

## COMMONWEALTH OF AUSTRALIA

### *Copyright Regulations 1969*

#### WARNING

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the *Copyright Act 1968* (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice.



# Acknowledgements

**This material includes content adapted from instructional resources made available by David Silver as part of his Reinforcement Learning course at UCL under CC-BY-NC 4.0.**

**Refer to <https://www.davidsilver.uk/teaching/> for full details.**



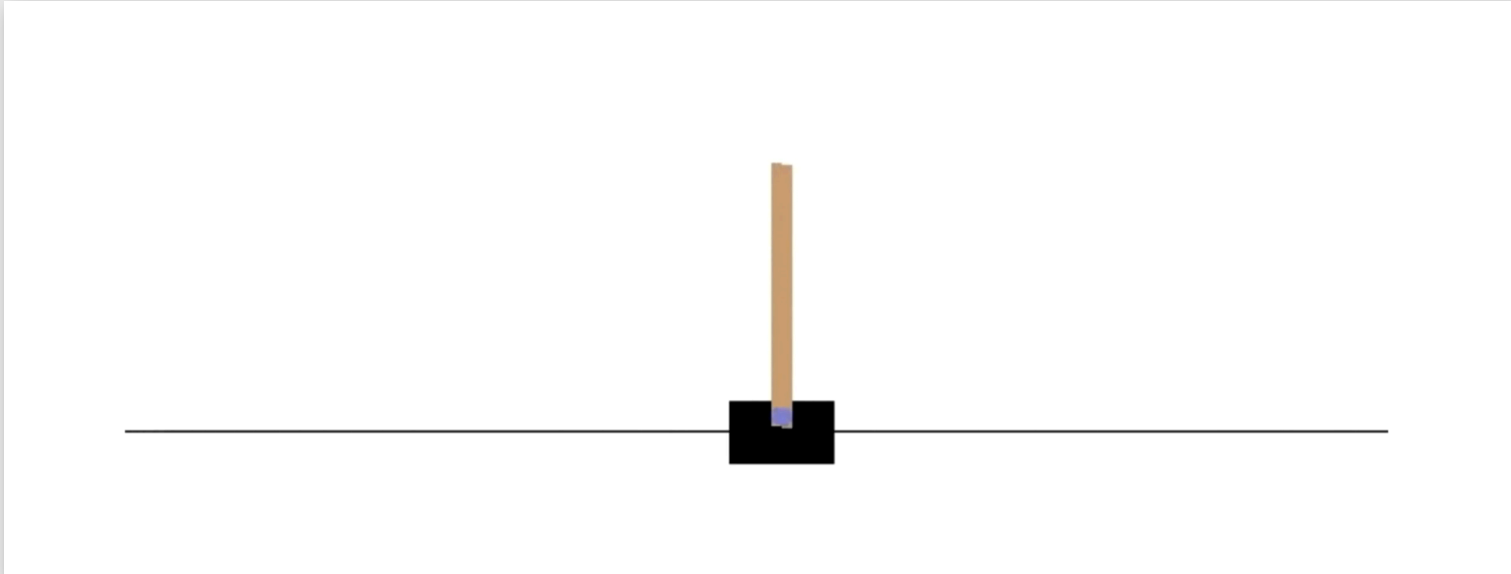


FIT5226

Multi-agent System & Collective Behaviour

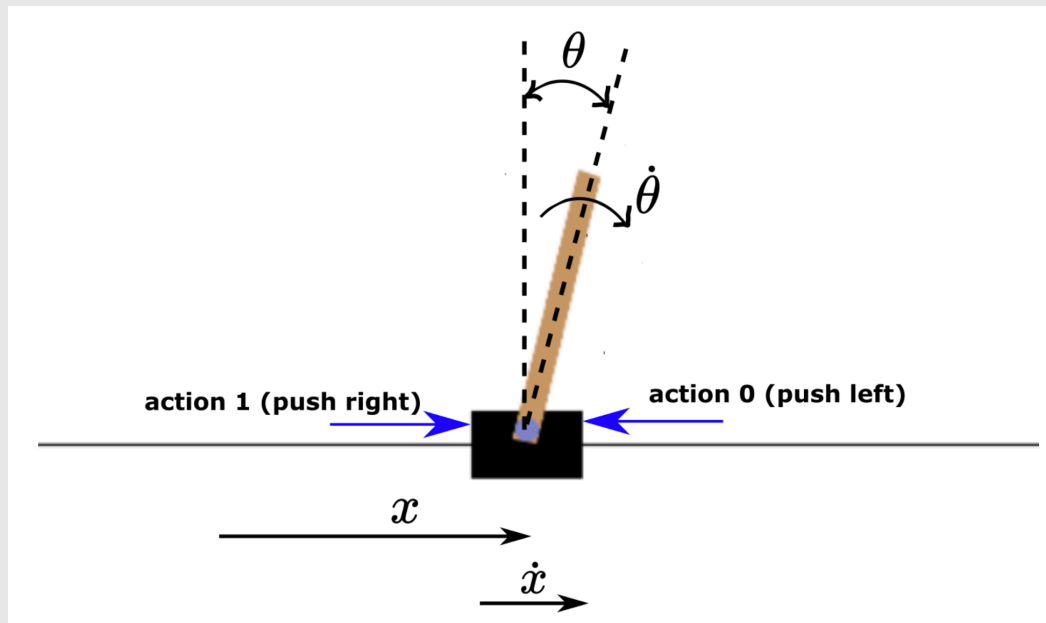
Wk 5: Deep Reinforcement and Deep-Q

# The Cart Pole Problem



cart pole or inverted pendulum

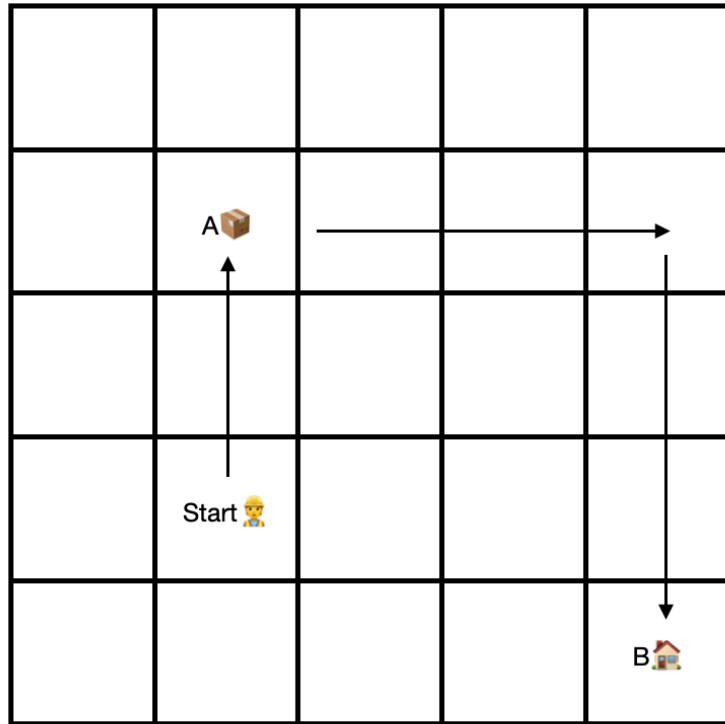
# Continuous State Spaces



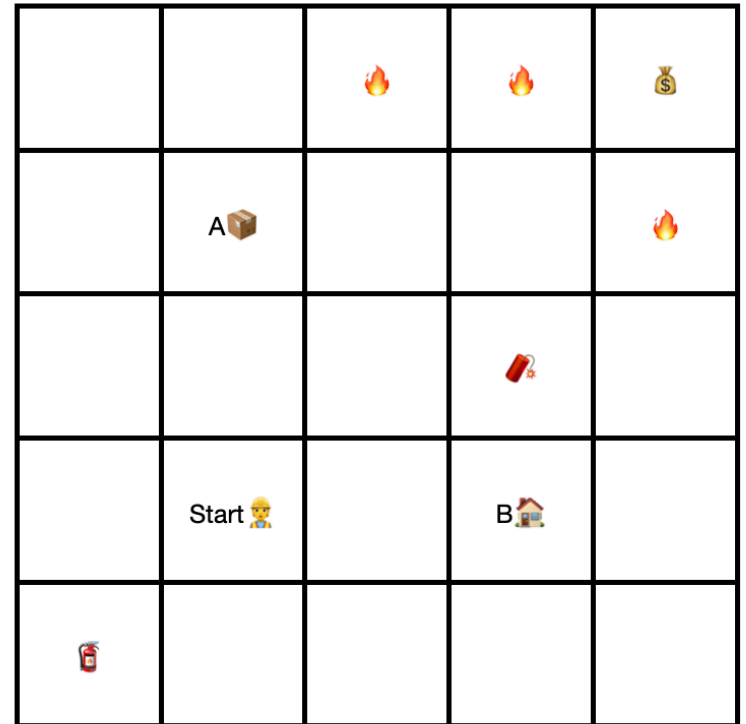
## Observation (state) space

- Cart Position ( $x$ ): The horizontal position of the cart on the track.
- Cart Velocity ( $dx/dt$ ): The velocity of the cart.
- Pole Angle ( $\theta$ ): The angle of the pole from the vertical upright position.
- Pole Angular Velocity ( $d\theta/dt$ ): The angular velocity of the pole.

# State Space Size



$$\approx 25^2 = 625$$



$$\approx 25^9 = 3,814,697,265,625$$

Backgammon:  $10^{20}$ ; Go:  $10^{170}$

# Generalisation in more complex state spaces

- Mars rover must learn high-level control to traverse terrain
- Driving over hazardous terrain 10x slower

4 actions

	↑		
←	●	→	
	↓		

Distance sensor

4	3	2	1
3	2	1	G
S	3	2	1

Terrain sensor

-1	-1	-10	-10
-1	-1	-1	-1
-1	-10	-1	-10

# State Space for Mars Rover

- State space is a vector of features

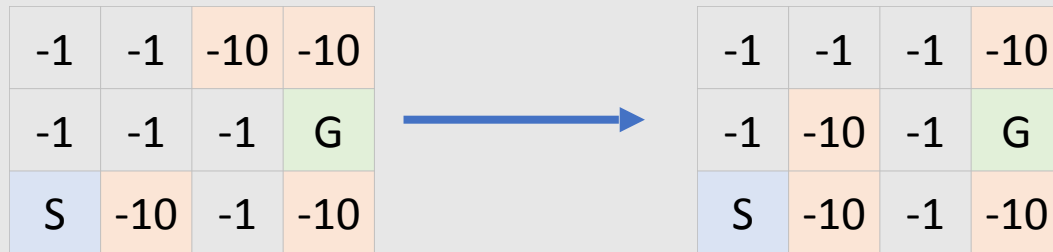
$$s = (D_N, D_E, D_S, D_W, H_N, H_E, H_s, H_W)$$
$$s = (3, 2, 3, 4, 0, 1, 0, 0)$$

- Suppose:
  - Max distance 5, Boolean hazard sensor,
  - Then,  $|S| = 5^4 + 2^4 = 10,000$  states,  $|A| = 4$  actions,
  - 40,000 Q-table entries (about 157 kB of memory)



# Generalisation

- After some exploration, our rover learned to navigate safely
- Now we change the map:



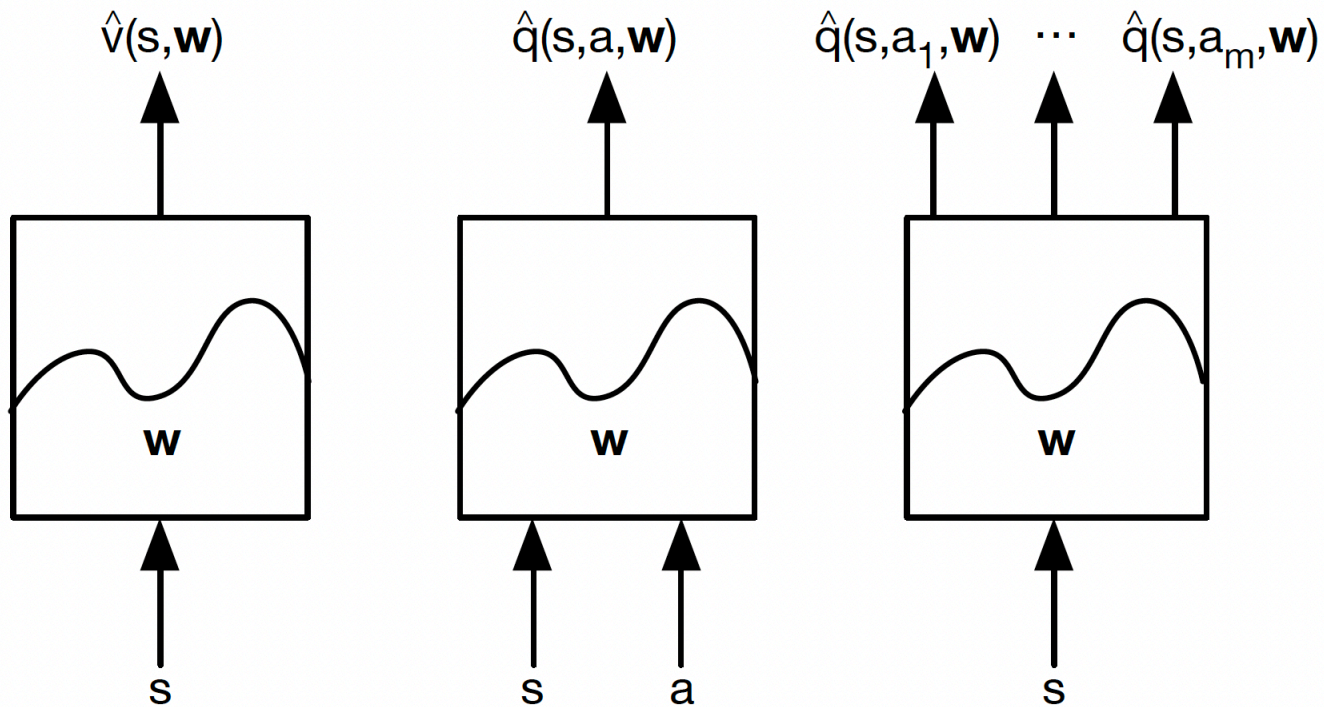
- What will happen?

# Value Function Approximation

---

- Very large state spaces
  - Continuous state spaces
  - Need for generalisation
- ➡ use a value function instead of a value table
- of course, this can only be approximate*

# Value Function Approximation Types

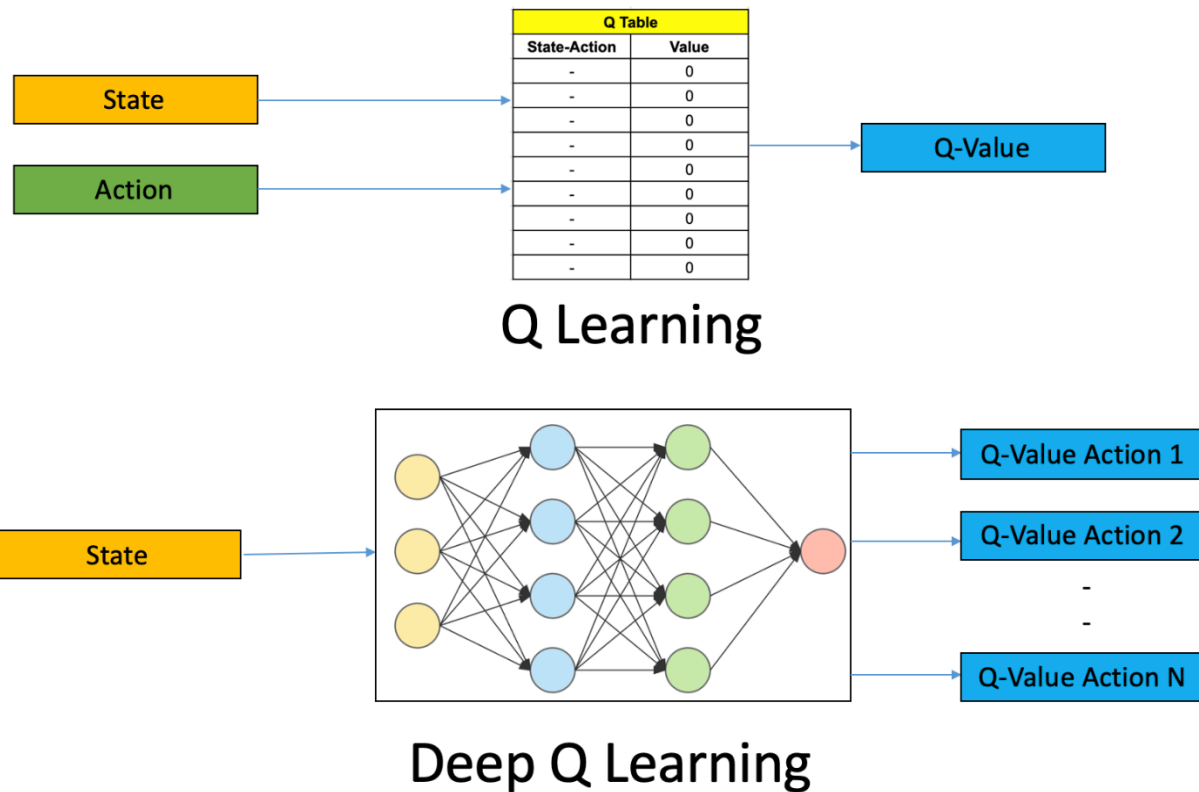


$$F : S \rightarrow \mathbb{R}$$

$$F : S \times A \rightarrow \mathbb{R}$$

$$F : S \rightarrow A \times \mathbb{R}$$

# Value Function Approximation with Deep-Q Network

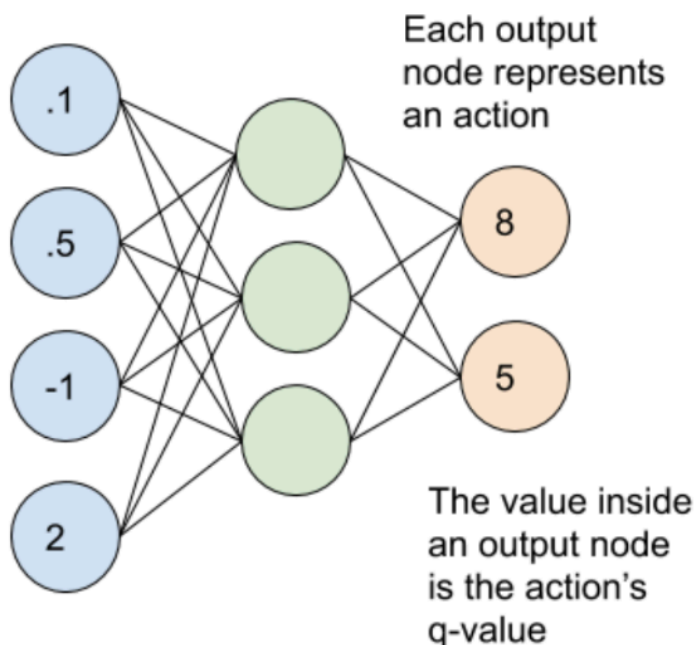


# From Bellman Update to Loss Function

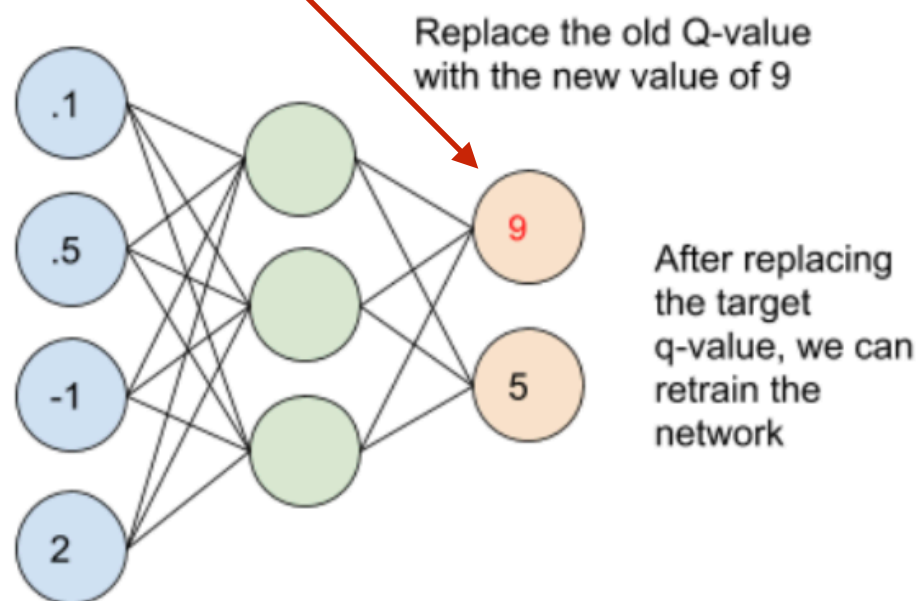
$$Q(S_t, A_t) = (1 - \alpha) Q(S_t, A_t) + \alpha * (R_t + \lambda * \max_a Q(S_{t+1}, a))$$

**TD-target**

Input States



Input States



# DQN Algorithm

Initialize the *Agent* to interact with the Environment

**while** *not converged* **do**

    /\* Sample phase

$\epsilon \leftarrow$  setting new epsilon with  $\epsilon$ -decay

    Choose an action  $a$  from state  $s$  using policy  $\epsilon$ -greedy( $Q$ )

*Agent* takes action  $a$ , observe reward  $r$ , and next state  $s'$

    Store transition  $(s, a, r, s', done)$  in the experience replay memory  $D$

**if** *enough experiences in  $D$*  **then**

        /\* Learn phase

        Sample a random *minibatch* of  $N$  transitions from  $D$

**for** *every transition*  $(s_i, a_i, r_i, s'_i, done_i)$  *in minibatch* **do**

**if**  $done_i$  **then**

$y_i = r_i$

**else**

$y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$

**end**

**end**

        Calculate the loss  $\mathcal{L} = 1/N \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$

        Update  $Q$  using the SGD algorithm by minimizing the loss  $\mathcal{L}$

        Every  $C$  steps, copy weights from  $Q$  to  $\hat{Q}$

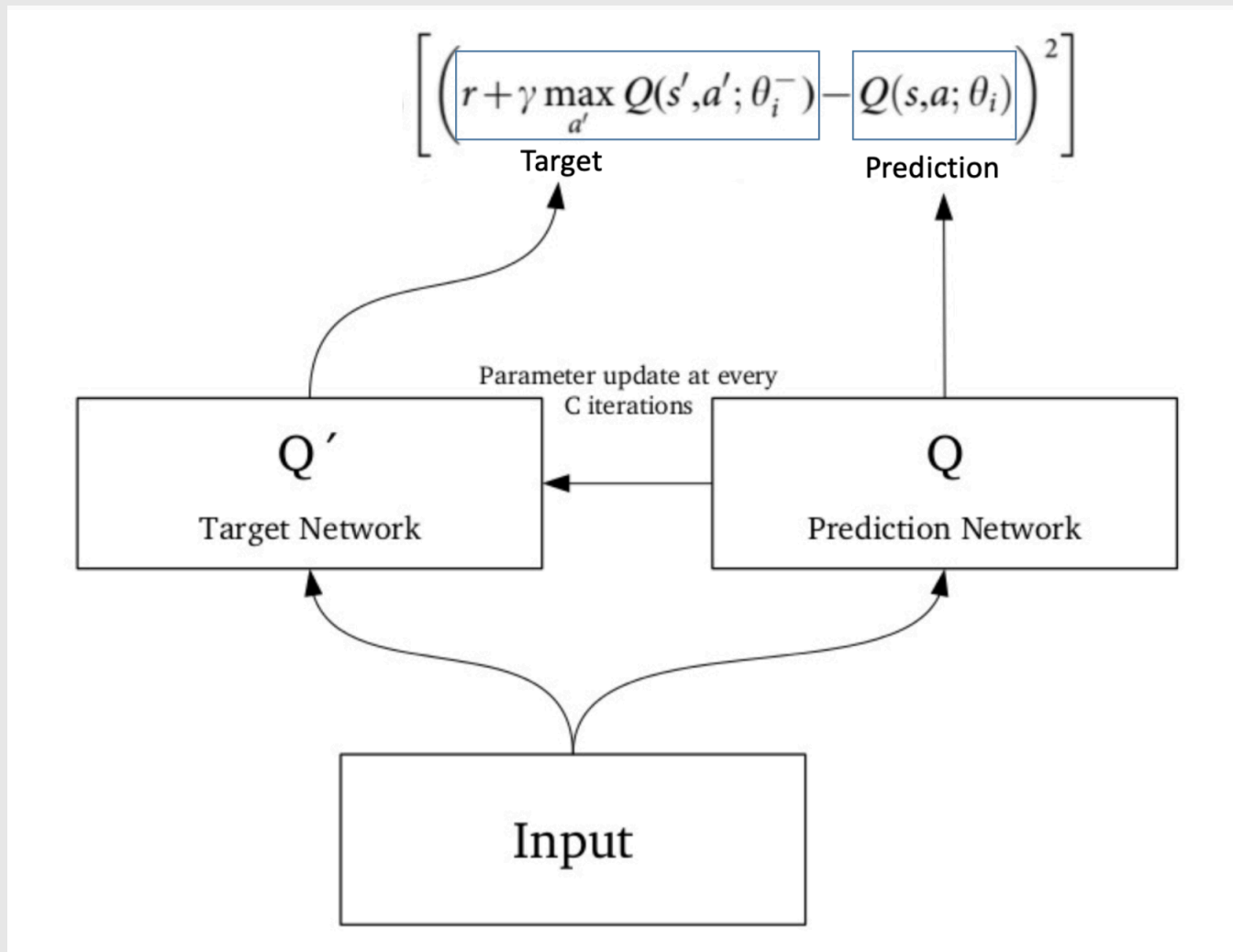
**end**

**end**

# Experience Replay

- Equivalent to standard Q-learning: adapt the network for every action as it is performed (usually using a single SGD step and MSE loss)
- **This can be unstable. To stabilise we perform *Experience Replay***
- All Transitions (state, action, reward, new state) are stored in a ring buffer of limited size
- Instead of just using the latest action for each training step we use a mini-batch to update
  - randomly sample a mini-batch from the buffer
  - compute loss and back-propagate for the whole mini-batch
  - we may do this only every  $k$  actions
- Note that this is off-policy  
(and thus requires an off-policy base algorithm like Q-learning)

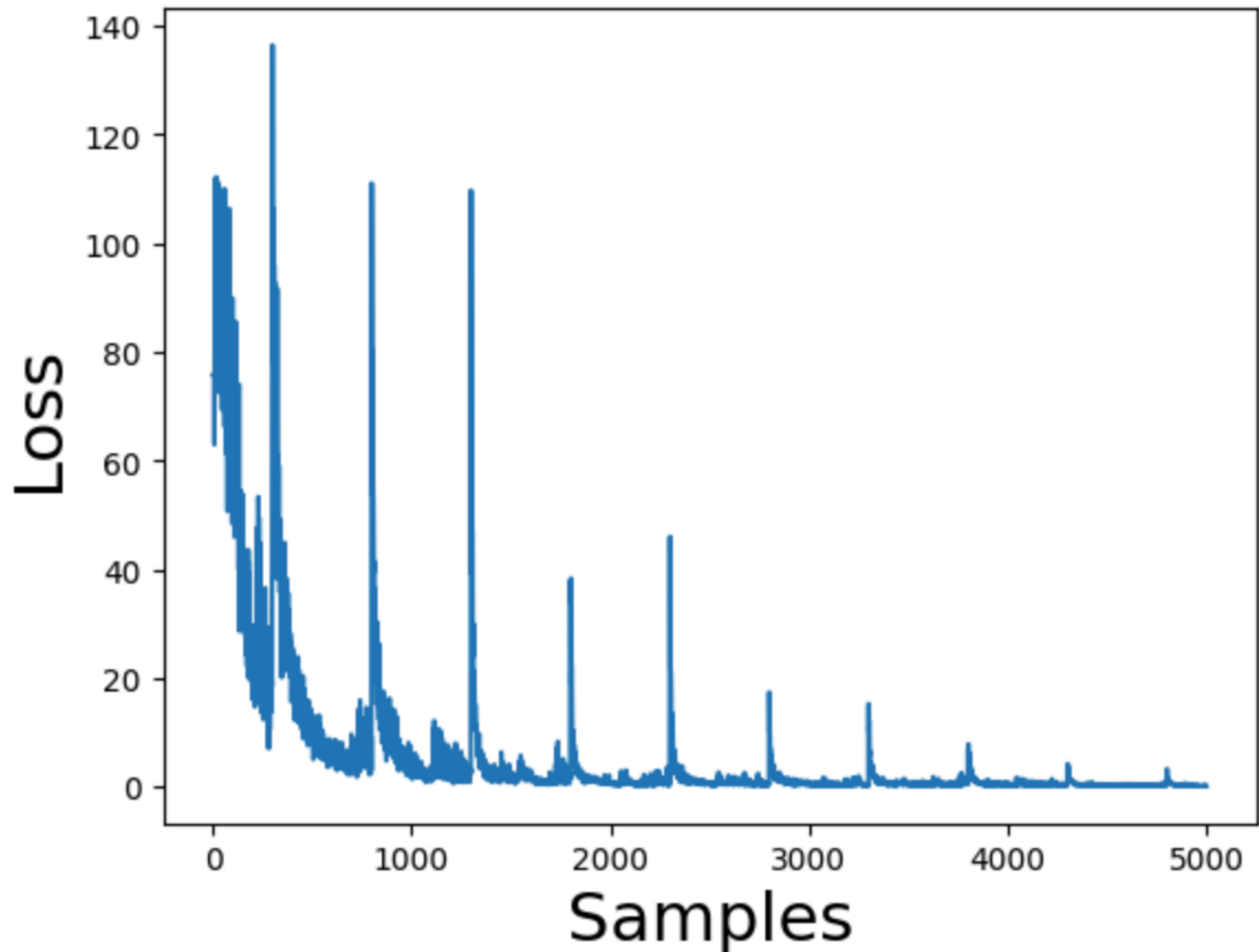
# Target Network versus Prediction (main) network



$$= \mathcal{L}$$



# Updating the Target Network

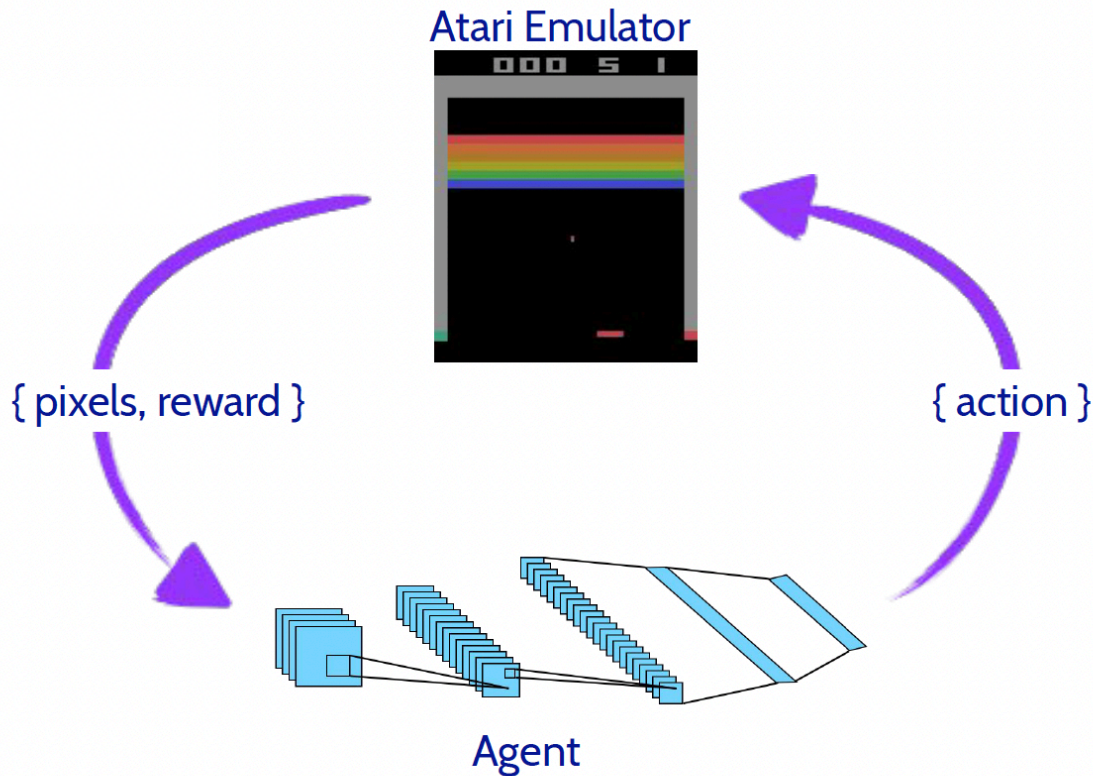


# Noisy-fying the state

- Discrete states (e.g. in grid worlds), especially one-hot can lead to “Dead Neurons”
- A trick to avoid this is to add a small amount of noise to the states
- This problem occurs specifically with ReLU (rectified linear unit) activation functions since these are non-differentiable at zero when the state vector is sparse (i.e. most its elements are zero) as is the case, for example, in our grid worlds in the lab.

```
state = state.reshape(1,k) + np.random.rand(1,k)/c
```

# End-to-end State Spaces



## LETTER

doi:10.1038/nature14236

### Human-level control through deep reinforcement learning

Vedant Mishra<sup>1</sup>, Koray Kavukcuoglu<sup>2</sup>, David Silver<sup>3</sup>, Andrej A. Rusu<sup>4</sup>, Joel Veness<sup>5</sup>, Marc G. Bellemare<sup>6</sup>, Alex Graves<sup>1</sup>, Martin Riedmiller<sup>7</sup>, Andreas K. Fylakiotou<sup>8</sup>, Georg Ostrovski<sup>9</sup>, Stig Petersen<sup>10</sup>, Charles Beattie<sup>11</sup>, Amir Sadik<sup>12</sup>, Iannis Antonoglou<sup>13</sup>, Helen King<sup>14</sup>, Dhruv Kumar<sup>15</sup>, Daan Wierstra<sup>16</sup>, Shane Legg<sup>17</sup> & Demis Hassabis<sup>18</sup>

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 situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems<sup>1</sup>, the former evidenced by a wealth of neural data revealing suitable parallels between the phasic signals emitted by dopamine neurons and temporal-difference reinforcement learning algorithms<sup>2</sup>. While reinforcement learning agents have achieved some success in a variety of domains<sup>3</sup>, their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Here we use recent advances in training deep neural networks<sup>4–11</sup> to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. We tested this agent on the challenging domain of classic Atari 2600 games<sup>12</sup>. We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters. This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

We set out to create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks—a central goal of general artificial intelligence<sup>13</sup> that has eluded previous efforts<sup>14,15</sup>. To achieve this, we developed a novel agent, a deep Q-network (DQN), which is able to combine reinforcement learning with a class of artificial neural networks<sup>16</sup> known as deep neural networks. Notably, recent advances in deep neural networks<sup>4–11</sup>, in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data. We use one particularly successful architecture, the deep convolutional network<sup>17</sup>, which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields—inspired by Hubel and Wiesel's seminal work on feedforward processing in early visual cortex<sup>18</sup>—thereby exploiting the local spatial correlations present in images, and building in robustness to natural transformations such as changes of viewpoint or scale.

We consider tasks in which the agent interacts with an environment through a sequence of observation, action and reward. The goal of the

agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E} [r_t + \gamma v_{t+1} + \dots | s_t = s, a_t = a, \pi],$$

which is the maximum sum of rewards  $r_t$  discounted by  $\gamma$  at each time-step  $t$ , achievable by a behaviour policy  $\pi = \text{Pol}(\theta)$ , after making an observation ( $s$ ) and taking an action ( $a$ ) (see Methods<sup>19</sup>).

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as  $Q$ ) function<sup>20</sup>. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to  $Q$  may significantly change the policy and therefore change the data distribution, and the correlation between the action-value ( $Q$ ) and the target values  $r + \gamma \max_{a'} Q(s', a')$ . We address these instabilities with a novel variant of  $Q$ -learning, which uses two key ideas. First, we use a biologically inspired mechanism termed experience replay<sup>21</sup> that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution (see below for details). Second, we used an iterative update that adjusts the action values ( $Q$ ) towards target values that are only periodically updated, thereby reducing correlations with the target.

While other stable methods exist for training neural networks in the reinforcement learning setting, such as neural fitted  $Q$ -iteration<sup>22</sup>, these methods involve the repeated training of networks de novo on hundreds of iterations. Consequently, these methods, unlike our algorithm, are too inefficient to be used successfully with large neural networks. We parameterize an approximate value function  $Q(s,a;\theta)$  using the deep convolutional neural network shown in Fig. 1, in which  $\theta$  are the parameters (that is, weights) of the  $Q$ -network at iteration  $i$ . To perform experience replay we store the agent's experiences  $s_t = (s_t, a_t, r_t, s_{t+1})$  at each time step  $t$  in a data set  $D_t = \{s_1, \dots, s_t\}$ . During learning, we apply  $Q$ -learning updates, on samples (or minibatches) of experience  $(s,a,r,s') = \mathcal{U}(D_t)$ , drawn uniformly at random from the pool of stored samples. The  $Q$ -learning update at iteration  $i$  uses the following loss function:

$$L_i(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{U}(D)} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s,a;\theta) \right)^2 \right]$$

in which  $\gamma$  is the discount factor determining the agent's horizon,  $\theta$  are the parameters of the  $Q$ -network at iteration  $i$  and  $\theta'$  are the network parameters used to compute the target at iteration  $i$ . The target network parameters  $\theta'$  are only updated with the  $Q$ -network parameters ( $\theta$ ) every  $C$  steps and are held fixed between individual updates (see Methods).

To evaluate our DQN agent, we took advantage of the Atari 2600 platform, which offers a diverse array of tasks ( $n = 49$ ) designed to be

Nature Journal  
https://www.nature.com/letters

### Human-level control through deep reinforcement learning

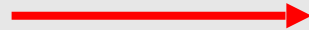
by V Mnih · 2015 · Cited by 25488 — The theory of reinforcement learning provides a normative account, deeply rooted in psychological and neuroscientific perspectives on animal ...

<sup>1</sup>Google DeepMind, 5 New Street Square, London EC4A 3DF, UK.  
<sup>2</sup>These authors contributed equally to this work.

# Breakout (DeepMind)



# Image Feature Vectors



RED	GREEN	BLUE
0	0	0
255	255	255
0	0	0
210	210	0
0	0	0
210	210	0
0	0	0

# State space for Atari Games

- We could use the image pixel values directly,
  - Vector  $(p_{1,1}^R, p_{1,1}^G, p_{1,1}^B, p_{1,2}^R, \dots, p_{h,w}^R, p_{h,w}^G, p_{h,w}^B)$
  - Atari frame,  $3 \times 210 \times 160 = 100,800$  inputs
- Two problems:
  - Huge input size;
  - Is a single frame enough for decisions?

# Movement



Which direction is the ball moving?

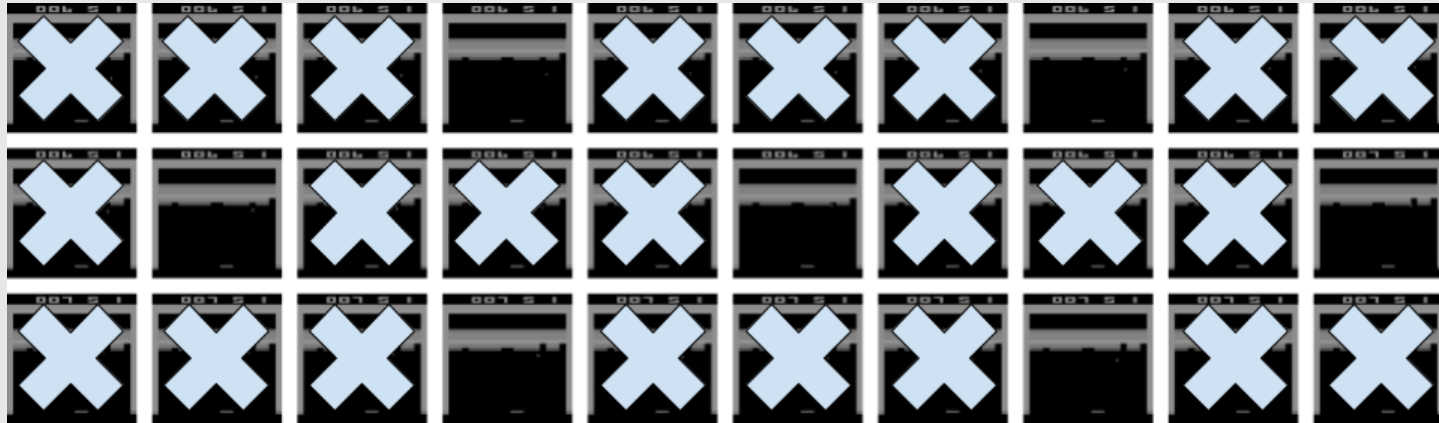
# Encoding Motion

---

- Accurate control needs motion information,
  - Use multiple frames
- Reduce size of the input image by
  - Using greyscale,
  - Rescaling frame size to 84 x 84



# Subsampling

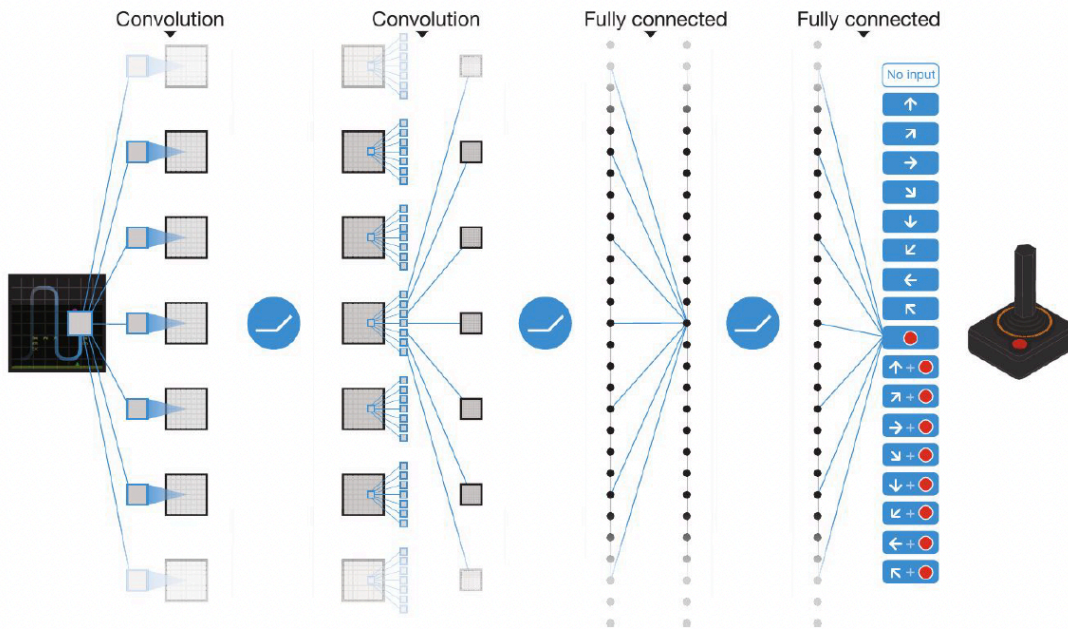


# 2-dimensional input (spatial context)

---

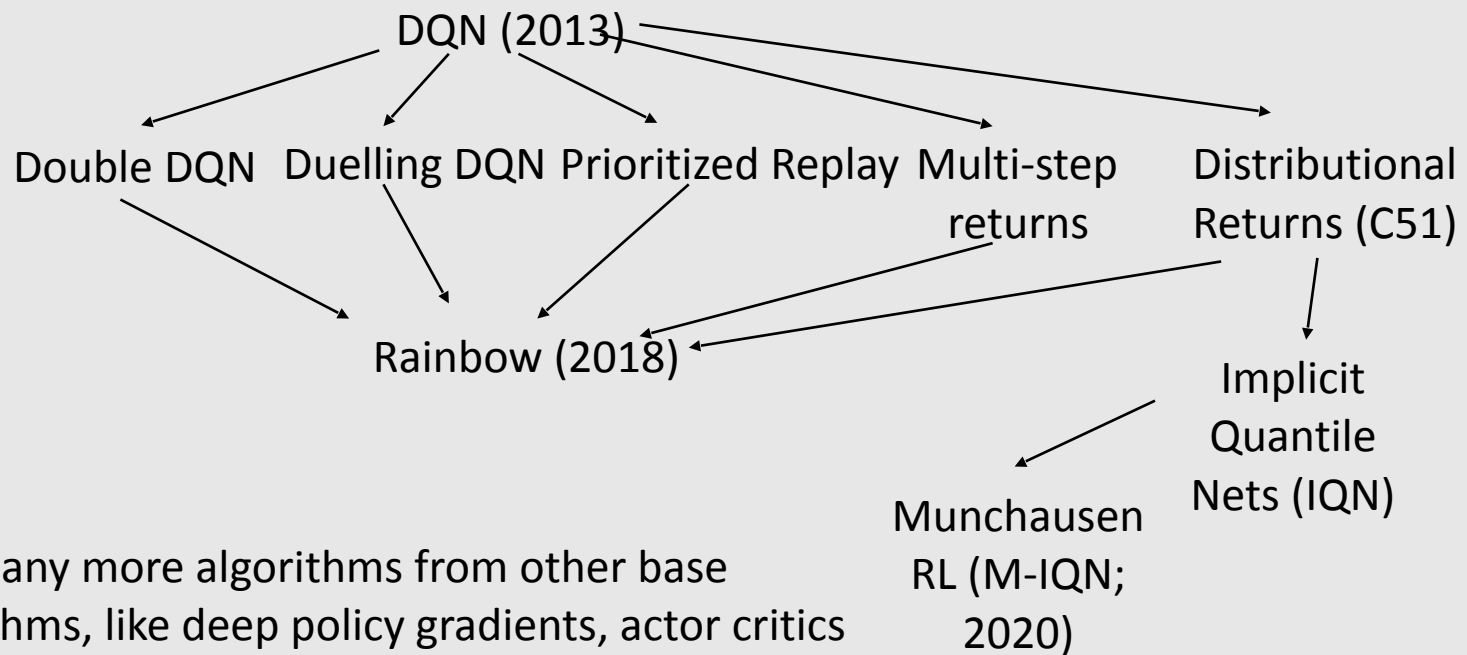
- By reducing input size we end up with  $4 \times 84 \times 84 = 28,224$  pixels
- Flattening these into an input layer is possible, **but loses spatial context**
- Solution: use Convolutional Neural Networks

# Q-Network Structure



- $s = 4 \times 84 \times 84$  grayscale input,
- 32 filters of 8x8 with stride 4,
- 64 filters of 4x4 with stride 2,
- 64 filters of 3x3 with stride 1,
- Fully connected, 512 units
- Fully connected,  $|A|=18$  output

# Advanced Deep-RL Methods



# Take home lessons

---

- Large state spaces, continuous state spaces and generalisation require function approximation
- We can use neural networks to approximate the state(action) value function
- Deep-Q = Q-Learning + (DNN substitutes Q-Table)
  - the loss is calculated as difference to the TD-target
- Experience replay is required to improve learning
- Dual networks are required to avoid catastrophic forgetting
- Adding Noise helps to avoid “Dead neurons”
- Deep-Q is only just the beginning ...