Reinforcement Learning: Practicals

Bellman equation

Temporal Difference Learning

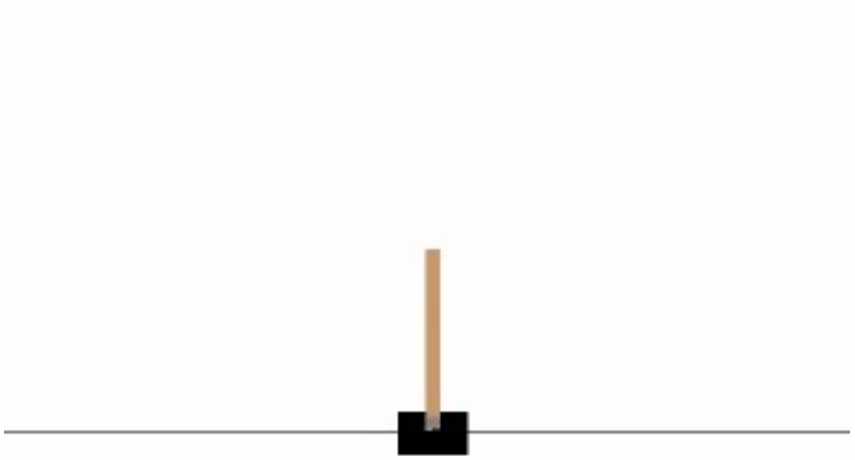
Feed Forward Neural Networks

Convolutional Neural Networks

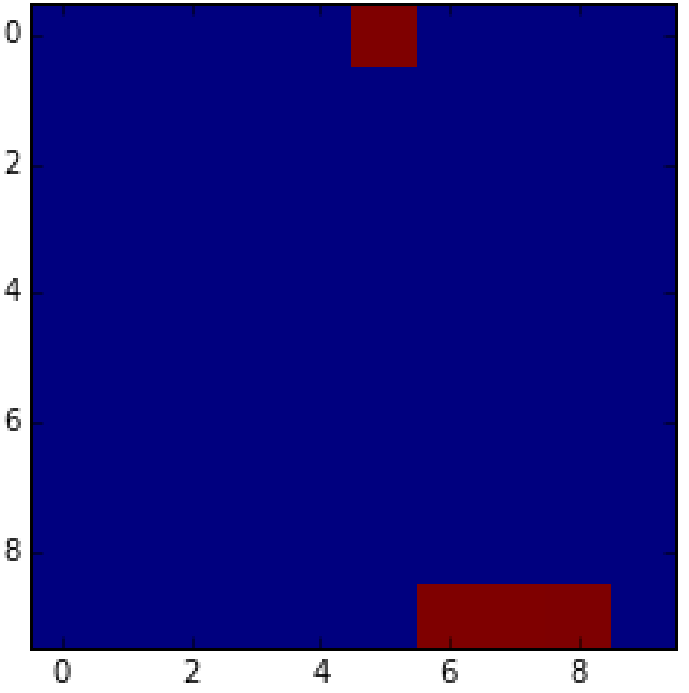
SGD

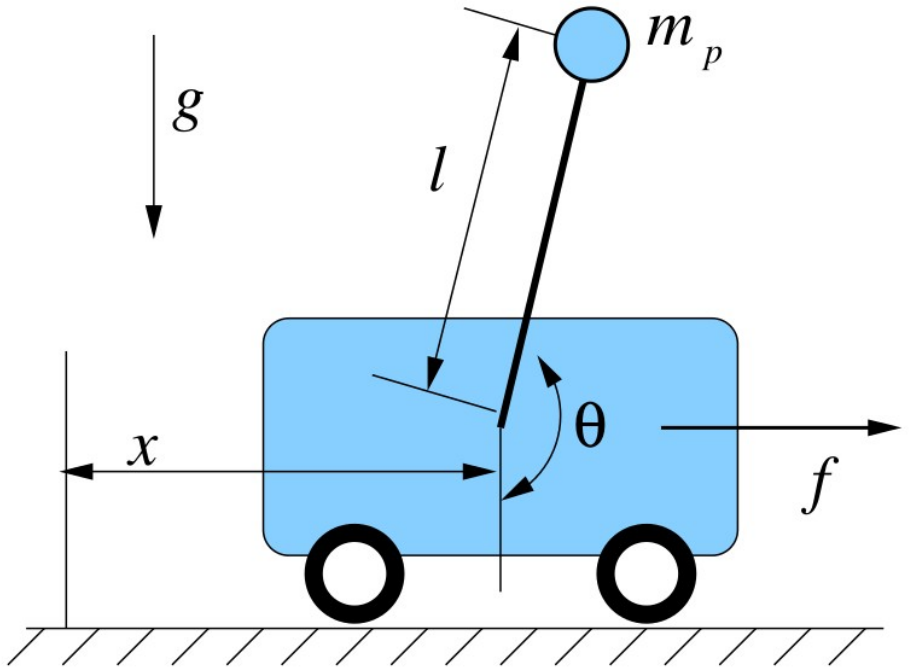
Backpropagation

Cartpole



Catch the pixel





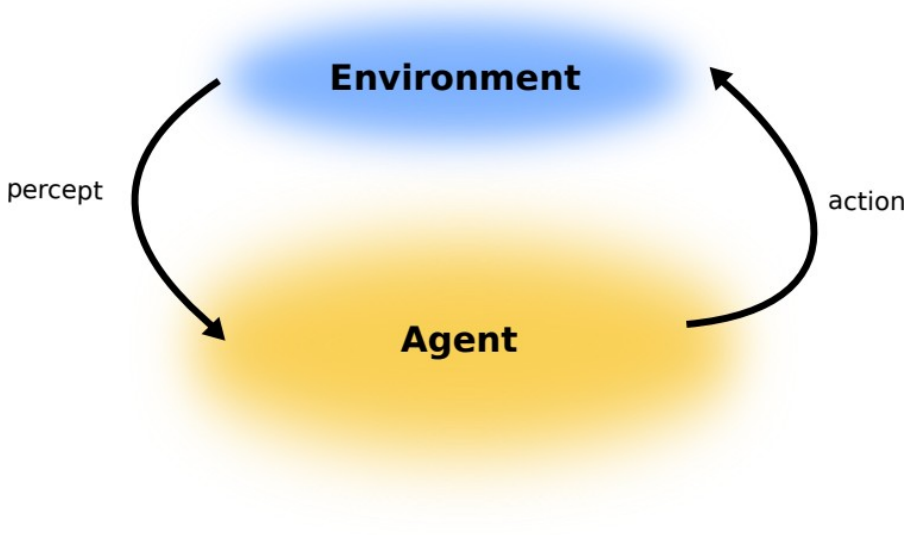
$$\begin{aligned}
\ddot{x} &= \frac{1}{M+m\sin^2\theta}\left[m\sin\theta(l\dot{\theta}^2 - g\cos\theta) + f\right] \\
\ddot{\theta} &= \frac{1}{l(M+m\sin^2\theta)}\left[-ml\dot{\theta}^2\sin\theta\cos\theta + (M+m)g\sin\theta - f\cos\theta\right]
\end{aligned}$$

$$s_t = [x_t, \dot{x}_t, \theta_t, \dot{\theta}_t] \rightarrow f_t = f_t(s_t)$$

$$x_t \in [x_{min}, x_{max}]$$
$$\dot{x}_t \in \mathbb{R}$$
$$\theta_t \in [0, 2\pi]$$
$$\dot{\theta}_t \in \mathbb{R}$$

$$\mathrm{s}_t = [x_t, \dot{x}_t, \theta_t, \dot{\theta}_t]$$

$$\mathrm{r} = +1 \quad \text{if balanced}$$



a_0 push left

a_1 push right

Terminal states

$$\theta_t > \pm 12°$$

x_t outside $[x_{min}, x_{max}]$

Episode length > T_{max}

Goal

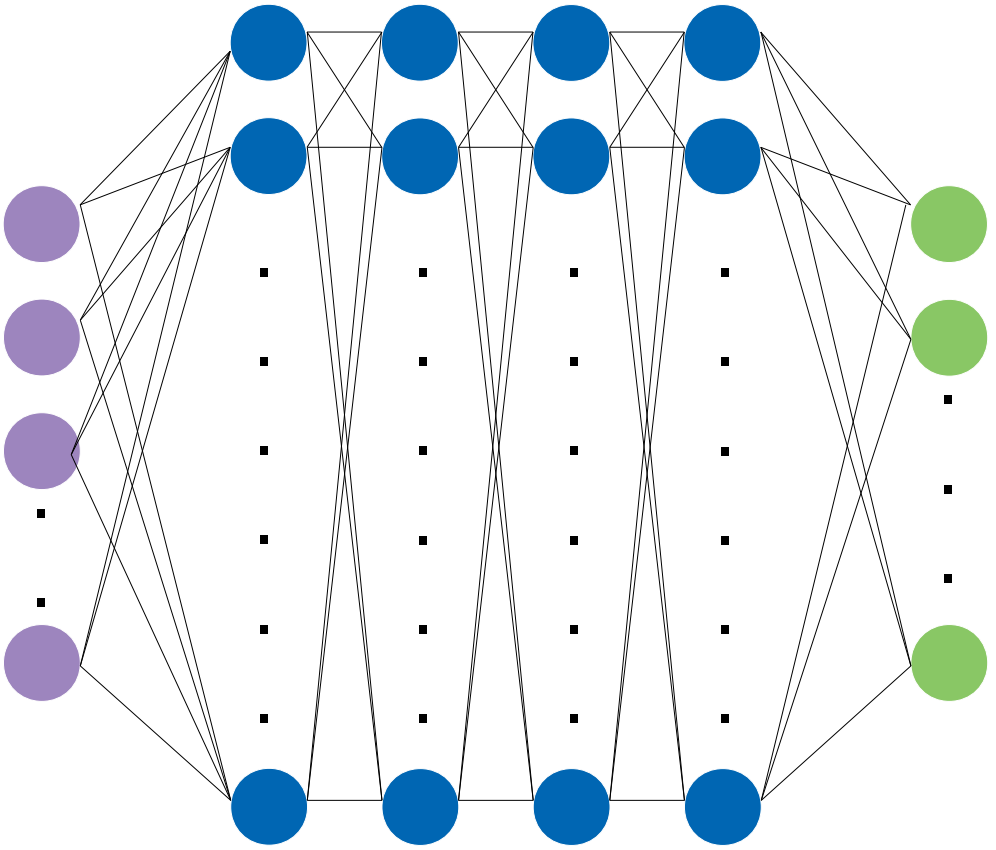
MAXIMIZE $\sum_t \gamma^t r_t$

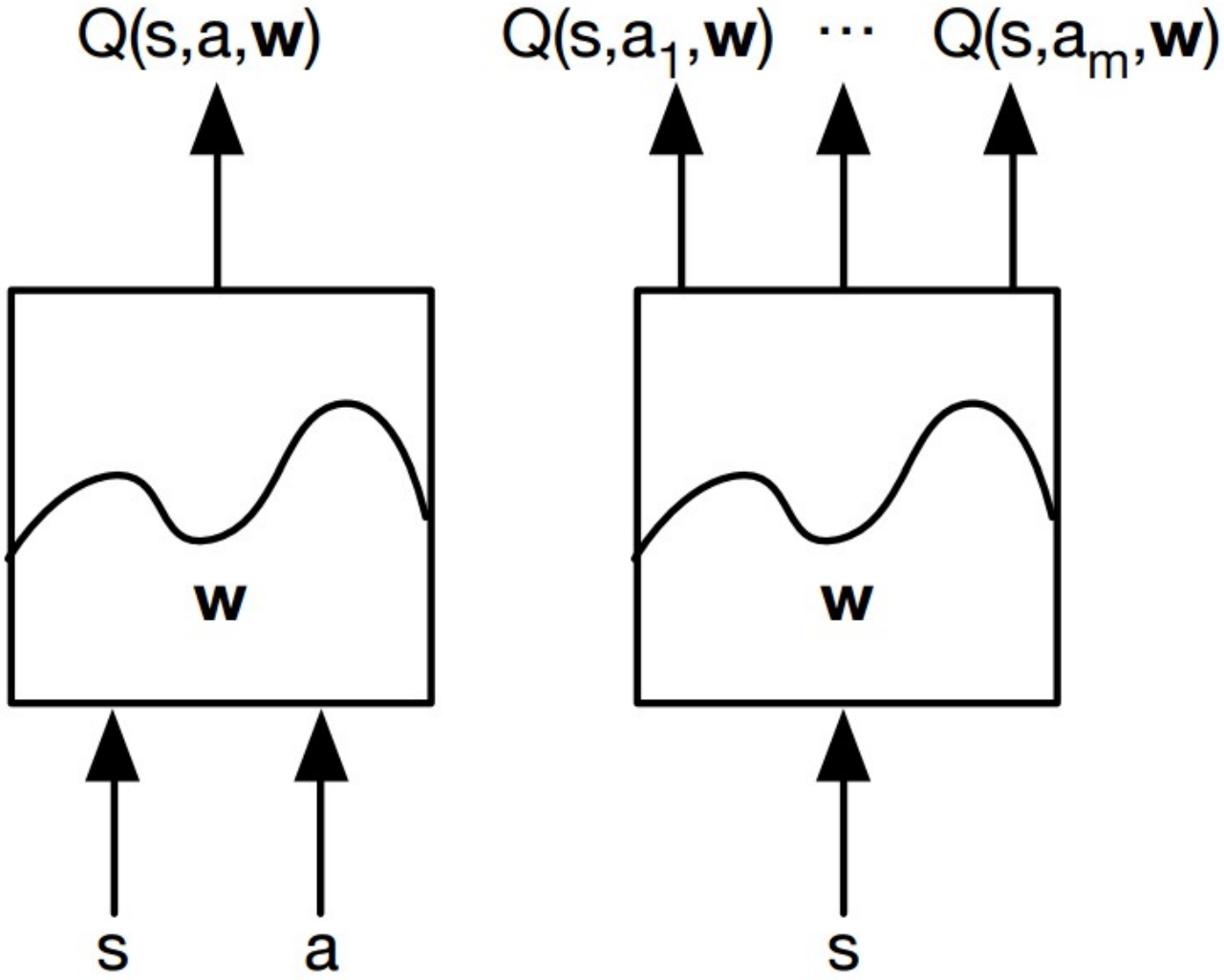
Deep neural network approximation of Q

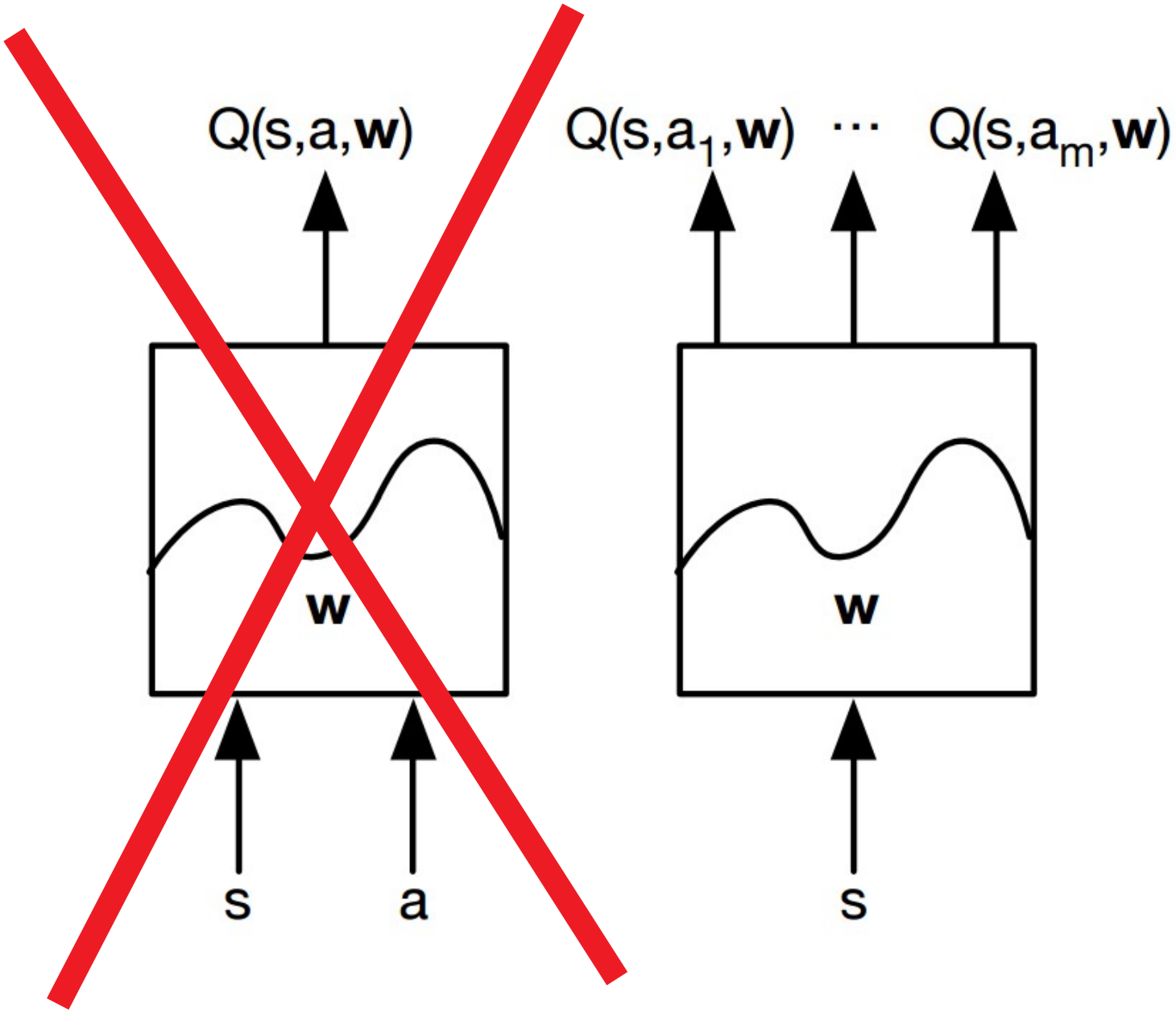
$$Q_\pi(s,a) \equiv \mathbb{E}_\pi \left[R_t | s_t = s,\ a_t = a \right]$$

$$Q(s,a) \approx Q(s,a,\mathbf{w})$$

{ state } { Hidden layers } { action }

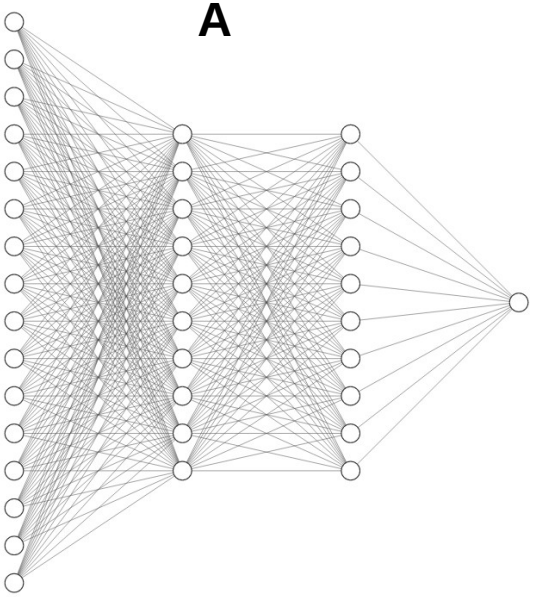



Q(s,a,w)
w
s
a
Q(s,a1,w)
···
Q(s,am,w)
w
s

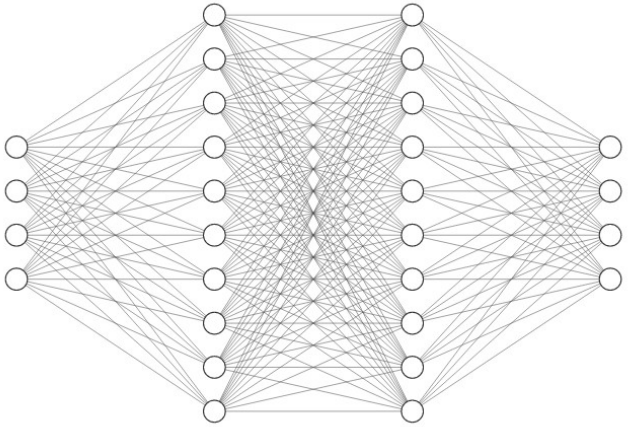

Q(s,a,w)
Q(s,a1,w) ··· Q(s,am,w)
w
w
s
a
s

Q What type of neural network do we need ?

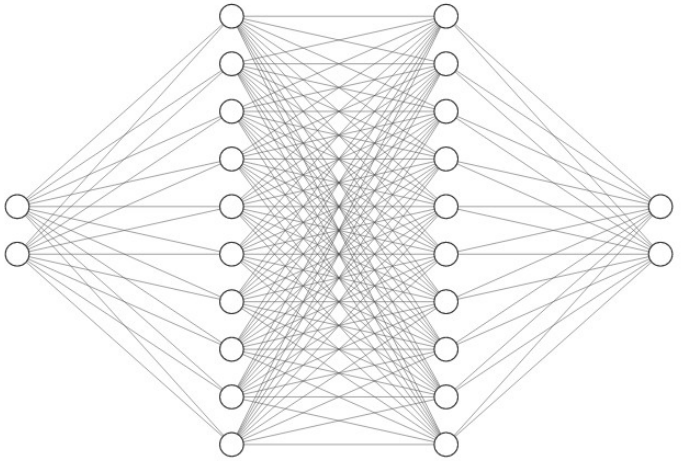
A



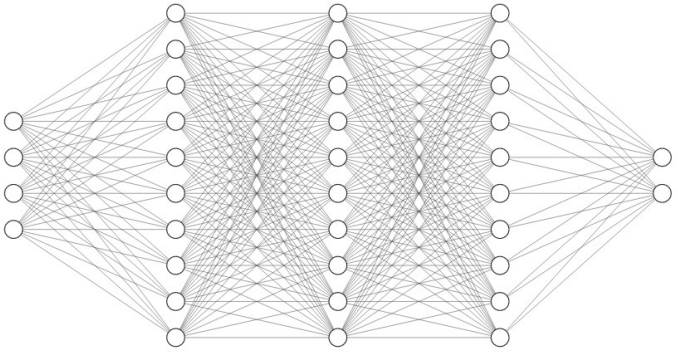
B



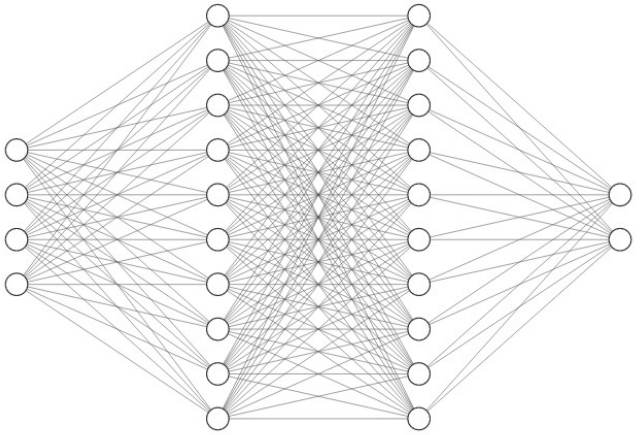
C



D



E



Training the Q-network

At the same time we want to

- Learn the value function
- Learn to act optimally

Bellman equation ...

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q^*(s',a')]$$

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

... still the Bellman equation ...

$$0 = \mathrm{E}_r\left[\mathrm{E}_{s'}\left[r + \gamma\ \max_{a'} Q(s',a',\mathbf{w}^*)\right] - Q(s,a,\mathbf{w}^*)\right]$$

TD-error

$$\delta = \mathrm{E}_{s'}\left[r + \gamma\ \max_{a'} Q(s',a',\mathbf{w}_{i-1})\right] - Q(s,a,\mathbf{w}_i)$$

Training the Q-DNN

Cost function ...

$$L(\mathbf{w}) = \frac{1}{2}\mathrm{E}_{s,a,r}\left[\left(\mathrm{E}_{s'}\left[y\right] - Q\right)^2\right]$$

$$= \frac{1}{2}\mathrm{E}_{s,a,r,s'}\left[(y-Q)^2\right] - \frac{1}{2}\mathrm{E}_{s,a,r}\left[\mathrm{V}_{s'}[y]\right]$$

experience

Optimized by SGD

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla_{\mathbf{w}}L$$

Computed by backpropagation

arXiv.org > cs > arXiv:1609.04747

Computer Science > Machine Learning

An overview of gradient descent optimization algorithms

Sebastian Ruder

(Submitted on 15 Sep 2016 (v1), last revised 15 Jun 2017 (this version, v2))

Experience replay: speeding up the learning

- correlated (in time) experiences slow down the learning

- the experience is immediately discarded

A possible solution*

keep a memory of the experiences and sample from it

$$\mathrm{M} = \{e_1, e_2, \cdots, e_n\}$$

Neuron
Review

Neuroscience-Inspired Artificial Intelligence

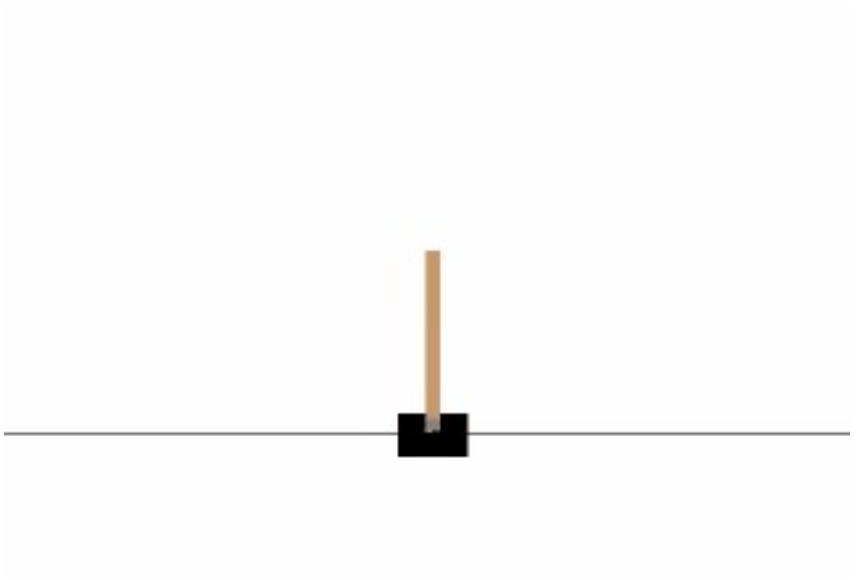
Demis Hassabis,[1,2,*] Dharshan Kumaran,[1,3] Christopher Summerfield,[1,4] and Matthew Botvinick[1,2]
[1]DeepMind, 5 New Street Square, London, UK
[2]Gatsby Computational Neuroscience Unit, 25 Howland Street, London, UK
[3]Institute of Cognitive Neuroscience, University College London, 17 Queen Square, London, UK
[4]Department of Experimental Psychology, University of Oxford, Oxford, UK
*Correspondence: dhcontact@google.com
http://dx.doi.org/10.1016/j.neuron.2017.06.011

*A3C Asynchronous Adavantage Actor-Critic methods

Enough talk! Let's code!

1. Open Lecture5 directory > Mouse Right click > Open in Terminal

2. > jupyter notebook cartpole.ipynb [enter]



Experiment with the code ...

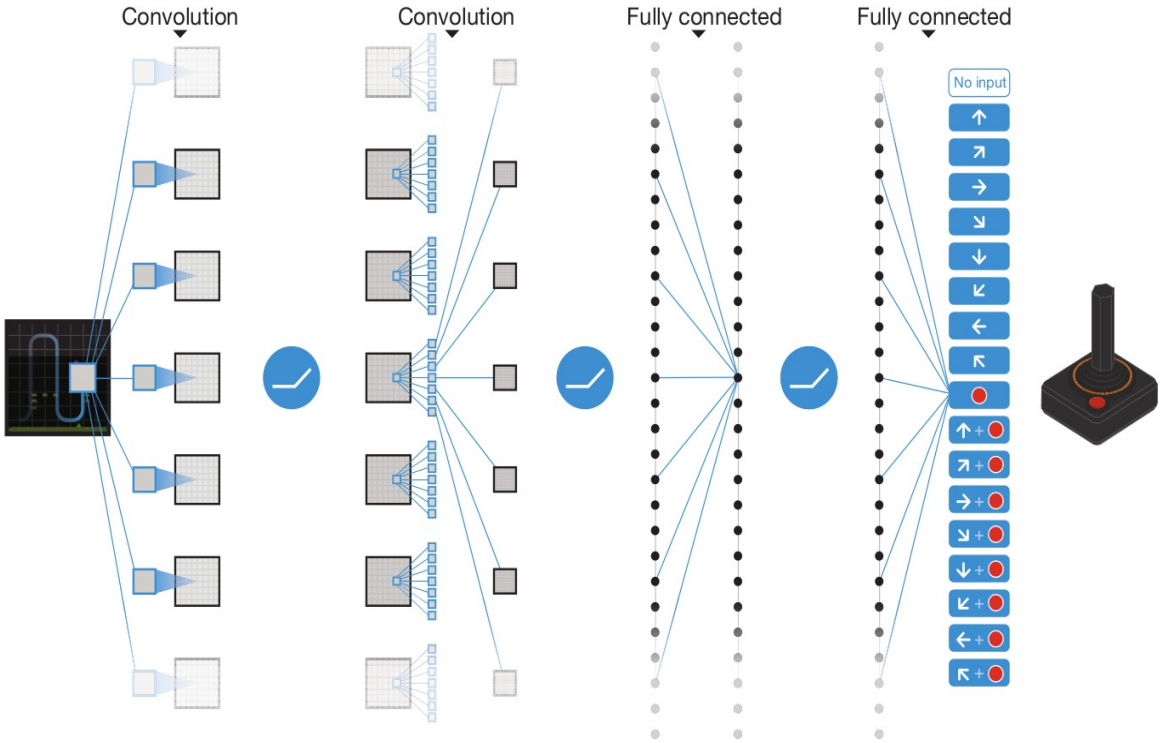
```
agent = DQNAgent(input_dim=4, output_dim=2,
                 batch_size=,
                 gamma = 1.,
                 learning_rate=)

epochs = 200

explore_p =
```

memory size

$\alpha : \mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L$

ϵ - greedy policy

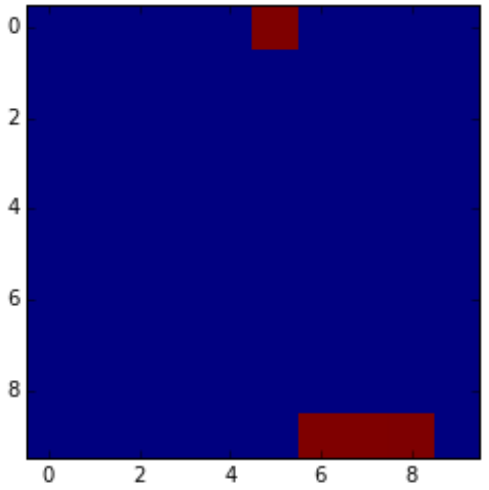
Learning to play looking the screen



Let's code again!

> Open terminal in `Lecture5/catch_the_pixel` **[enter]**

> `jupyter notebook catch_the_pixel.ipynb` **[enter]**



Reward at the end

+1 catched
-1 otherwise