Reinforcement Learning (RL)

Chapter 9:

Deep Reinforcement Learning (DRL)

Deep Q-Learning algorithm

Saeed Saeedvand, Ph.D.

Contents

In this Chapter:

- ✓ Deep Reinforcement Learning
- ✓ Deep Q-Learning.

Aim of this chapter:

- ✓ Understand the general concepts of the Deep Reinforcement Learning and some implementation notes.

Deep Reinforcement Learning

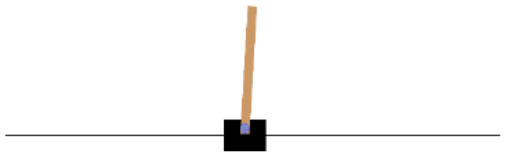
✓ **Deep reinforcement learning** (**deep RL**) combines Reinforcement Learning (RL) and Deep Learning (DL).

- Deep neural networks can be directly used as function approximation
- States as input can be directly feed to deep network without requiring an explicit specification of features.

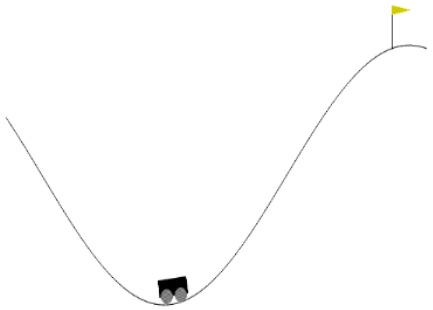
In Deep RL we try to allowing agents to **make decisions** from **pure input data** inputs as **state space** (We avoid manual engineering)

Deep Reinforcement Learning

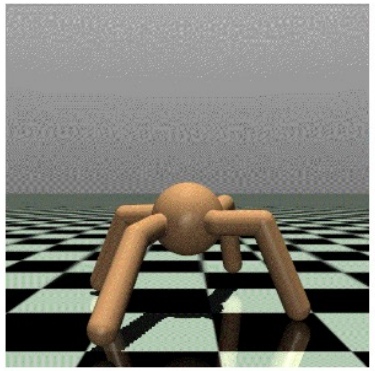
- ✓ Learning to control agents directly from **high-dimensional sensory** inputs like vision and speech
- ✓ In hand-crafted features performance is heavily relies on the quality of the feature representation
- ✓ **Convolutional networks**, **multilayer perceptrons** enable us to use very large inputs with using Deep RL algorithms can be considered



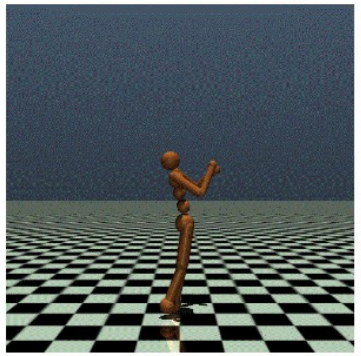
Cart Pole



Mountain Car Continuous



Ant



Humanoid

Pixel rendered to the screen in a video game

Deep Reinforcement Learning

Differences of RL and Supervised Learning (like DL)

- ✓Deep learning applications require large amounts of hand- labelled training data **while** RL must be able to learn from a scalar reward
- ✓In DL we calculate loss directly **while** reward signal that is noisy and delayed between actions (may seen after many actions).

Deep Reinforcement Learning

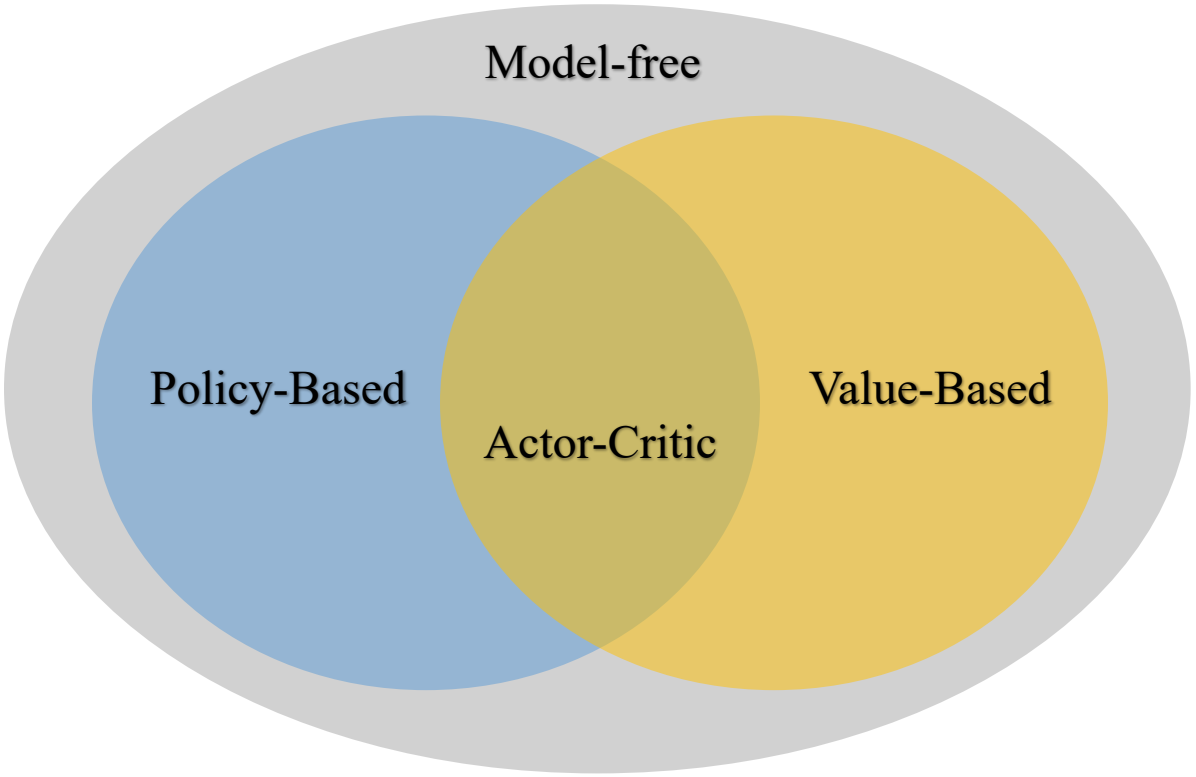
Differences of RL and Supervised Learning (like DL)

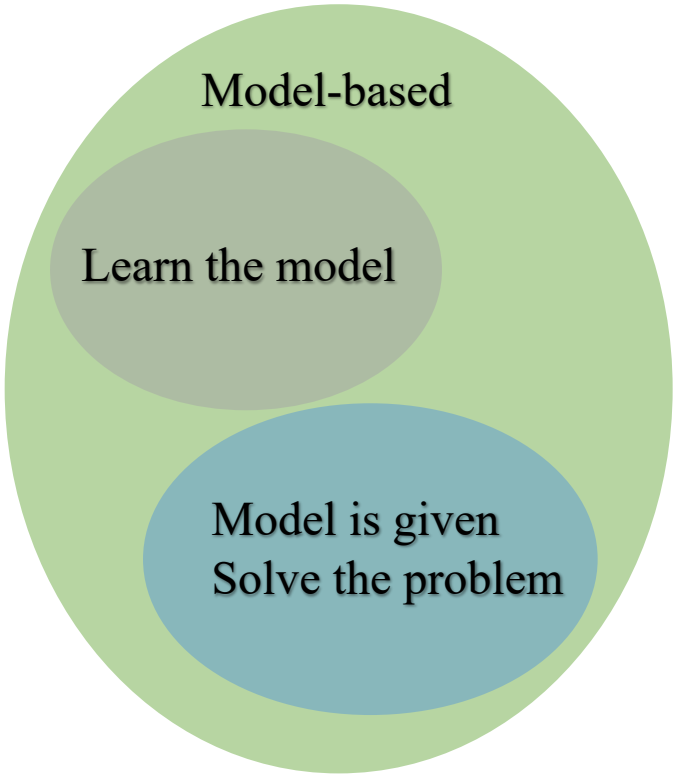
- ✓Data samples usually are independent **while** in RL data sequence usually are dependent

- ✓In DL generally data is steady **while** In RL data distribution changes as the algorithm learns new action chosen.

Deep Reinforcement Learning

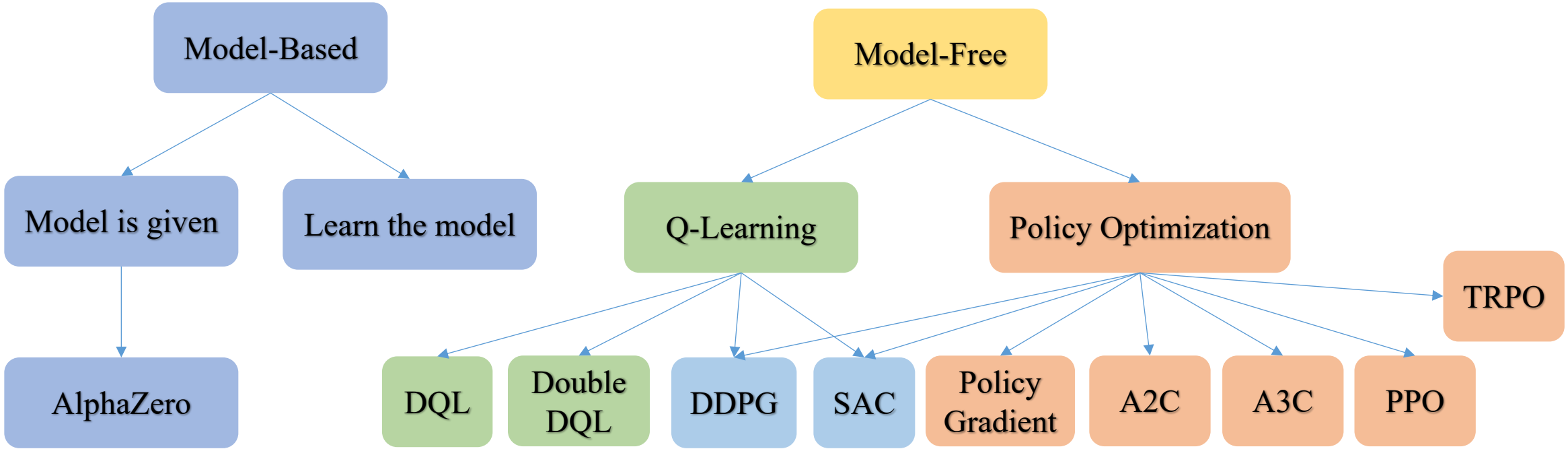
Different class of DRL algorithms:





Deep Reinforcement Learning

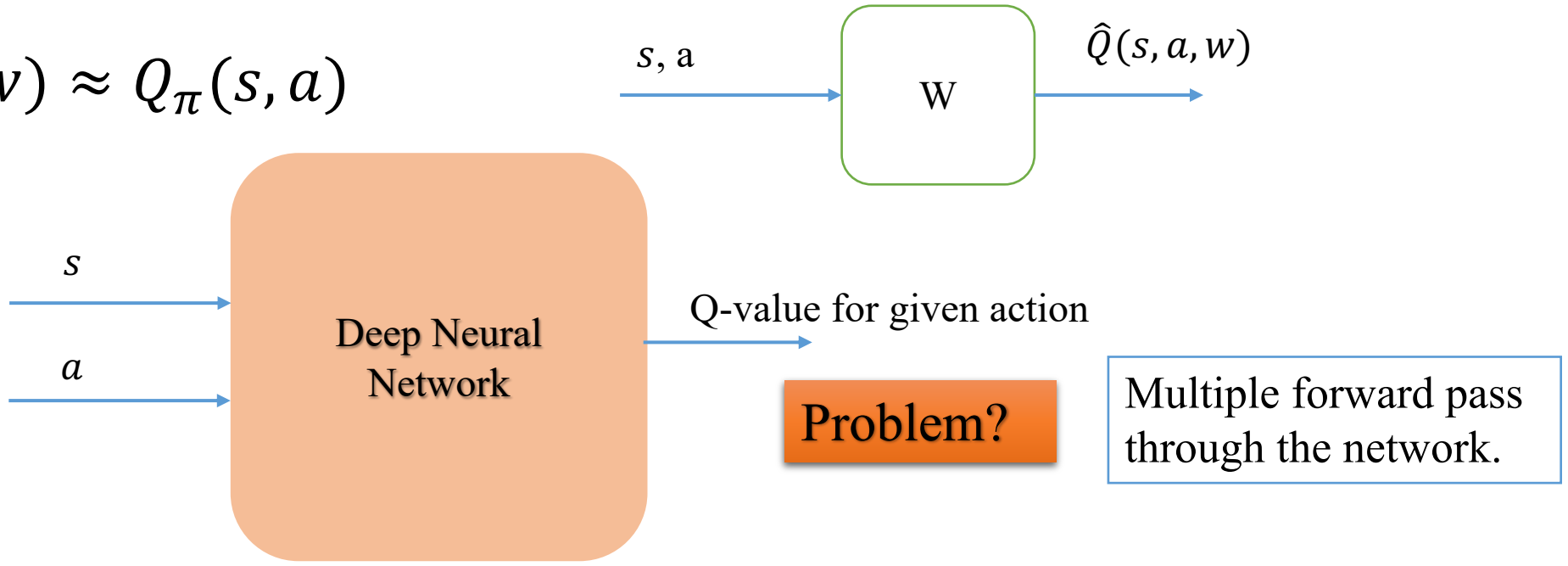
Different class of DRL algorithms:



Deep Q-Learning (DQL)

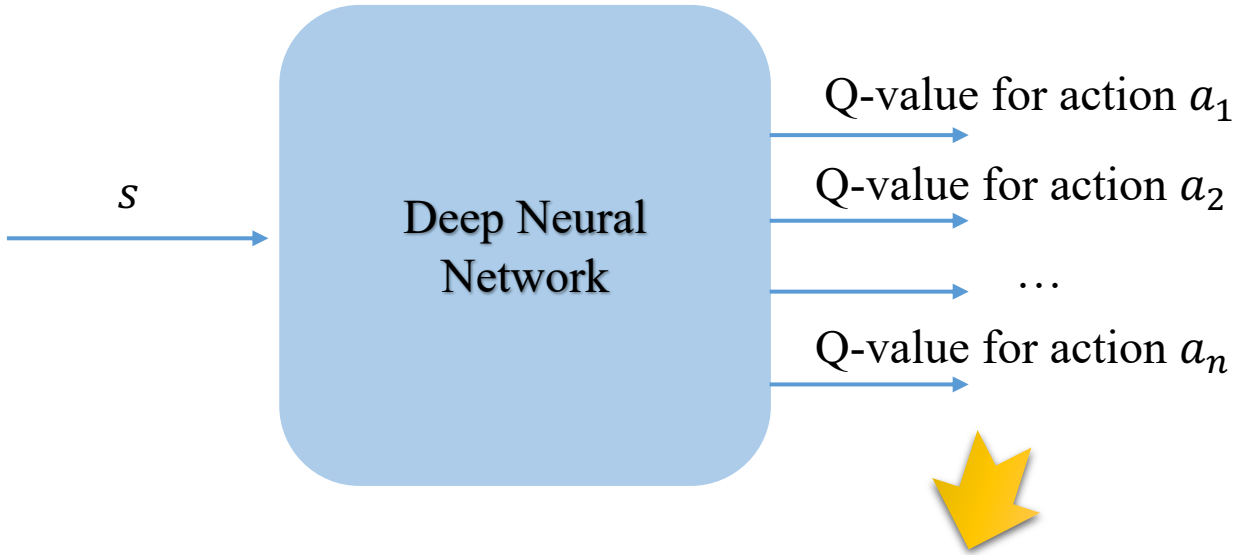
- ✓ Deep Q-learning is one of the ways to use neural networks.
- ✓ DQL has been proposed for *"Playing Atari with DRL at 2013"*.

$$\hat{Q}(s,a,w) \approx Q_\pi(s,a)$$



Deep Q-Learning

✓ Define a **separate output unit for each possible action**, and only the state representation is an input to the neural.



Gives us the ability to compute Q-values for all possible actions in a given state with only a **single forward pass through the network**.

Deep Q-Learning

- ✓ DQL uses the Q-learning idea
- ✓ **Convolutional neural network (CNN)** can be useful to deal this problem
- ✓ Agent can learn successful control policies **from raw video data** in **complex RL environments**
- ✓ A **deep network** can be trained with a **Q-learning algorithm**, with stochastic gradient descent to update the weights.
- ✓ To **solve data correlation (dependency)** and **non-stationary distributions** of data **experience replay mechanism** can be used.

Deep Q-Learning

What is the experience replay or reply buffer:

✓ If we update network each time by one new state and value pairs the network only learns the most recent data (old ones will be forgotten).

Solution:

- We store the last N experience tuples in the replay memory (s, a, r, s', a') - pairs from prior episodes
- Aim is to uniformly and randomly samples previous transitions, and thereby smooths the training distribution over many past behaviors.
- Samples of experience randomly drawn from the pool of stored samples and **minibatch updates** during inner loop.

Deep Q-Learning

Challenge: Using histories of **arbitrary length as inputs** to a neural network can be very **difficult**

Solution:

- To solve this the Q-function only works on fixed length representation of histories produced by a function ∅

Challenge: Update of one network can be very unstable

Solution:

- The reason is if we change the target each time the model updates itself and makes instability (we will discuss more)

Value Function Approximation

DQL objectives

- ✓ Create a **single neural network agent** that is able to **successfully learn to play as many of the games** as possible
- ✓ **No game-specific information** or hand-designed visual features used
- ✓ Learn **only by video input**, the **reward and terminal signals**, and the **set of possible action,** (same as human player).
- ✓ So information like agent position and environment information **only agent observes an image (**vector of **raw pixel** values representing the current screen**)**

Value Function Approximation

DQL objective

- ✓ Receiving a **reward that representing** the change in **game score**
- ✓ **Feedback** about an action may only be received after **a very long time**
- ✓ Observation of **current situation** from **only the current screen is not always possible**
- ✓ **S**equences of actions and observations is important (finite MDP), so that each **finite sequence is a distinct state**
- ✓ Using **same network architecture** and all **hyperparameters** for all different games

Deep Q-Learning

- ✓ A challenging RL testbed that presents agents with a high dimensional visual inputs to play Atari games
  - ➢ For example 210 × 160 RGB video at 60Hz

- ✓ Diverse and interesting set of tasks that were designed to be difficult for humans players.

Deep Q-Learning's original paper

Playing atari with **deep reinforcement learning**
V Mnih, K Kavukcuoglu, D Silver, A Graves… - arXiv preprint arXiv …, 2013 - arxiv.org
… **deep learning** model for **reinforcement learning**, and demonstrated its ability to master difficult control policies for **Atari** … We also presented a variant of online Q-**learning** that combines …
☆ Save 99 Cite Cited by 10379 Related articles All 34 versions »

Reminder to Value Function Approximation

General Rule for VFA

- ✓ We can define an **objective function** $E(w)$ (also known as $J(w)$) as **minimizing loss** between target value and predicted value.

$$E(w) = \mathbb{E}_{\pi}[(V_{\pi}(s) - \hat{V}(s, w))^2]$$

- ✓ This is an optimization problem (find parameter vector w to minimize MSE).
- ✓ For instance **Stochastic gradient descent (SGD)** algorithm can be used to solve it.

Deep Q-Learning

- ✓ As standard assumption the **future rewards** are discounted by **γ per time-step**
- ✓ The **network can be trained** by minimizing a sequence of loss functions $\boldsymbol{E_i(\theta_i)}$ that calculated and changes at each iteration i.

$$E_i(\theta_i) = \mathbb{E}[(y_i - \hat{Q}(s_t, a_t; \theta_i))^2]$$

Where, target value for iteration i is:

$$y_i = R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; \theta_{i-1})]$$

Note: The parameters from the previous iteration θ_{i-1} are **held fixed** when optimizing the loss function $E_i(\theta_i)$ **or we use two network.**

Deep Q-Learning

Calculate the target and predicted values

$$E(w) = \mathbb{E}_{\pi}[(V_{\pi}(s) - \hat{V}(s,w))^2]$$

VFA as Q-learning

✓ Differentiating the loss function with respect to the weights with the following loss and gradient:

$$E(w) = \mathbb{E}[(R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)] - \hat{Q}(s_t, a_t; w))^2]$$

$$\Delta w = \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)] - \hat{Q}(s_t, a_t; w) \nabla_w \hat{Q}(s_t, a_t; w)]$$

Gradient (Nambla)

Important Note: The real target value $\mathrm{Q}(s_{t+1}, a_{t+1})$ in practice is also prediction from model as $\hat{Q}(s_{t+1}, a_{t+1}; w)$ while in supervised learning are fixed before learning begins..

Deep Q-Learning

- ✓ The **network weights** are updated after **every time-step**
- ✓ Then the **expectations** are replaced by another single samples from the behavior distribution from experiences
- ✓ **To optimize** the loss function instead of computing the full expectations in the above gradient, (computationally expensive) **we can use stochastic gradient descent (SGD)**

Deep Q-Learning – Target Network

What is the Target Network

- ✓ In fact same network is calculating the target value and predicted value, which can cause problems and instability

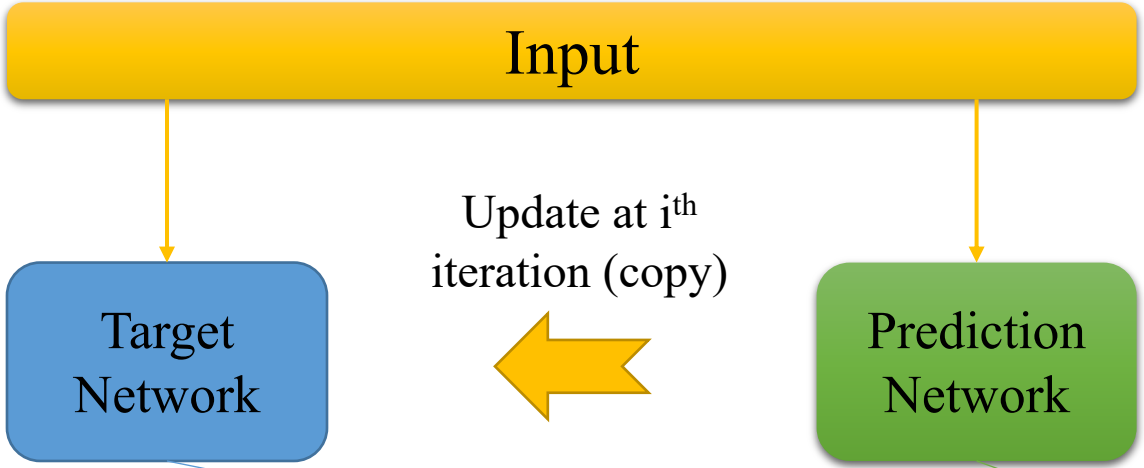
$$E(w) = \mathbb{E}[(R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)] - \hat{Q}(s_t, a_t; w))^2]$$

- ✓ Solution is we use two networks instead of using one:
  - Target Network
  - Prediction Network

- ✓ Target network with same architecture and we updated it only some times (copy from prediction Network)

Deep Q-Learning – Target Network

Update of Target Network



$$E(w) = \mathbb{E}[(R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)] - \hat{Q}(s_t, a_t; w))^2]$$

Deep Q-Learning

Reducing the input dimensionality is well-known

✓Frames can be preprocessed by converting from RGB to gray-scale

✓ Images can be down-sampling it to a smaller image sizes:

- No need for details most of the times
- For example: 210 × 160 pixels to the 110×84 pixels.

✓Captures the playing or important area

Deep Q-Learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

$$\Delta w = \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)] - \hat{Q}(s_t, a_t; w) \nabla_w \hat{Q}(s_t, a_t; w)]$$

Deep Q-Learning

DQL is model-free:

- ✓ It solves the reinforcement learning task **directly using samples from the environment**, without explicitly constructing an estimate of environment.

DQL is off-policy:

- ✓ Learns about the **greedy strategy** $\text{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)]$ **while** following a agent actions can come from **epsilon-greedy strategy** and can be different.

Deep Q-Learning

DQL for Atary game (original presentation)

- ✓ The input to the neural network consists is an 84 × 84 × 4 image produced by ∅
- ✓ First hidden layer convolves 16 of 8 × 8 filters with stride 4 with the input image and applies a ReLU
- ✓ Second hidden layer convolves 32 of 4 × 4 filters with stride 2 and applies a ReLU
- ✓ Final hidden layer is fully-connected and consists of 256 units
- ✓ The output layer is a fully- connected linear layer with a single output for each valid action

Deep Q-Learning

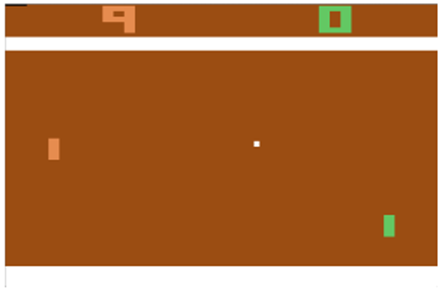
DQL for Atary game (original presentation)

- ✓ The number of valid actions varied between 4 and 18 on the games we considered
- ✓ All positive rewards to be 1 and all negative rewards to be −1, leaving 0 rewards unchanged
- ✓ Trained for a total of 10 million frames and used a replay memory of one million most recent frames
- ✓ Use a simple frame-skipping technique; agent sees and selects actions on every kth frame instead of every frame, and last action is repeated on skipped frame (like 4)

Value Function Approximation

Beam Rider

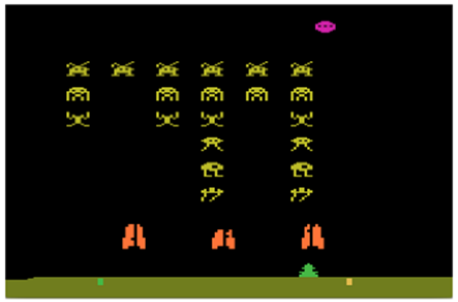
Breakout



Pong

Seaquest

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690



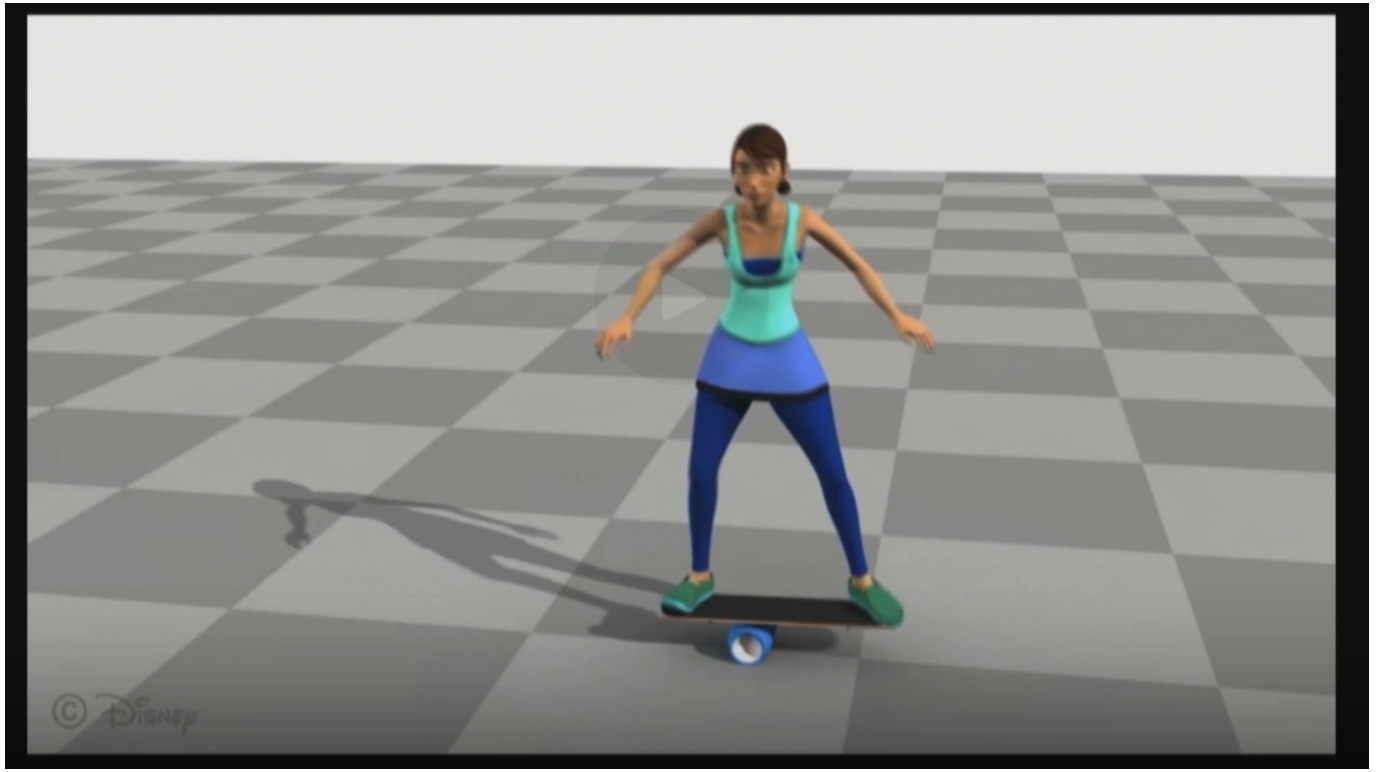
Space Invaders

,,,

,

Implementation Example

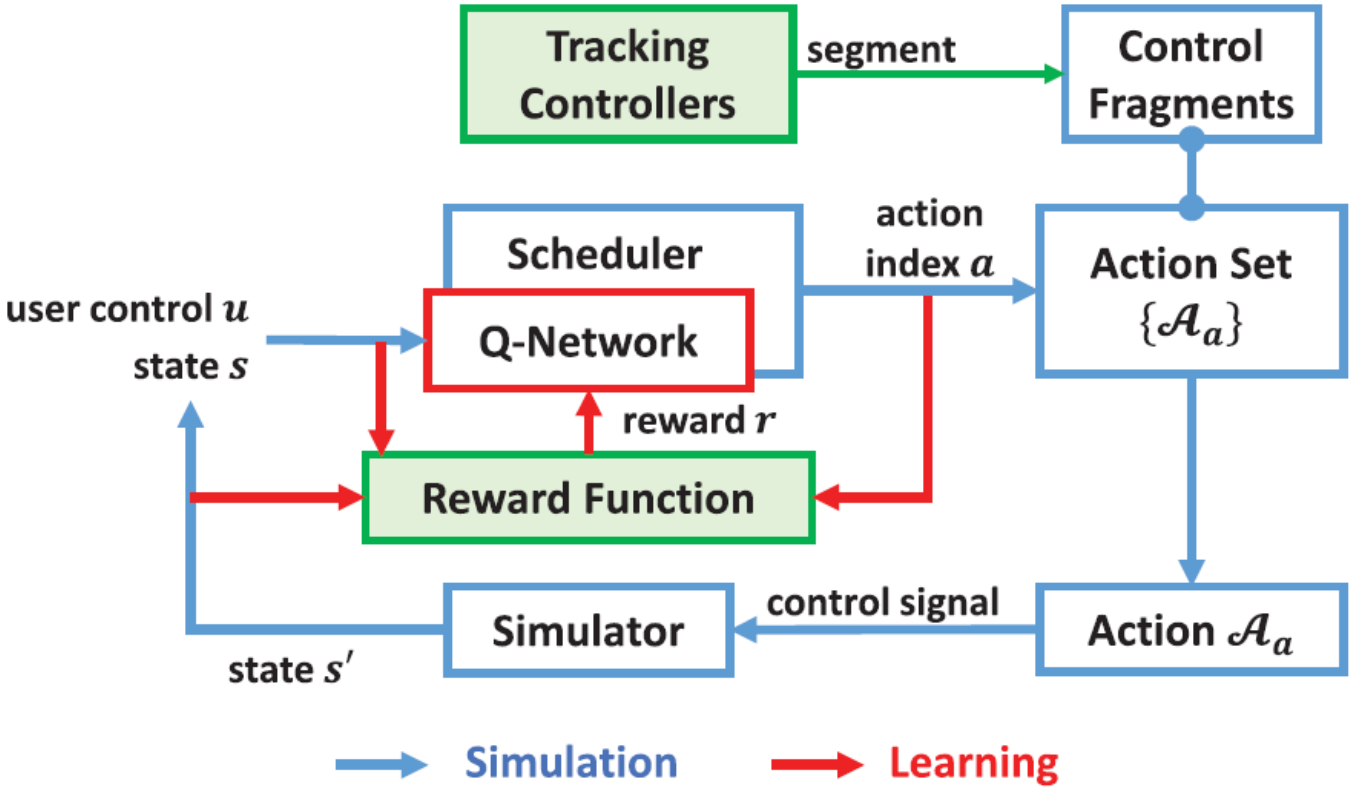
- ✓ Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning



Liu, Libin, and Jessica Hodgins. "Learning to schedule control fragments for physics-based characters using deep q-learning." ACM Transactions on Graphics (TOG) 36.3 (2017): 1-14.

Implementation Example

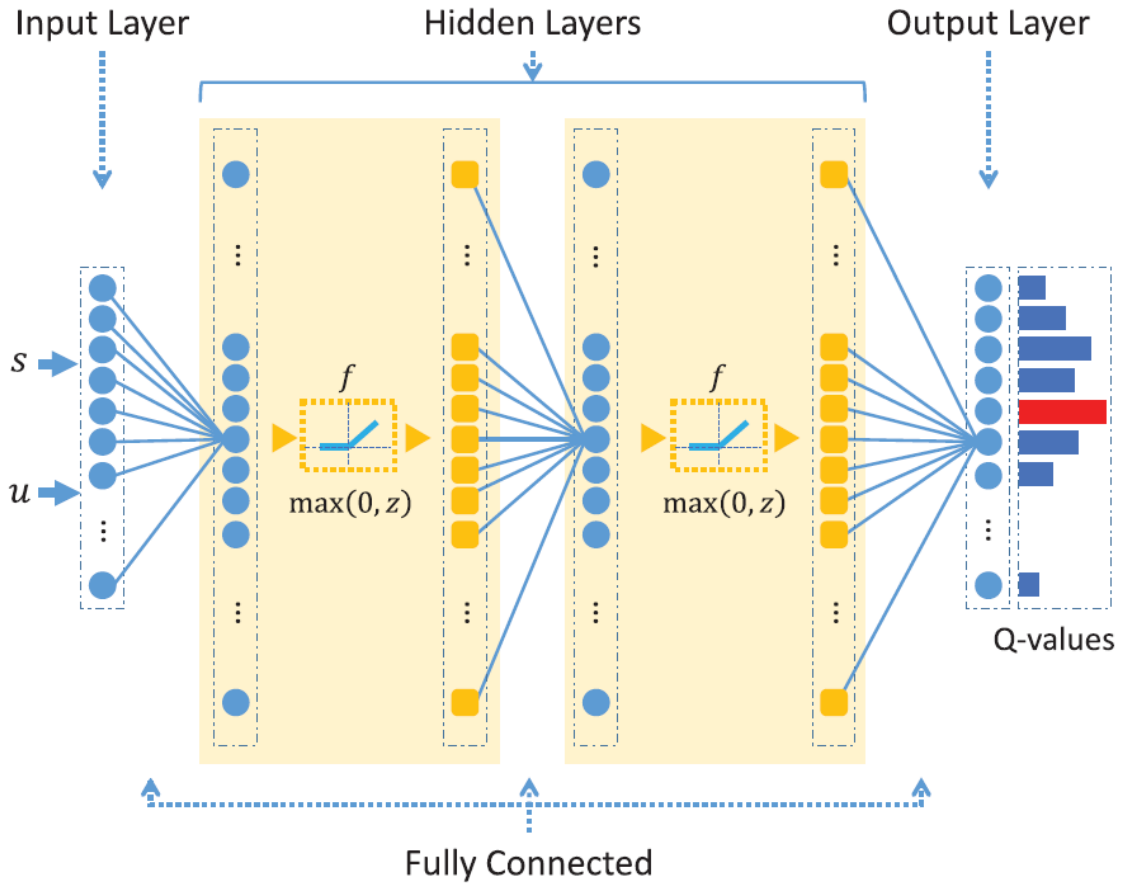
✓ Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning



- ✓ Finite action set consisting of control fragments.
- ✓ extracted from precomputed tracking controllers

Implementation Example

✓ Q-Network: The input layer is a state vector that models the simulation state s and the task parameter u



Implementation Example

ALGORITHM 1: Learning of Q-network

initialize $D \leftarrow \emptyset$
initialize a Q-network with random parameters $\boldsymbol{\theta}$
backup current parameters $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$
loop
    choose a starting state $x_0 = (s_0, u_0, \tilde{a}_0)$
    $t \leftarrow 0$
    while $x_t \notin X_{\text{fail}}$ **and** $t < T_{\text{episode}}$ **do**
        select an action a_t according to $x_t = (s_t, u_t, \tilde{a}_t)$:
            with probability ε_r select a random action
            with probability ε_o select a_t s.t. $\langle \tilde{a}_t, a_t \rangle \in O$
            otherwise select $a_t = \text{argmax}_a \, Q(x_t, a, \boldsymbol{\theta})$
        $x_{t+1} \leftarrow T(x_t, a_t)$; $r_t \leftarrow R(x_t, a_t, x_{t+1})$
        store transition tuple (x_t, a_t, r_t, x_{t+1}) in D
        update $\boldsymbol{\theta}$ with batch stochastic gradient descent
        every N_{backup} steps backup $\boldsymbol{\theta}$ to $\hat{\boldsymbol{\theta}}$
        $t \leftarrow t + 1$
    end while
end loop

Value Function Approximation

Advantage

Solving memory problem

- ✓ State action pair table is very big

Generalization

- ✓ For non-seen states there can be approximation of some actions