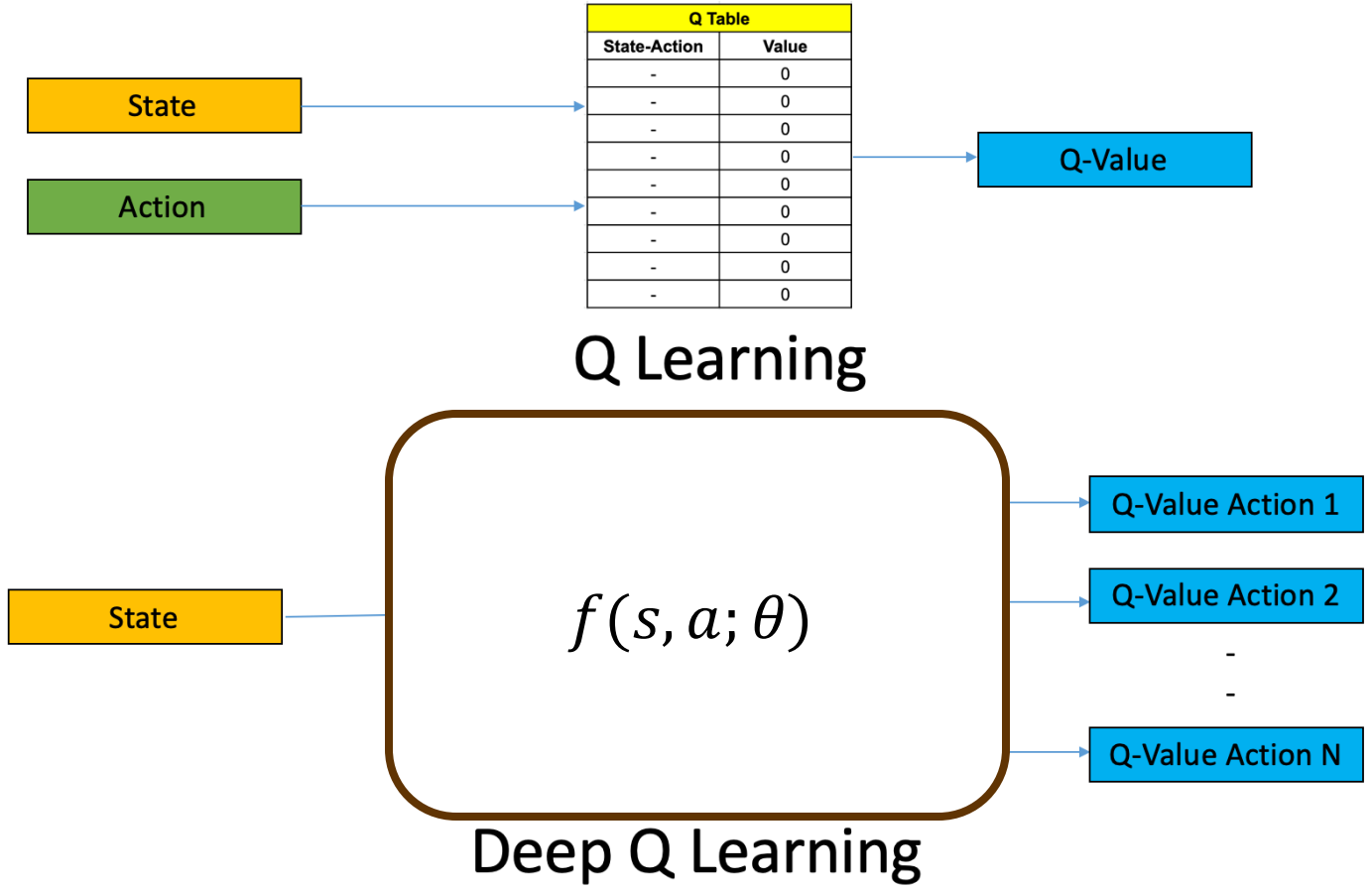
지능 시스템
Intelligent Systems

Lecture 7 – Deep Q Network

From Last Lecture



Revision – Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$h_1 = a\left(w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2\right)$$
$$h_2 = a\left(w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2\right)$$

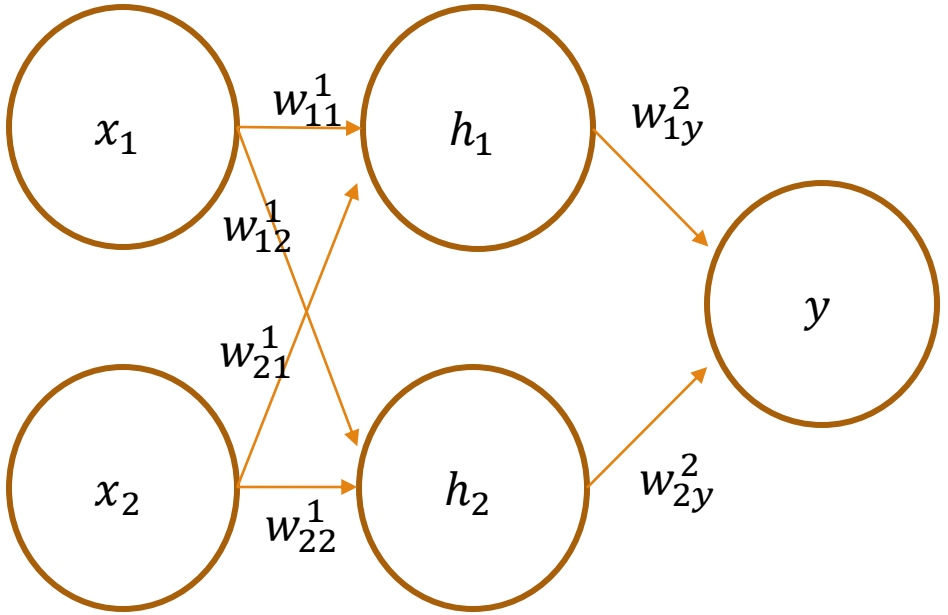
And we can combine these expressions,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Where (1) denotes that the weights are between input and the first hidden layer. (Not power)



We denote the activation function as $a()$

Revision – Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

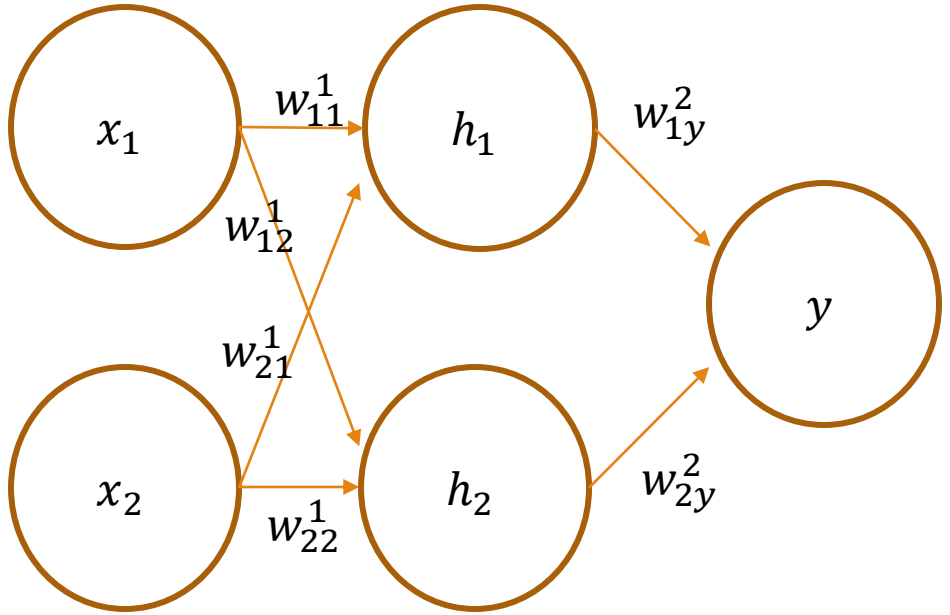
$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Then,

$$y = a(\boldsymbol{w}^{(2)}\boldsymbol{h})$$

Where

$$\boldsymbol{w}^{(2)} = \begin{pmatrix} w_{by}^2 & w_{1y}^2 & w_{2y}^2 \end{pmatrix}$$



We denote the activation function as $a()$

Revision – Finding optimal weights – Loss function

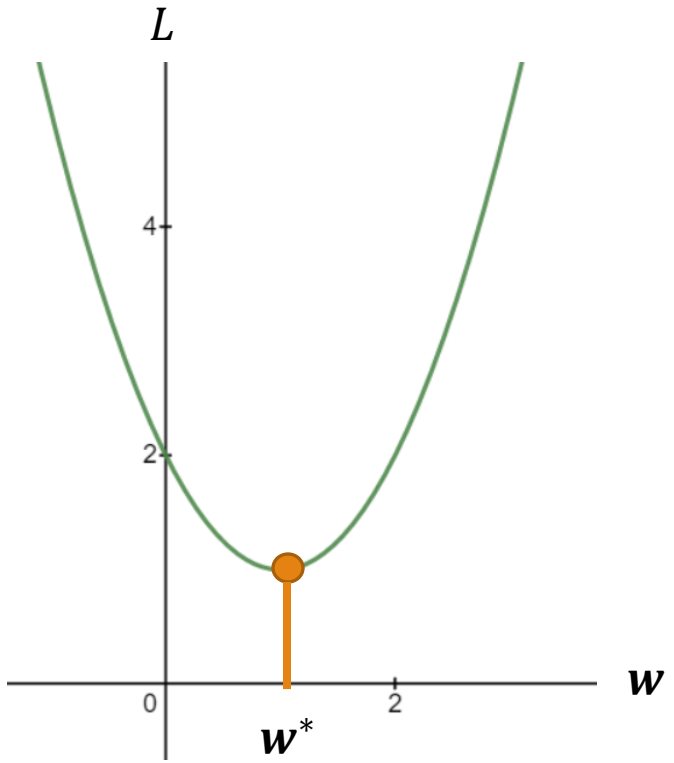
Consider: $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$

Our task is finding weights, $\boldsymbol{w}$, that minimizes the loss, L.

From intuition, we can simply find the optimal weights is finding the turning points of the loss function by differentiating it.

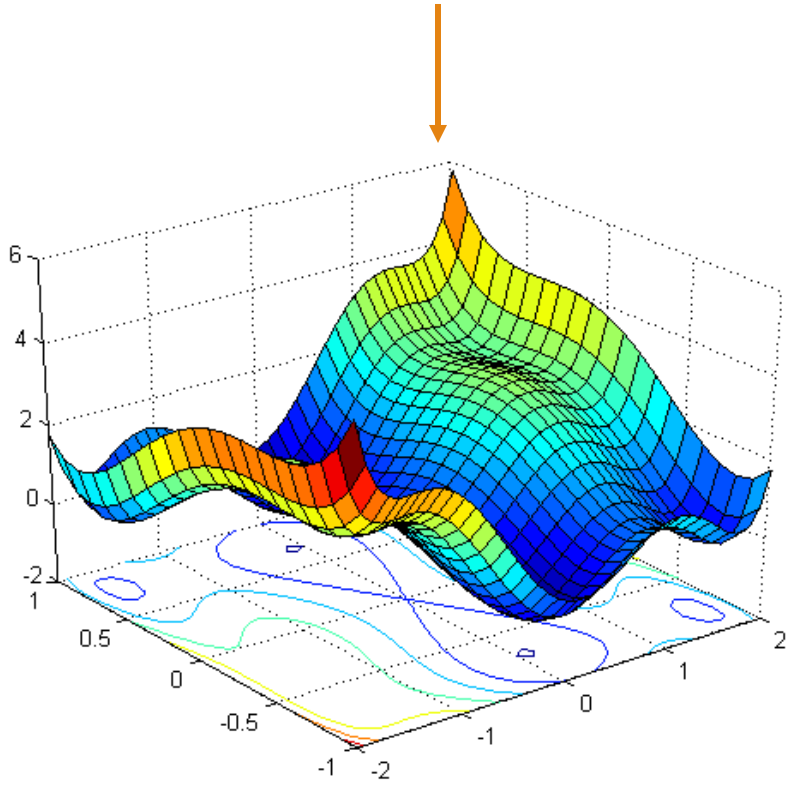
$$\boldsymbol{w}^* = \boldsymbol{argmin}_{\boldsymbol{w}}(L(\boldsymbol{x}, \boldsymbol{w}))$$

$$0 = \frac{\partial L}{\partial \boldsymbol{w}} |_{\boldsymbol{w}=\boldsymbol{w}^*}$$



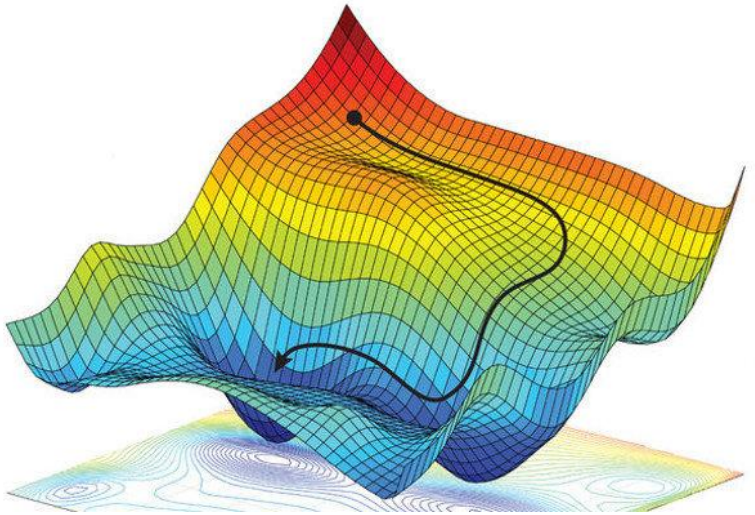
Revision – Finding optimal weights – Loss function

Example loss function with two weights



Actual loss functions with billions of weight parameters are computationally impossible to compute the full derivative.

So, without the derivative, how can we find the optimal weights?

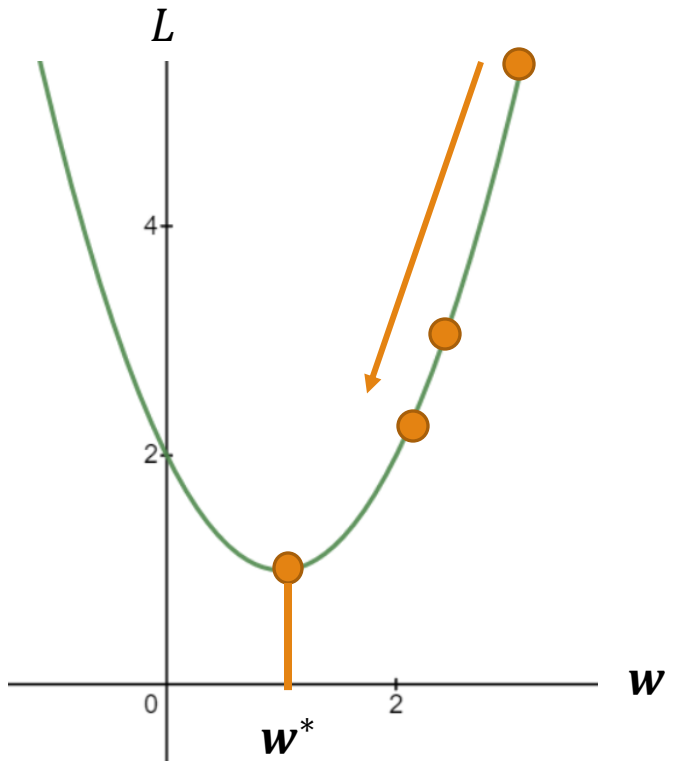


Revision – Stochastic Gradient Descent

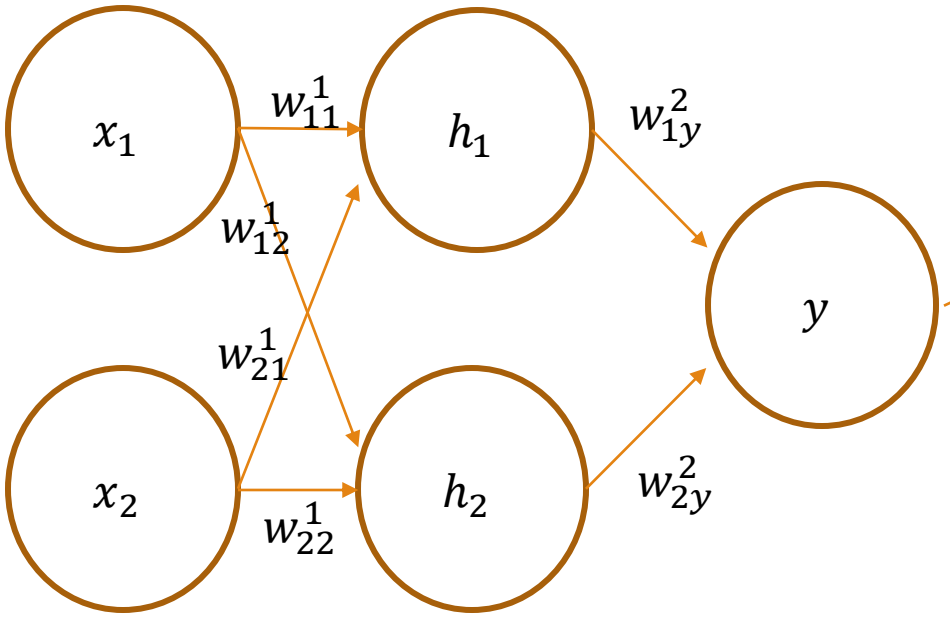
Stochastic Gradient Descent: Applying mean loss of randomly sampled minibatch data and performing gradient descent algorithm.

$$L = \frac{1}{|B|}\sum_{B}\frac{1}{2}\sum_{k}(y_k - t_k)^2$$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta\frac{\partial L}{\partial \boldsymbol{w}}$$



Revision – Gradient Computation



$$L = \frac{1}{|B|}\sum_B \frac{1}{2}\sum_k (y_k - t_k)^2$$

Label, t

Say, we want to compute $\frac{\partial L}{\partial w_{11}^1}$, then,

$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{|B|}\sum_B \sum_k (y_k - t_k)\frac{\partial y_k}{\partial w_{11}^1}$$

Where,

$$\frac{\partial y_k}{\partial w_{11}^1} = \frac{\partial}{\partial \boldsymbol{h}} a(\boldsymbol{w}^{(2)}\boldsymbol{h})\frac{\partial \boldsymbol{h}}{\partial w_{11}^1}$$

And so on…….

Revision – Loss function in Q-Learning

Unlike standard Supervised Learning(SL), in Reinforcement Learning(RL), the correct answer, which we call **a label, does not exist.**

So how can we compute a loss?

Consider the Q update equation of Q-Learning:

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a^*) - \mathrm{Q}(s_t, a_t)\big)$$

When we reach the Optimal Q values, the error term becomes 0. → Our Objective!
Similar to the goal is SL to make a loss 0 (or near zero to avoid overfitting).

We can compute $\mathrm{Q}(s_t, a_t)$ and $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ by forward propagation of our current Neural Network.

Revision – Loss function in Q-Learning

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a^*) - \mathrm{Q}(s_t, a_t)\big)$$

When we reach the Optimal Q values, the error term becomes 0. → Our Objective!
Similar to the goal is SL to make a loss 0 (or near zero to avoid overfitting).

We can compute $\mathrm{Q}(s_t, a_t)$ and $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ by forward propagation of our current Neural Network.

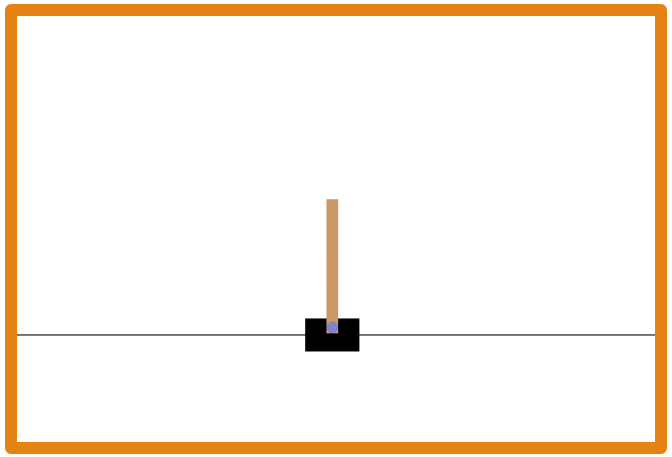
We assume that $r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$ is a more accurate Q value than $\mathrm{Q}(s_t, a_t)$, hence we call it a target that $\mathrm{Q}(s_t, a_t)$ must follow.

$$\mathrm{y} = \mathrm{Q}(s_t, a_t)$$

$$t = r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$$

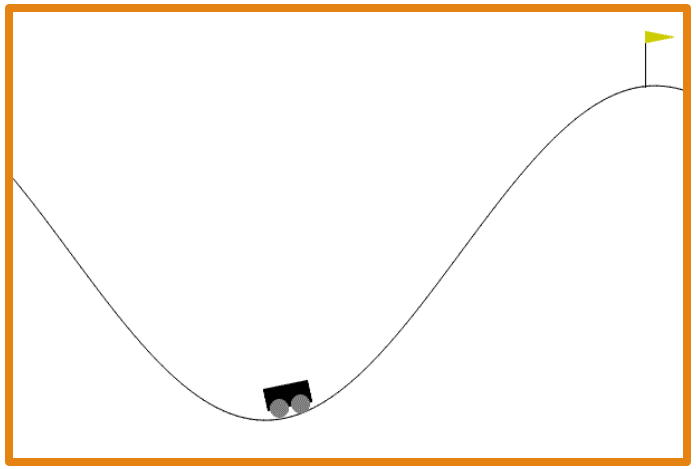
$$L = \frac{1}{2}\sum_k (y_k - t_k)^2$$

State Representation



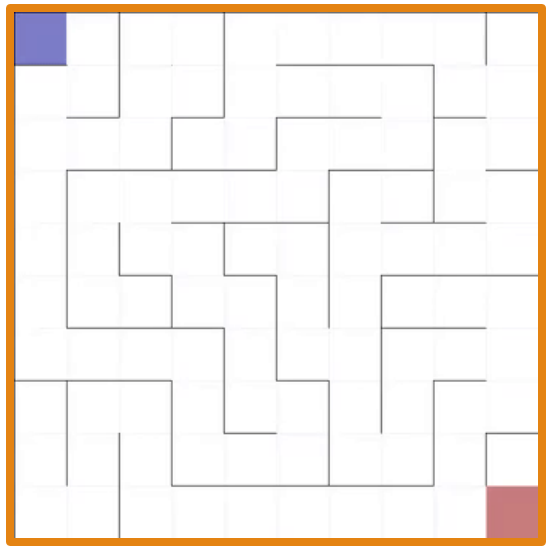
Cartpole:

- Cart position
- Pole angle
- Cart velocity
- Pole tip's angular velocity



Mountain Car:

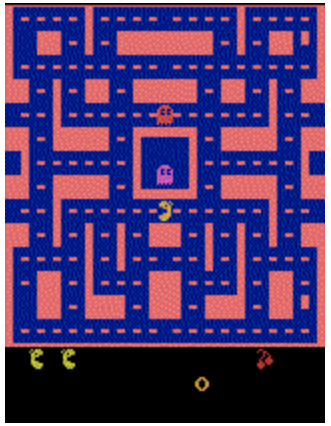
- Car position
- Car velocity

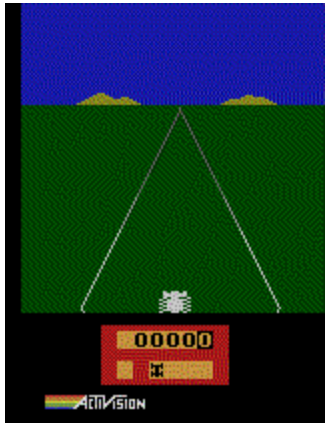


Maze:

- Current position

State Representation

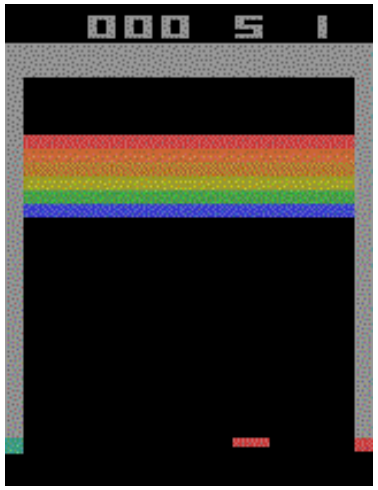


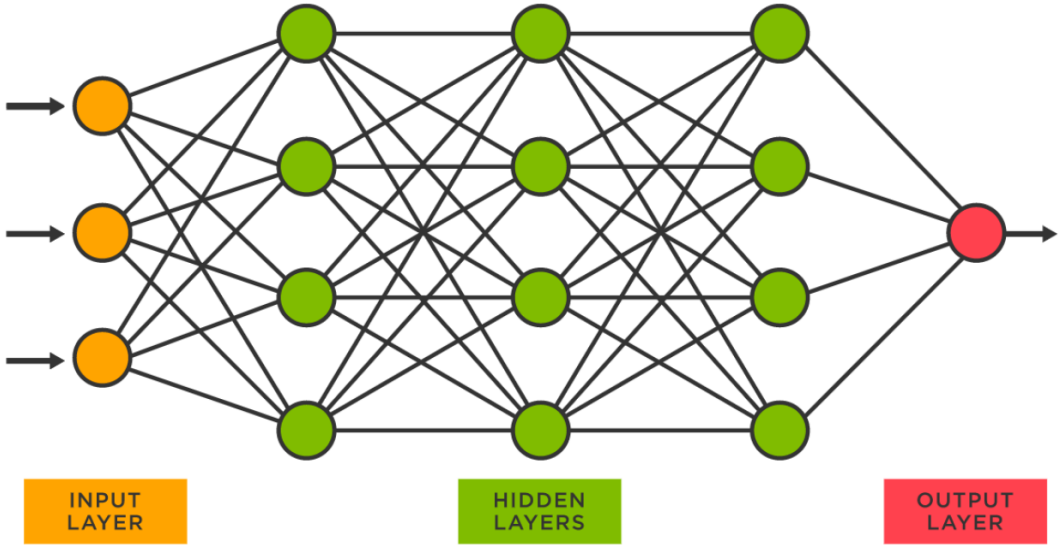


Atari Games:

States….?

Image as an input

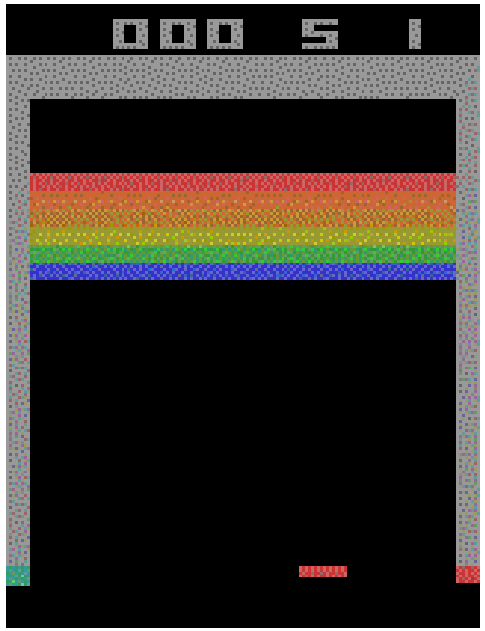




$Q(state)$

Image Representation

Pixel-wise Representation



$$\begin{pmatrix} 0.2342 \\ 0.1235 \\ 1.3456 \\ 4.3156 \\ 0.0024 \\ 0.0255 \\ 1.1032 \\ 2.3109 \\ \vdots \\ 0.8724 \end{pmatrix}$$

- Pixel information(grey scale/RGB) are sorted in one column vector.

Pros:

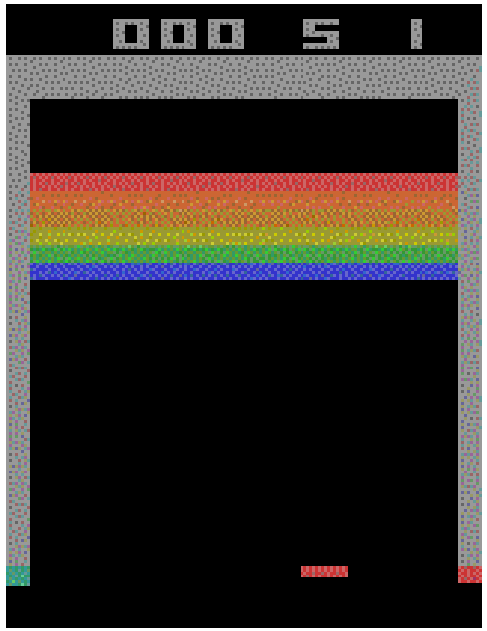
- Easy to extract from an image.
- Raw-wise information can easily be obtained due to pixel-data in the same raw being close to each other in the vector.

Cons:

- No information about the features of an image.
  - Edges
  - 2D Shapes
  - Location of particular objects

Image Representation

Feature Representation



$$\begin{pmatrix} Position\ of\ the\ bar \\ Number\ of\ bricks\ left \\ Position\ of\ the\ ball \\ Velocity\ of\ the\ ball \\ Current\ Score \\ \vdots \\ Background \end{pmatrix}$$

- Features of an image are sorted in one column vector.

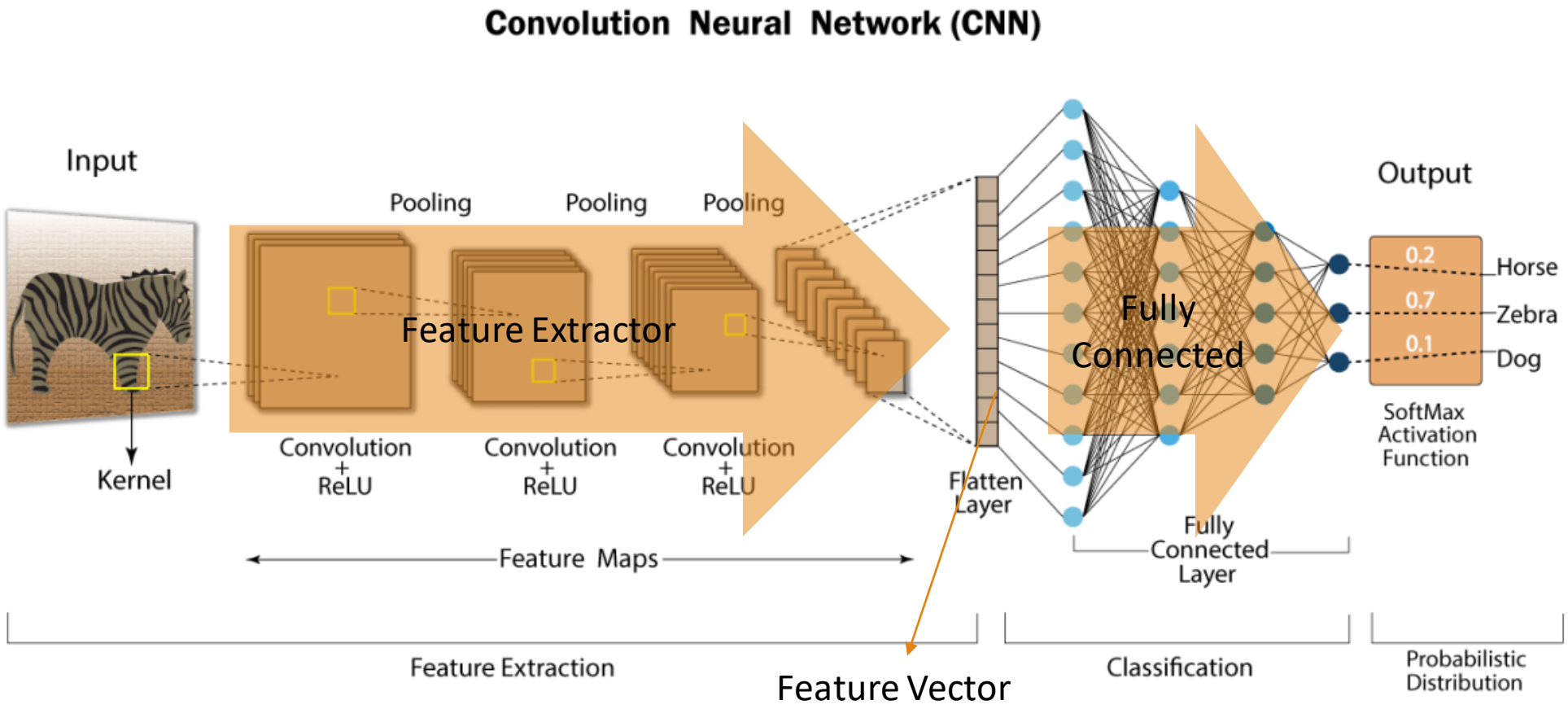
Pros:

- The vector holds information about the features of an image.
  - Edges
  - 2D Shapes
  - Location of particular objects

Cons:

- Difficult to extract the feature vector

Convolutional Neural Network (CNN)



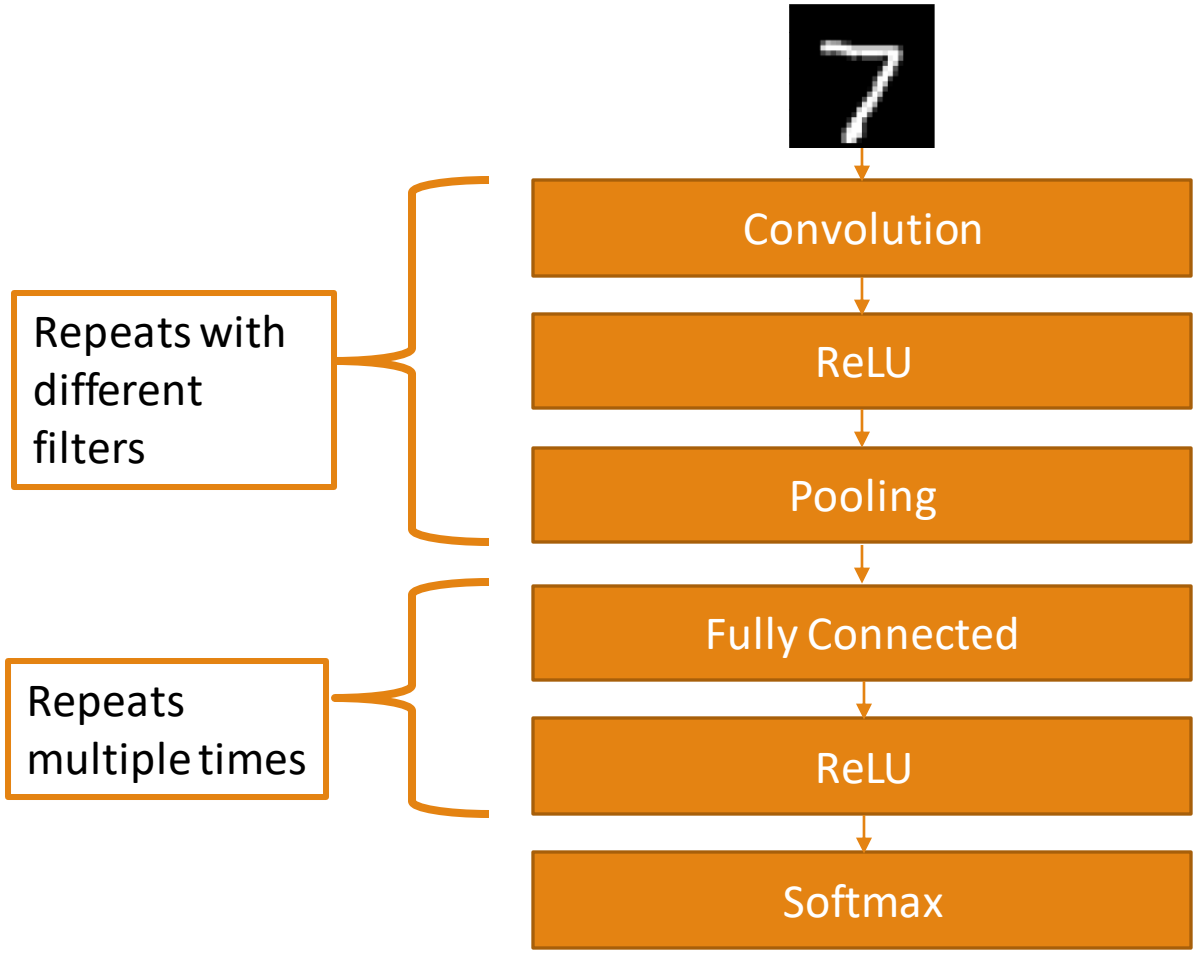
CNN Structure

CNN의 기본 구조:

Feature Extractor:
Convolution → ReLU → Pooling(optional)
Fully Connected:
Linear Layer → ReLU

Convolution: Utilizes various filters to extract features from the input image.

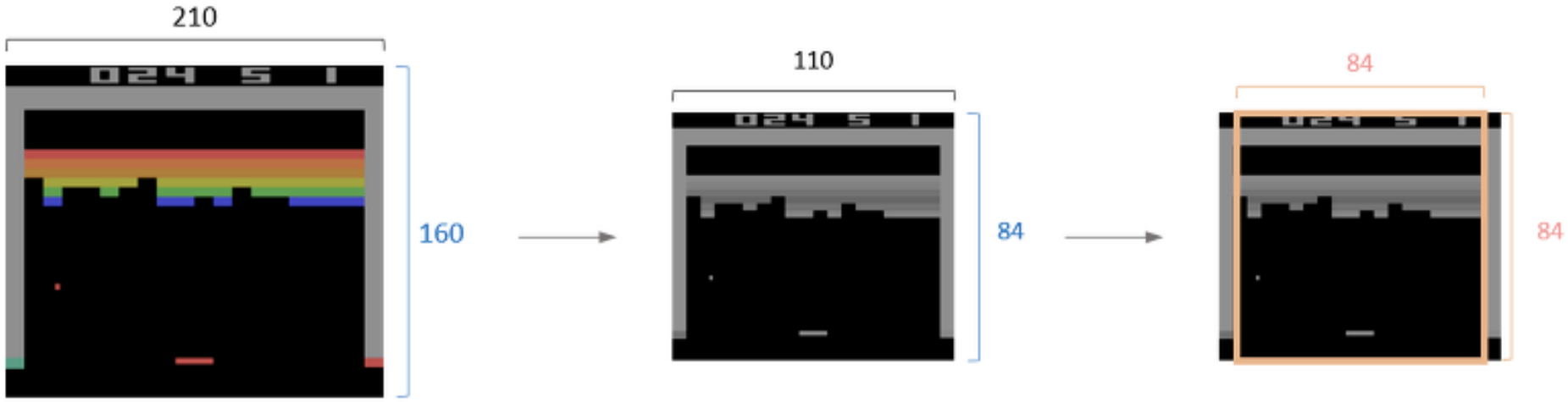
Pooling(optional): Downsampling/subsampling
→ Accelerates computation by reducing the size of a matrix.



Deep Q Network (DQN)

- Convolutional Neural Network
  - Image Pre-processing
  - Network Structure
- Replay Buffer
- Q-Learning
  - Behaviour and Target Network

Image Pre-processing



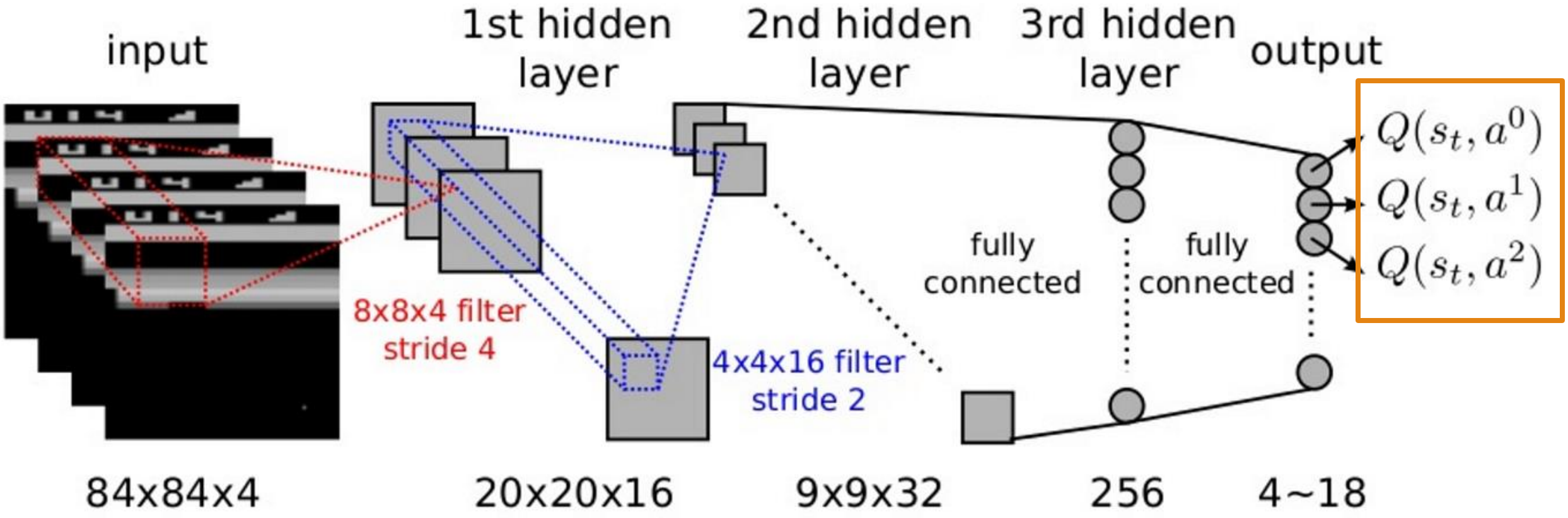
128 color,
210 X 160 pixels

gray scale,
110 X 84 pixels

cropping
84 X 84 pixels

Transforms the image to gray scale & Cuts the image to a fixed size.
→ Enables to use the same network structure for all 57 Atari Games
→ Simplifies the learning of CNN

DQN – Network Structure

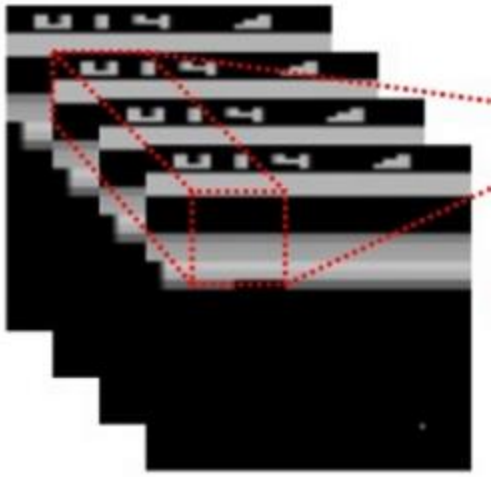


Replay Buffer

Experience Tuple

(State, Action, Reward, Next_State, Done)

State:

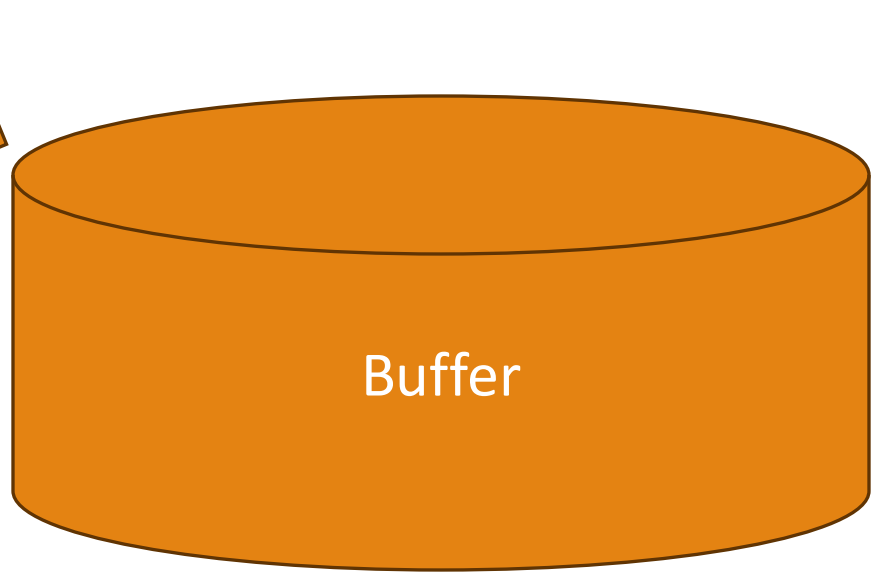


Action:

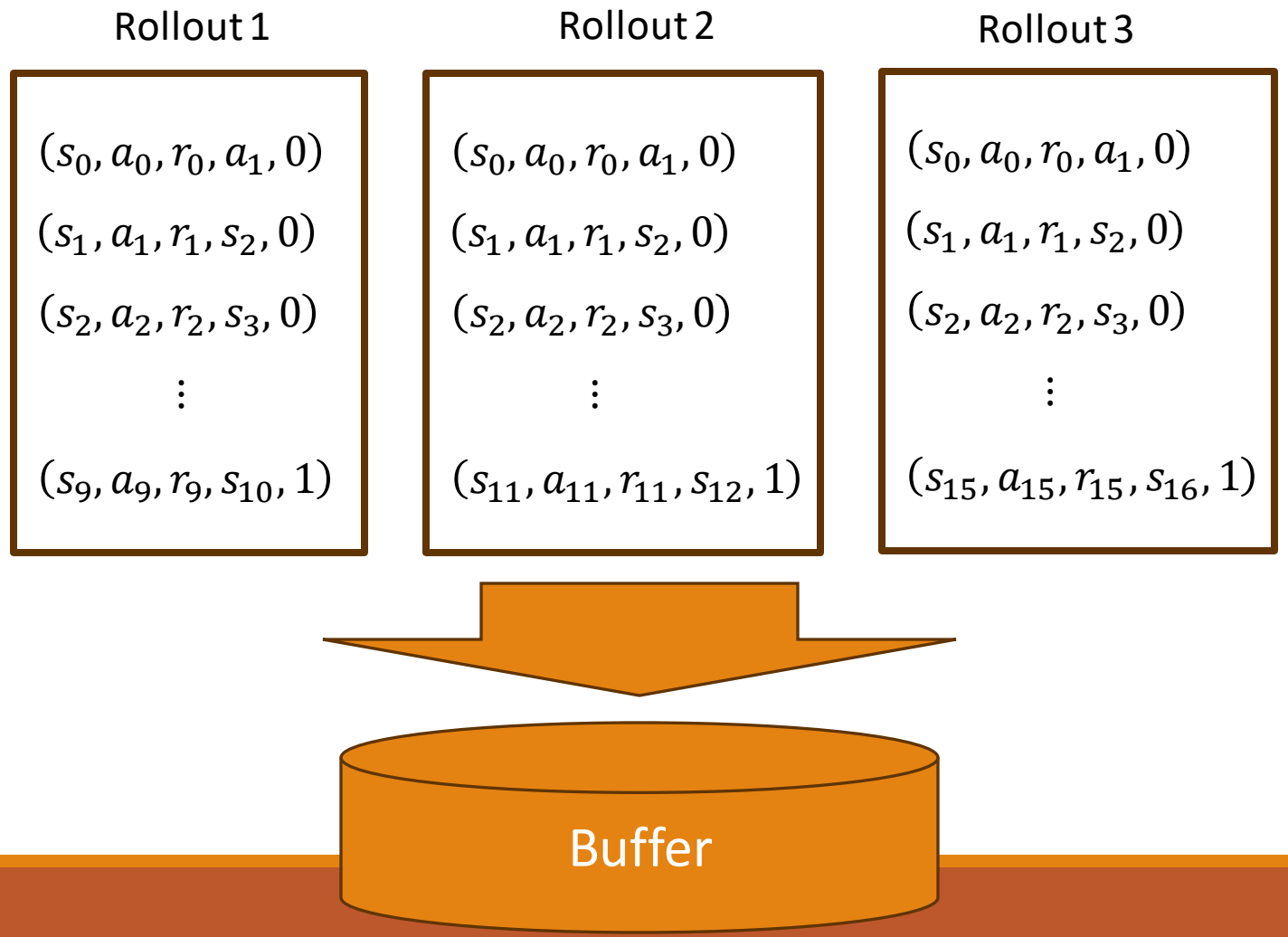
Num	Action
0	NOOP
1	FIRE
2	RIGHT
3	LEFT

Reward:

Internal Score of the game



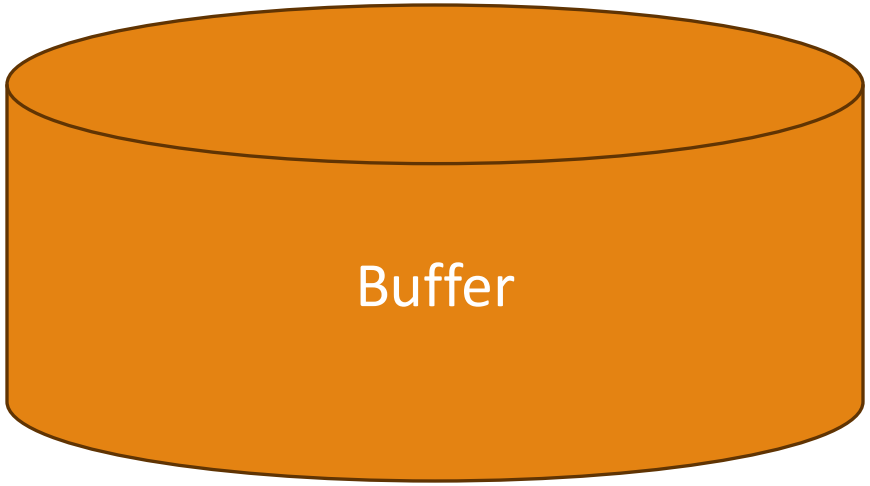
Replay Buffer



Number of step data in buffer?

Replay Buffer

- The size of the buffer is arbitrary.
  - DQN algorithms used the buffer with a size of **1M**
- The algorithm **initially only samples rollout data** and fills the buffer without learning.
- Once the buffer is fully filled, the learning then starts by extracting data from the buffer with **uniform probability**.



Q-Learning

$$Q(s_t, a_t) \cong r_t + \gamma Q(s_{t+1}, a^*)$$

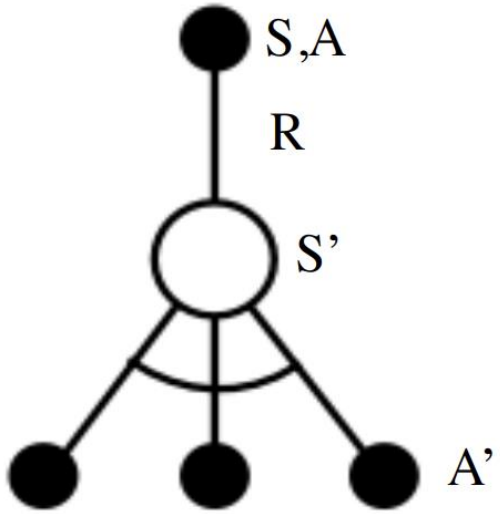
where, $a^* = \underset{a}{\text{argmax}}\, Q(s_{t+1}, a)$

Next state

Alternative successor action

$$Q(s_t, a_t) \cong r_t + \gamma Q(s_{t+1}, a^*) = r_t + \gamma \max_a Q(s_{t+1}, a)$$

Q-Learning takes the best possible value of Q as the target.



Deep Q Learning – Loss function

$$Loss_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \,|s,a\right]$$

$\rho(.)$: uniform distribution sampling from the replay buffer

$\mathcal{E}$: Emulator (the Atari Environment)

θ_i : the parameter set of the CNN at i^{th} iteration of training(learning)

s', a' : Next State, Next Action

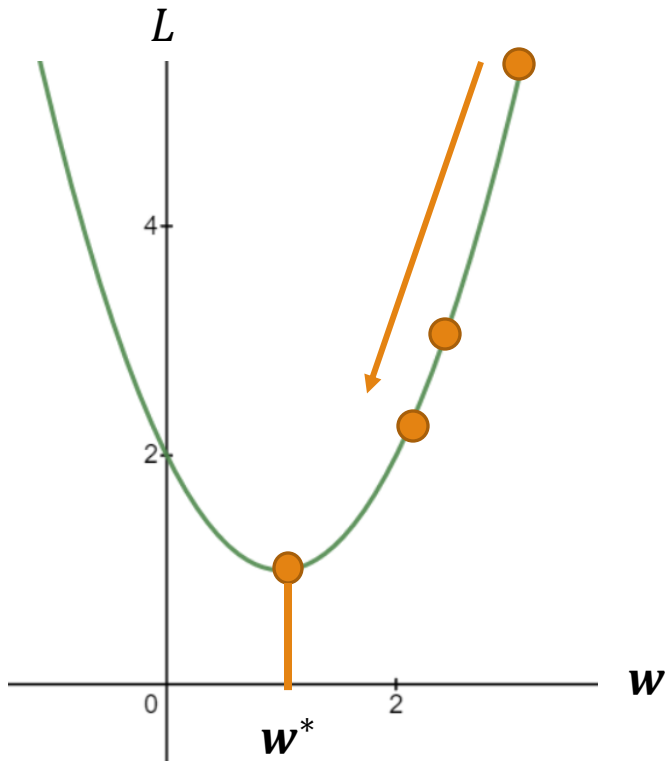
Revision – Stochastic Gradient Descent

Stochastic Gradient Descent: Applying mean loss of randomly sampled minibatch data and performing gradient descent algorithm.

$$L = \frac{1}{|B|}\sum_{B}\frac{1}{2}\sum_{k}(y_k - t_k)^2$$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta\frac{\partial L}{\partial \boldsymbol{w}}$$

Remember? We needed the derivative of the loss wrt the parameter



Deep Q Learning – Loss function

$$Loss_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \,|s,a\right]$$

$$\nabla_{\theta_i} Loss_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[\left(y_i - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

The gradient of loss

Computed via backward propagation

Deep Q Learning – Loss function

$$Loss_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$$ → Behavior Policy

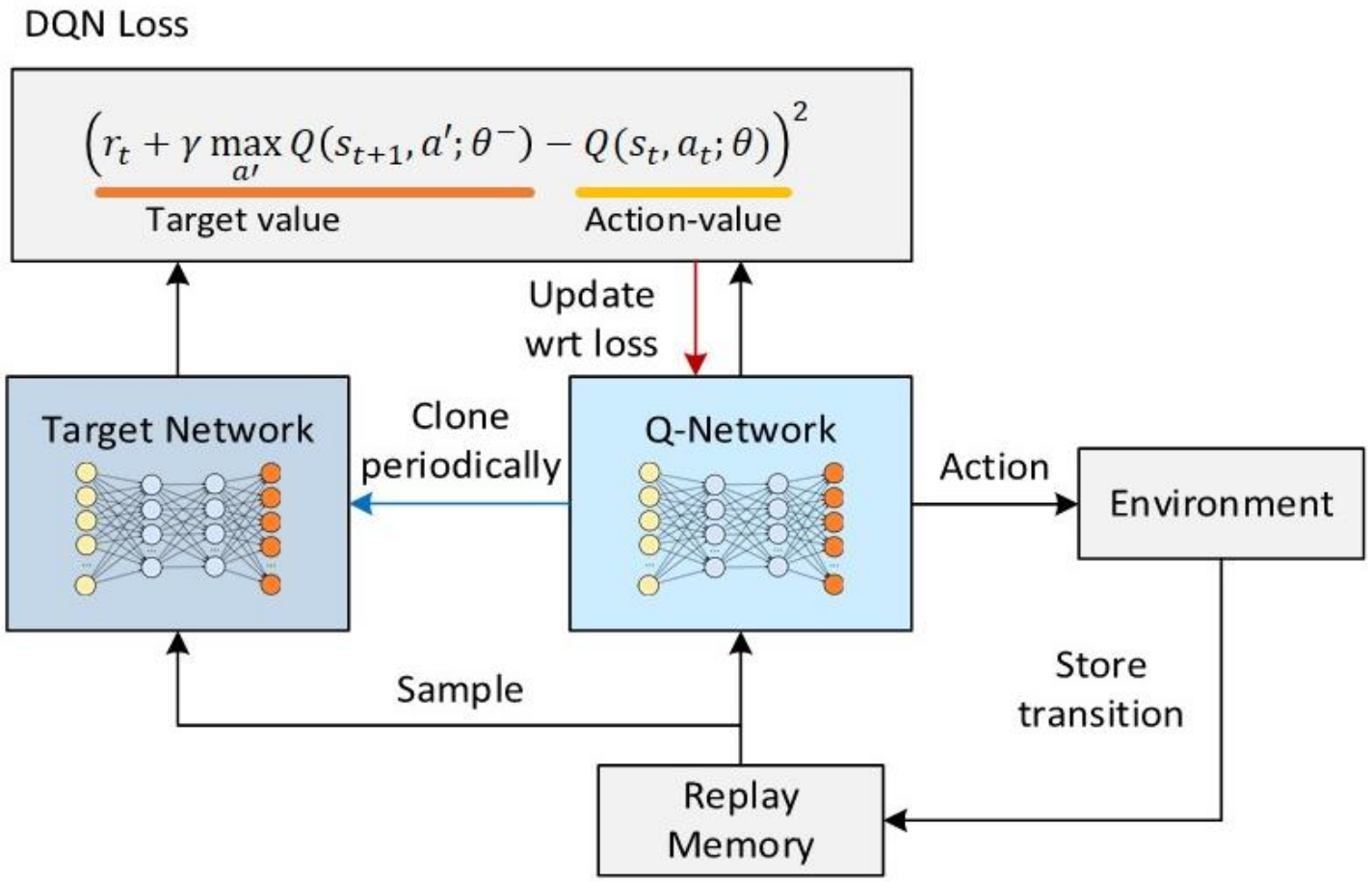
$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a\right]$$ → Target Policy

$$\nabla_{\theta_i} Loss_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[\left(y_i - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

The DQN algorithm is an off-policy algorithm.
It uses different behavior policy and target policy

Here, the target policy is an old version of the behavior policy

DQN algorithm flow



DQN algorithm

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

추가 설명

(1) 변수 초기화

- replay memory D 초기화 (D의 메모리 용량은 N)
- action-value function Q 초기화 (= random한 weight를 지니도록 초기화)

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights

변수 초기화

(2-1) 반복문 : 에피소드 별 반복 (학습 횟수)

- sequence 초기화 : s1 = {x1}
- sequence 전처리

* 전처리에 대한 내용은 아래 4.1절에서 나옴 (단순한 이미지 resize)

for episode $= 1, M$ **do**
Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

에피소드 반복문

Ref: https://ropiens.tistory.com/75

추가 설명

(2-2) 반복문 : 한 에피소드 내에서 시간에 따라 반복

- ε의 확률로 랜덤한 action a_t를 선택
- 이때 a_t를 선택하는 기준은 Q-value를 최대화 시키는 a_t를 선택
- a_t를 emulator(Atari)에서 실행하고 reward r_t와 다음 이미지 x_t+1를 관찰

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}

(a_t, r_t, x_t+1)를 취득하는 과정

- sequence를 다음과 같이 정의 : s_t+1 = s_t, a_t, x_t+1
- 그 후 sequence를 전처리하고 replay memory D에 저장
  * 다음의 sequence를 transition이라고 정의 : [Φ_t, a_t, r_t, Φ_t+1]
- D로 부터 transition을 minibatch 형태로 랜덤하게 추출함

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 $\mathcal{D}$

DQN 데이터 형태 : transitions

Ref: https://ropiens.tistory.com/75

추가 설명

- sequence를 다음과 같이 정의 : s_t+1 = s_t, a_t, x_t+1
- 그 후 sequence를 전처리하고 replay memory D에 저장
  * 다음의 sequence를 transition이라고 정의 : [Φ_t, a_t, r_t, Φ_t+1]
- D로 부터 transition을 minibatch 형태로 랜덤하게 추출함

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 $\mathcal{D}$

DQN 데이터 형태 : transitions

- 정답 값 y_j은 아래와 같이 정의하고, 이 정답값과 DQN의 판단값의 차이를 이용하여 SGD로 DQN을 최적화 시킴

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

DQN 최적화 과정

Ref: https://ropiens.tistory.com/75