In the Name of God

Iran University
of Science and
Technology

Reinforcement Learning in Control

Dr. Saeed Shamaghdari

Electrical Engineering Department
Control Group

Fall 2025 | 4041

Deep Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

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]

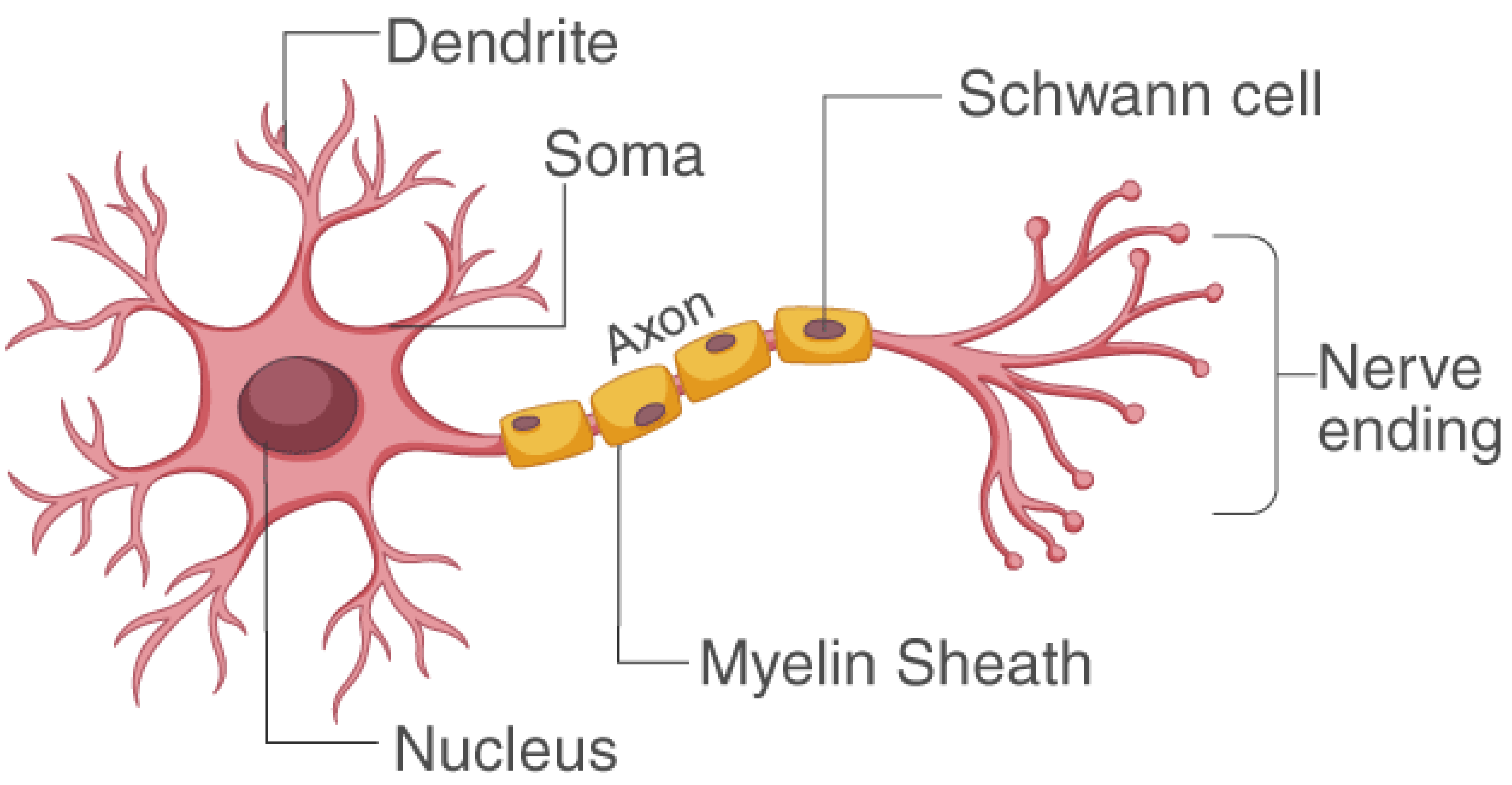
[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK.

*These authors contributed equally to this work.

26 FEBRUARY 2015 | VOL 518 | NATURE | 529

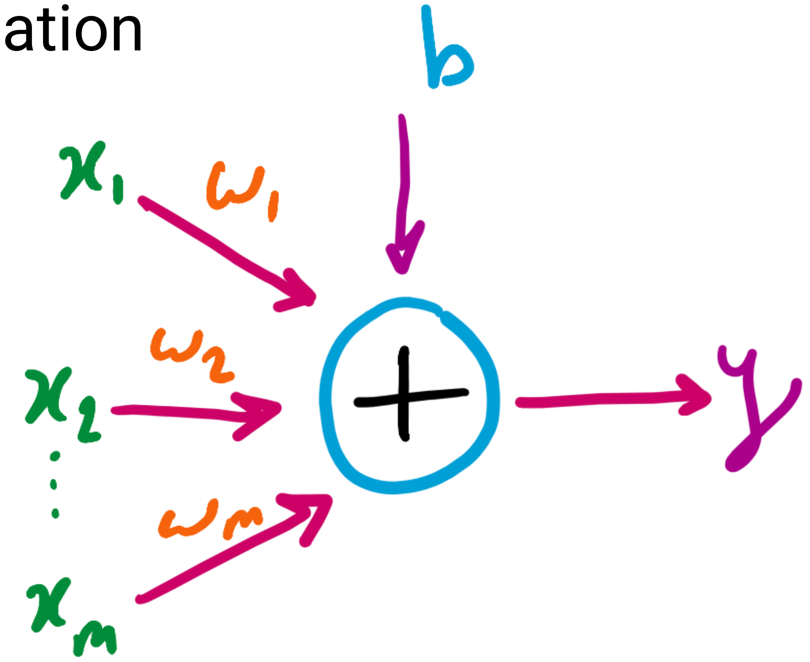
Neural Networks and Deep Learning: A Simple Review

Neuron



Neural Networks and Deep Learning: A Simple Review

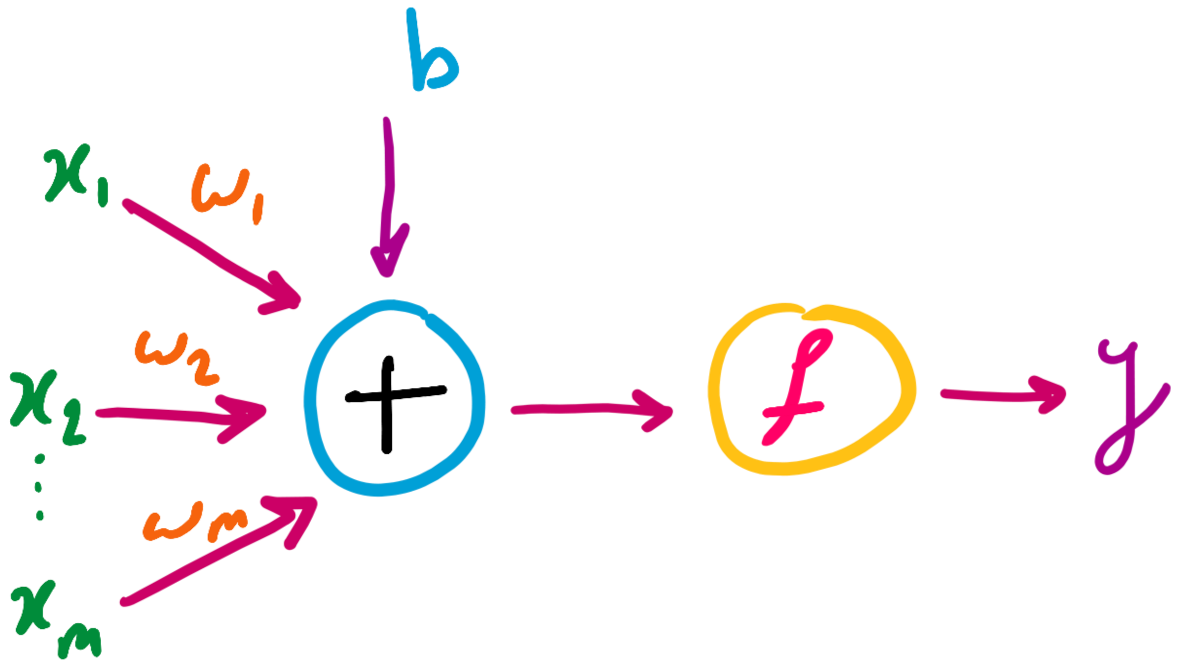
Artificial Neuron Formulation



$$y = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + b$$

Neural Networks and Deep Learning: A Simple Review

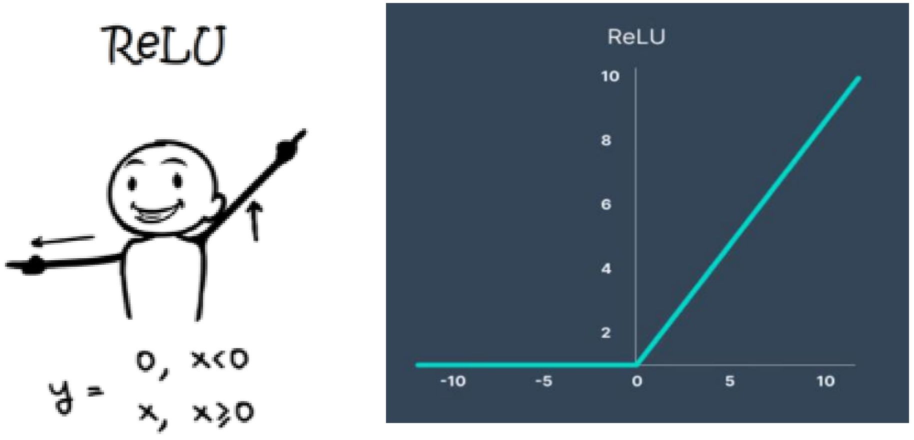
Q: Problems?

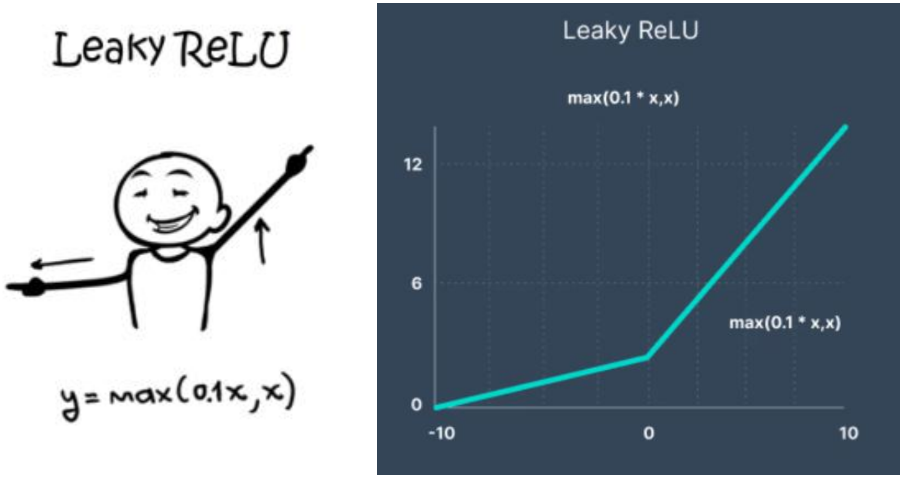


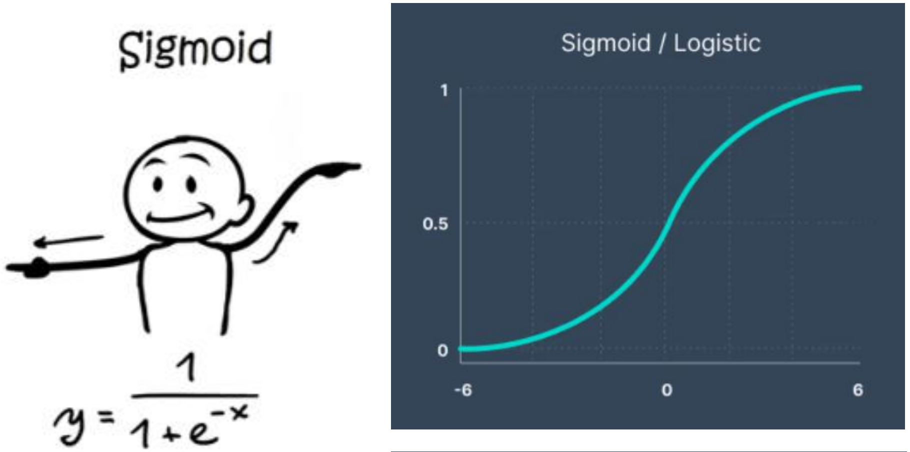
$$y = f(w_1x_1 + w_2x_2 + \cdots + w_mx_m + b) = f(w^Tx + b)$$

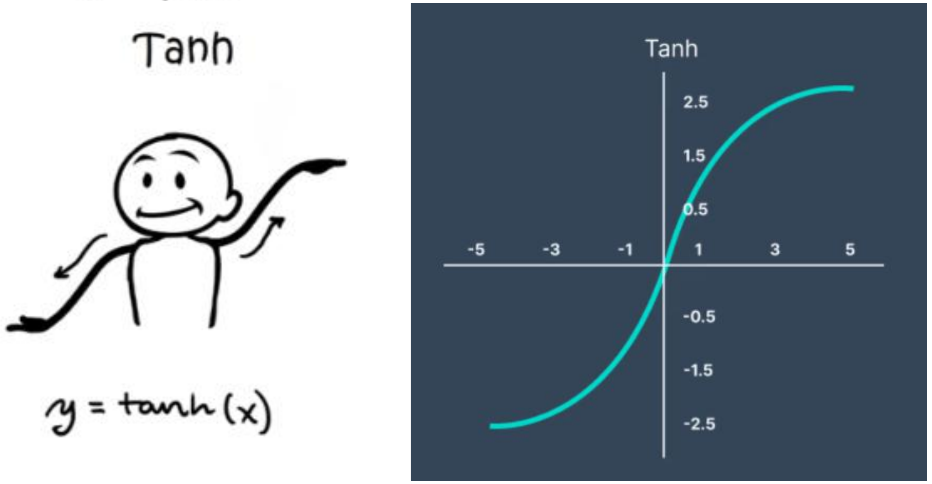
Neural Networks and Deep Learning: A Simple Review

Activation Functions



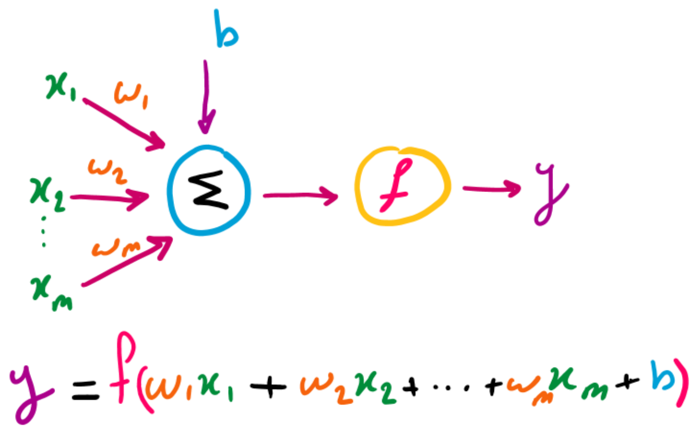


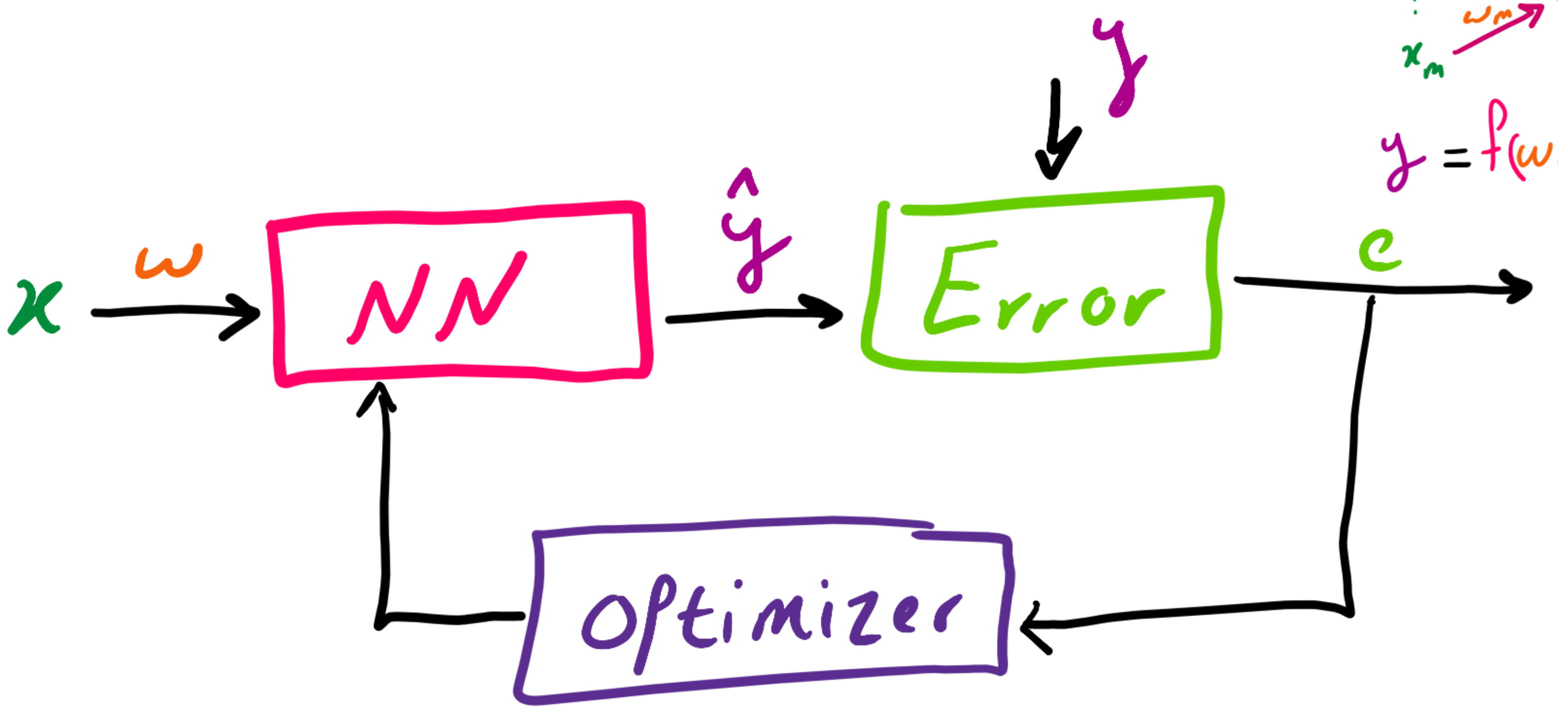




Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

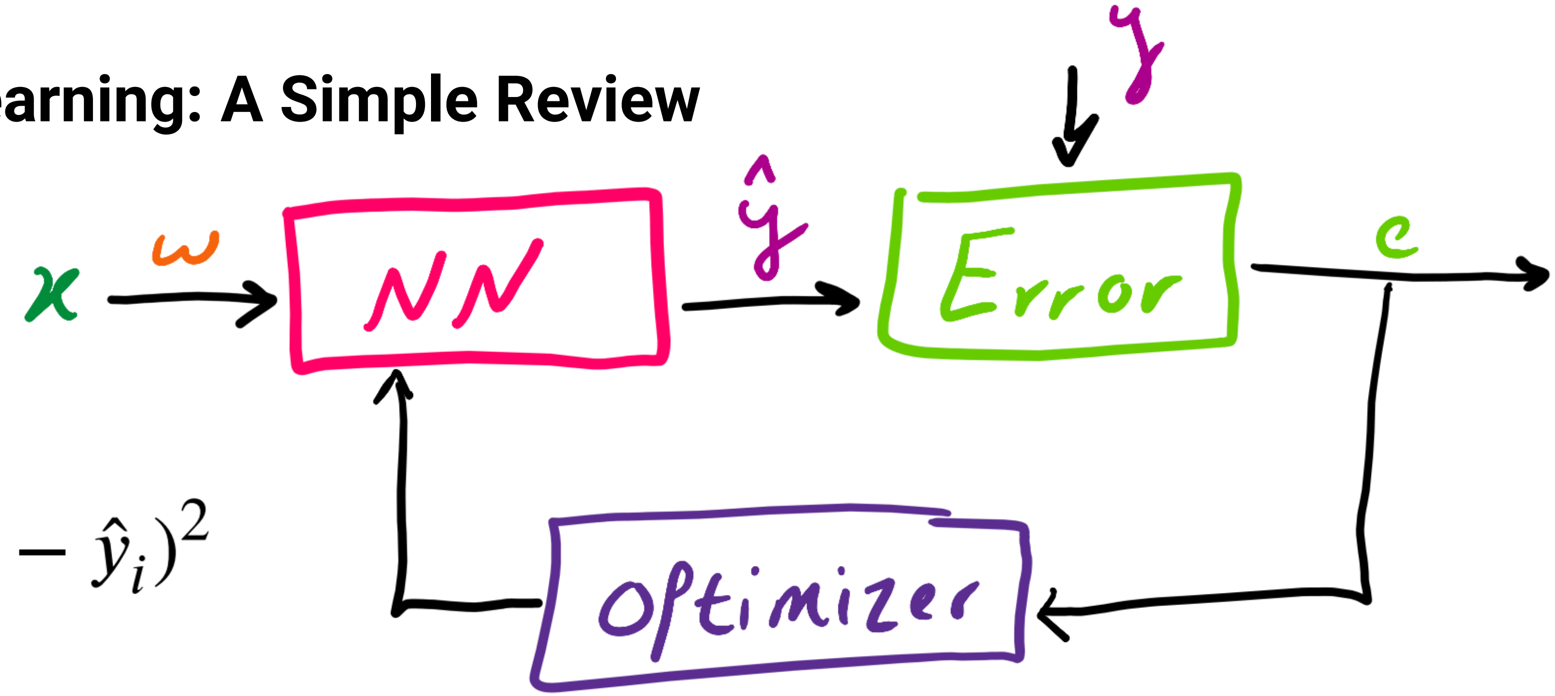




Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

Loss Function:



***MSE*:**

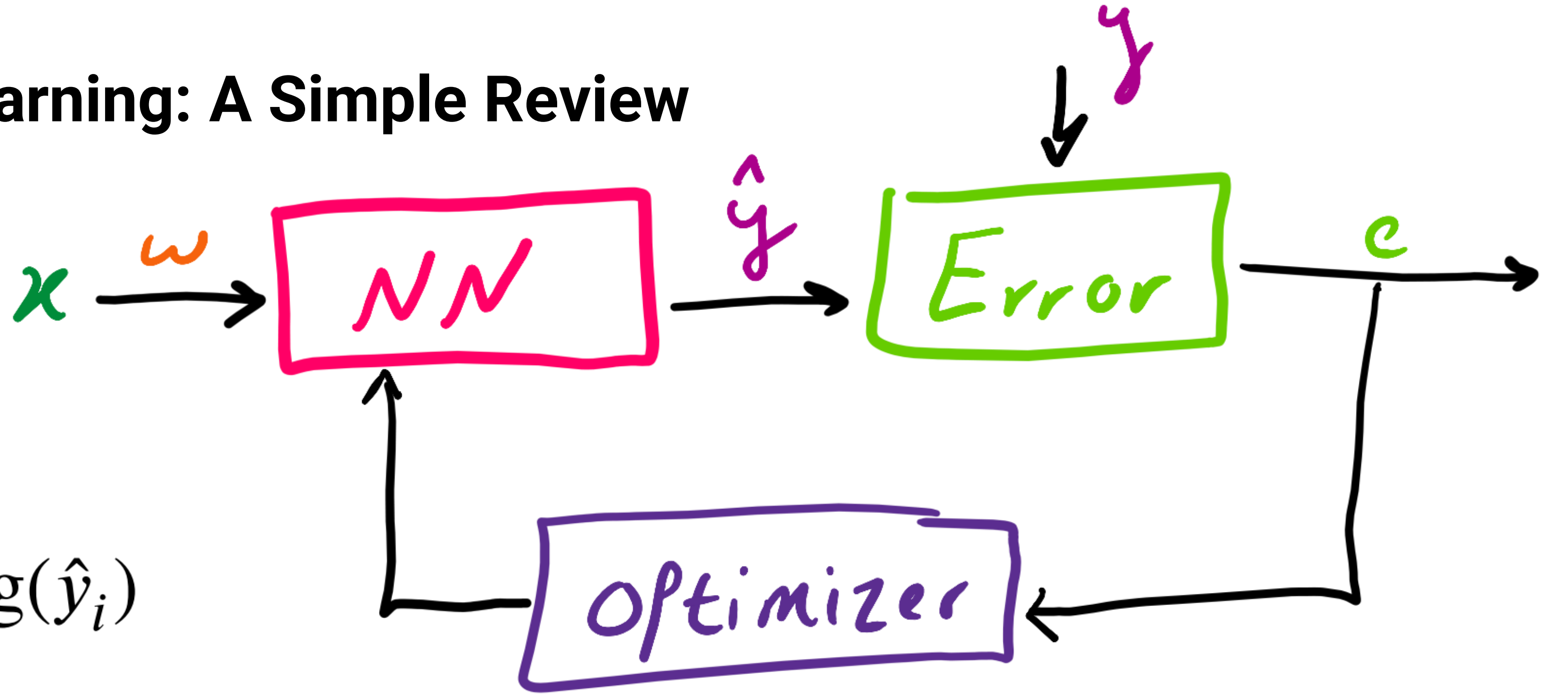
$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

***MAE*:**

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram



Loss Function:

Cross Entropy:

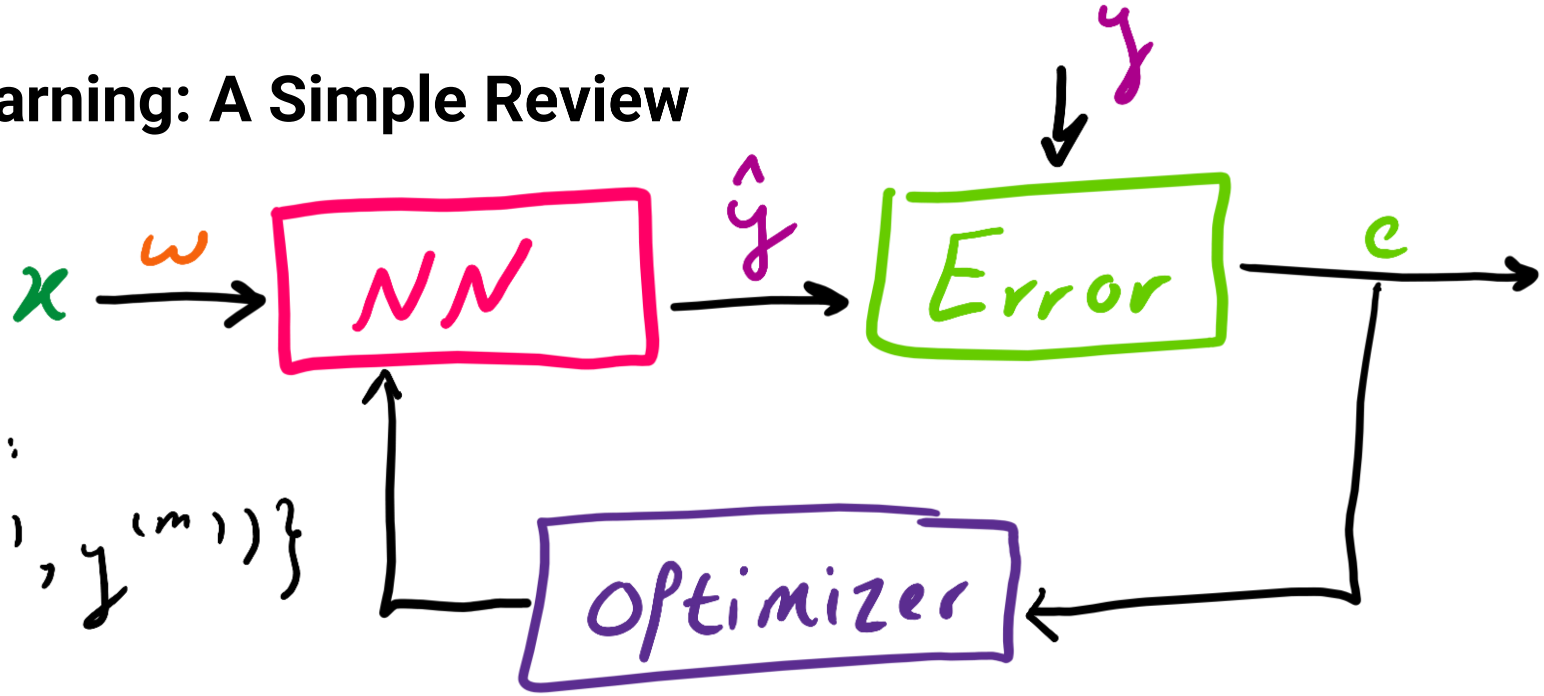
$$L(y,\hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Binary Cross Entropy:

$$L(y,\hat{y}) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram



Given m train examples:

$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

want $\hat{y}^{(i)} \approx y^{(i)}$

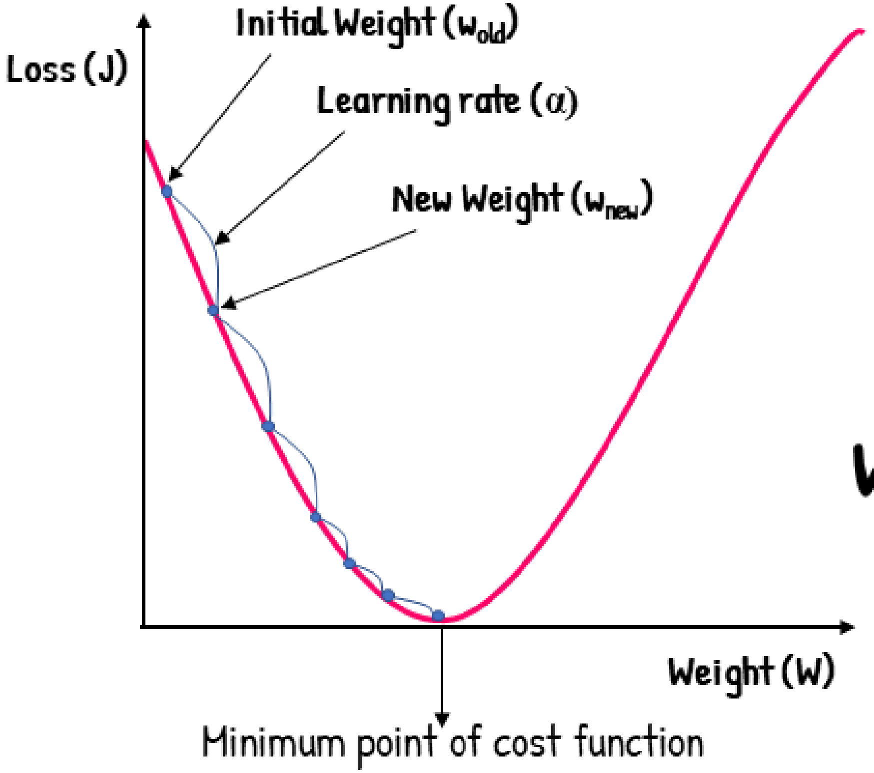
So the cost function is:

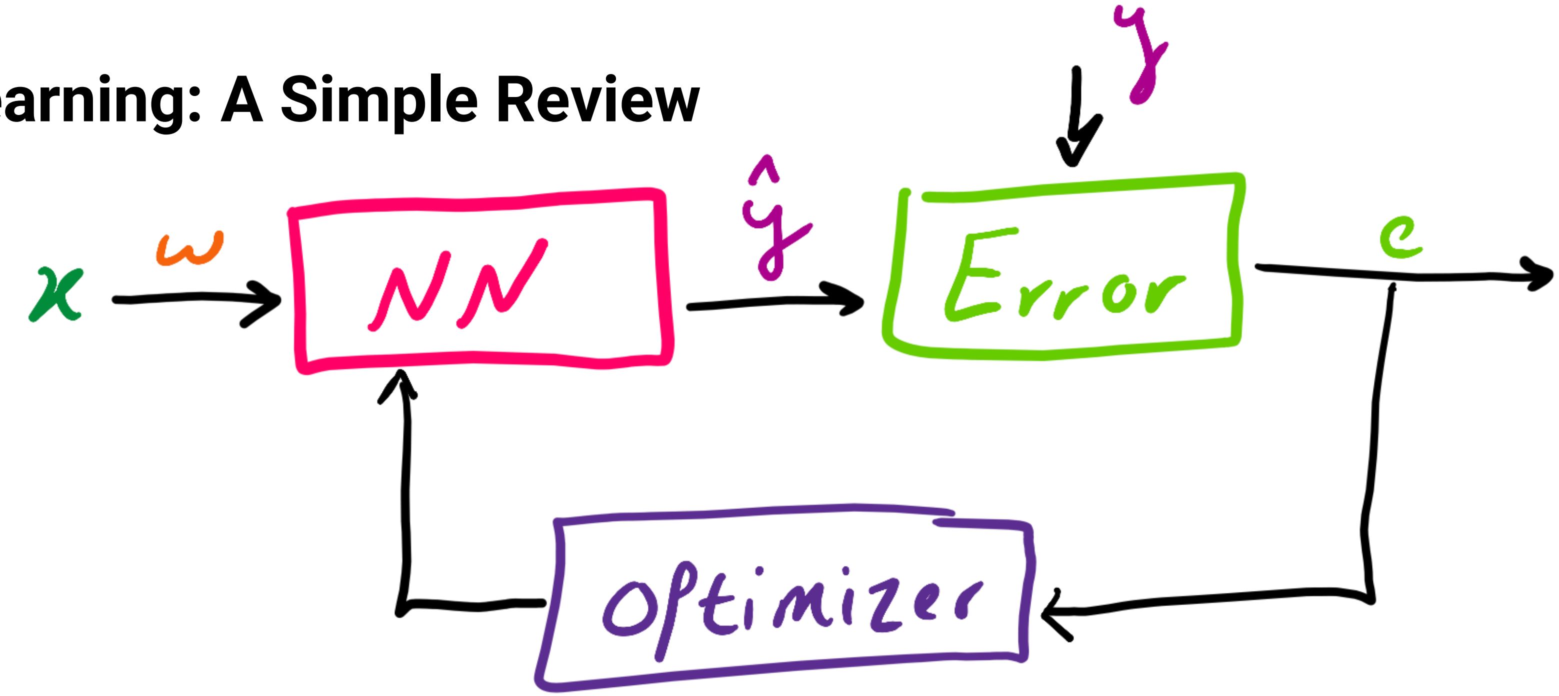
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(y,\hat{y})$$

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

Gradient Descent:

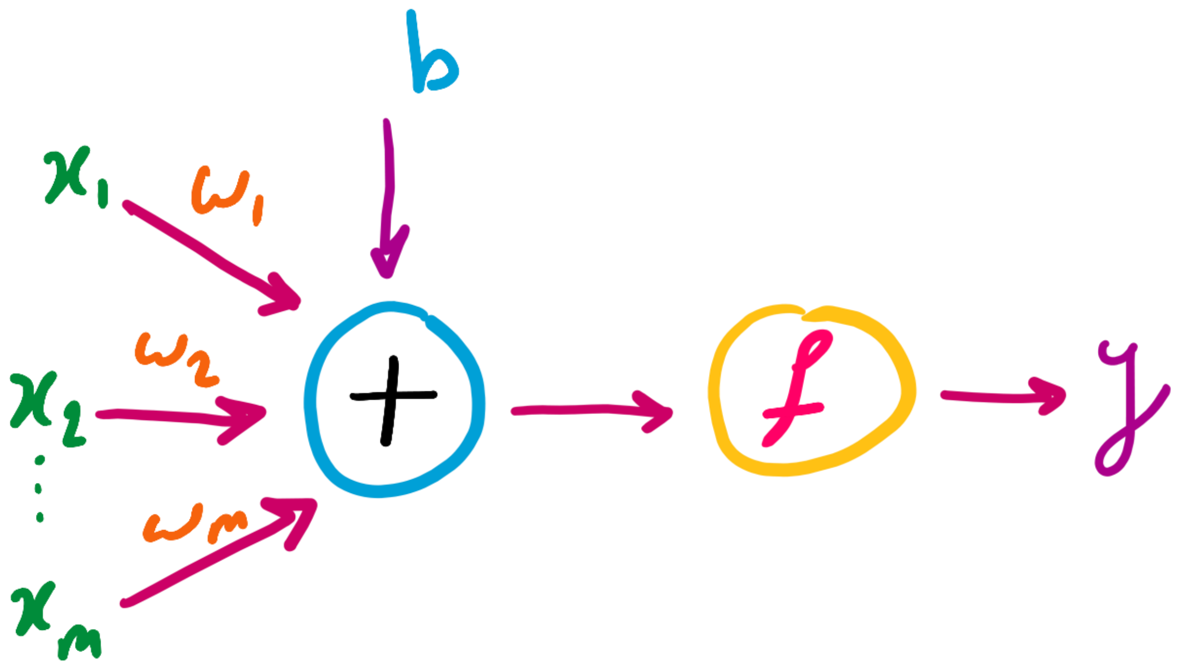




$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

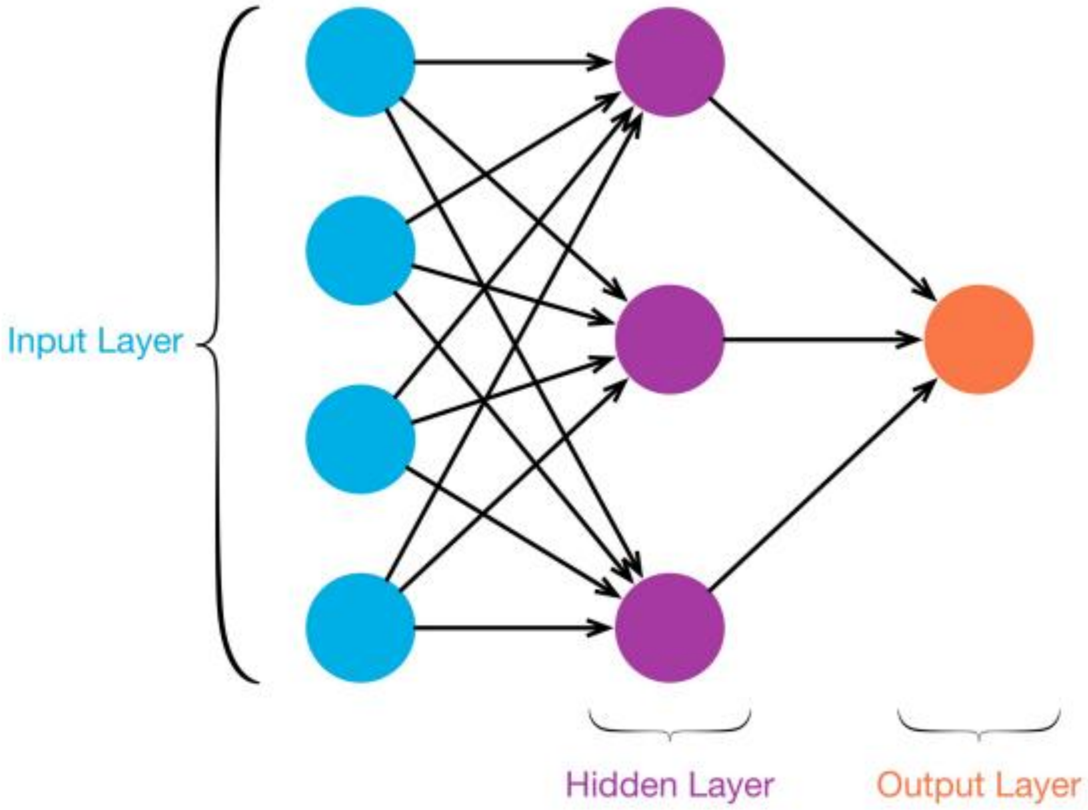
Neural Networks and Deep Learning: A Simple Review

Single-Layer Perceptron

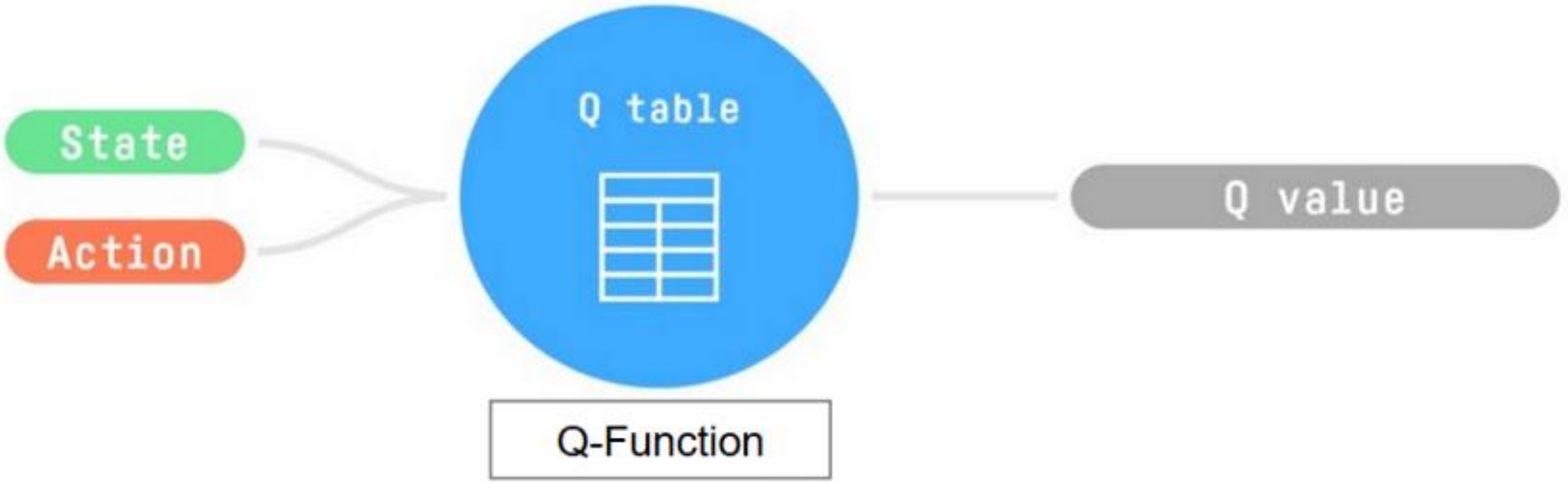


Neural Networks and Deep Learning: A Simple Review

Multi-Layer Perceptron (MLP) (Play!)



Q-Learning Recap …



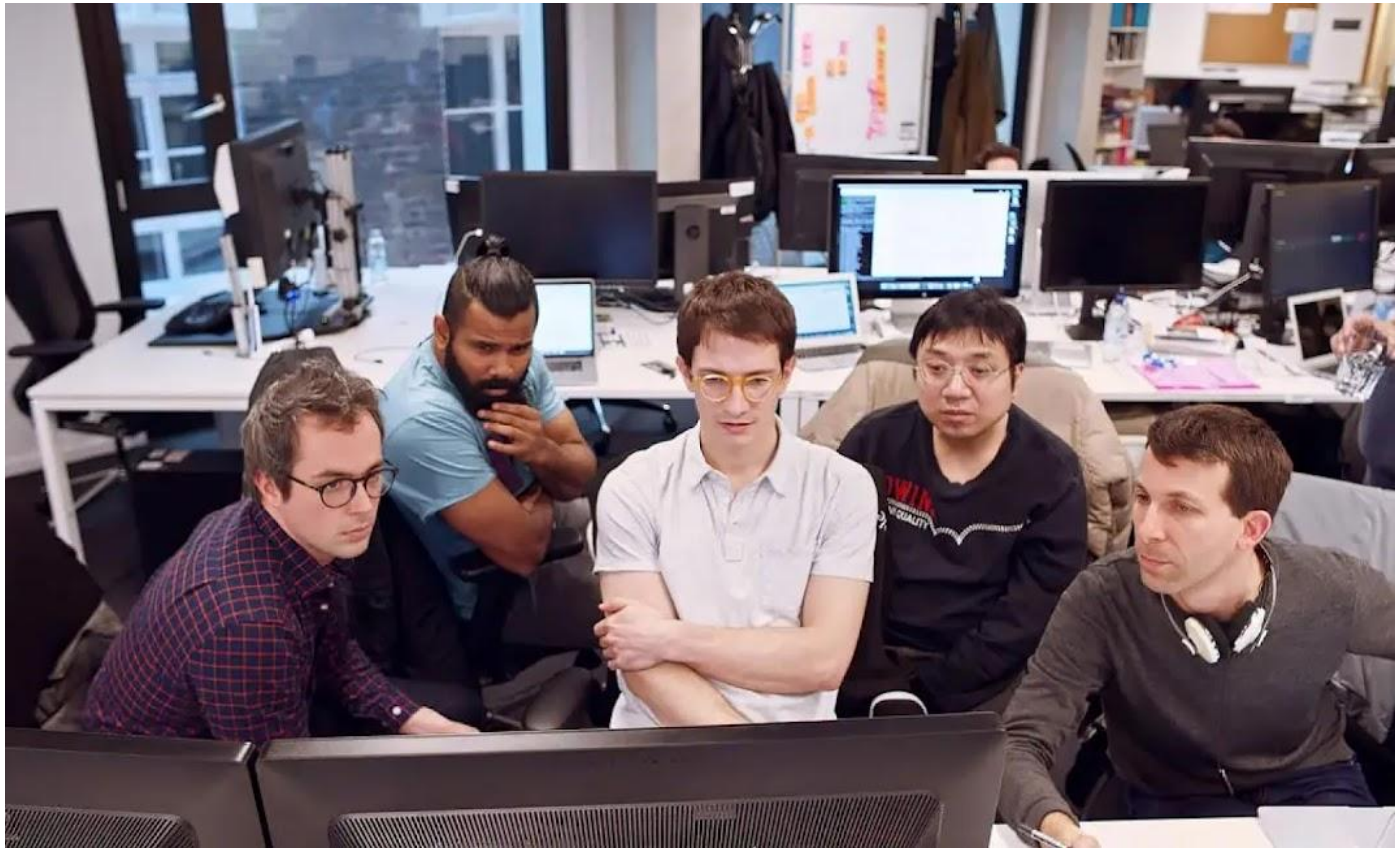
Pseudocode

Q-Learning

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ *to* *num_episodes* **do** Step 1
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy) Step 2
    Take action A_t and observe R_{t+1}, S_{t+1} Step 3
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ Step 4
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Why **Deep** Reinforcement Learning?



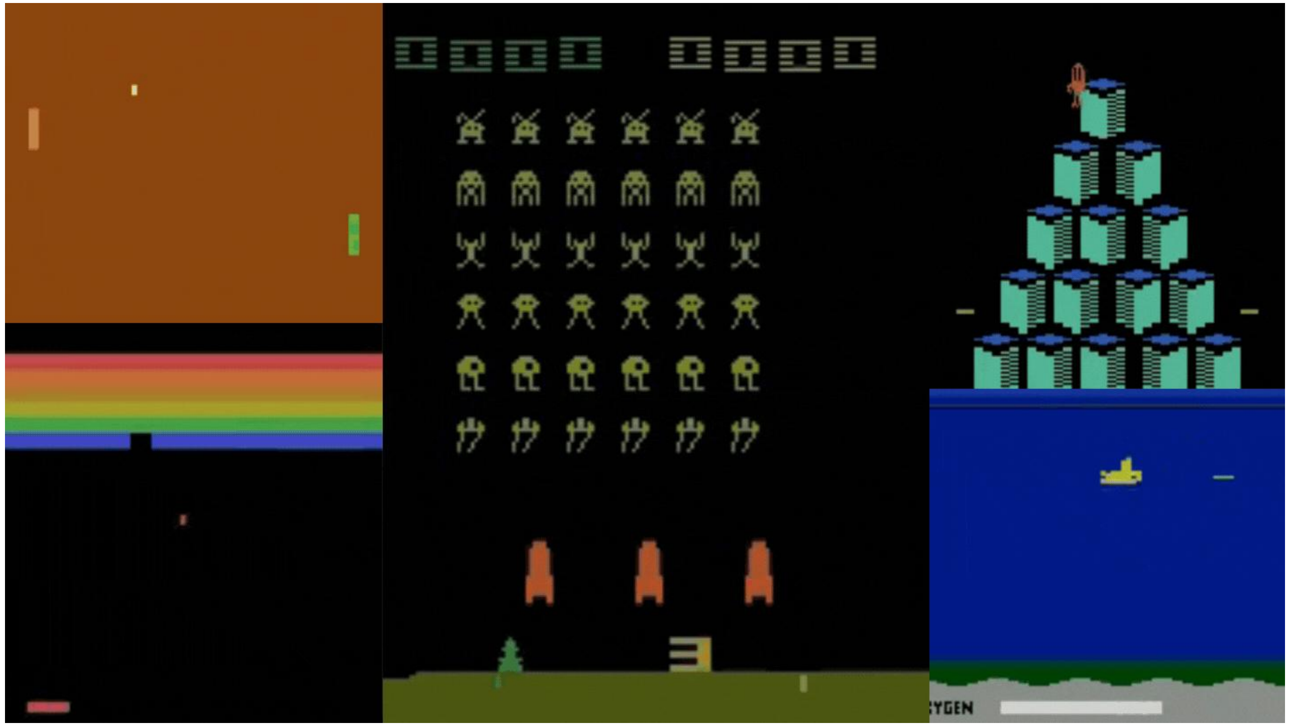
Playing Atari with Deep Reinforcement Learning

(Paper)

Why Deep Reinforcement Learning?

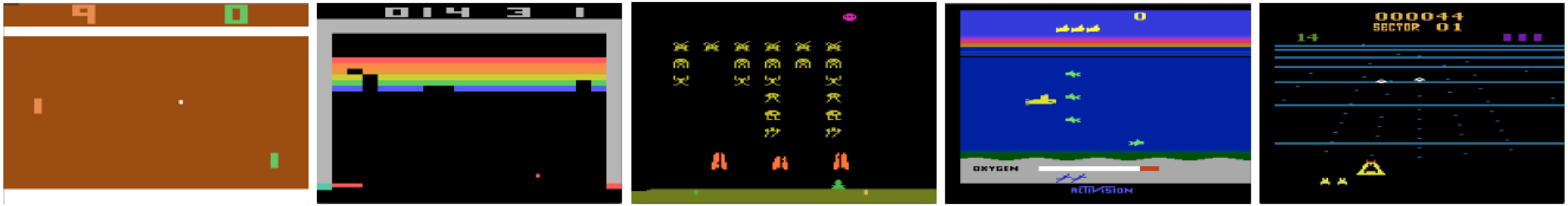
چالش جدی RL: کنترل مبتنی بر یادگیری مستقیما از دیتای با ابعاد بزرگ (تصویر یا صوت یا سنسورها)

Q: Number of states in an 8*8 Gridworld?
What about an Atari game?



Playing Atari with Deep Reinforcement Learning ([Paper])

Why Deep Reinforcement Learning?

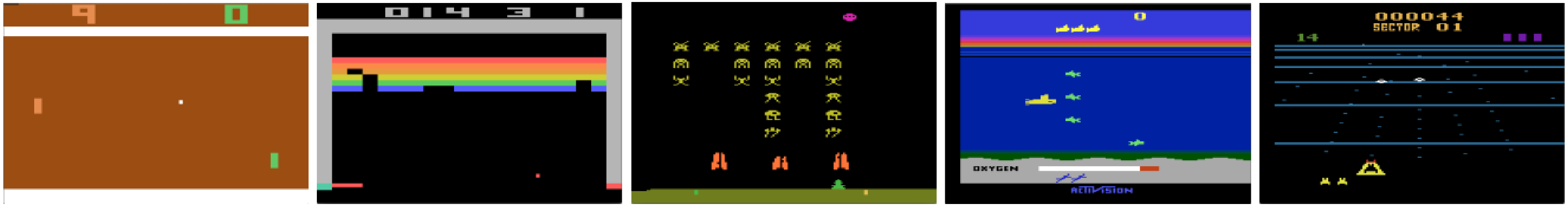


(*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

"Atari 2600 :
*visual input (**210 × 160 RGB** video at 60Hz)*

Playing Atari with Deep Reinforcement Learning ([Paper]())

Why Deep Reinforcement Learning?



(*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Each Frame: (210, 160, 3) containing values ranging from 0 to 255

Q: Number of states?

A: $256^{210\times160\times3} = 256^{100800}$

Idea: approximate Q-values

Using parametrized Q-function $Q_\theta(s, a)$

Playing Atari with Deep Reinforcement Learning (Paper)

DeepRL: Why Is It Still Hard?

- روش های موفق Deep Learning: دیتاهای عظیم لیبل دار

- یادگیری RL از پاداش اسکالربه صورت sparse و نویزی و تاخیر دار (برعکس DL)

Playing Atari with Deep Reinforcement Learning (Paper)

DeepRL: Why Is It Still Hard?

DL data samples: IID

RL: Highly correlated states.

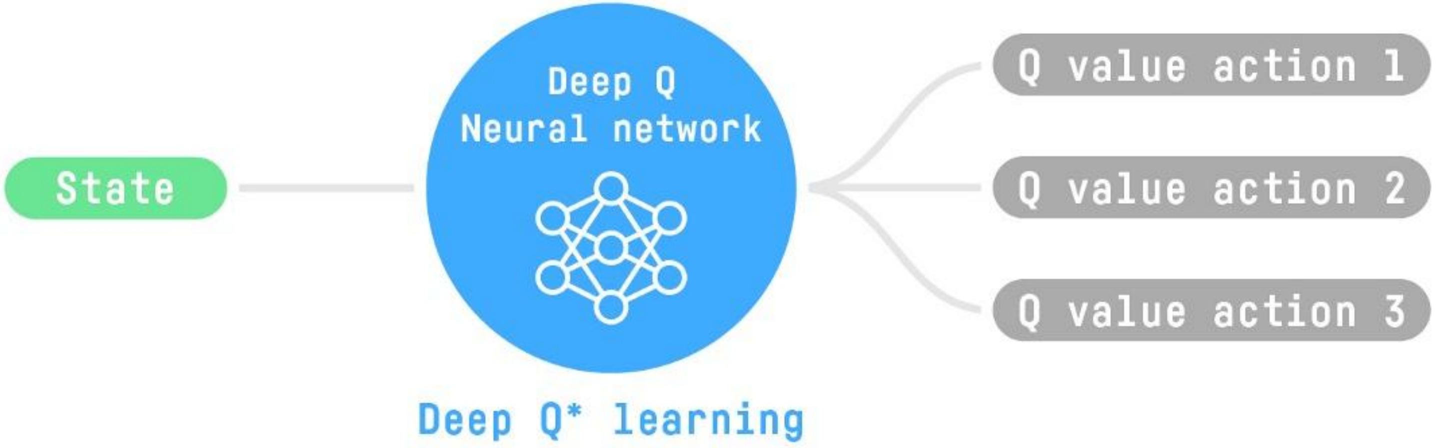
تغییر توزیع دیتا در RL در فرایند یادگیری

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

The best idea is to approximate the Q-values using a parametrized Q-function $Q_\theta(s, a)$.

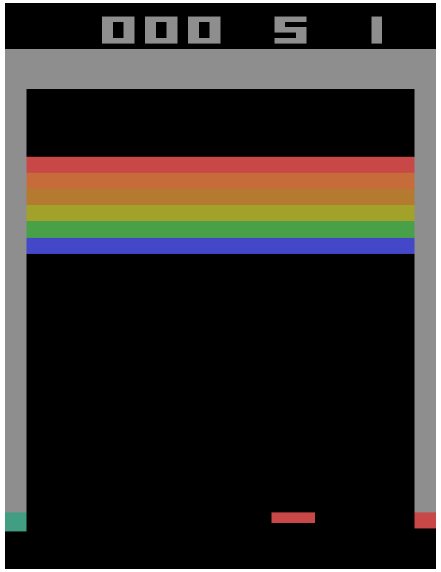
مشابه DL:



Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

Emulator: agent interacts with an environment ε



Goal: maximizes future rewards.

The future discounted *return* at time *t:*

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

r_t: *reward*

The optimal action-value function:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

Playing Atari with Deep Reinforcement Learning ([Paper])

Deep Q-Learning (DQN)

optimal strategy :
select a' :

maximizing the expected value of $r + \gamma Q^*(s', a')$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \Big| s, a \right]$$

Reminder Box: Value Iteration

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s', a') | s, a\right]$$

$$Q_i \rightarrow Q^* \text{ as } i \rightarrow \infty$$

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

A function approximator to estimate the action-value function:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

A Neural Network!

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

Neural network function approximator with weights θ : ***Q-network***

Training:

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

"Where

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

$\rho(s,a)$: behaviour

تارگت وابسته به وزنهای شبکه است برعکس Suprvised DL که ثابت است

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

مشتق تابع Loss نسبت به وزنها:

Reminder Box

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

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

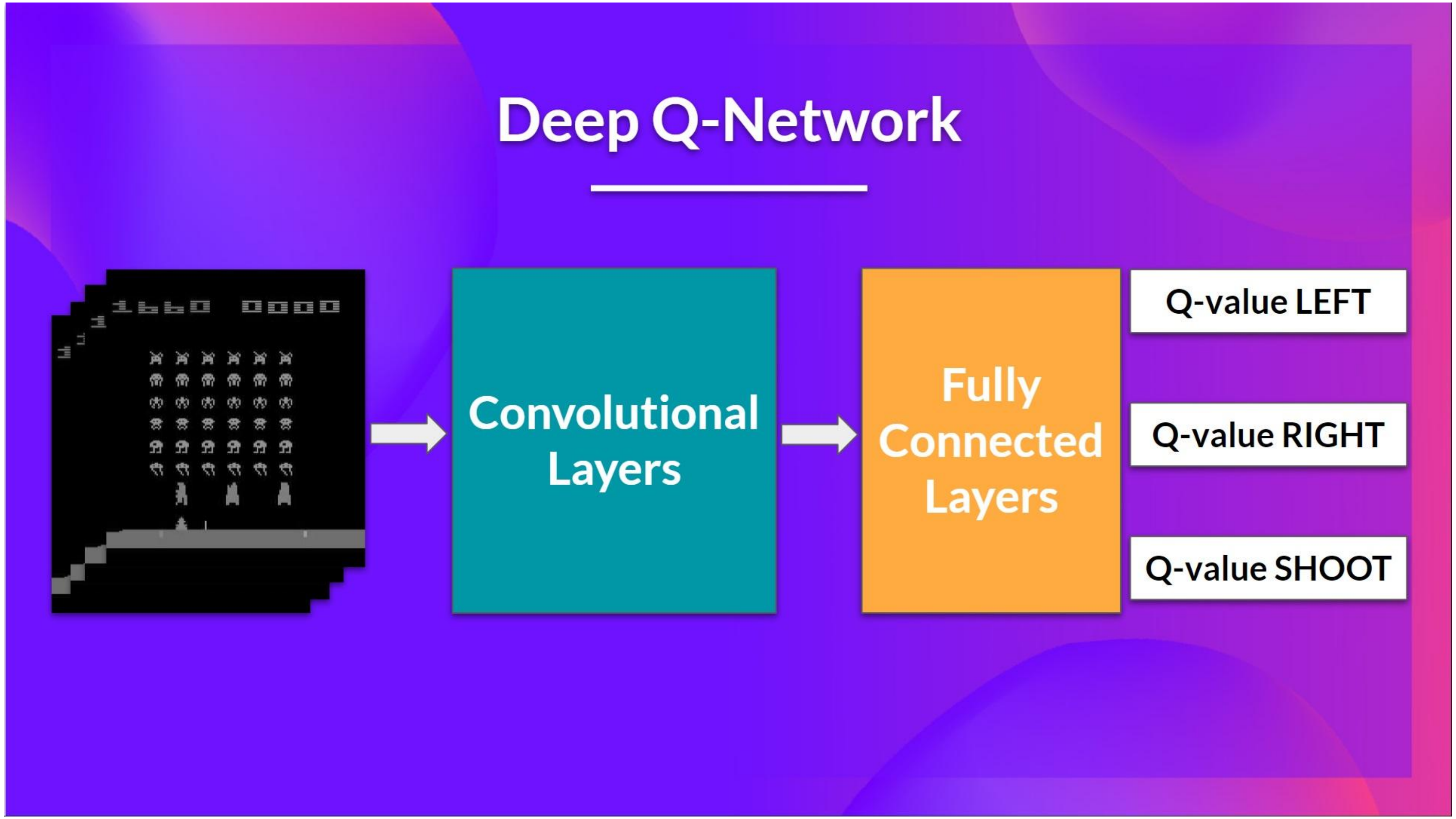
stochastic gradient descent

model-free and off-policy.

Playing Atari with Deep Reinforcement Learning (Paper)

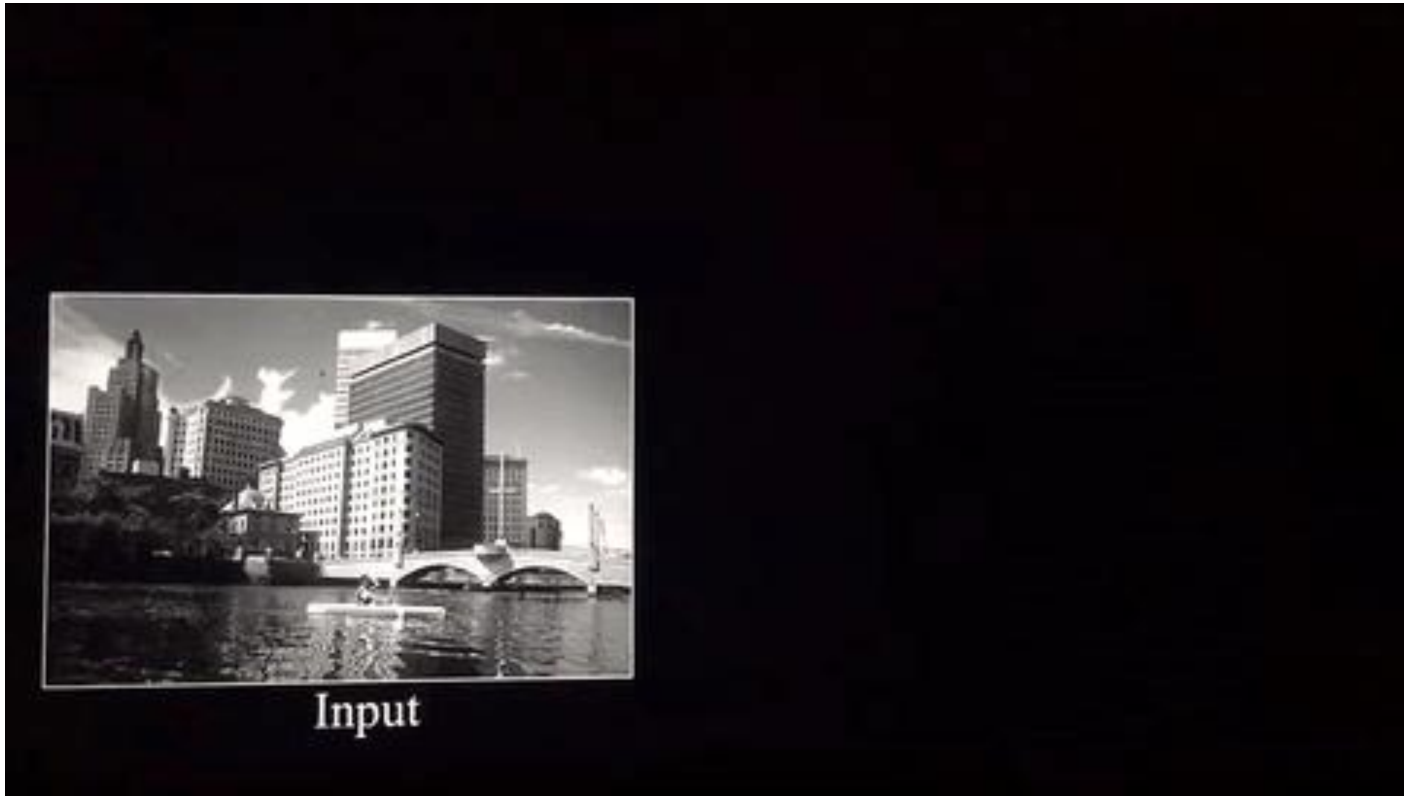
Architecture

Input: Stack of 4 gray-scale frames
Q: Why?
Output: A vector of Q-values for each possible action at that state



Playing Atari with Deep Reinforcement Learning ([Paper])

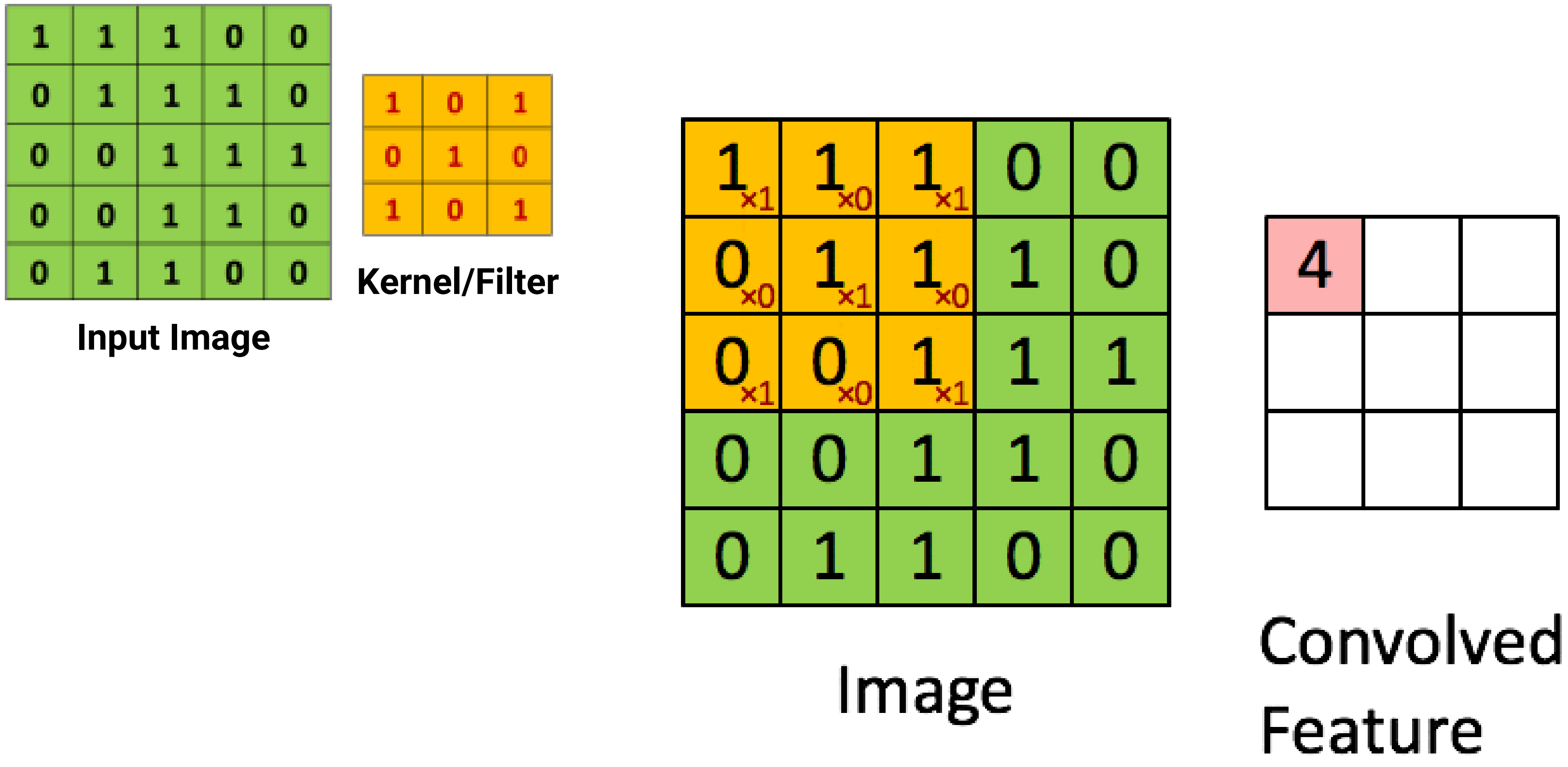
Architecture: What Is a Convolutional Layer?



Convolutional Layers

Playing Atari with Deep Reinforcement Learning (Paper)

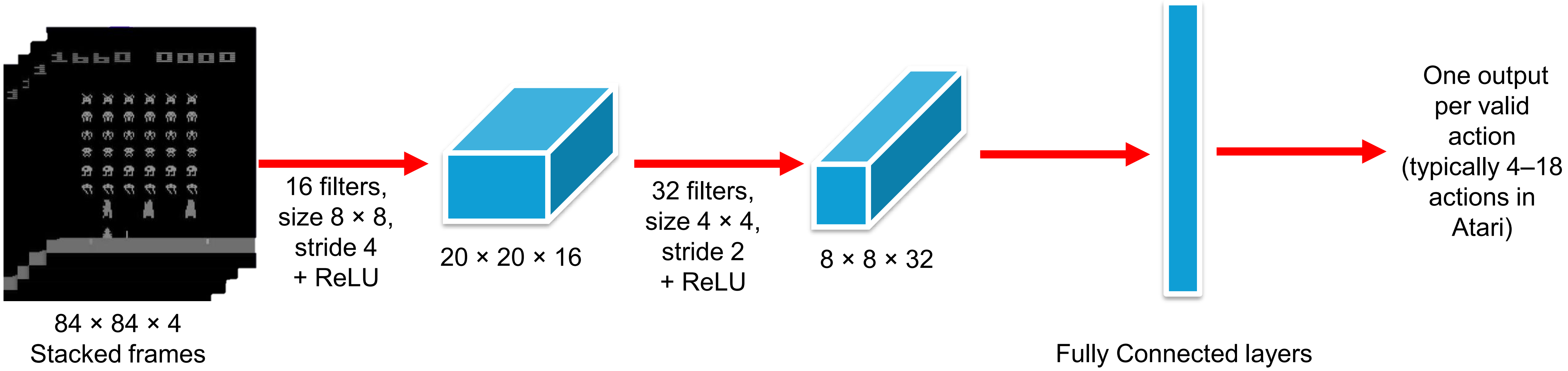
Architecture: What Is a Convolutional Layer?



Convolutional Layers

Playing Atari with Deep Reinforcement Learning (Paper)

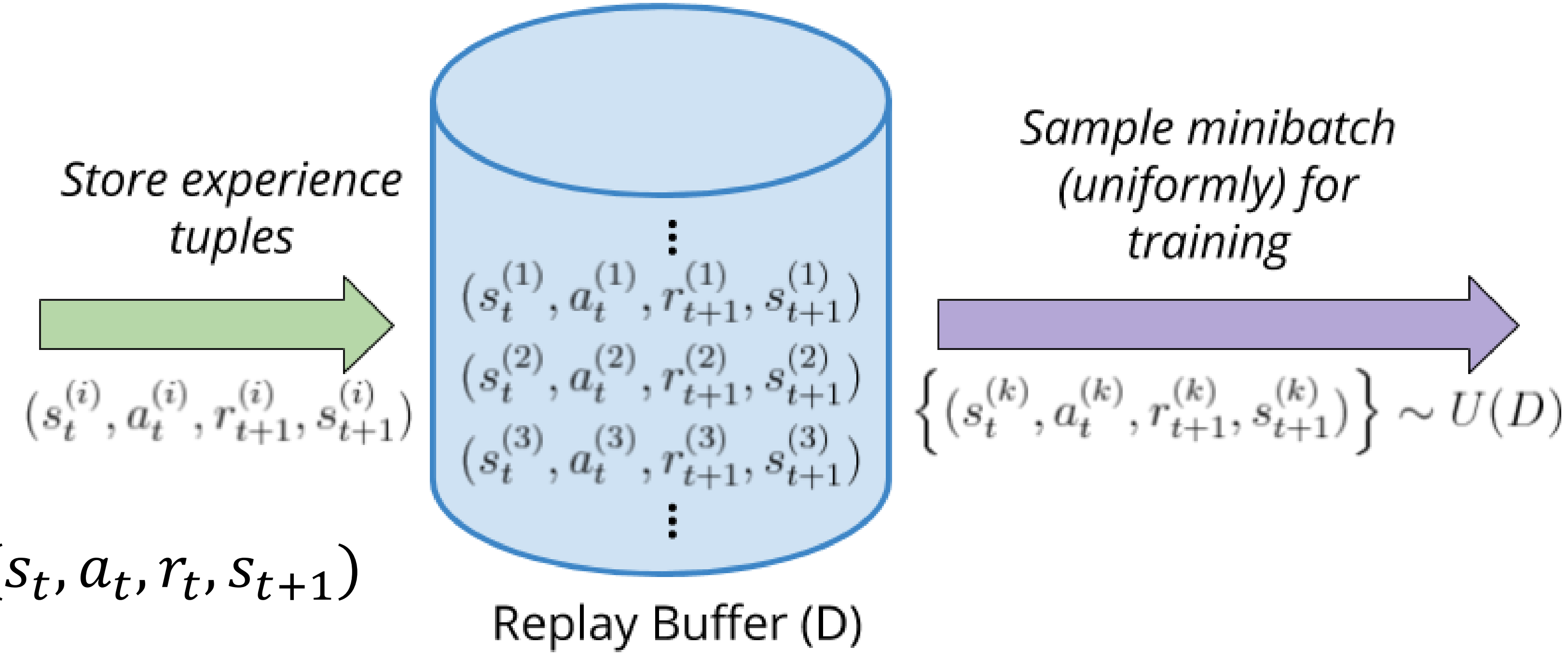
Architecture



This architecture is known as a Deep Q-Network (DQN)

Playing Atari with Deep Reinforcement Learning ([Paper])

Experience Replay
Replay Buffer



Agent's experiences : $e_t = (s_t, a_t, r_t, s_{t+1})$

Data-set $D = e_1, \dots, e_N$

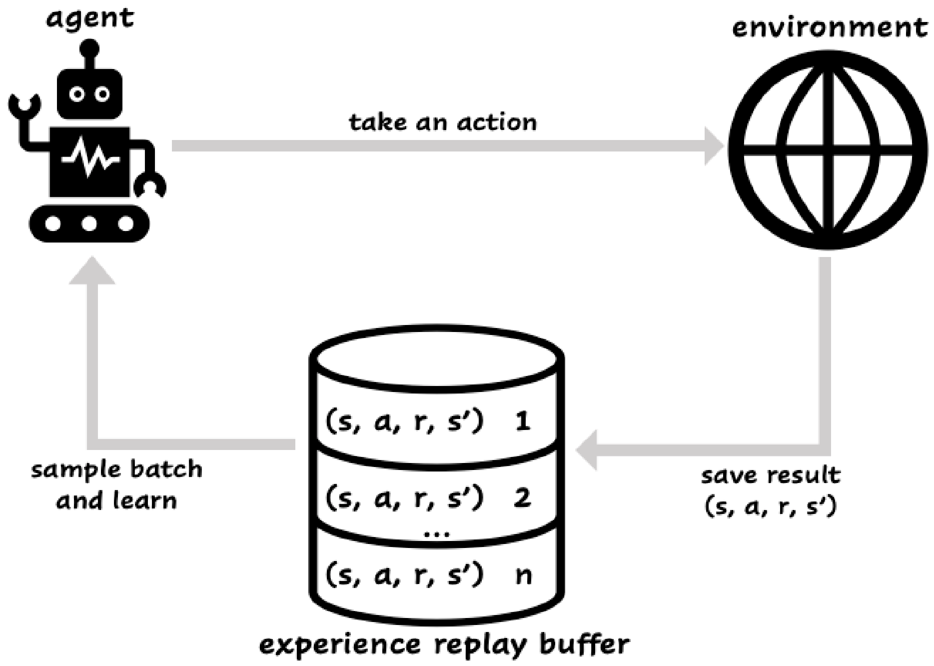
Q-learning updates, or minibatch updates :

$$Random : e \sim D$$

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm: Replay Buffer

Stores the last N experience
Samples uniformly at random from D



Q: Why Replay Buffer?

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm

Reminder Box

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation

Former Q-value estimation

Learning Rate

Immediate Reward

Discounted Estimate optimal Q-value of next state

Former Q-value estimation

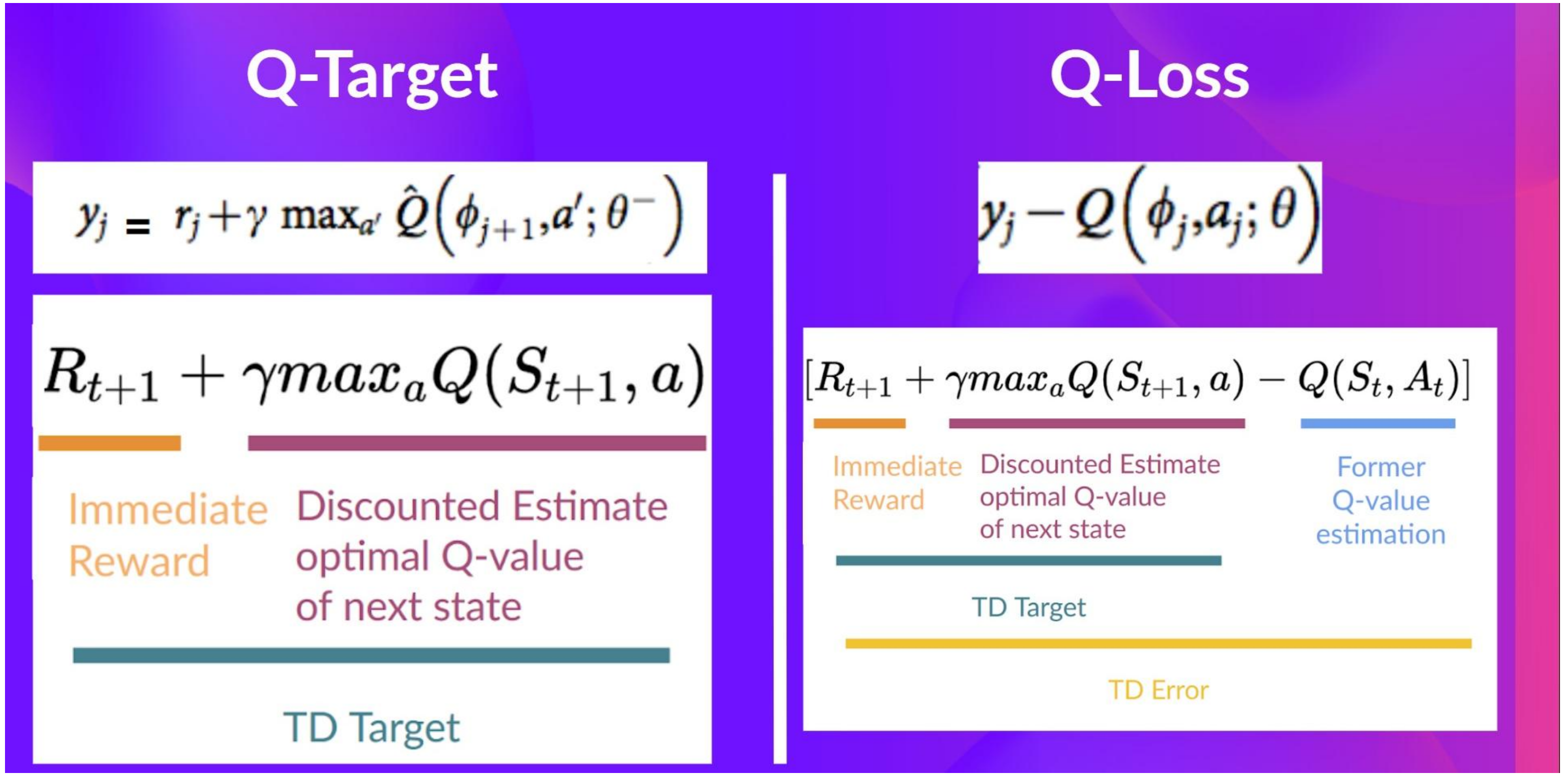
TD Target

TD Error

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm

Intuition



Playing Atari with Deep Reinforcement Learning (Paper)

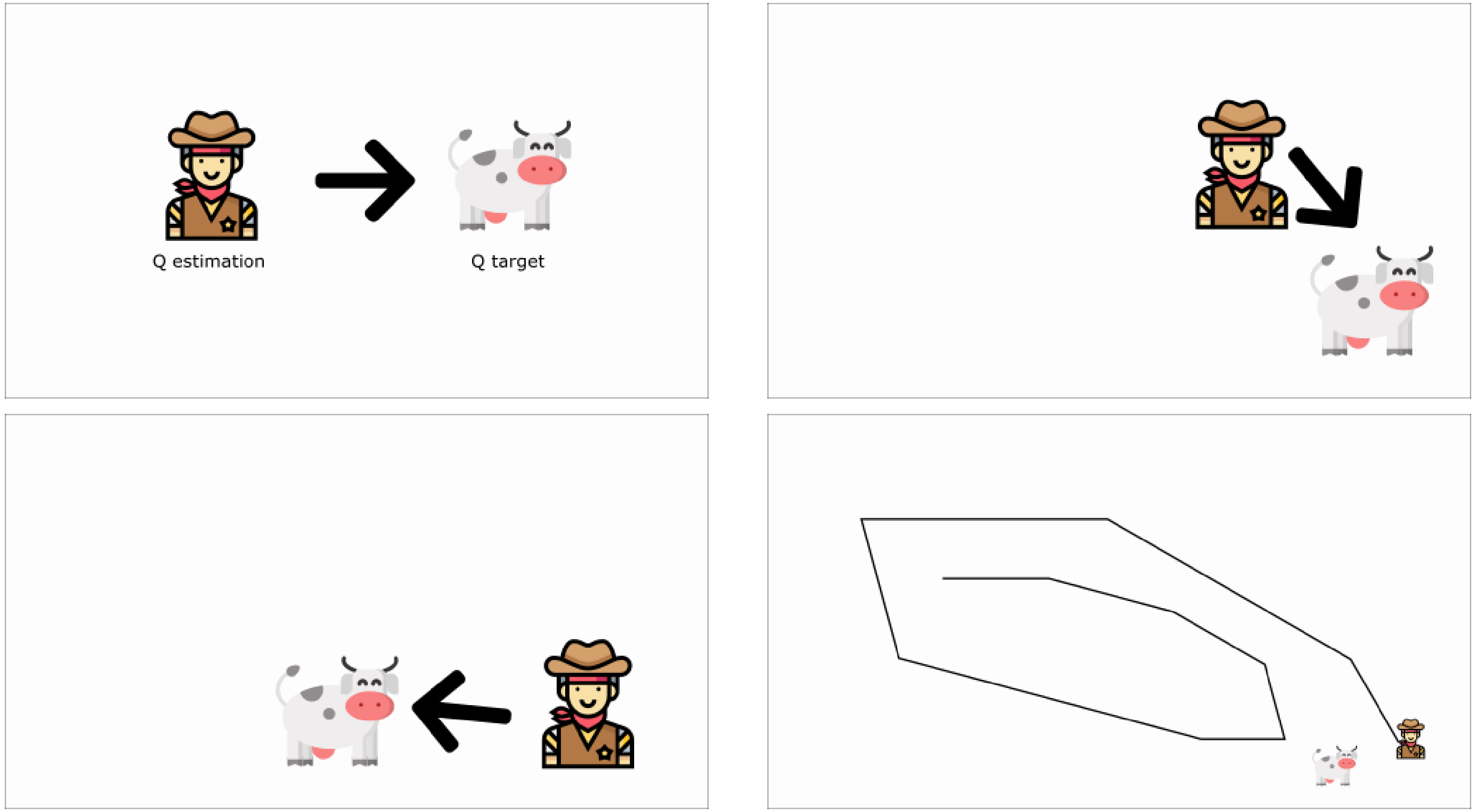
Algorithm

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 $\boxed{3}$
  end for
end for

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm: A Challenge!



Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm: A Challenge!

Solution:

- Use a **separate network** with fixed parameters for estimating the TD Target
- Copy the parameters from our Deep Q-Network **every C steps** to update the target network.

Playing Atari with Deep Reinforcement Learning (Paper, Paper)

Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Playing Atari with Deep Reinforcement Learning (Paper, Paper)

Conclusion

"We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them, with no adjustment of the architecture or hyperparameters."

Playing Atari with Deep Reinforcement Learning (Paper, Paper)

Appendix

What is ϕ from both algorithms?

Preprocessing. Working directly with raw Atari 2600 frames, which are 210×160 pixel images with a 128-colour palette, can be demanding in terms of computation and memory requirements. We apply a basic preprocessing step aimed at reducing the input dimensionality and dealing with some artefacts of the Atari 2600 emulator. First, to encode a single frame we take the maximum value for each pixel colour value over the frame being encoded and the previous frame. This was necessary to remove flickering that is present in games where some objects appear only in even frames while other objects appear only in odd frames, an artefact caused by the limited number of sprites Atari 2600 can display at once. Second, we then extract the Y channel, also known as luminance, from the RGB frame and rescale it to 84×84. The function ϕ from algorithm 1 described below applies this preprocessing to the m most recent frames and stacks them to produce the input to the Q-function, in which $m = 4$, although the algorithm is robust to different values of m (for example, 3 or 5).

Playing Atari with Deep Reinforcement Learning (Paper, Paper)