In the Name of God

Iran University
of Science and
Technology

دانشگاه علم و صنعت ایران

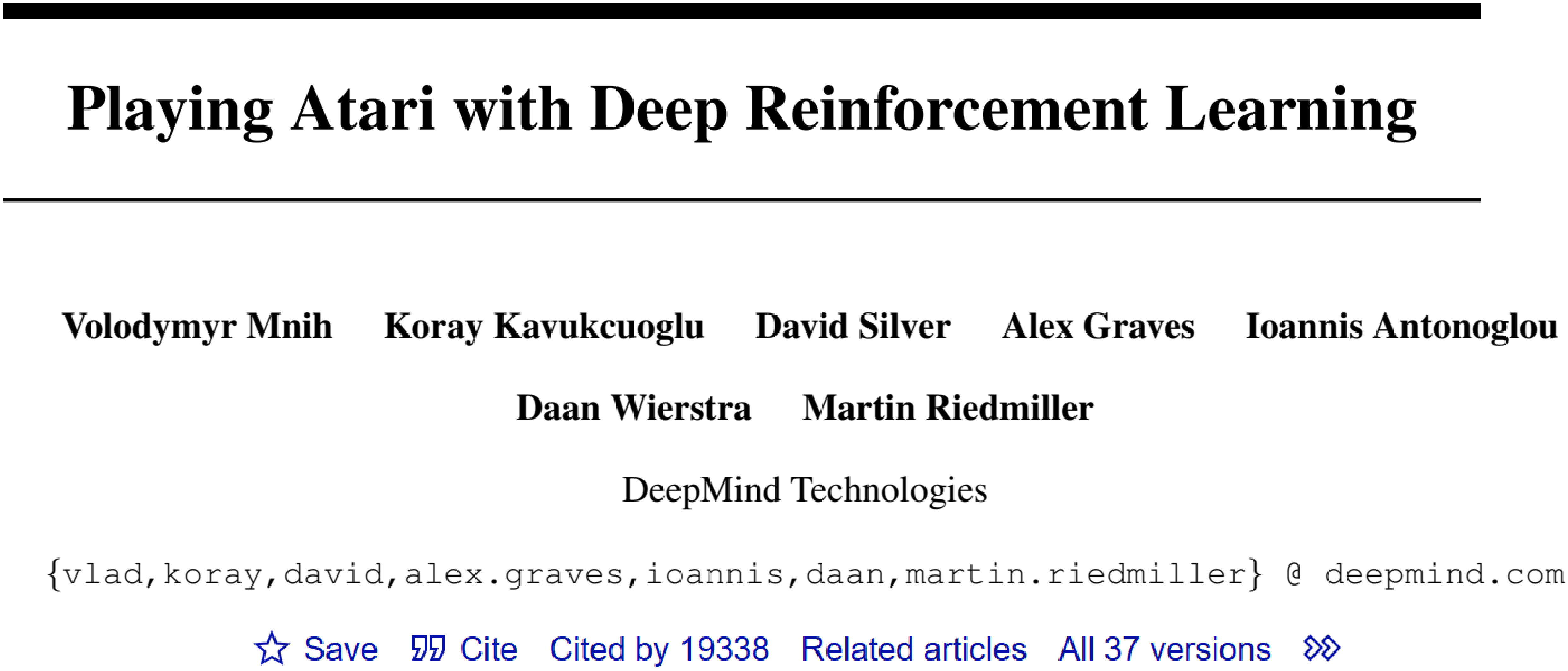
Reinforcement Learning in Control

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]

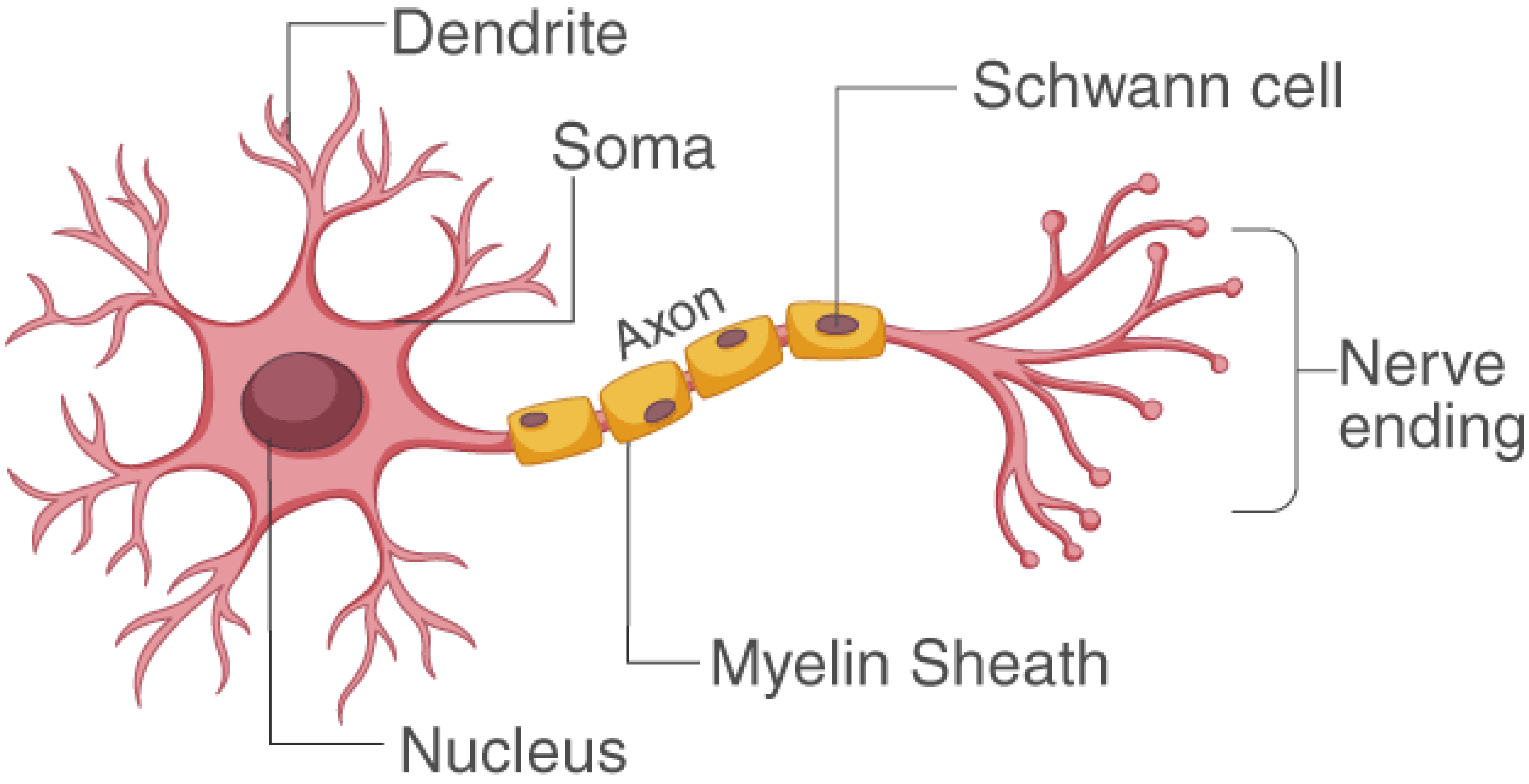
[1] Google DeepMind, 5 New Street Square, London EC4A 3TW, UK.

*These authors contributed equally to this work.

26 FEBRUARY 2015 | VOL 518 | NATURE | 529

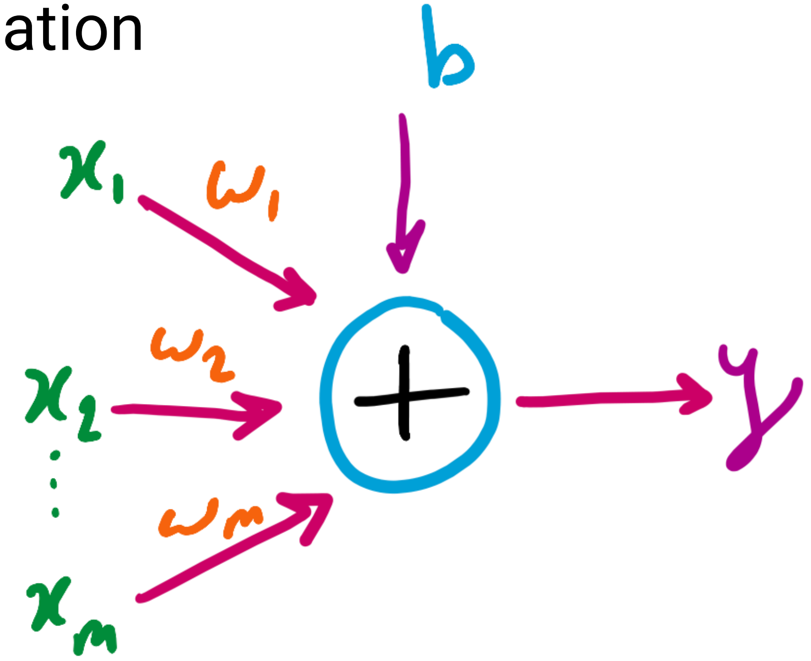
Neural Networks and Deep Learning: A Simple Review

Neuron



Neural Networks and Deep Learning: A Simple Review

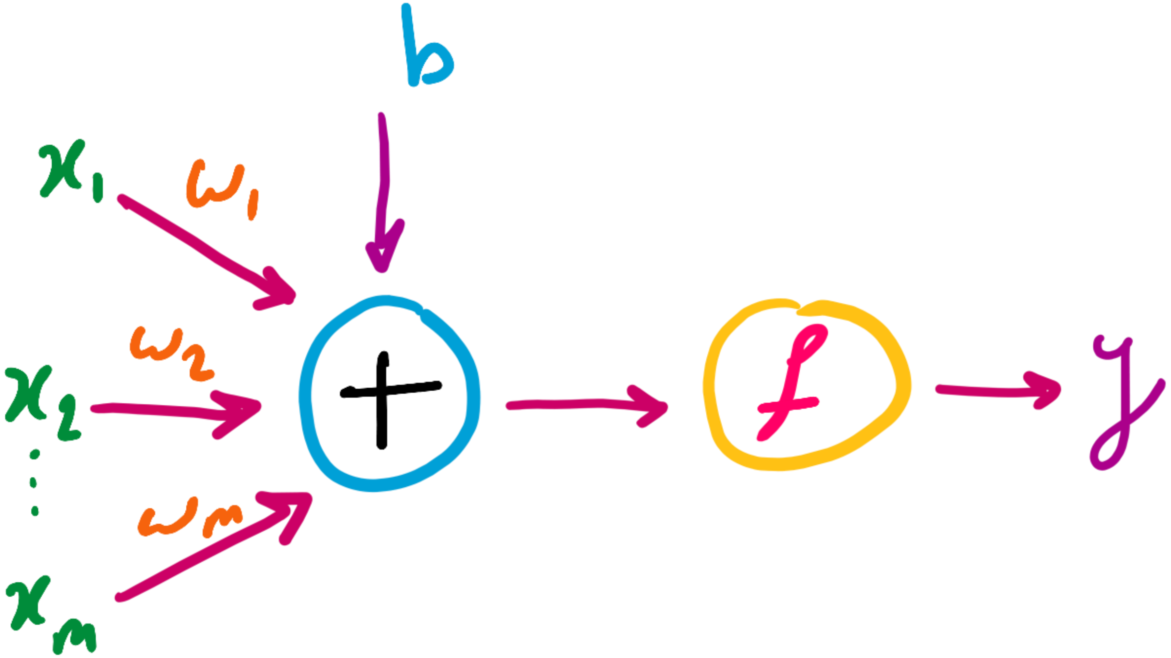
Artificial Neuron Formulation



$$y = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b$$

Neural Networks and Deep Learning: A Simple Review

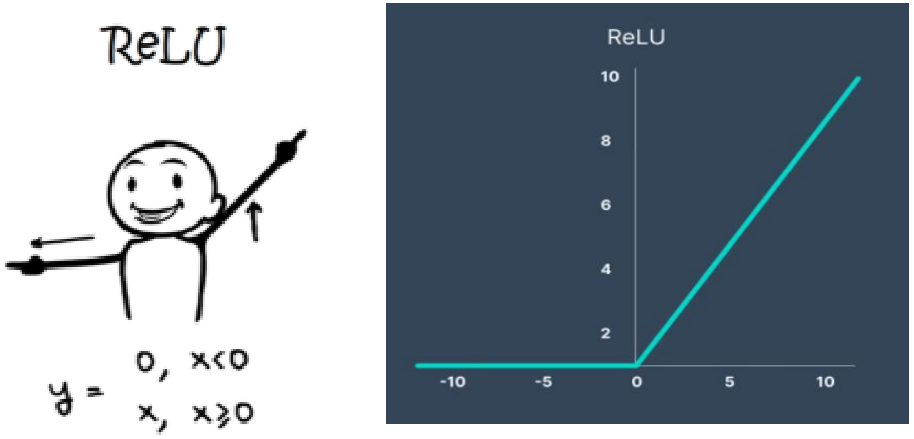
Q: Problems?

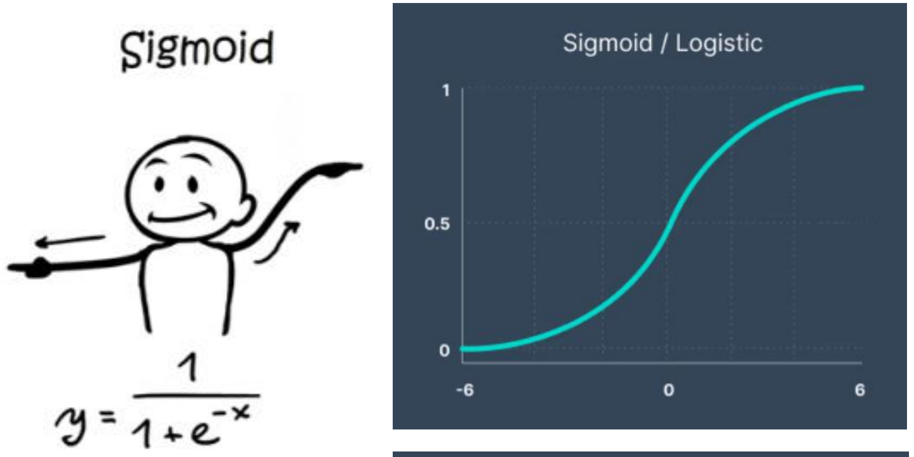


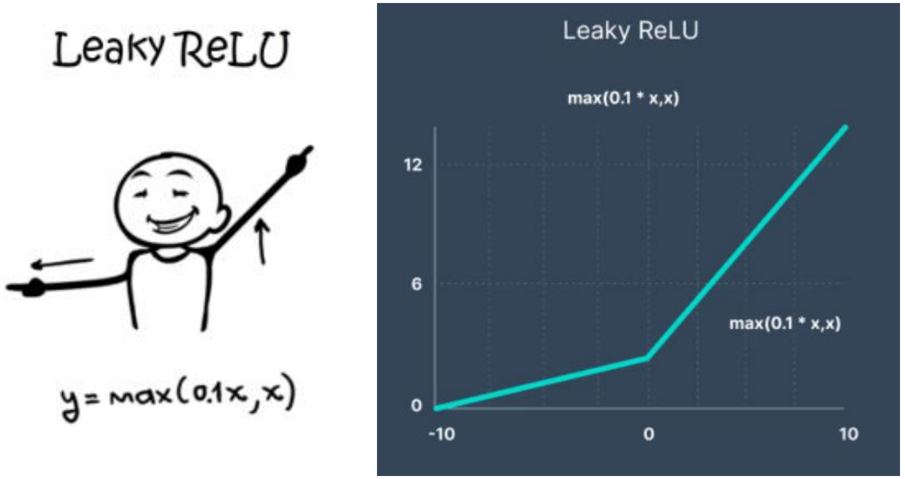
$$y = f(w_1x_1 + w_2x_2 + \dots + w_mx_m + b) = f(w^Tx + b)$$

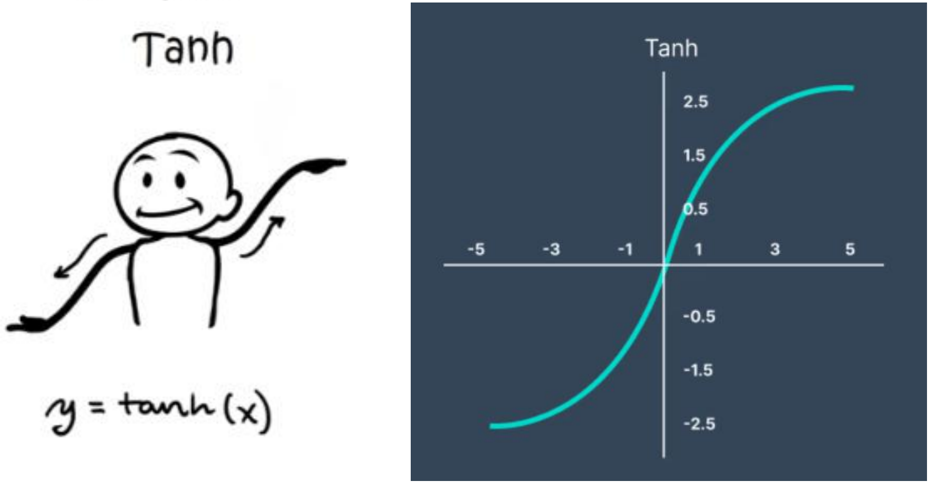
Neural Networks and Deep Learning: A Simple Review

Activation Functions



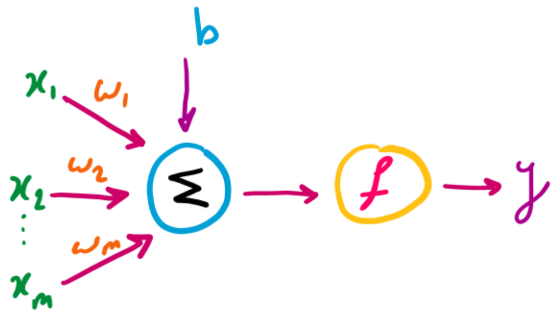




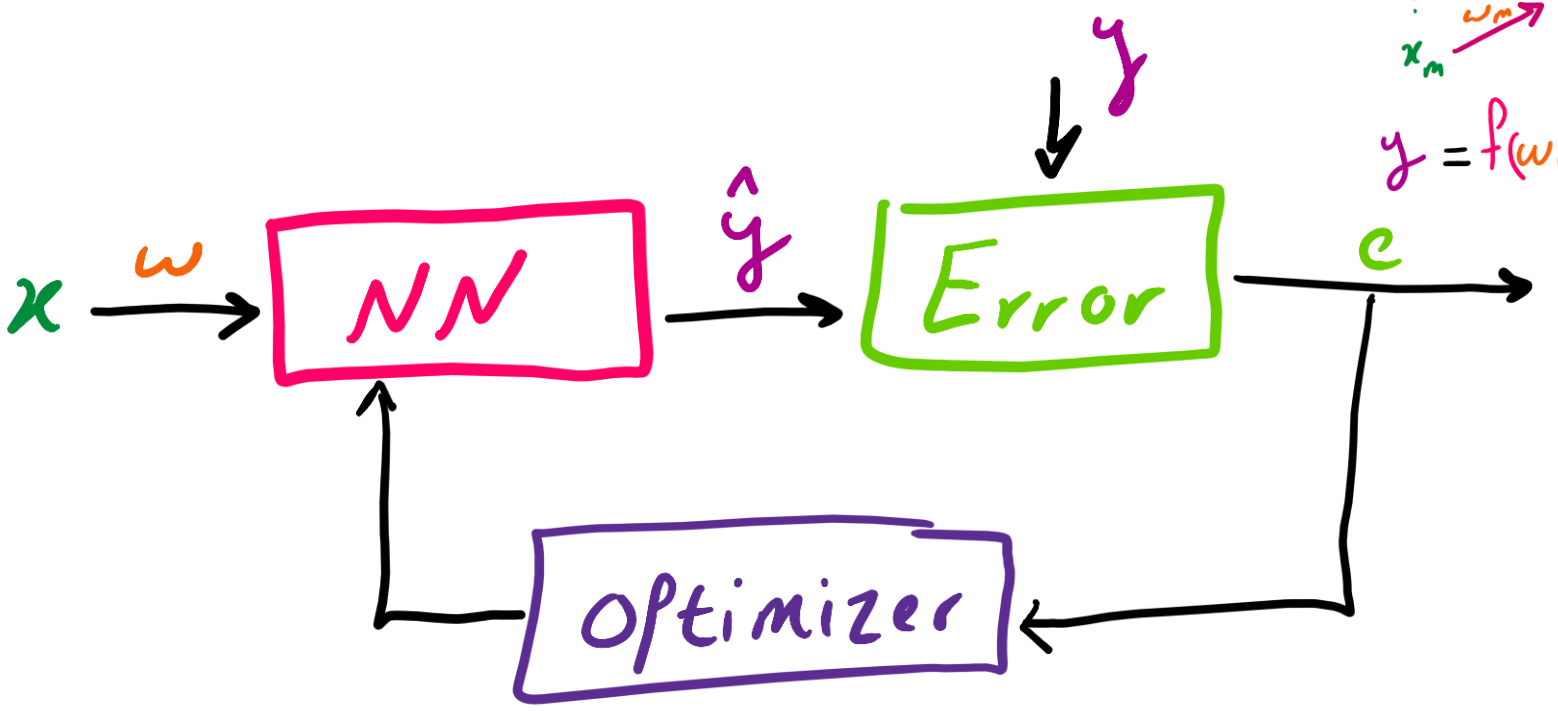


Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram



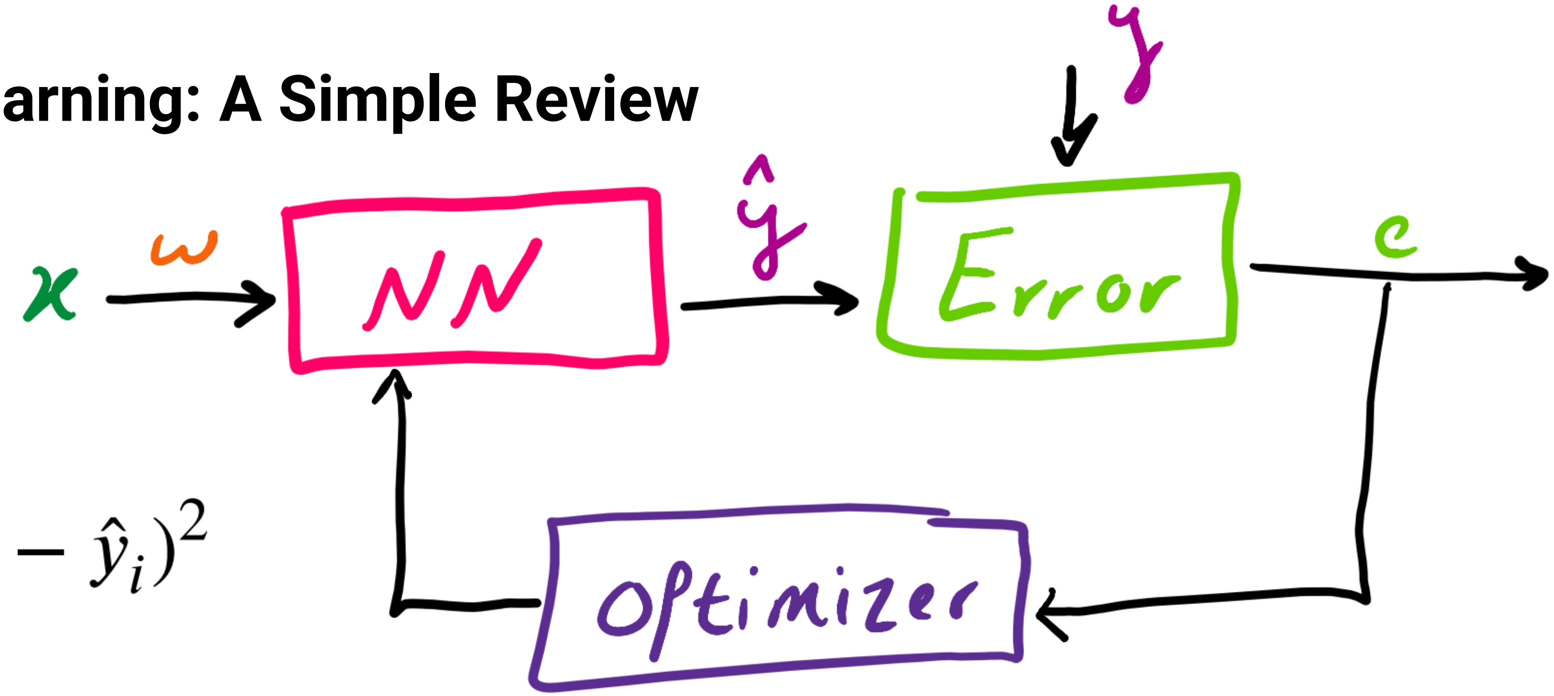
$$y = f(w_1x_1 + w_2x_2 + \cdots + w_mx_m + b)$$



Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

Loss Function:



***MSE*:**

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

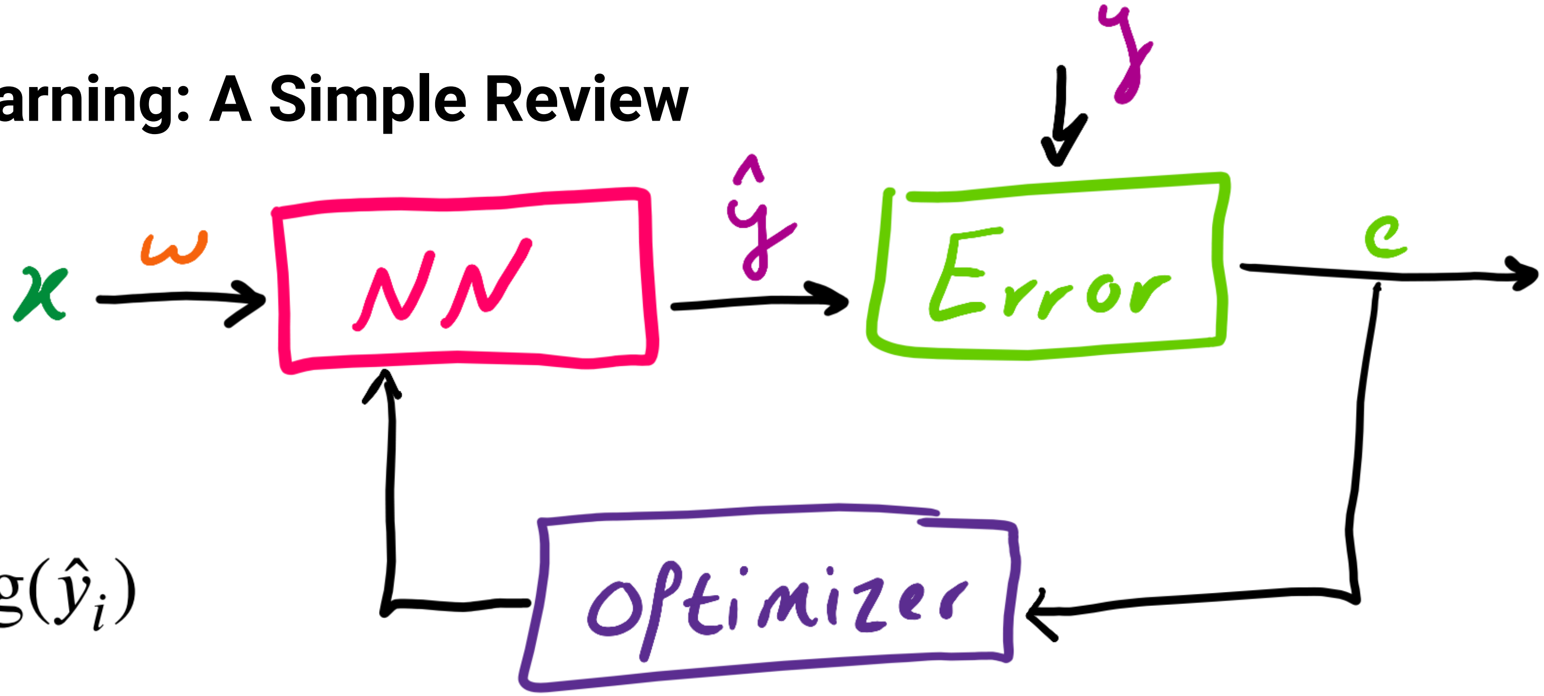
***MAE*:**

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

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

Loss Function:



Cross Entropy:

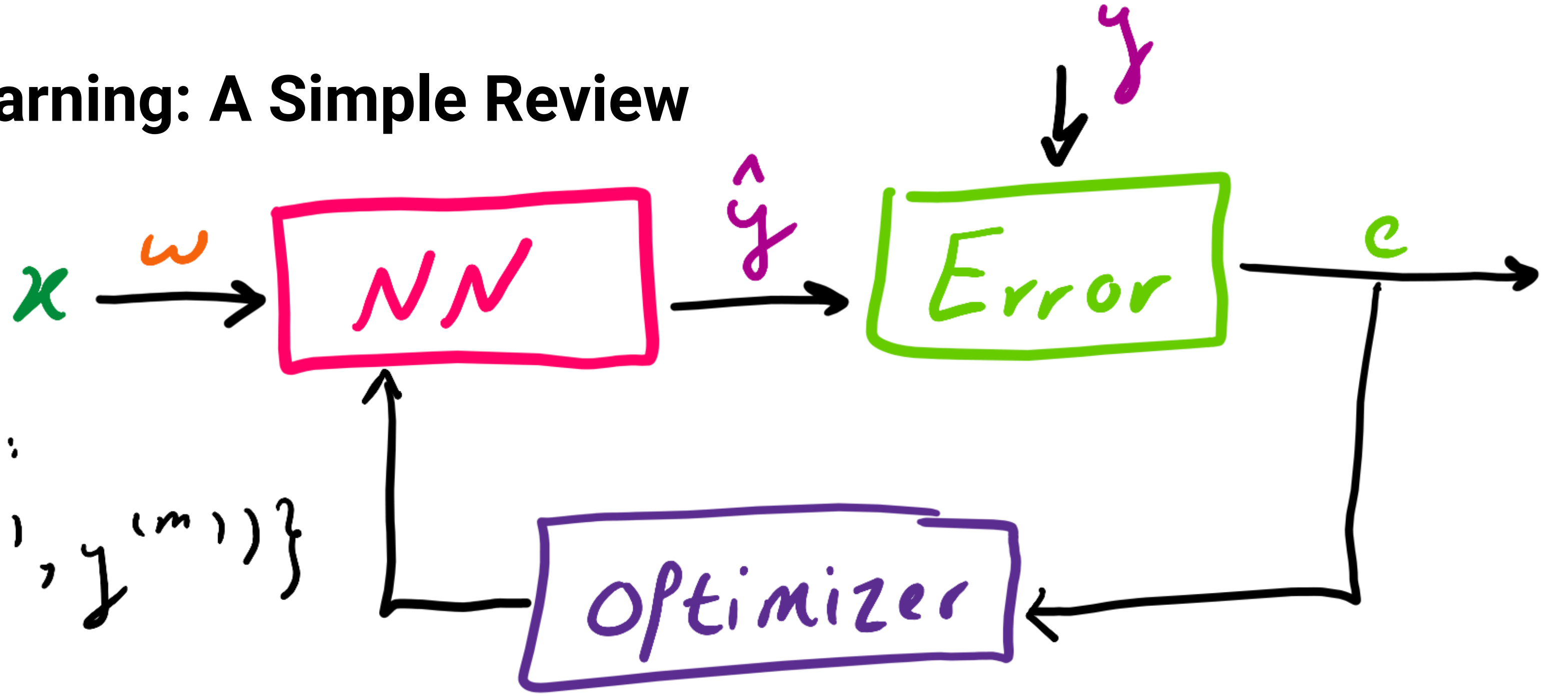
$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Binary Cross Entropy:

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

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram



Given m train examples:

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

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

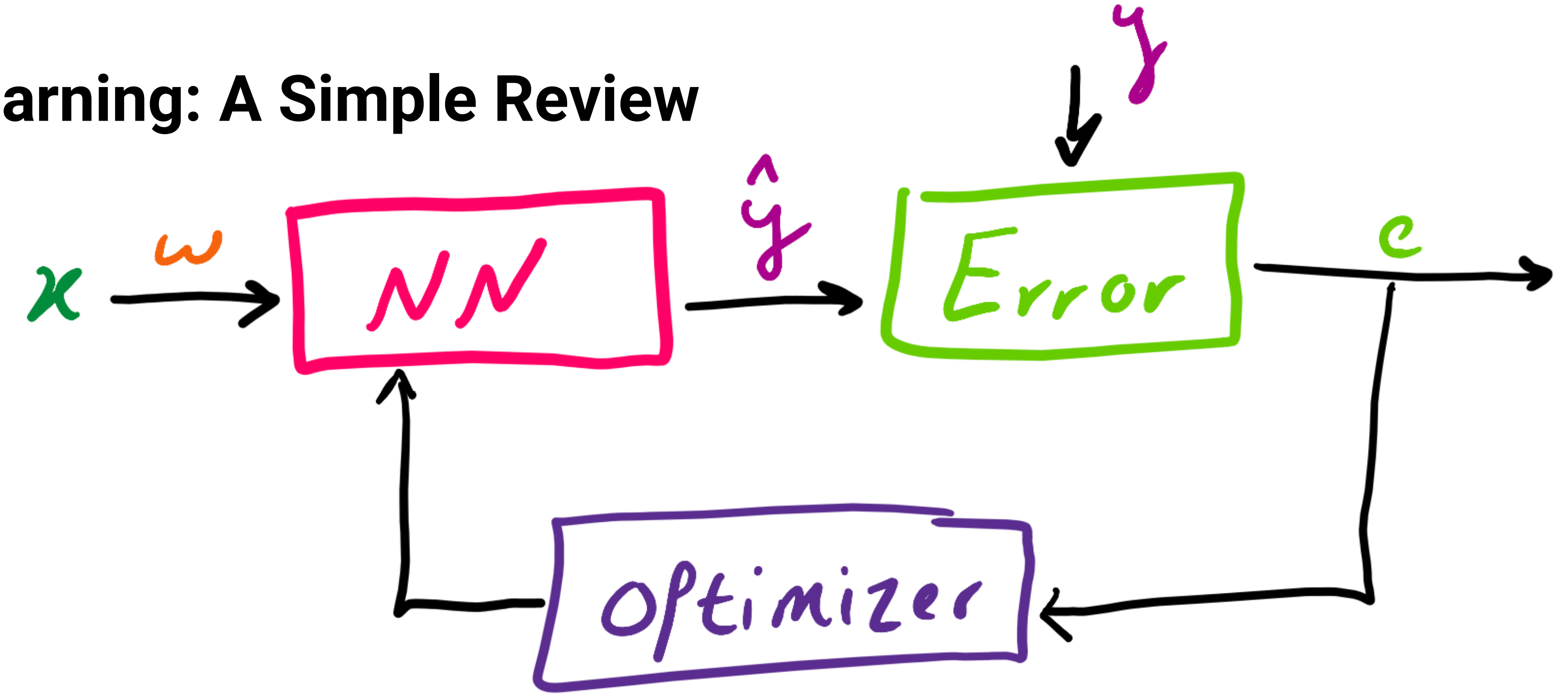
So the cost function is:

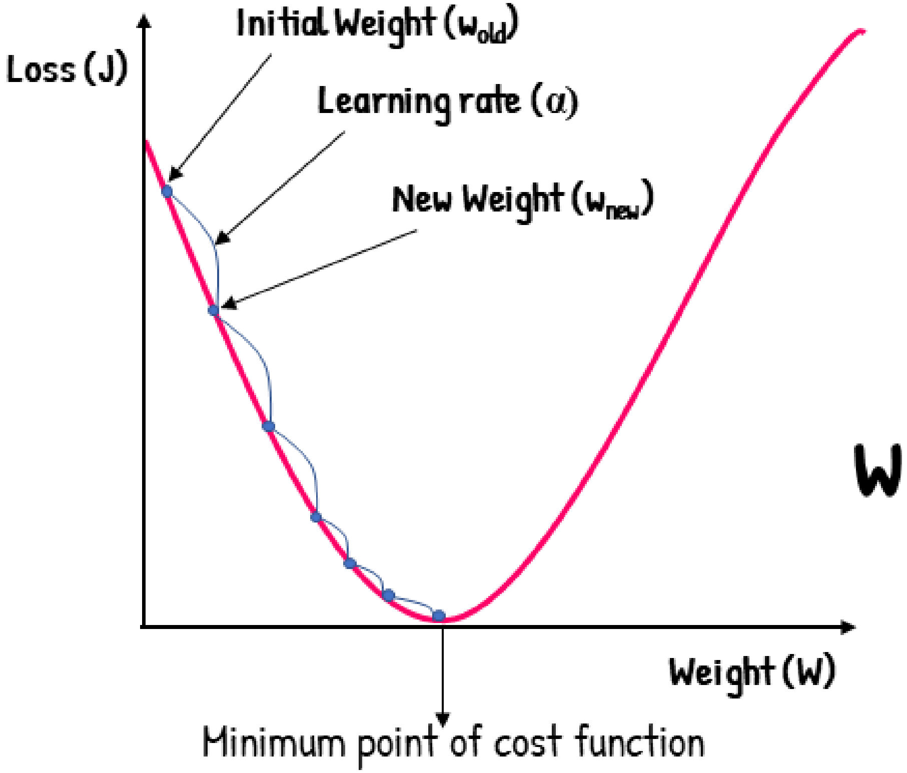
$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(y, \hat{y})$$

Neural Networks and Deep Learning: A Simple Review

Learning Block Diagram

Gradient Descent:

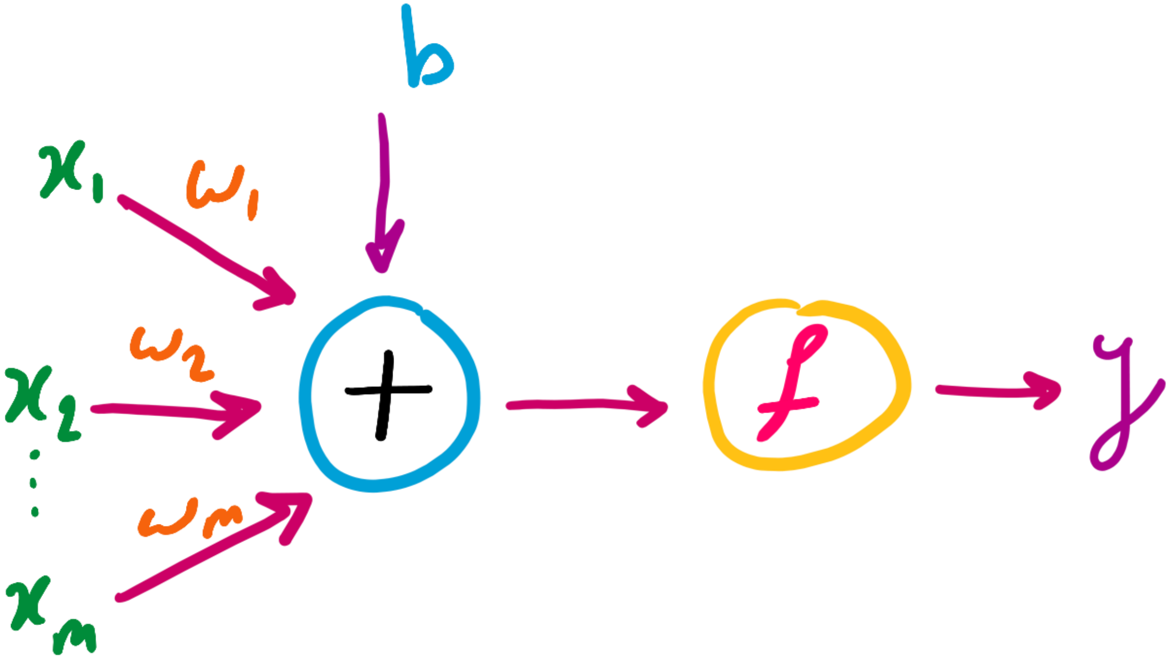




$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

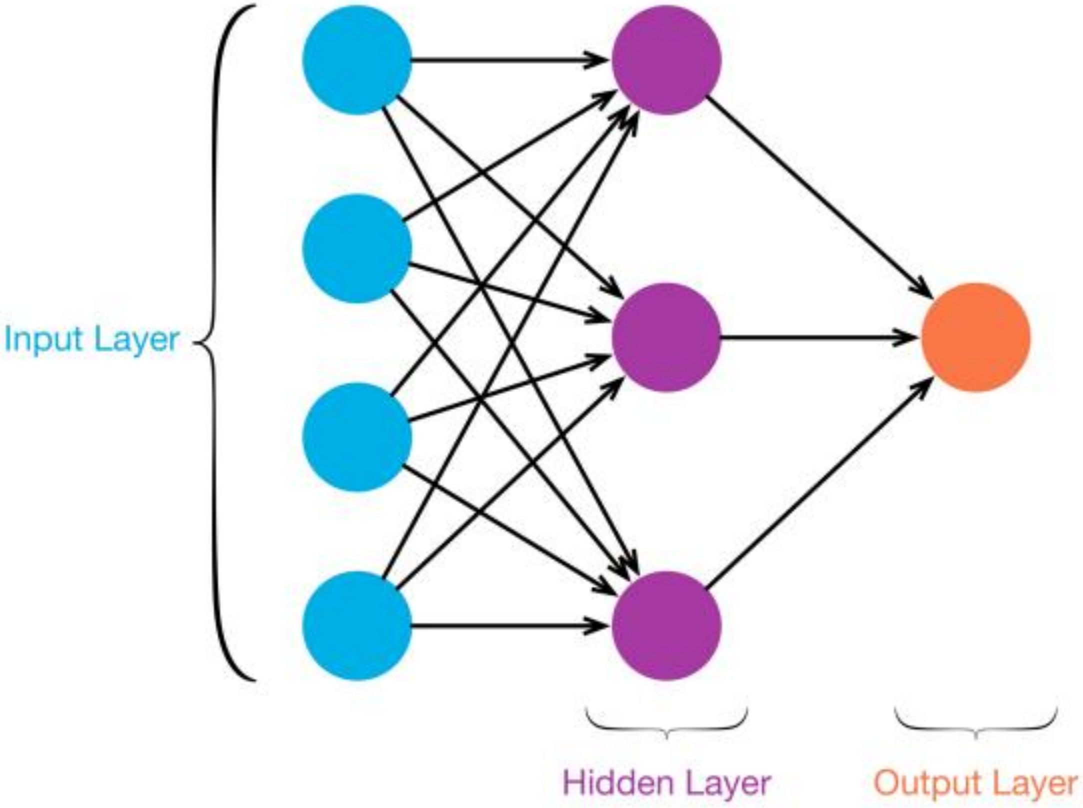
Neural Networks and Deep Learning: A Simple Review

Single-Layer Perceptron

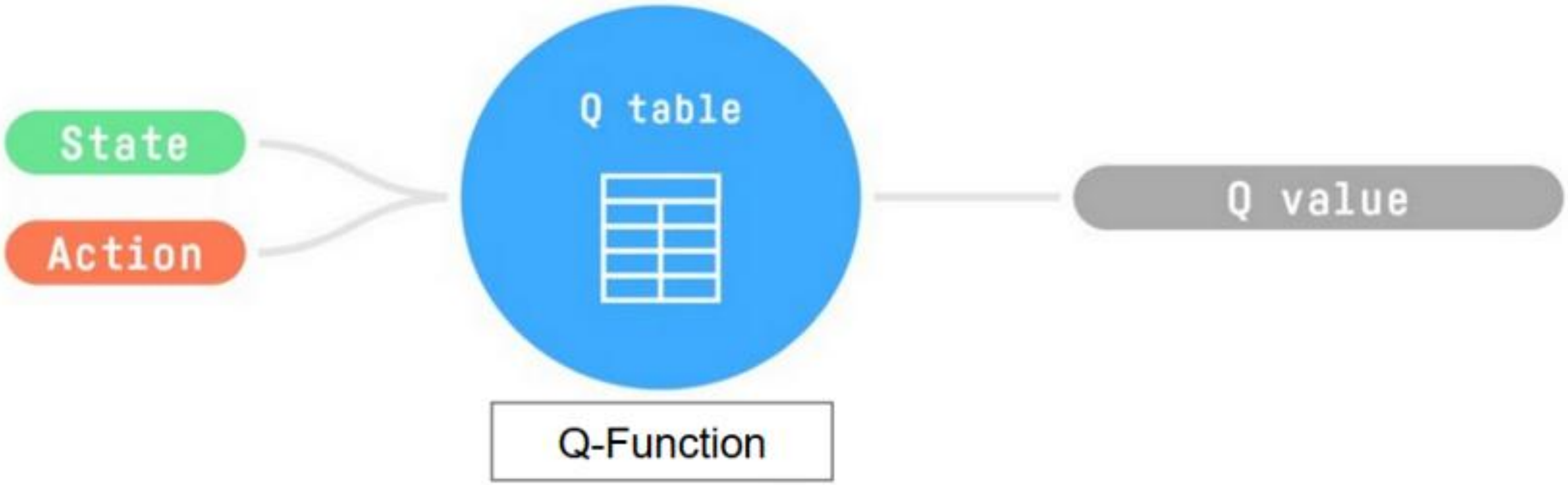


Neural Networks and Deep Learning: A Simple Review

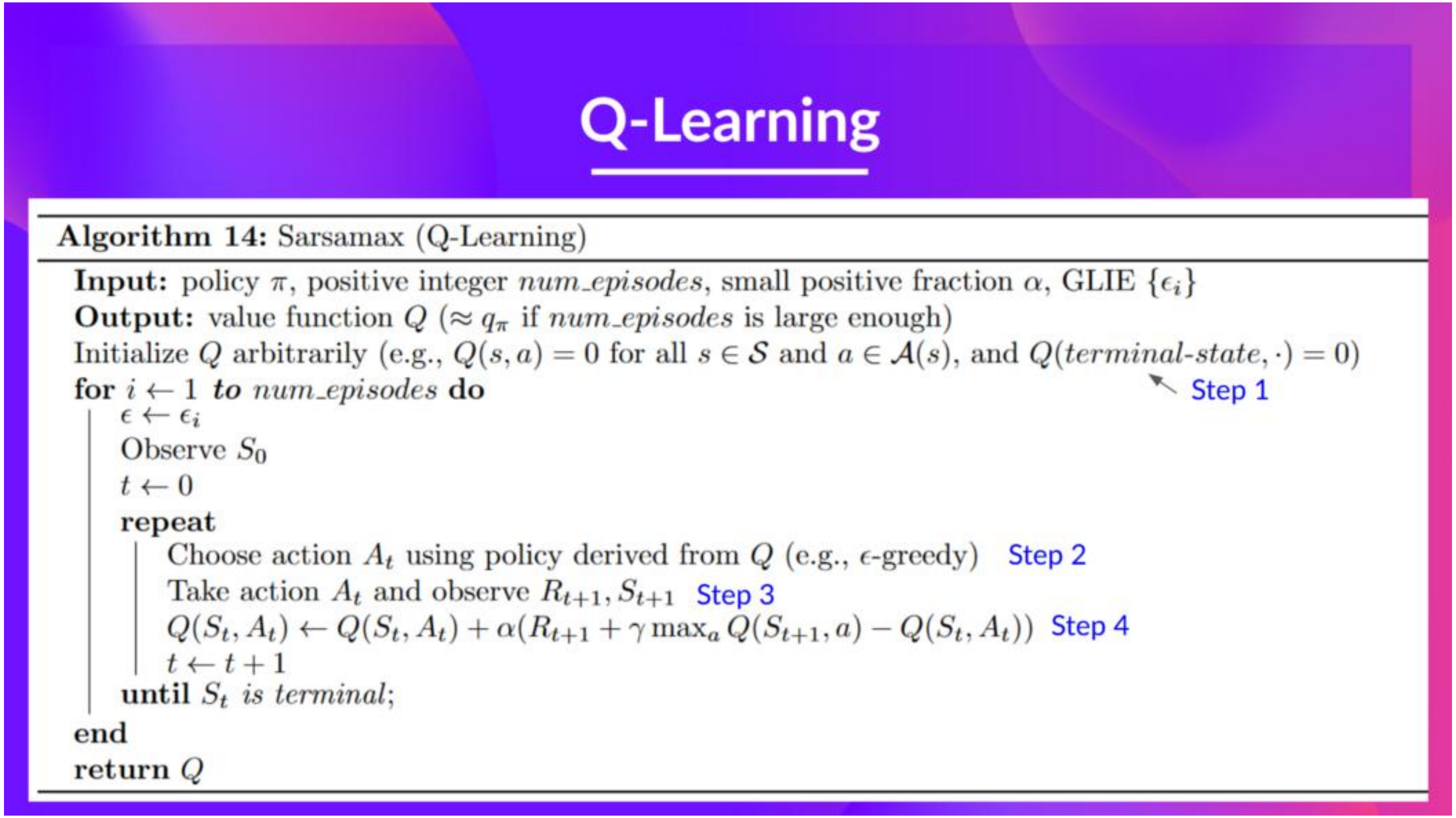
Multi-Layer Perceptron (MLP) (Play!)



Q-Learning Recap …



Pseudocode

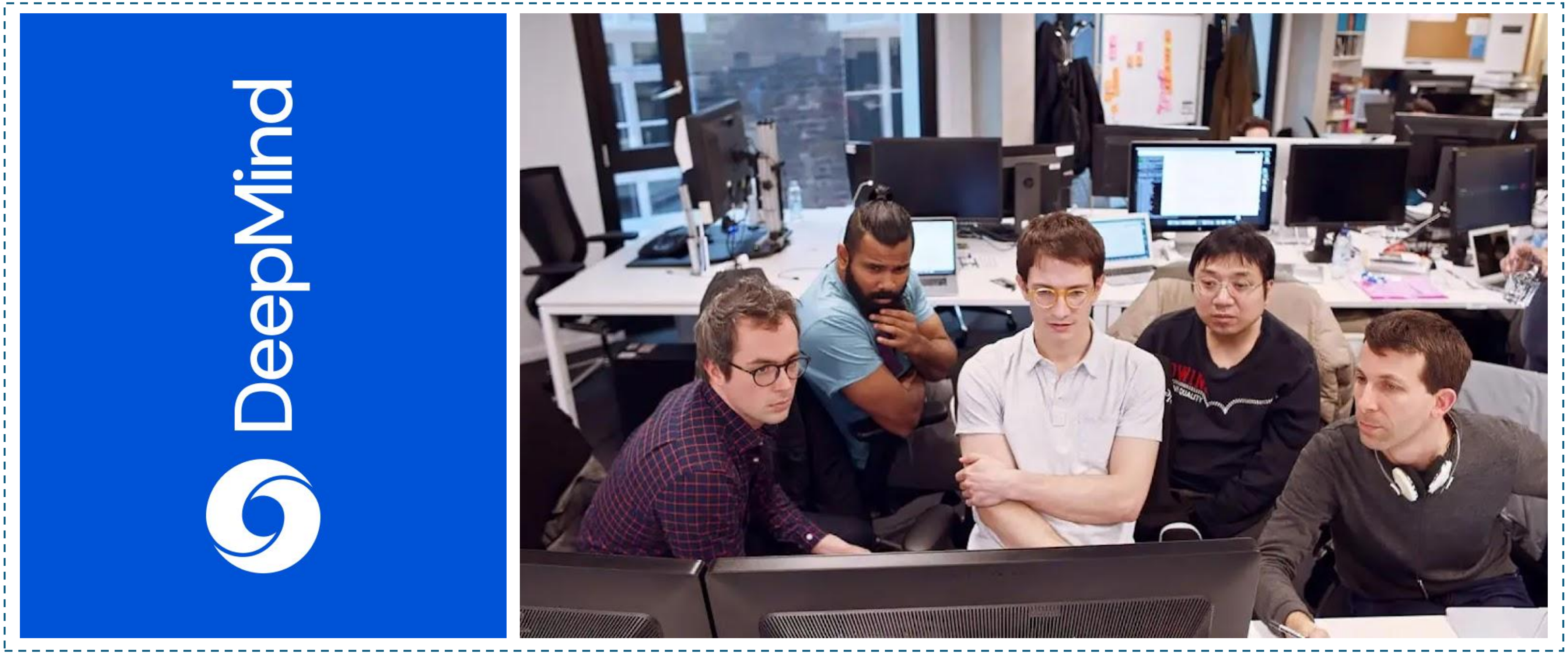


Q-Learning

Algorithm 14: Sarsamax (Q-Learning)

```
Input: policy $\pi$, positive integer $num\_episodes$, small positive fraction $\alpha$, 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 $Q(terminal\text{-}state, \cdot) = 0$)   Step 1
for $i \leftarrow 1$ to $num\_episodes$ do
    $\epsilon \leftarrow \epsilon_i$
    Observe $S_0$
    $t \leftarrow 0$
    repeat
        Choose action $A_t$ using policy derived from $Q$ (e.g., $\epsilon$-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?



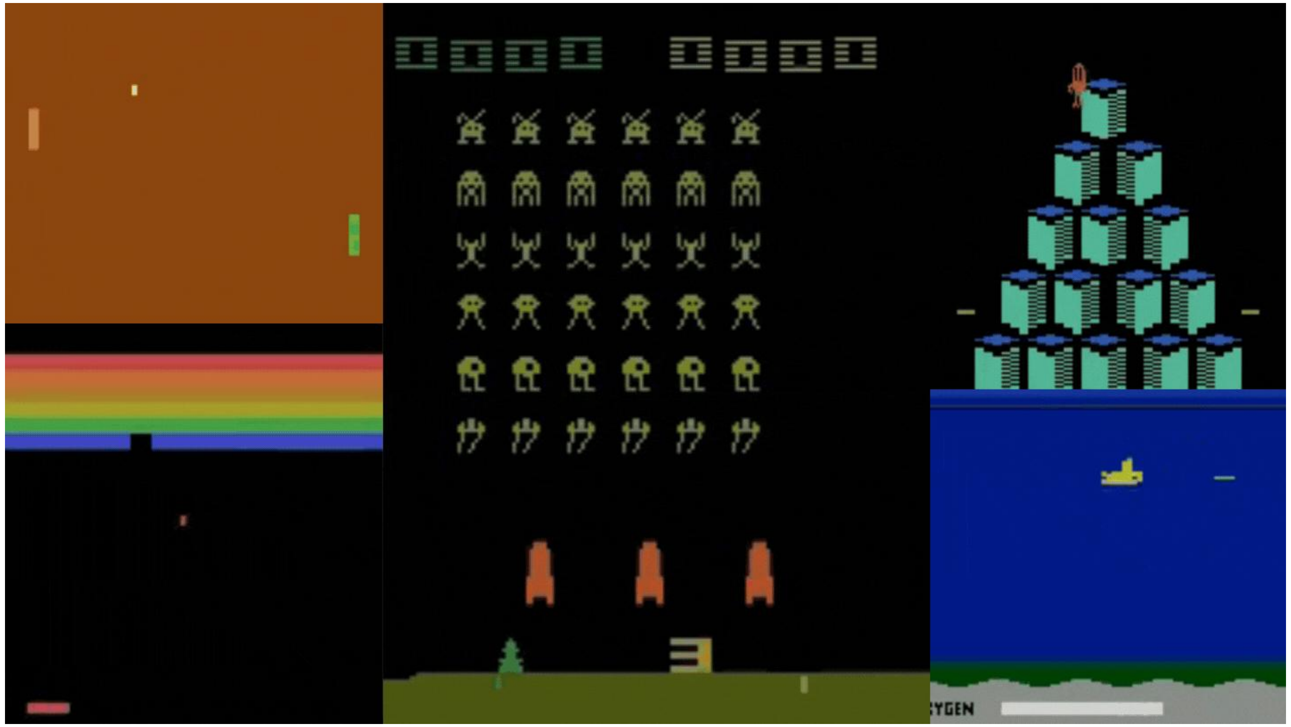
Playing Atari with Deep Reinforcement Learning

(Paper)

Why Deep Reinforcement Learning?

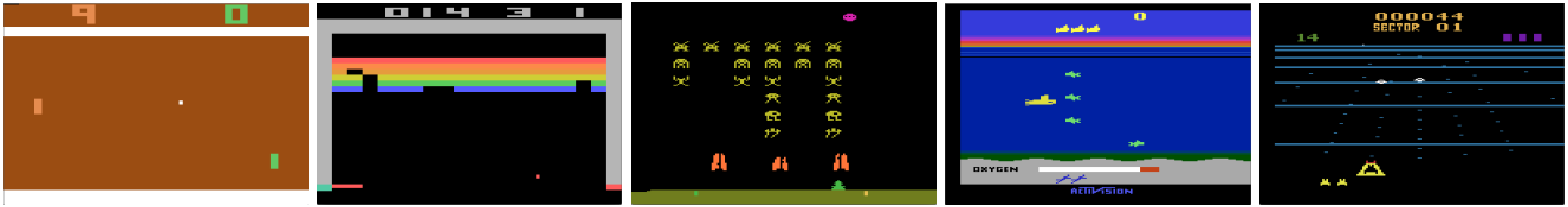
"Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL)."

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?

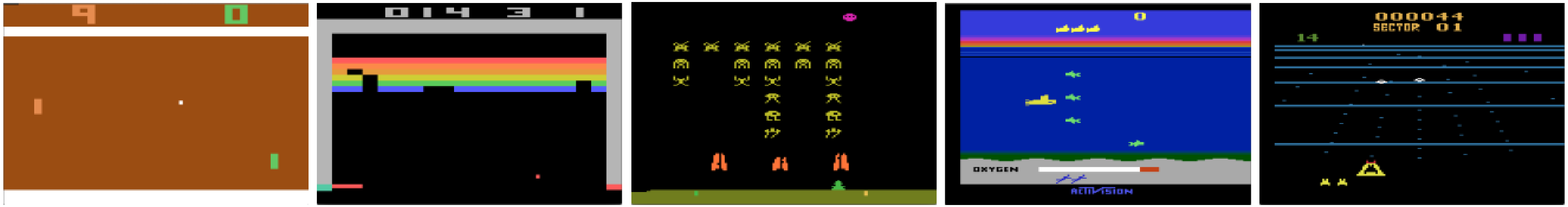


(*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

*“Atari 2600 is a challenging RL testbed that presents agents with a high dimensional visual input (**210 × 160 RGB** video at 60Hz) and a diverse and interesting set of tasks that were designed to be difficult for humans players. “*

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

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

Playing Atari with Deep Reinforcement Learning (Paper)

DeepRL: Why Is It Still Hard?

Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data.

- RL algorithms, on the other hand, must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed.

“The delay between actions and resulting rewards, which can be thousands of timesteps long, seems particularly daunting when compared to the direct association between inputs and targets found in supervised learning.”

Playing Atari with Deep Reinforcement Learning (Paper)

DeepRL: Why Is It Still Hard?

Another issue is that most deep learning algorithms assume the data samples to be independent (IID), while in reinforcement learning one typically encounters sequences of highly correlated states.

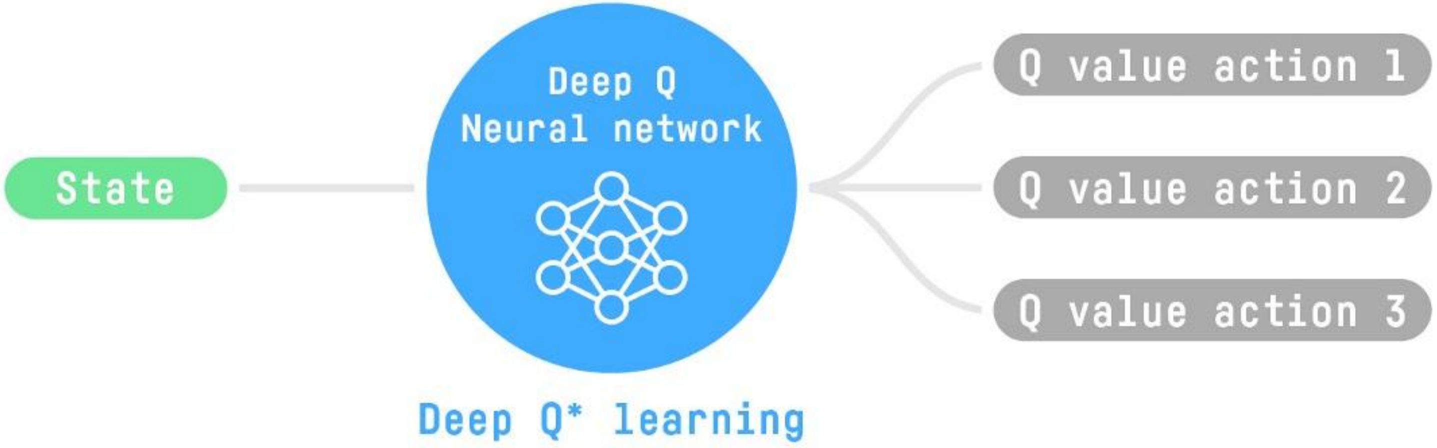
Furthermore, in RL the data distribution changes as the algorithm learns new behaviours, which can be problematic for deep learning methods that assume a fixed underlying distribution.

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

This neural network will approximate, given a state, the different Q-values for each possible action at that state.
And that's exactly what Deep Q-Learning does.

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

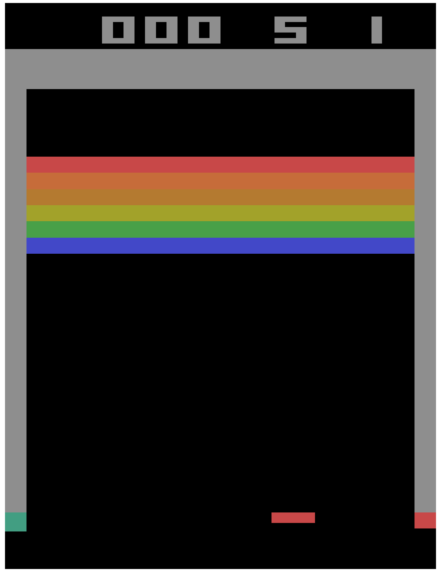


Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

“We consider tasks in which an agent interacts with an environment ε, in this case the Atari emulator.”

Goal: Interact with the emulator ε by selecting actions in a way that 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)

Intuition: If the optimal value $Q^*(s', a')$ of the sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximizing the expected value of $r + \gamma Q^*(s', a')$

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

Reminder Box: Value Iteration

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

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

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)

“We refer to a neural network function approximator with weights θ as a Q-network. A Q-network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i:”

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

“Where

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

is the target for iteration i, and ρ(s,a) is a probability distribution over sequences s and actions a that we refer to as the behaviour distribution.”

Note: The targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins.

Playing Atari with Deep Reinforcement Learning (Paper)

Deep Q-Learning (DQN)

Reminder Box

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

“Differentiating the loss function with respect to the weights we arrive at the following gradient:”

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

“Rather than computing the full expectations in the above gradient, it is often computationally expedient to optimize the loss function by stochastic gradient descent.”

Note: This algorithm is model-free and off-policy.

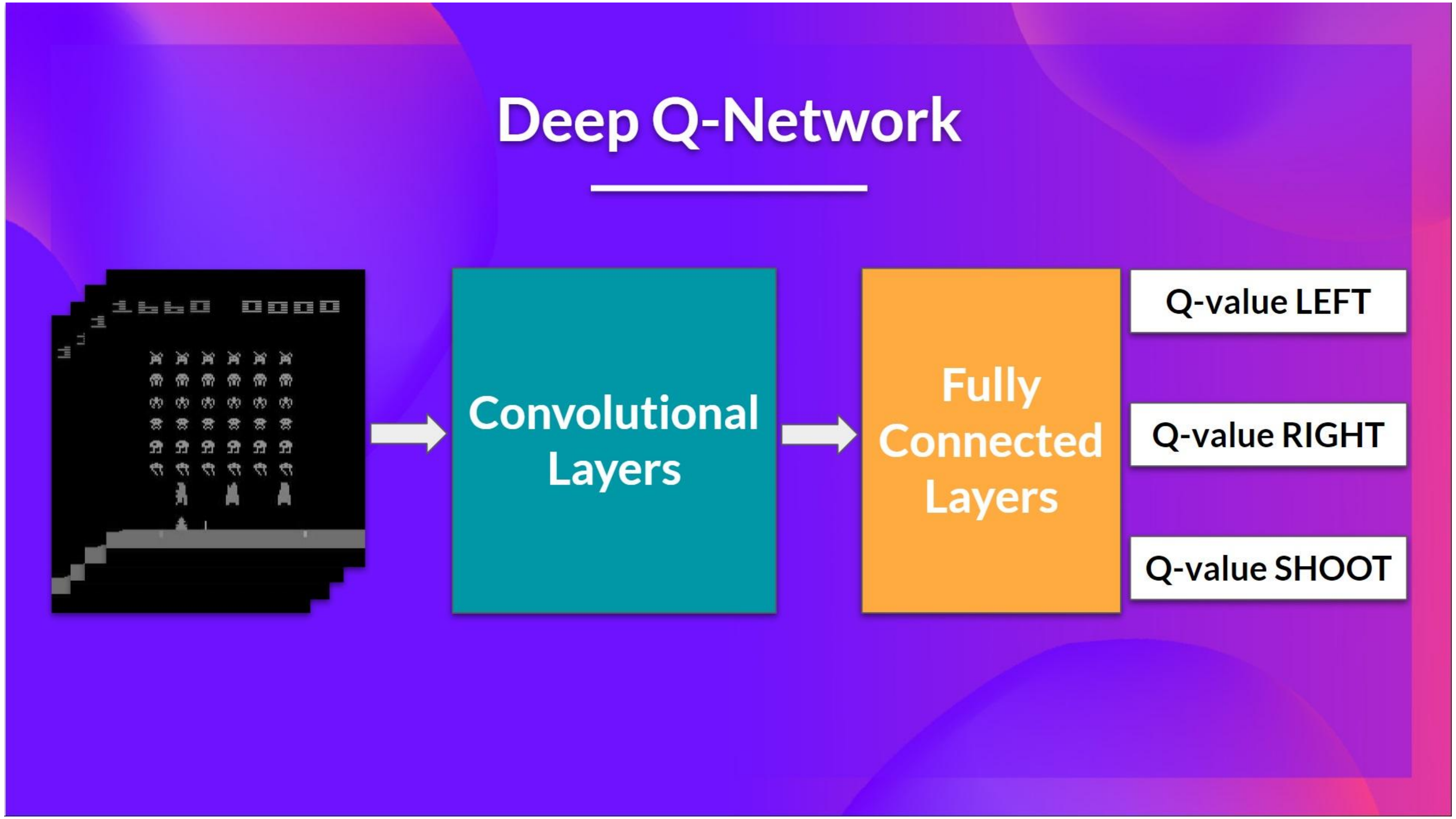
Playing Atari with Deep Reinforcement Learning (Paper)

Architecture

Input: Stack of 4 gray-scale frames

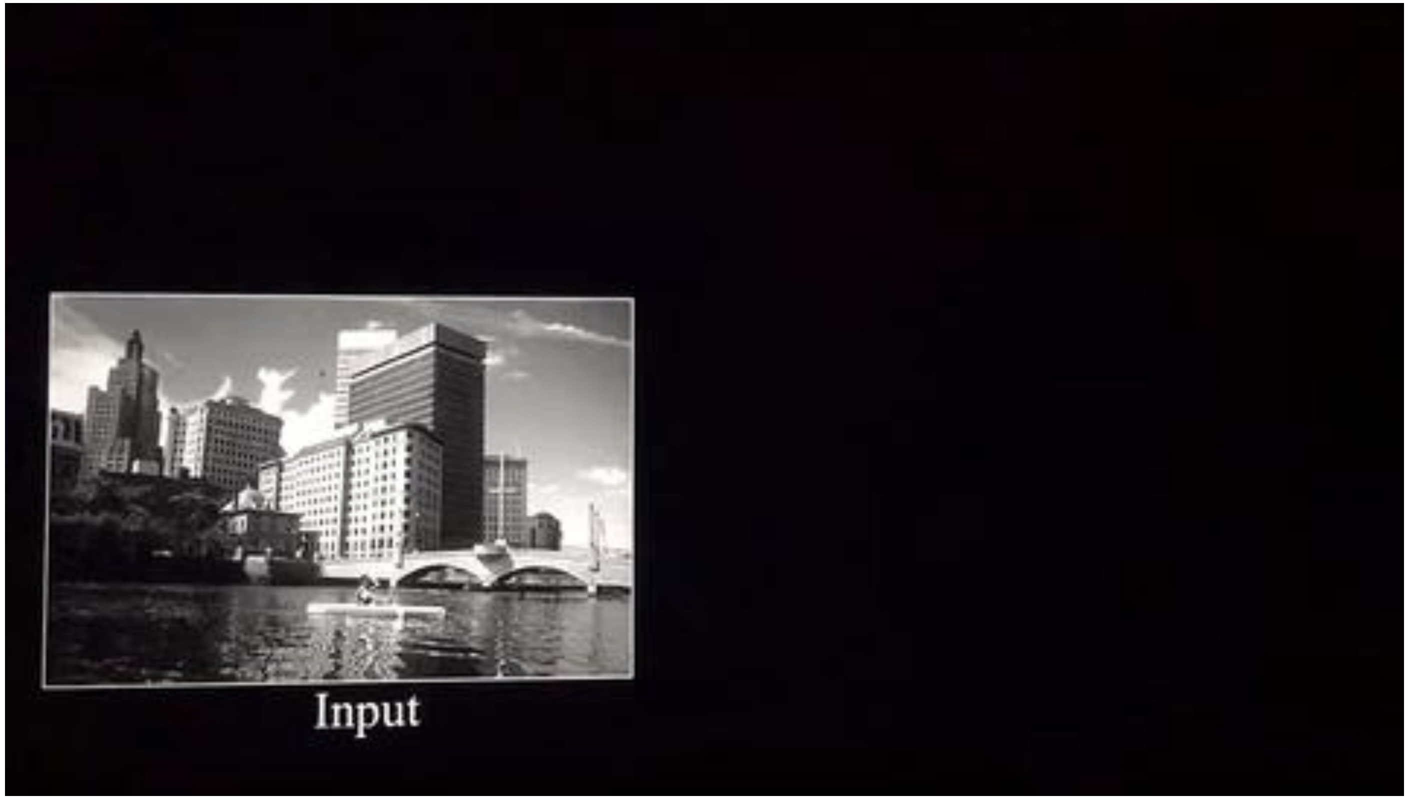
Q: Why?

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)

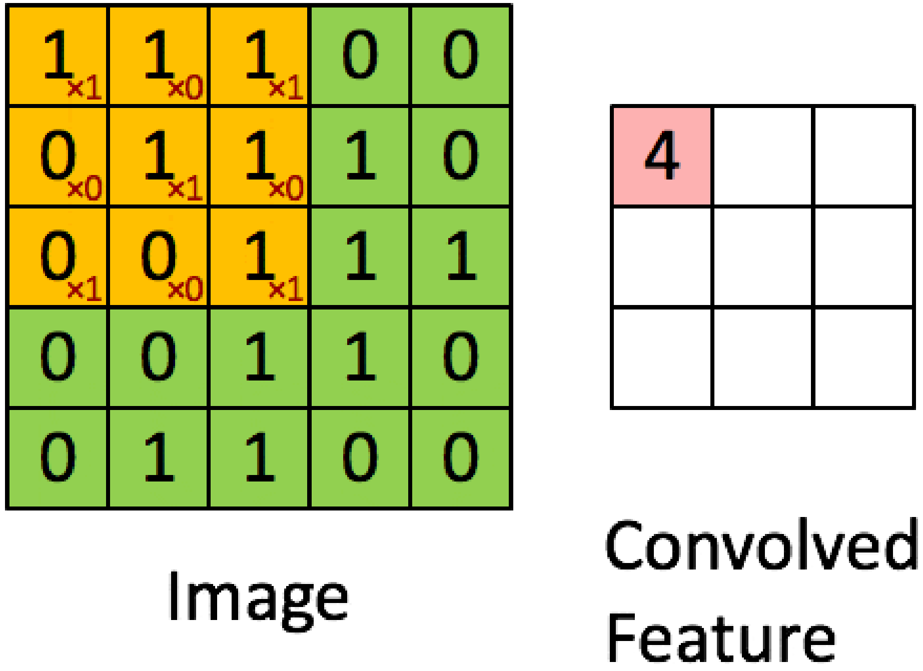
Architecture: What Is a Convolutional Layer?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

1	0	1
0	1	0
1	0	1

Kernel/Filter



$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

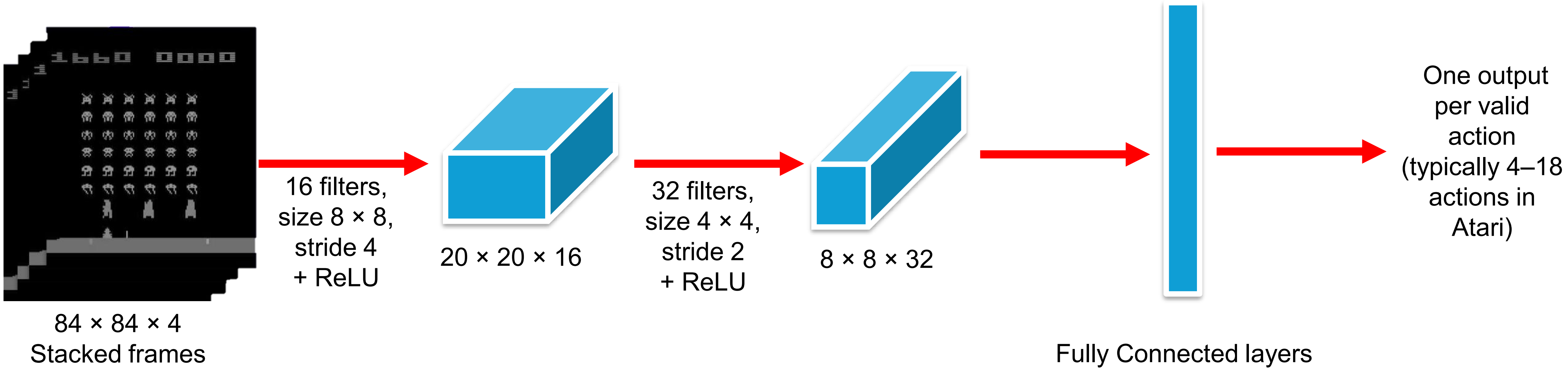
4		

Convolved Feature

Convolutional Layers

Playing Atari with Deep Reinforcement Learning (Paper)

Architecture

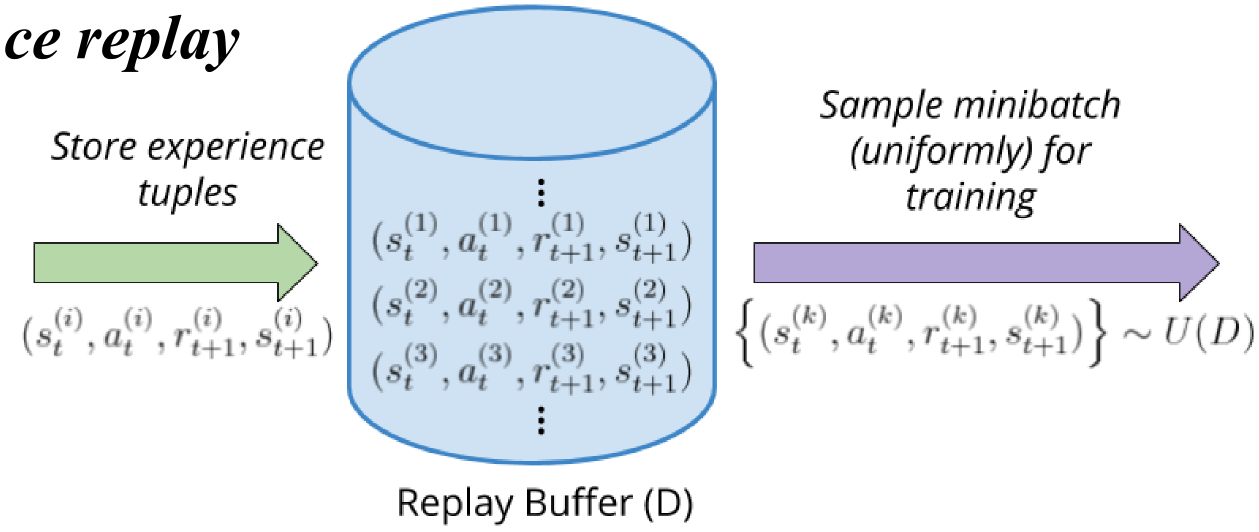


This architecture is known as a Deep Q-Network (DQN)

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm: Replay Buffer

“We utilize a technique known as ***experience replay***

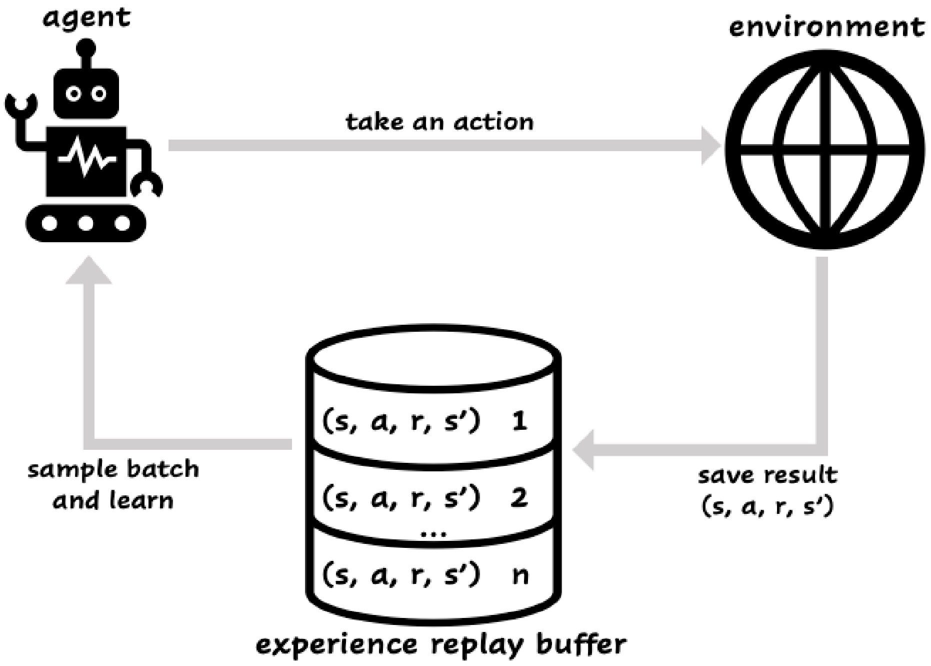

Store experience tuples
$(s_t^{(i)}, a_t^{(i)}, r_{t+1}^{(i)}, s_{t+1}^{(i)})$
⋮
$(s_t^{(1)}, a_t^{(1)}, r_{t+1}^{(1)}, s_{t+1}^{(1)})$
$(s_t^{(2)}, a_t^{(2)}, r_{t+1}^{(2)}, s_{t+1}^{(2)})$
$(s_t^{(3)}, a_t^{(3)}, r_{t+1}^{(3)}, s_{t+1}^{(3)})$
⋮
Replay Buffer (D)
Sample minibatch (uniformly) for training
$\{(s_t^{(k)}, a_t^{(k)}, r_{t+1}^{(k)}, s_{t+1}^{(k)})\} \sim U(D)$

where we store the agent’s experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data-set $D = e_1, \ldots, e_N$, pooled over many episodes into a replay memory. During the inner loop of the algorithm, we apply Q-learning updates, or minibatch updates, to samples of experience, $e \sim D$, drawn at random from the pool of stored samples.”

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm: Replay Buffer

"In practice, our algorithm only stores the last N experience tuples in the replay memory, and samples uniformly at random from D when performing updates."

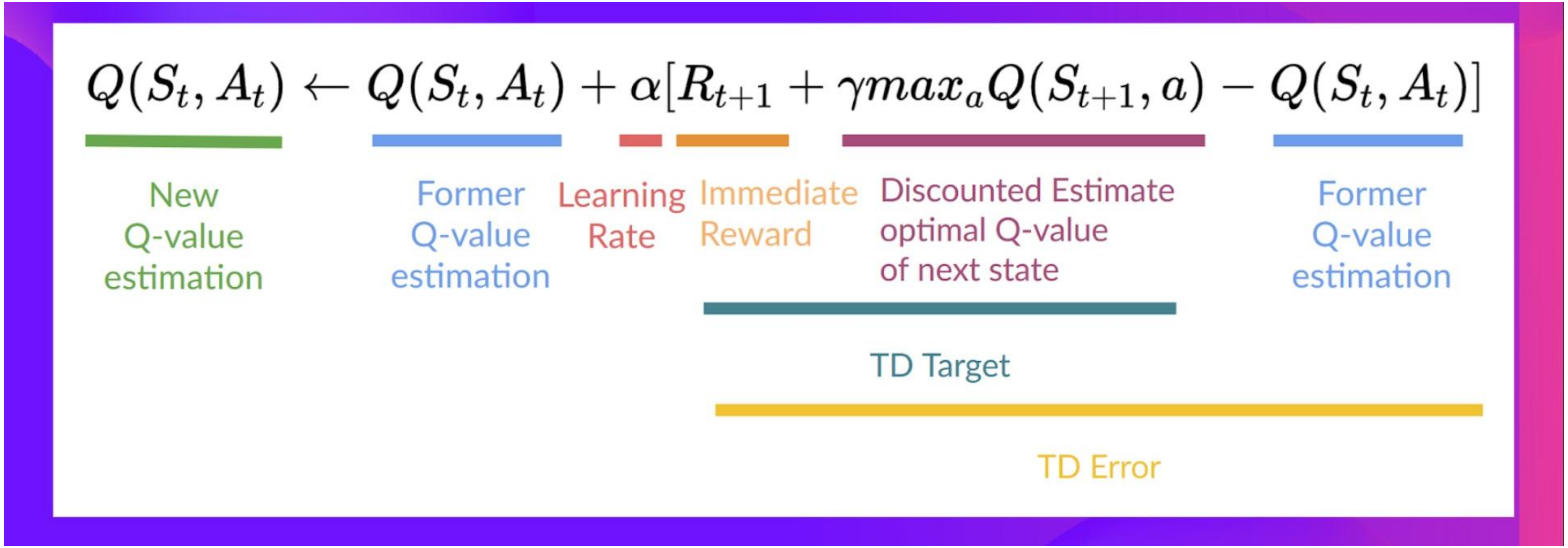


Q: Why Replay Buffer?

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm

Reminder Box



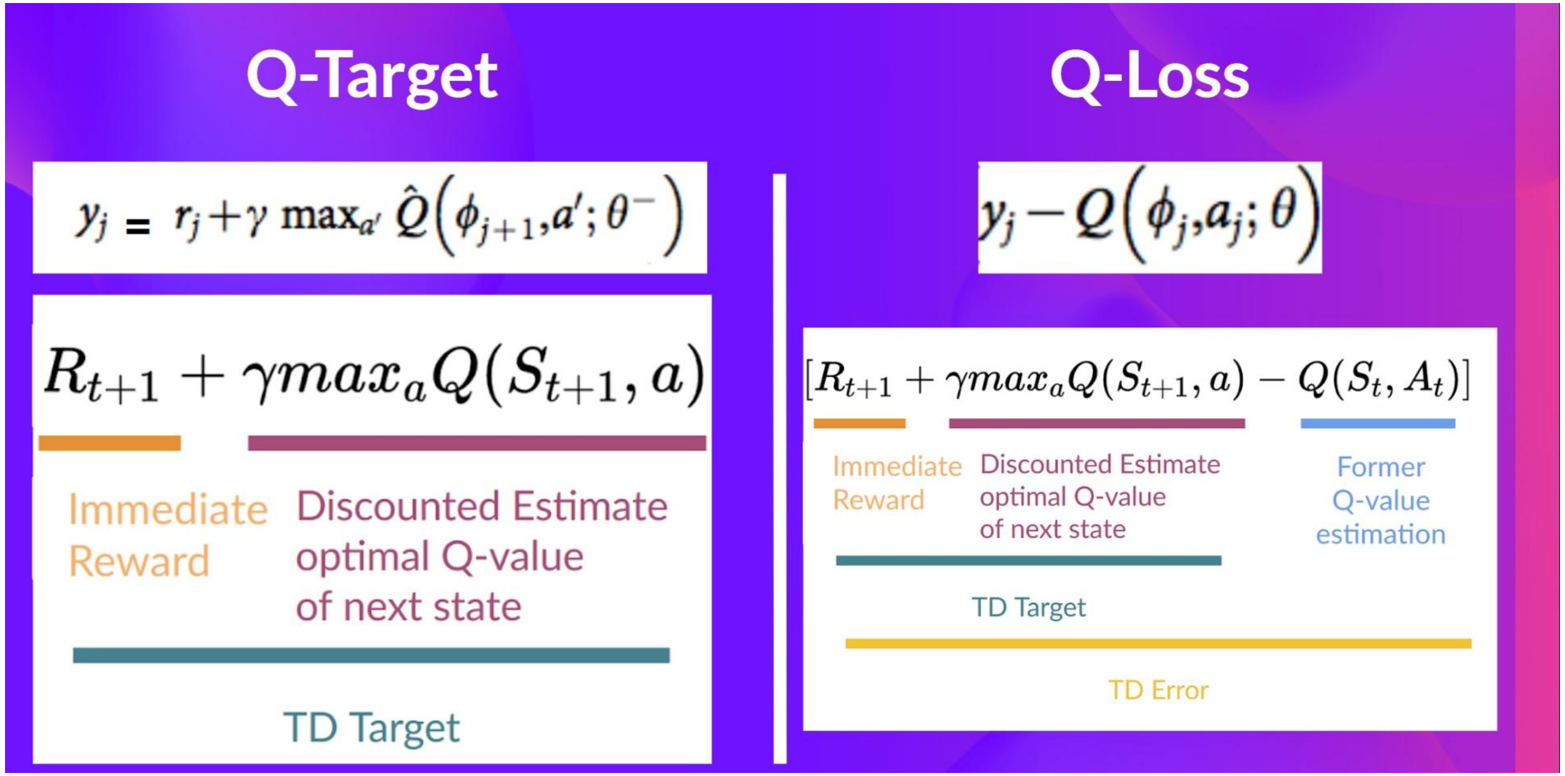
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

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



Q-Target

$$y_j = r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right)$$

$$\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}}$$

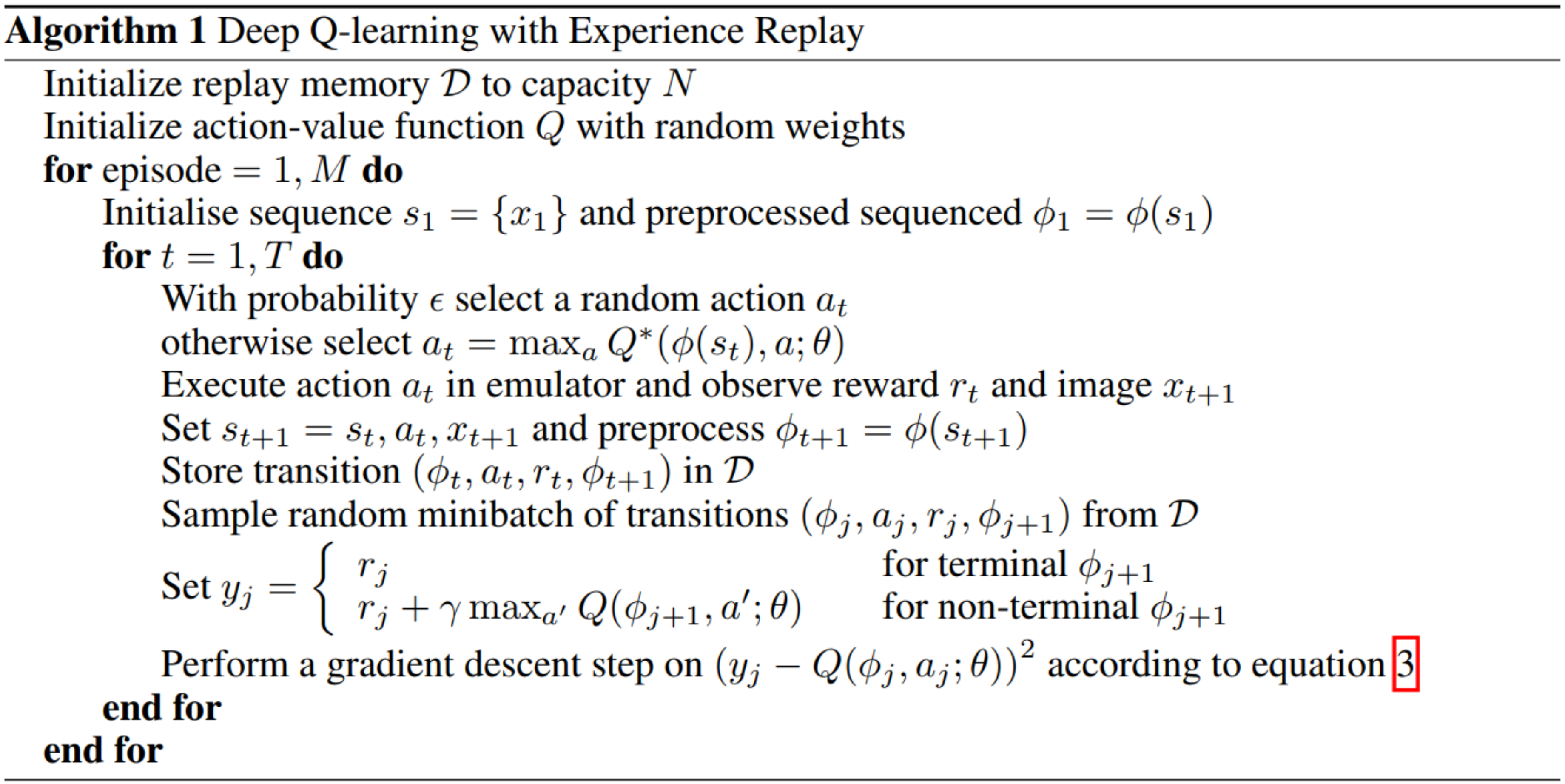
Q-Loss

$$y_j - Q\left(\phi_j, a_j; \theta\right)$$

$$\underbrace{[\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}]}_{\text{TD Error}}$$

Playing Atari with Deep Reinforcement Learning (Paper)

Algorithm

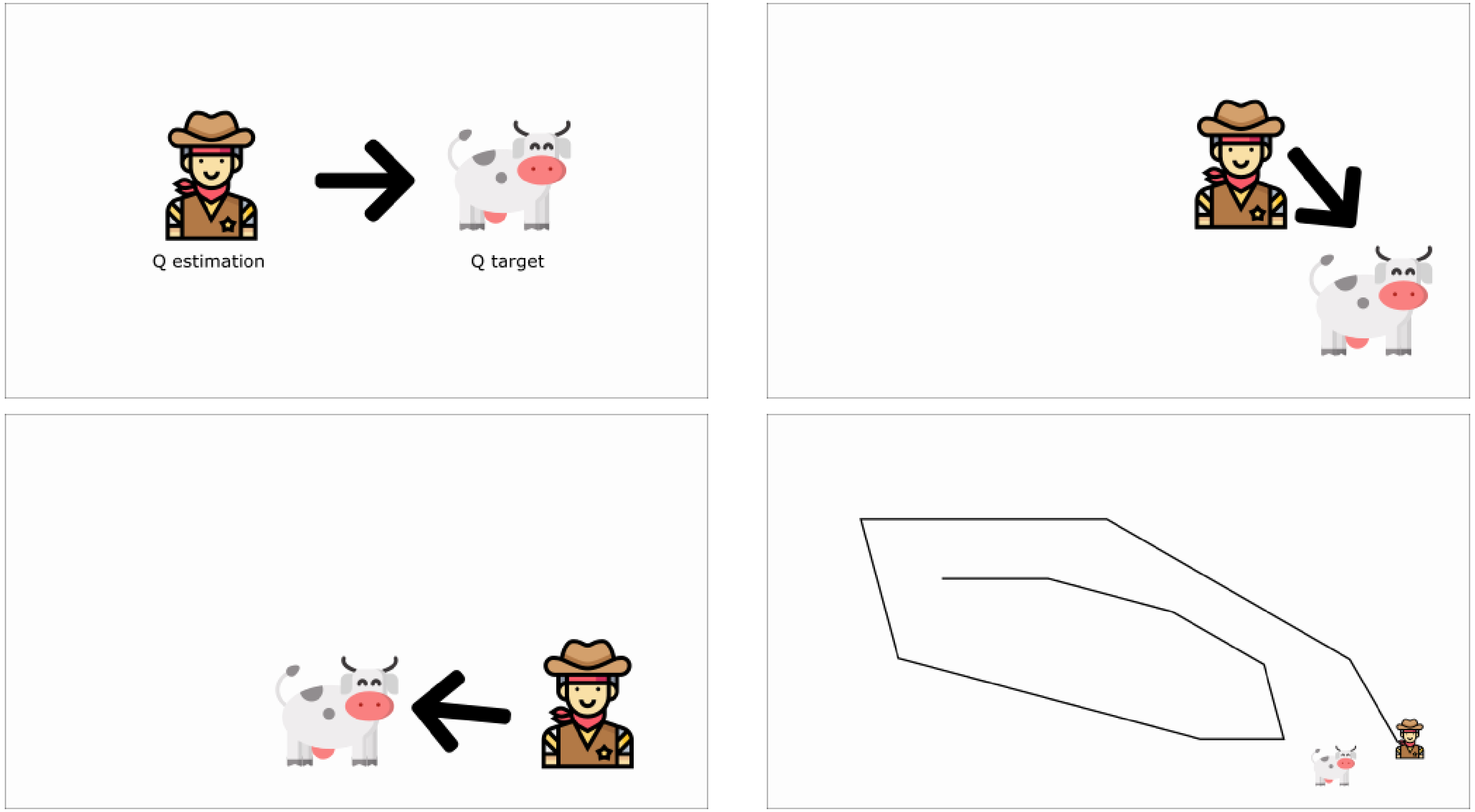


Algorithm 1 Deep Q-learning with Experience Replay

```
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 $\epsilon$ 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)

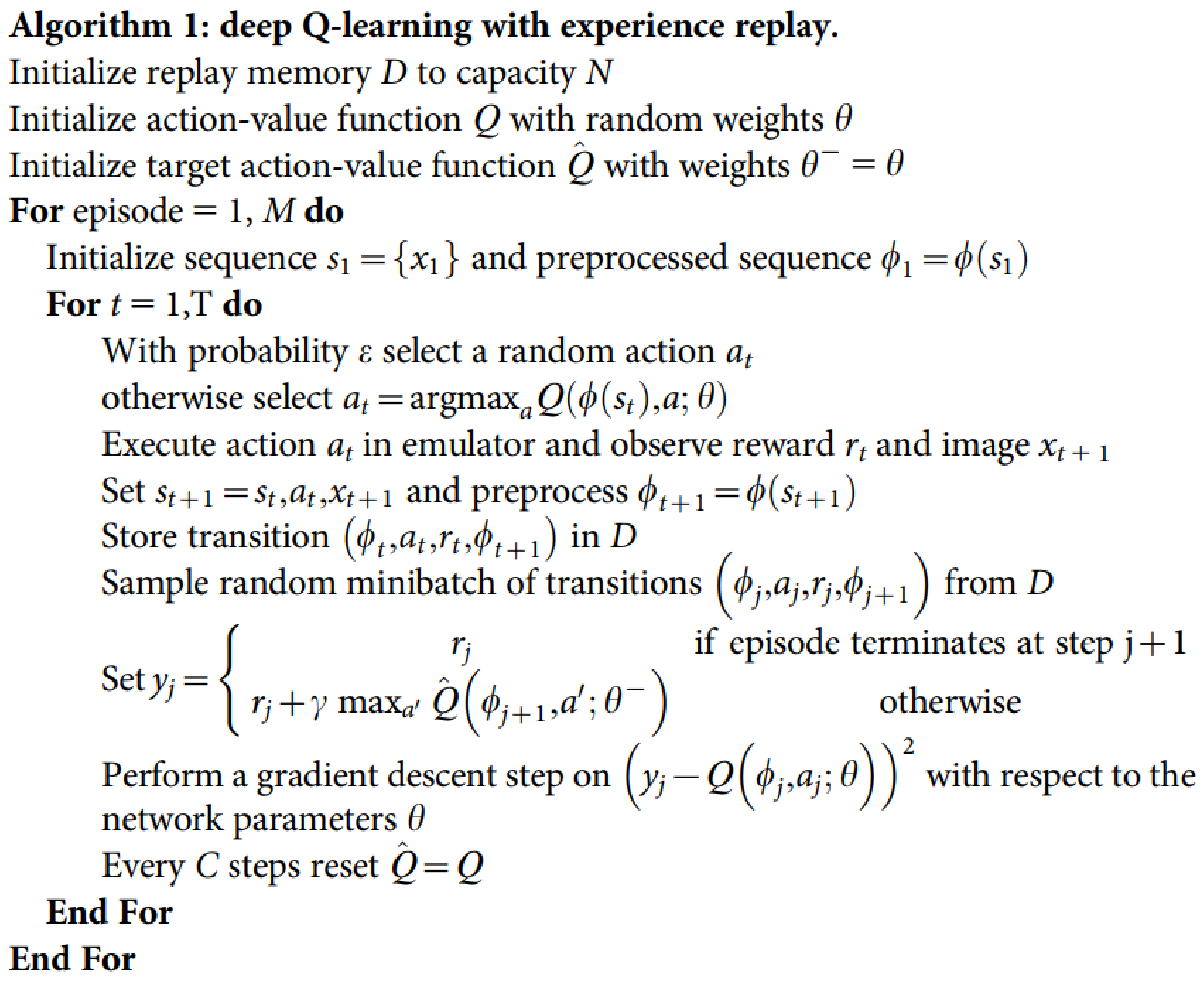
Algorithm: A Challenge!

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 $\theta$
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 $\varepsilon$ 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 $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```

Playing Atari with Deep Reinforcement Learning (Paper, Paper)

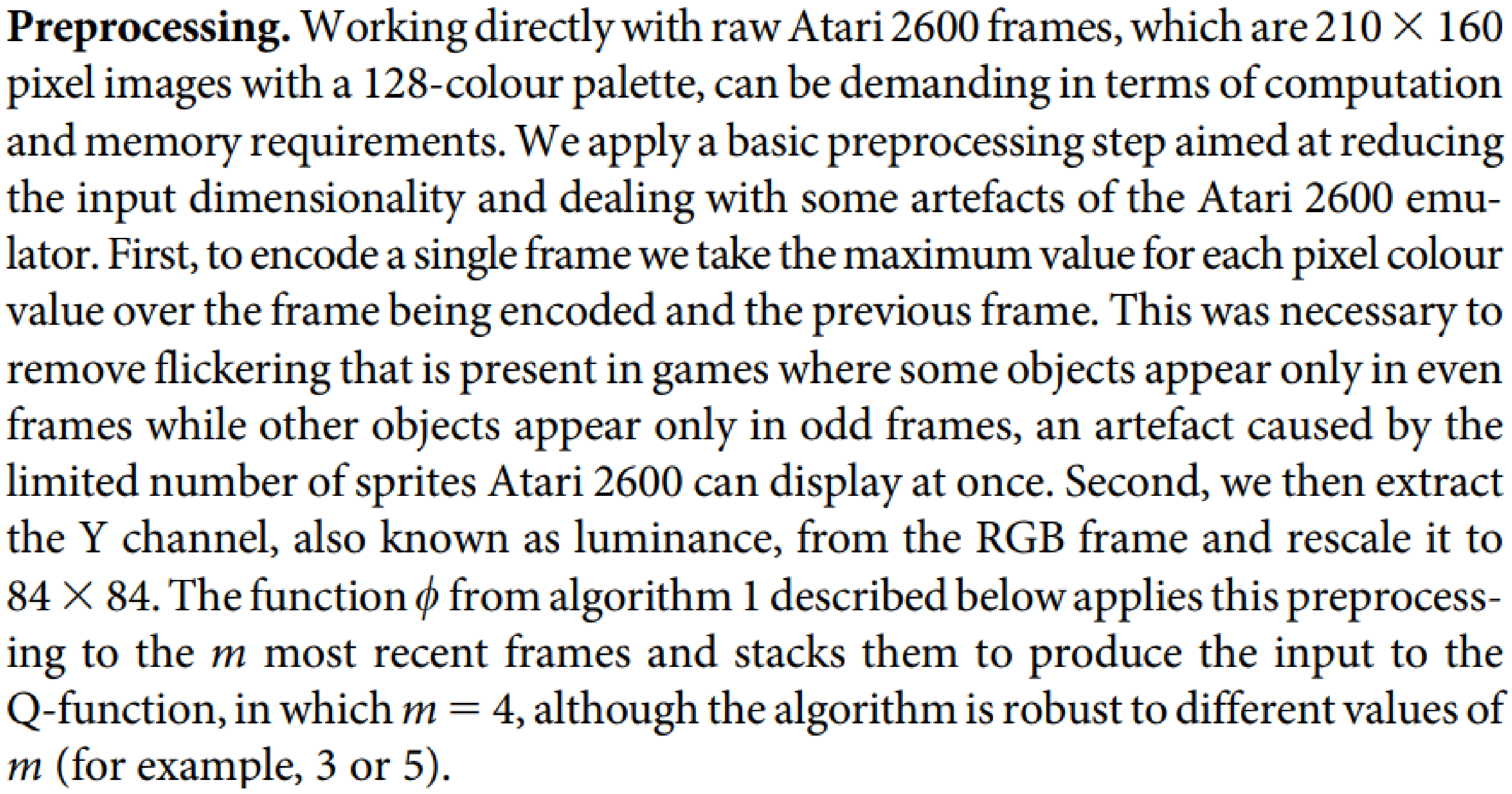
Conclusion

“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)