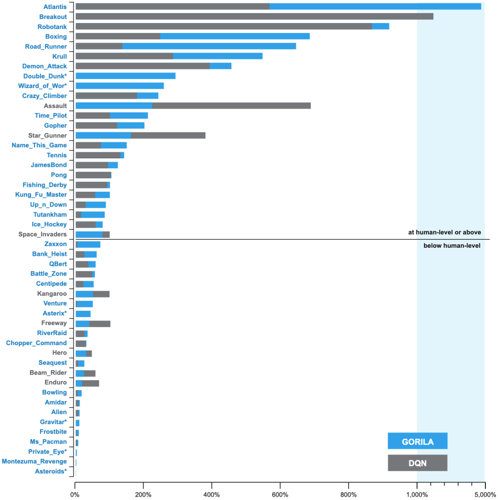
1. *Parallel actors generating acting behavior*
2. *Parallel DQNs trained by stored experience*
3. *Distributed storage of experience*
4. *A distributed DQN that specifies acting behavior policy*

Gorila massively parallelize implementation of DQL

this enables implementation of DQL for realistic hard control loops

Distributed DQL: *Gorila*

With significantly lower training time, Gorila starts to beat classic DQL



Dealing with Continuous Actions

Once DQL was established a new question raised

- ? *How can we handle settings with continuous action space?*

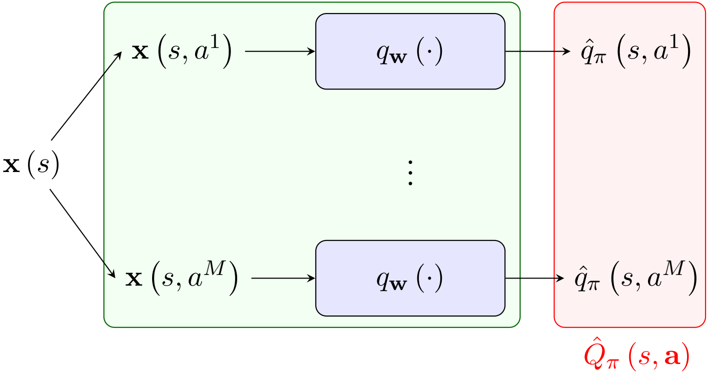
This is a very practical setting that show up in robotics, autonomous driving, etc

---

- + *What about it? Why should be a challenge to use DQL with continuous action space?*
- – Well! How could we maximize action-value in this case?!

Let's take a look to see the challenge clearly

DQN: *Recalling the Output*



With continuous actions, we cannot enumerate the output!

+ *Why don't we use action-value approximator of form I?!*

– Well! Let's do this

Recall: *Action-Value Approximator – Form I*

This is what we called Form I

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

Let's now see how the DQL algorithm can be applied

↳ *Let's consider the vanilla DQL*

1: *Update policy to* $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(Q_{\mathbf{w}}(S_t, \mathbf{a}))$
2: *Draw action* A_t *from* $\pi(\cdot|S_t)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
3: $\Delta \leftarrow R_{t+1} + \gamma \max_m Q_{\mathbf{w}}(S_{t+1}, a^m) - Q_{\mathbf{w}}(S_t, A_t)$
4: *Update* $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$

We obviously can replace $Q_{\mathbf{w}}(\cdot)$ *with* $q_{\mathbf{w}}(\cdot)$

DQL with *Action-Value Approximator – Form I*

1: *Update policy to* $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(Q_{\mathbf{w}}(S_t, \mathbf{a}))$
2: *Draw action* A_t *from* $\pi(\cdot|S_t)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
3: $\Delta \leftarrow R_{t+1} + \gamma \max_m Q_{\mathbf{w}}(S_{t+1}, a^m) - Q_{\mathbf{w}}(S_t, A_t)$
4: *Update* $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$

We can replace these updates with

1: *Update policy to* $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(q_{\mathbf{w}}(S_t, a))$
2: *Draw action* A_t *from* $\pi(\cdot|S_t)$ *and observe* $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
3: $\Delta \leftarrow R_{t+1} + \gamma \max_a q_{\mathbf{w}}(S_{t+1}, a) - q_{\mathbf{w}}(S_t, A_t)$
4: *Update* $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Delta \nabla q_{\mathbf{w}}(S_t, A_t)$

Well! We need to optimize over a continuous variable!

+ *Where exactly we need it?*
– In lines 1 and 3: *once for* ϵ*-greedy update and once for off-policy control*

DQL with *Action-Value Approximator – Form I*

This is an essential challenge: *we need to optimize over a continuous variable*

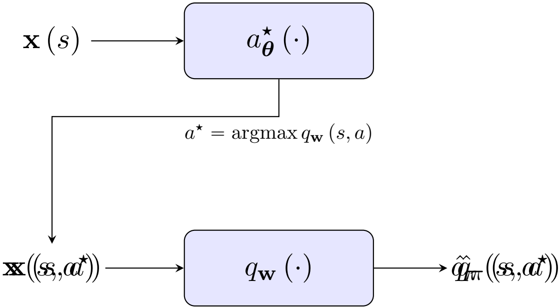
- *We may grid the action space*
  - ↳ *It's computationally expensive: we are back at square one!*
- *We may apply gradient descent*
  - ↳ *It makes a two-tier loop: it's again computationally expensive!*

\+ *It sounds like impossible!*

– *Only impossible is impossible*

In practice, we solve the target optimization via a DNN!

Learning Optimal Action



We could then update in DQL algorithm as

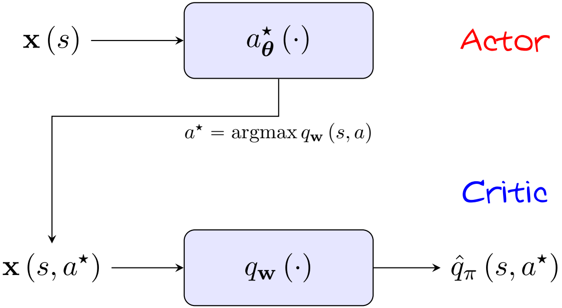
$$\Delta \leftarrow R_{t+1} + \gamma q_{\mathbf{w}} \left(S_{t+1}, a^\star \right) - q_{\mathbf{w}} \left(S_t, A_t \right)$$

and for ϵ-greedy improvement, we could

act $a^\star$ with probability $1 - \epsilon$ and random with probability ϵ

Learning Optimal Action

+ *Say we trained the network after some time; then, what do we do?*
– We act $a^\star$ for each state s



This is a particular example if actor-critic algorithm!

↳ *We will discuss these algorithms*

Solution to Continuous Action: *Policy Networks*

Our actor learns an optimal greedy policy

$$\mathbf{x}(s) \longrightarrow \boxed{a^\star_{\boldsymbol{\theta}}(\cdot)} \longrightarrow a^\star(s) \longrightarrow \begin{cases} 1 & a = a^\star(s) \\ 0 & a \neq a^\star(s) \end{cases}$$

This is a simplified form of the so-called policy network

$$\mathbf{x}(s,a) \longrightarrow \boxed{\pi_{\boldsymbol{\theta}}(\cdot)} \longrightarrow \hat{\pi}^\star(a|s)$$

Policy Network

Policy network is an approximation model that maps state-action features to the optimal policy

Solution to Continuous Action: *Policy Networks*

+ *What is the point in doing DQL anymore when we have a ? We already learn the optimal policy that we are looking for!*
– Great! This is what we do in *policy gradient algorithms*

Policy networks are used in two sets of deep RL approaches

- *Policy gradient approaches*
  ↳ *We do not use a value network and directly approximate optimal policy*
  ↳ *This is what we study next*
- *Actor-critic approaches*
  ↳ *We keep the value network to examine the approximated optimal policy*
  ↳ *This is the most practically-robust approach we can use*