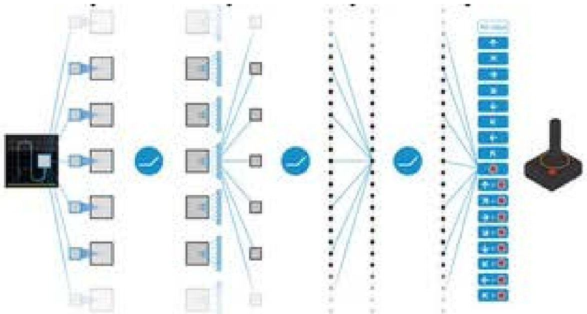
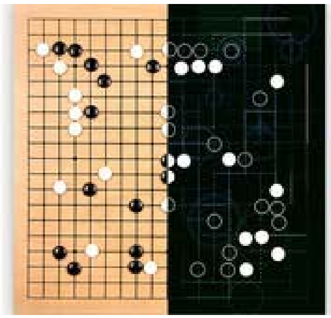
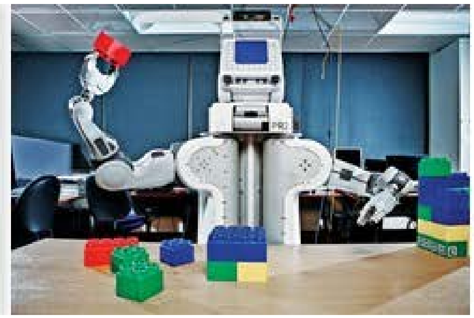
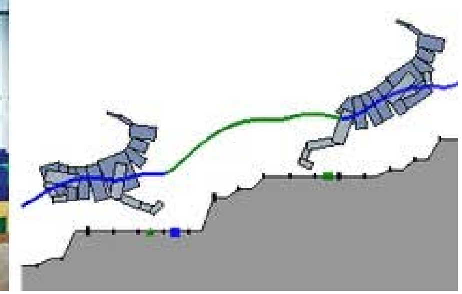
Deep Reinforcement Learning

Deep Learning, Spring 2017



Deep Reinforcement Learning

- Plays Atari video games
- Beats human champions at Poker and Go
- Robot learns to pick up, stack blocks
- Simulated quadruped learns to run

What is reinforcement learning?

Deep Reinforcement Learning

Types of learning

Supervised

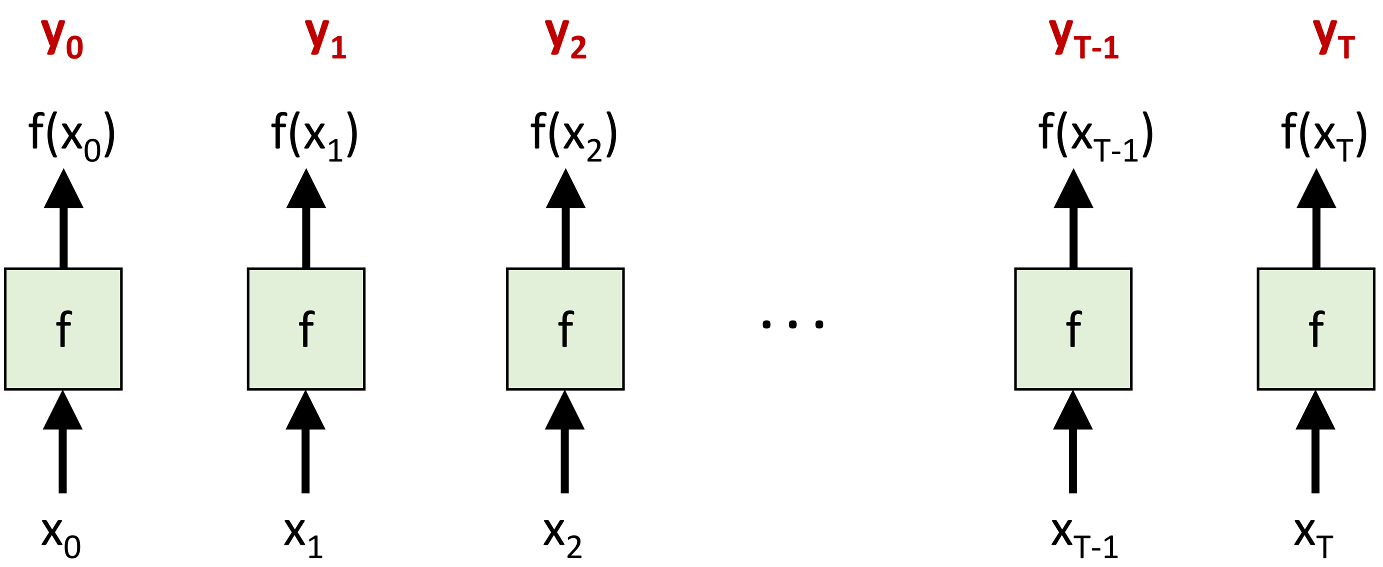
Unsupervised

Reinforcement

Supervised learning

- model receives input *x*
- also gets correct output y
- predictions do not change future inputs

Supervised learning: (in arbitrary order of examples)

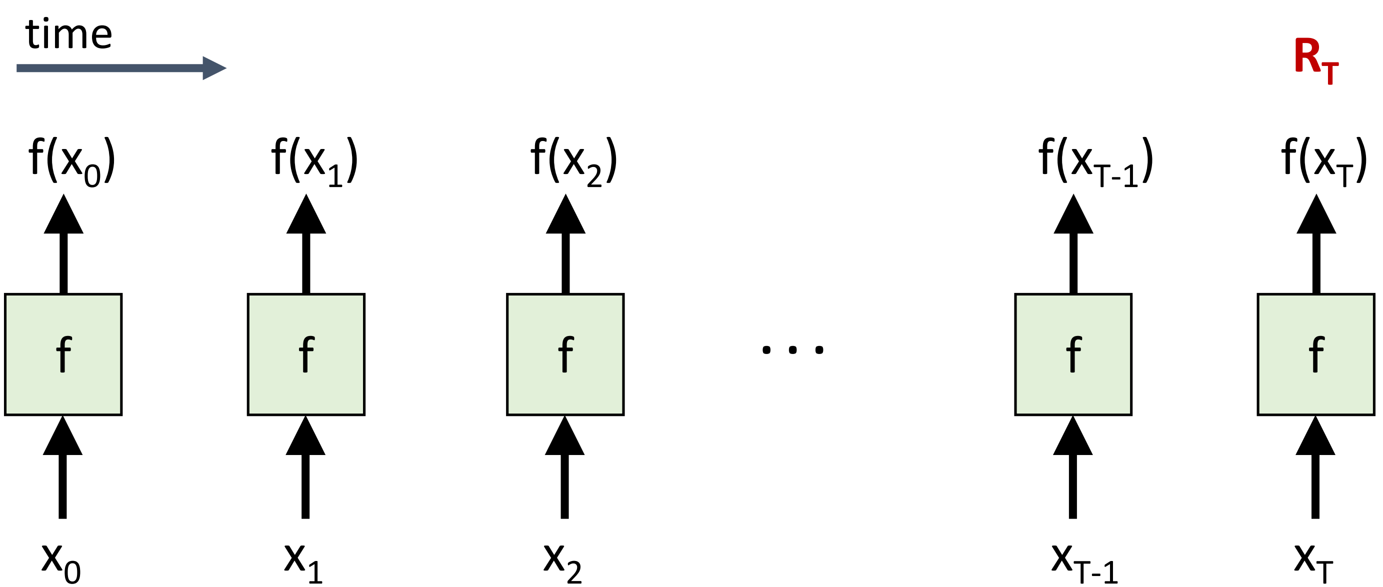


This is not how humans learn!

Reinforcement learning

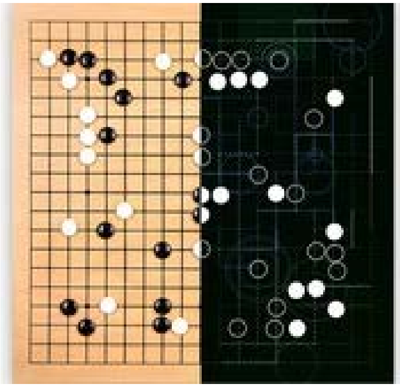
- agent receives input *x*, chooses action
- gets *R* (reward) after *T* time steps
- actions affect the next input (state)

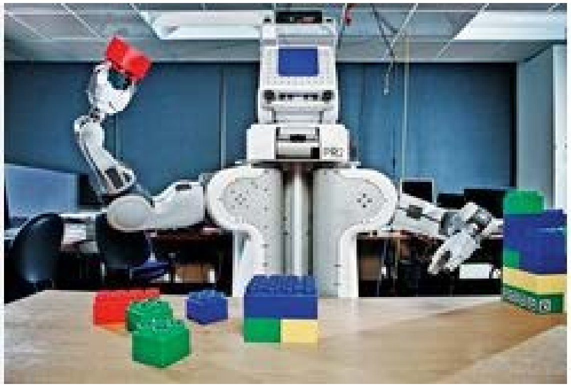
Reinforcement learning:

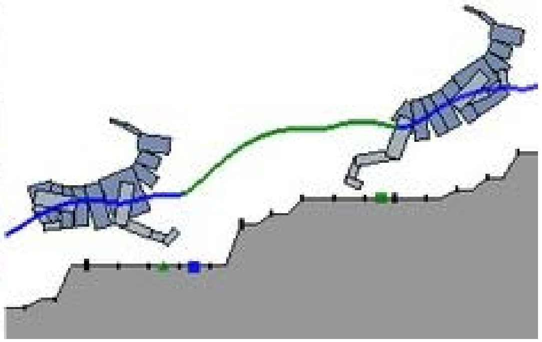


Input is the "world's" state

- Current game board layout
- Picture of table with blocks
- Quadriped position and orientation

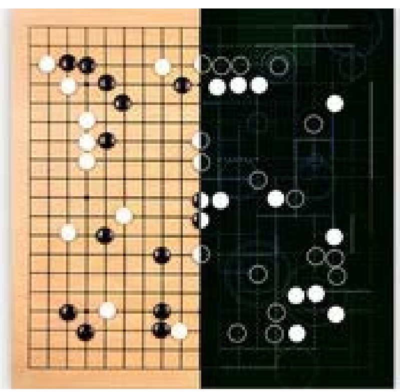
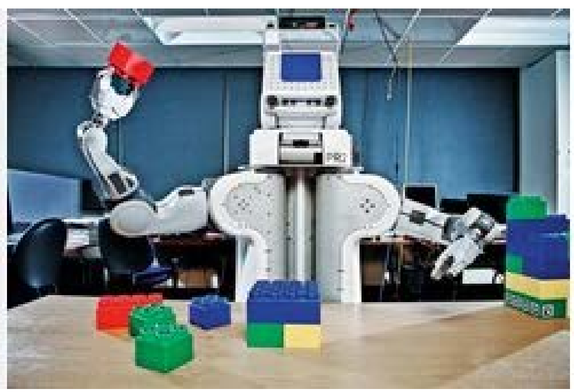
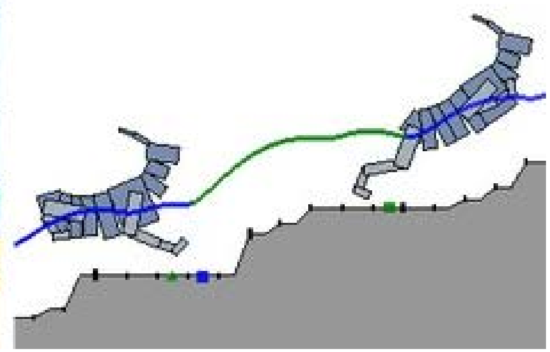






Output is an action

- Which game piece to move where (discrete)
- Orientation and position of robot arm (continuous)
- Joint angles of quadruped legs (continuous)

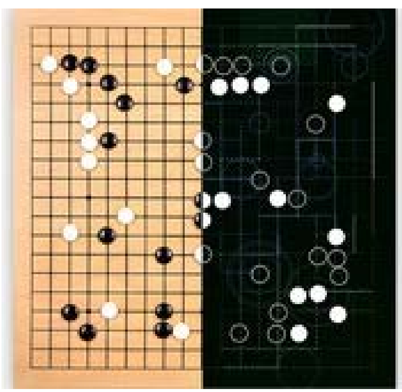
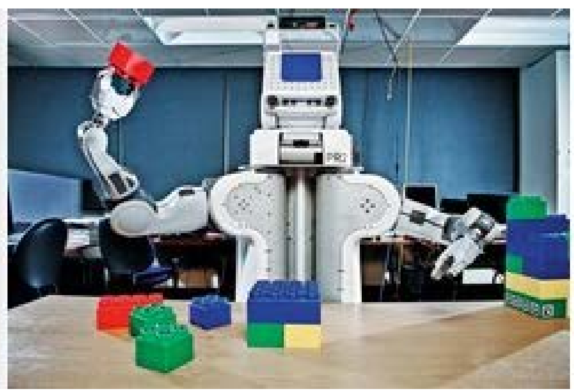
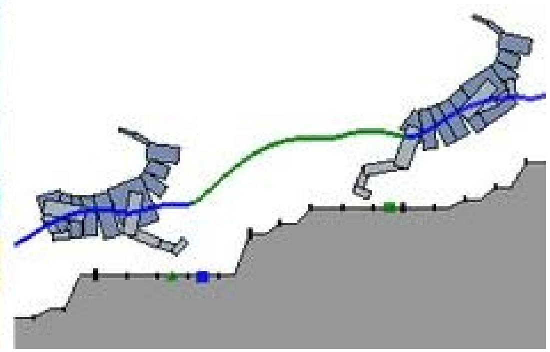
Actions affect state!

action→reward

some rewards are negative

Reward examples

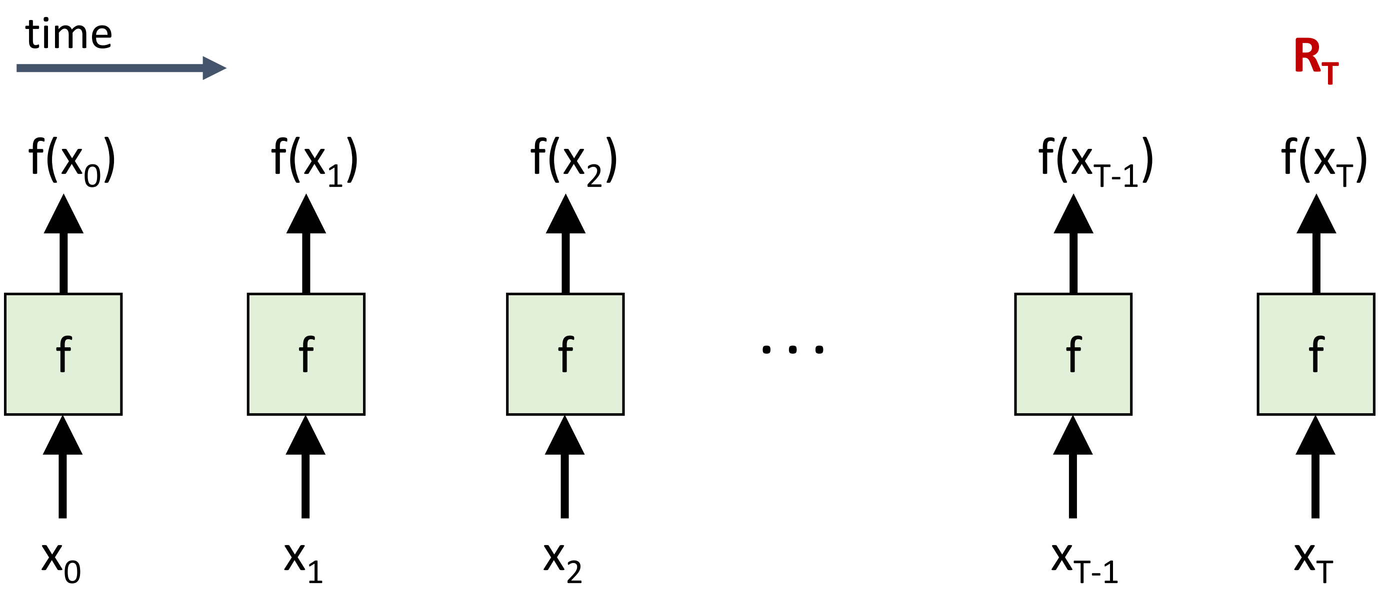
- Wining the game (positive)
- Successfully picking up block (positive)
- Falling (negative)

Goal of reinforcement learning

- Learn to predict actions that maximize future rewards
- Need a new mathematical framework

Reinforcement learning:

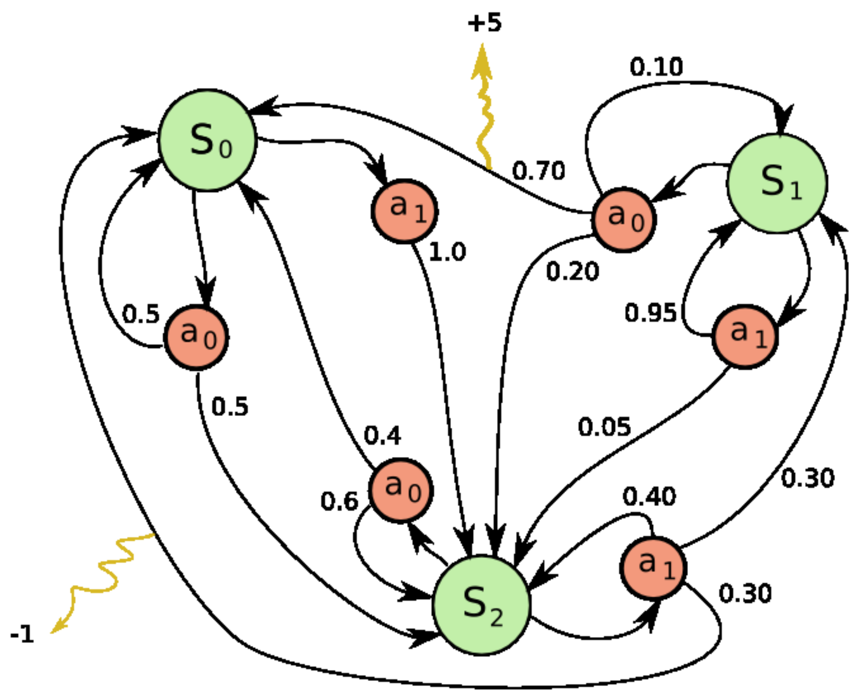


Markov Decision Process

Deep Reinforcement Learning

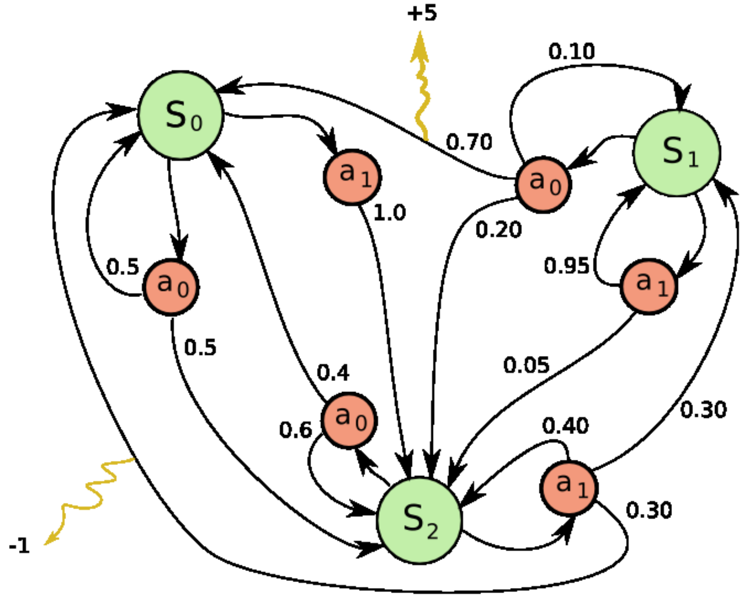
Markov Decision Process (MPD)

Definition: a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.



https://en.wikipedia.org/wiki/Markov_decision_process

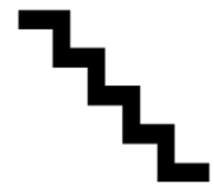
MDP notation



- S – set of States
- A – set of Actions
- $R: S \rightarrow \mathbb{R}$ (Reward)
- Psa – transition probabilities ($p(s, a, s') \in \mathbb{R}$)
- γ – discount factor

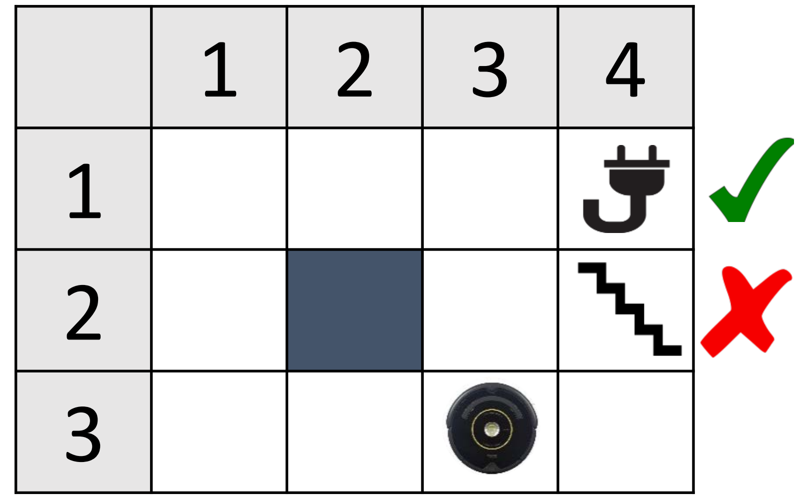
MDP = (S, A, R, Psa, γ)

MDP (Simple example)

	1	2	3	4
1				
2				
3				

MDP (Simple example)

- States S = locations
- Actions A = {↑,→,←,↓}



MDP (Simple example)

- States S = locations
- Actions A = {↑,→,←,↓}
- Reward $R: S \to \mathbb{R}$

	1	2	3	4
1	0	0	0	+1
2	0		0	-1
3	0	0		0

MDP (Simple example)

- States S = locations
- Actions A = {↑,→,←,↓}
- Reward $R: S \rightarrow \mathbb{R}$

	1	2	3	4
1	-.02	-.02	-.02	+1
2	-.02		-.02	-1
3	-.02	-.02		-.02

MDP (Simple example)

- States S = locations
- Actions A = {↑,→,←, ↓}
- Reward $R: S \rightarrow \mathbb{R}$
- Transition Psa

$$P_{(3,3),\uparrow}\big((2,3)\big) = 0.8$$
$$P_{(3,3),\uparrow}\big((3,4)\big) = 0.1$$
$$P_{(3,3),\uparrow}\big((3,2)\big) = 0.1$$
$$P_{(3,3),\uparrow}\big((1,3)\big) = 0$$
$$\vdots$$

	1	2	3	4
1	-.02	-.02	-.02	+1
2	-.02		-.02	-1
3	-.02	-.02		-.02

MDP - Dynamics

	1	2	3	4
1	-.02	-.02	-.02	+1
2	-.02		-.02	-1
3				-.02

-.02 -.02 -.02

- Start from state S_0
- Choose action A_0
- Transit to $S_1 \sim P_{s_0 a_0}$
- Continue…

- Total payoff:

-.02 -.02 -.02

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

How do we choose good actions?

Choosing actions in MDP

States S = locations

Actions A = {↑ ,→ ,← ,↓ }

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

Transition Psa

	1	2	3	4
1	-.02	-.02	-.02	+1
2	-.02		-.02	-1
3	-.02	-.02		-.02

- Goal - Choose actions as to maximize expected total payoff:

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

- In our example:

  R – get to charge station, avoid stairs

  γ – discourage long paths

MDP – Policy π

States S = locations

Actions A = {↑ ,→ ,← ,↓ }

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

Transition Psa

	1	2	3	4
1	↓	→	→	+1
2	←		↑	-1
3	→	←	↑	↓

- Goal - Choose actions as to maximize expected total payoff:

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

- Policy $\pi: S \rightarrow A$

MDP – Policy value function

States S = locations

Actions A = {↑ ,→ ,← ,↓ }

Reward $R: S \to \mathbb{R}$

Transition Psa

Policy $\pi: S \to A$

	1	2	3	4
1	↓	→	→	+1
2	←		↑	-1
3	→	←	↑	↓

- Value function - $V: S \to \mathbb{R}$

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right]$$

Expected sum of discounted rewards

MDP – Policy value function

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right]$$

$$\Rightarrow V^{\pi}(s) = E[R(s_0)] + E[\gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

this is recursion!

Bellman's equation:

$$V^{\pi}(s) = R(s) + \gamma E_{s' \sim P_{s,\pi(s)}}[V(s')]$$

$$\Updownarrow$$

expectation over values of next state

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

MDP – Policy value function

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

$$\begin{array}{c} \textit{solve} \\ |S| \\ \text{eqs.} \end{array} \begin{cases} V_1 = R(1) + \gamma P_{1,\downarrow}(2)\, V_2 + \dots \\ V_2 = \dots \\ \dots \\ V_{10} = R(10) + \gamma\big(P_{10,\uparrow}(6) V_6 + P_{10,\uparrow}(9) V_9 + \\ \qquad\qquad P_{10,\uparrow}(11) V_{11}\big) \\ \dots \end{cases}$$

	1	2	3	4
1	1	2	3	4
2	5		6	7
3	8	9	10	11

	1	2	3	4
1	↓	→	→	+1
2	←		↑	-1
3	→	←	↑	↓

Policy π

MDP – ways to solve

Optimal value function — need to define π

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman: $V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$

Optimal policy — need to define P_{sa}

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Optimal state-action value function Q: — easier!

Define $Q: S \times A \to \mathbb{R}$

Bellman: $Q(s_t, a) = R(s_t) + \gamma \max_{a'} Q(s', a')$

Q-learning

The agent interacts with the environment, updates Q recursively

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

reward

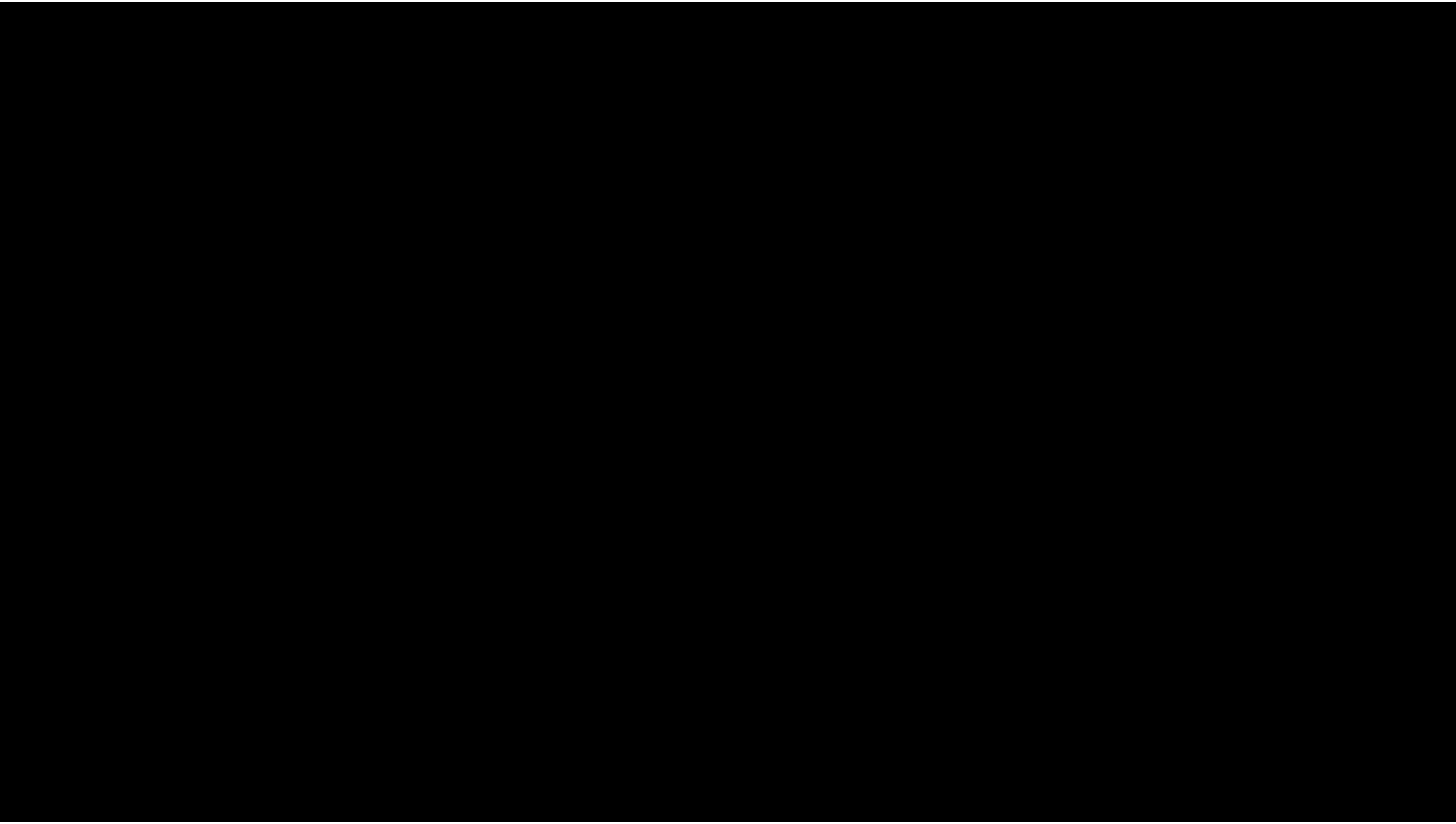
current value

discount

largest increase over all possible actions in new state

learning rate

Q-learning example



https://www.youtube.com/watch?v=R88CiN7dTZc

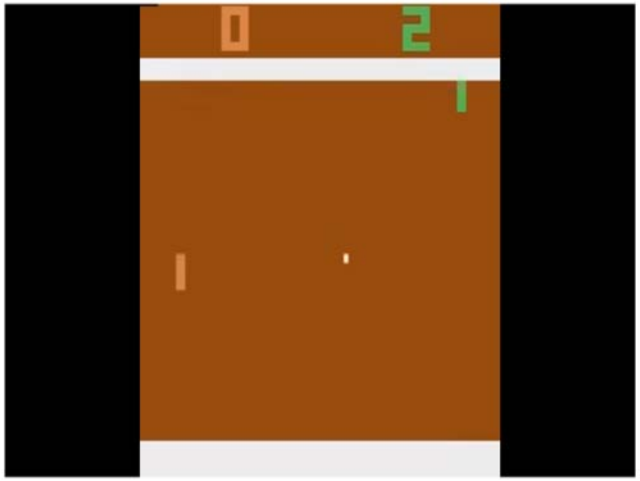
Continuous state

Deep Reinforcement Learning

Continuous state - Pong

https://www.youtube.com/watch?v=YOW8m2YGtRg

MDP for Pong



In this case, what are these?

- S – set of States
- A – set of Actions
- $R: S \rightarrow \mathbb{R}$ (Reward)
- Psa – transition probabilities ($p(s, a, s') \in \mathbb{R}$)

Can we learn Q-value?

- Can discretize state space, but it may be too large
- Can simplify state by adding domain knowledge (e.g. paddle, ball), but it may not be available
- Instead, use a neural net to learn good features!

Deep RL

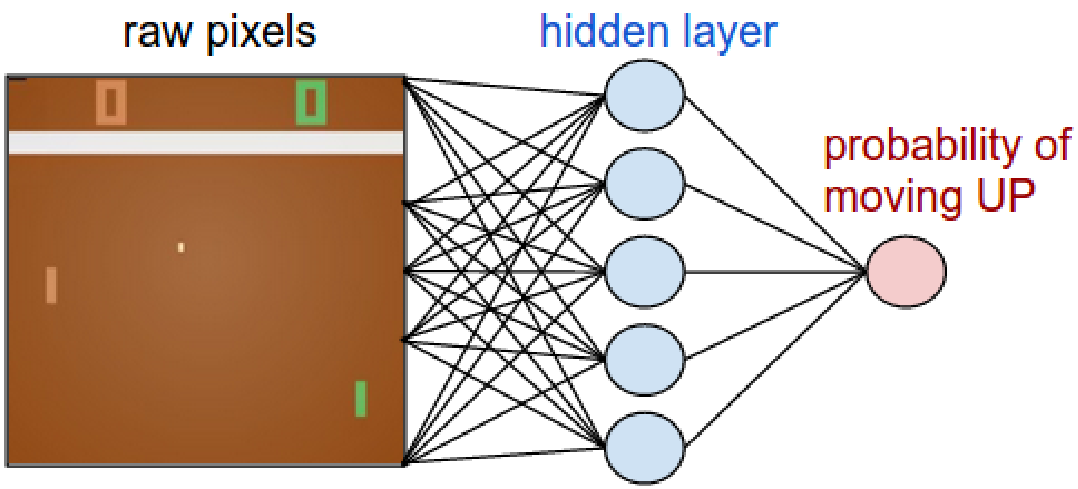
Deep Reinforcement Learning

Deep RL

- V, Q or π can be approximated with deep network
- Deep Q-Learning
  - Input: state, action
  - Output: Q-value
- Alternative: learn a Policy Network
  - Input: state
  - Output: distribution over actions

Cover today

Policy network for pong

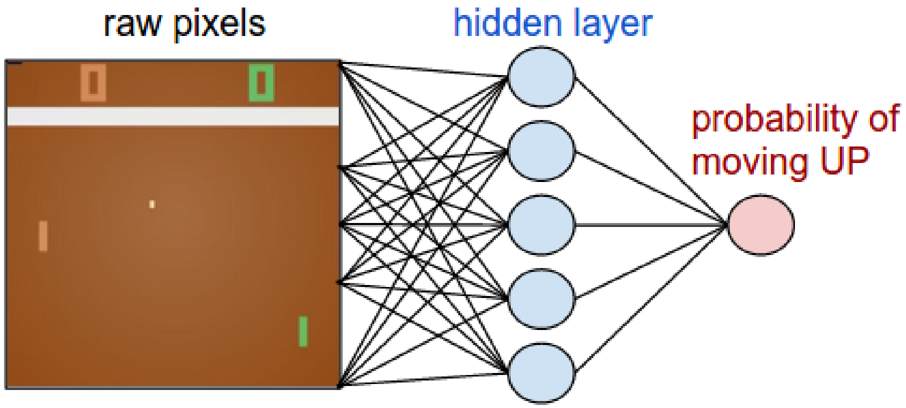


- define a *policy network* that implements the player
- takes the state of the game and decides what to do (move UP or DOWN)
- 2-layer neural network that takes the raw image pixels* (100,800 = 210x160x3), outputs the probability of going UP

*feed at least 2 frames to the policy network so that it can detect motion.

http://karpathy.github.io/2016/05/31/rl/

Policy gradient



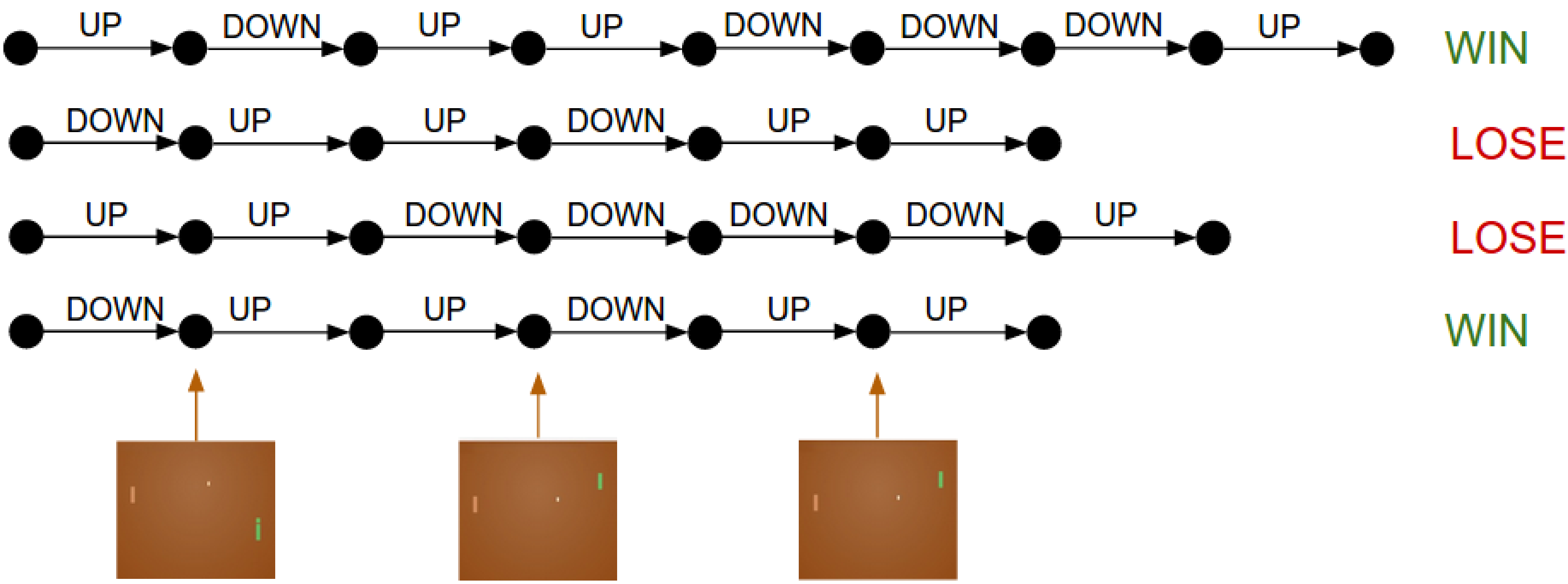
- Suppose network predicts
  - p(UP) = 30%
  - p(DOWN)=70%
- Can sample an action from this distribution and execute it
- Can immediately use gradient of 1.0 for DOWN and backprop to find the gradient vector that would encourage the network to predict DOWN

Problem: do not yet know if going DOWN is good!

Solution: simply wait until the end of the game, then take the reward we get (either +1 if we won or -1 if we lost), and enter that as the gradient for taken actions

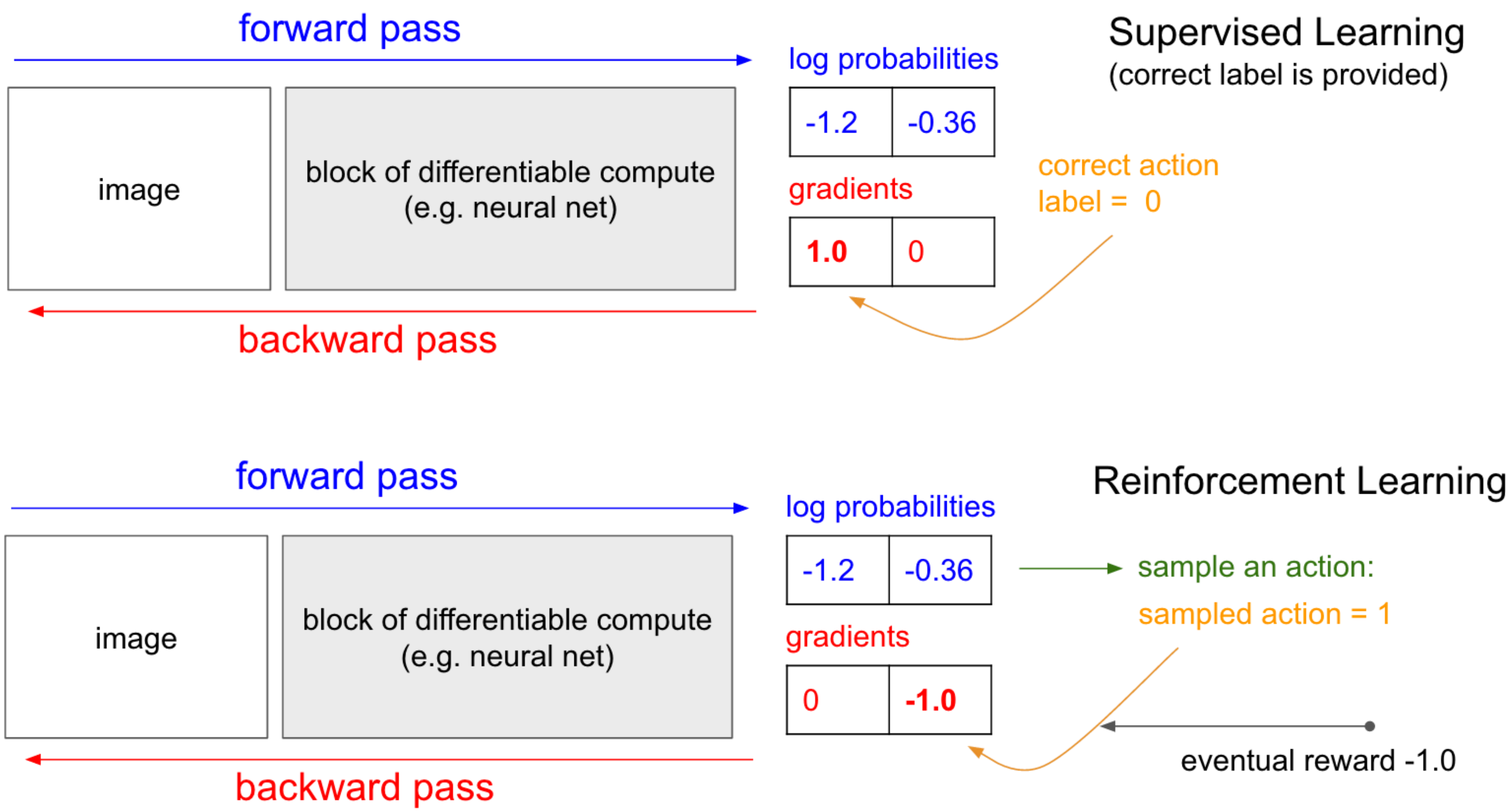
http://karpathy.github.io/2016/05/31/rl/

Policy gradient



http://karpathy.github.io/2016/05/31/rl/

Policy gradient vs supervised



http://karpathy.github.io/2016/05/31/rl/

Problems with this?

- what if we made a good action in frame 50 (bouncing the ball back correctly), but then missed the ball in frame 150?
- If every single action is now labeled as bad (because we lost), wouldn't that discourage the correct bounce on frame 50?
- Yes, but after thousands/millions of games, network will learn a good policy

http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

Want to maximize

$$E_{x \sim p(x|\theta)}[f(x)]$$

$f(x)$ is the reward function

$\mathrm{p}(x)$ is the policy network with parameters θ

(i.e. change the network's parameters so that action samples get higher rewards)

http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

$$\nabla_\theta E_x[f(x)] = \nabla_\theta \sum_x p(x) f(x)$$

http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) \\ &= \sum_x \nabla_\theta p(x) f(x)\end{aligned}$$

http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) \\
&= \sum_x \nabla_\theta p(x) f(x) \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x)
\end{aligned}
$$

http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

$$\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) \\
&= \sum_x \nabla_\theta p(x) f(x) \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x)
\end{aligned}$$

use the fact that $\nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z$

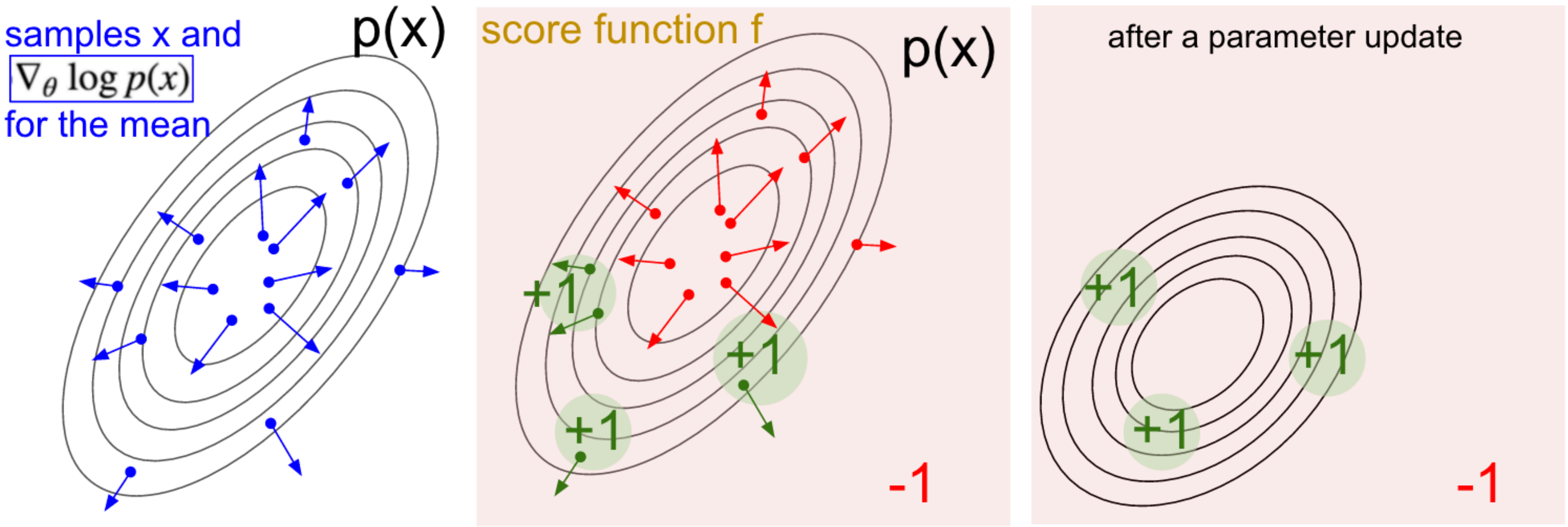
http://karpathy.github.io/2016/05/31/rl/

Derivation of policy gradient

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) \\
&= \sum_x \nabla_\theta p(x) f(x) \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) \\
&= E_x[f(x) \nabla_\theta \log p(x)]
\end{aligned}
$$

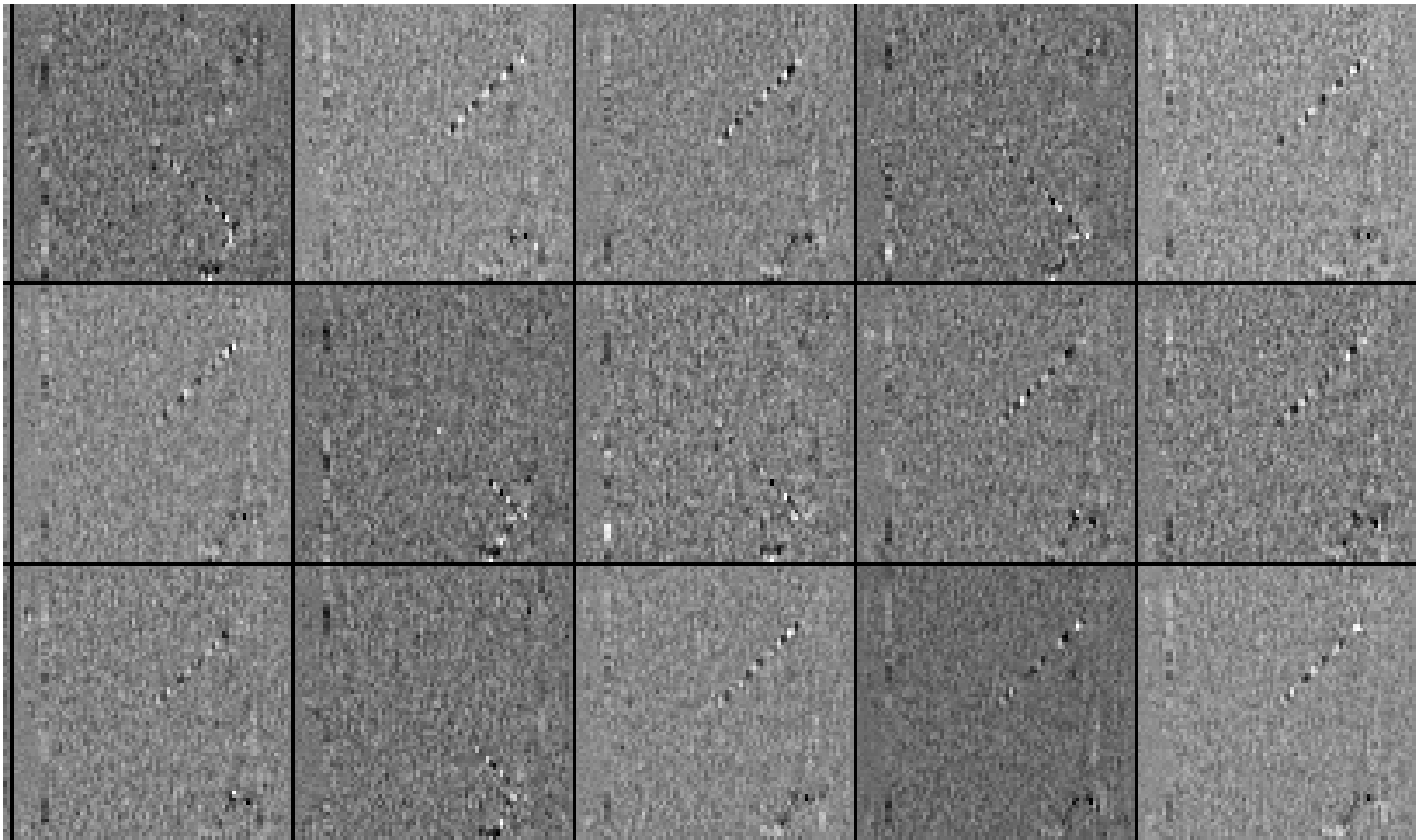
http://karpathy.github.io/2016/05/31/rl/

Policy gradient

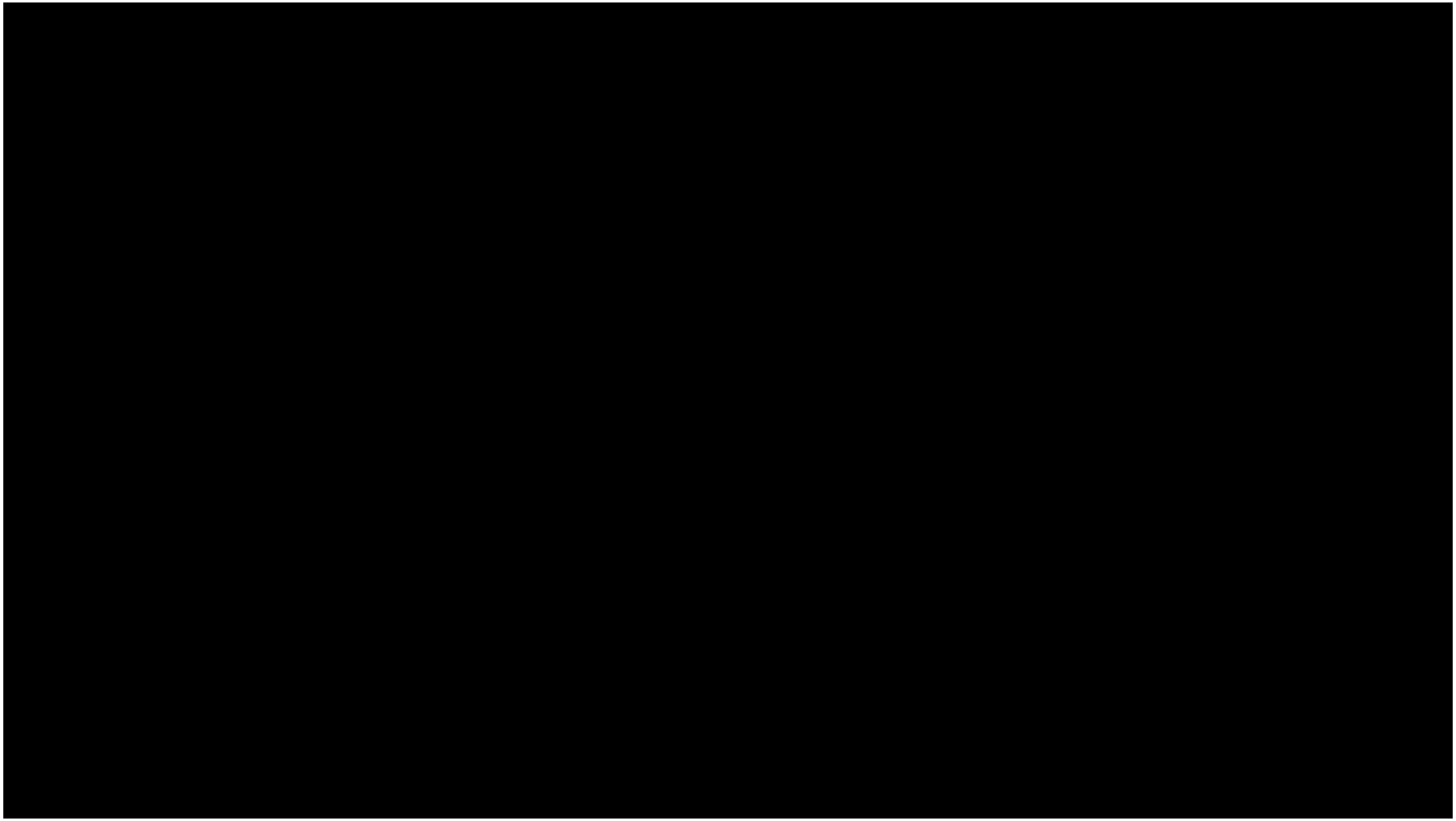


http://karpathy.github.io/2016/05/31/rl/

Learned weights for Pong



Deep Mind's bot playing Atari Breakout



https://www.youtube.com/watch?v=TmPfTpjtdgg

References

Andrew Ng's Reinforcement Learning course, lecture 16
https://www.youtube.com/watch?v=RtxI449ZjSc

Andrej Karpathy's blog post on policy gradient
http://karpathy.github.io/2016/05/31/rl/

Mnih et. al, Playing Atari with Deep Reinforcement Learning (DeepMind)
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Intuitive explanation of deep Q-learning
https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

DQN - Playing Atari

- Human-level control through deep reinforcement learning (Mnih et al. 2015)

DQN - Playing Atari

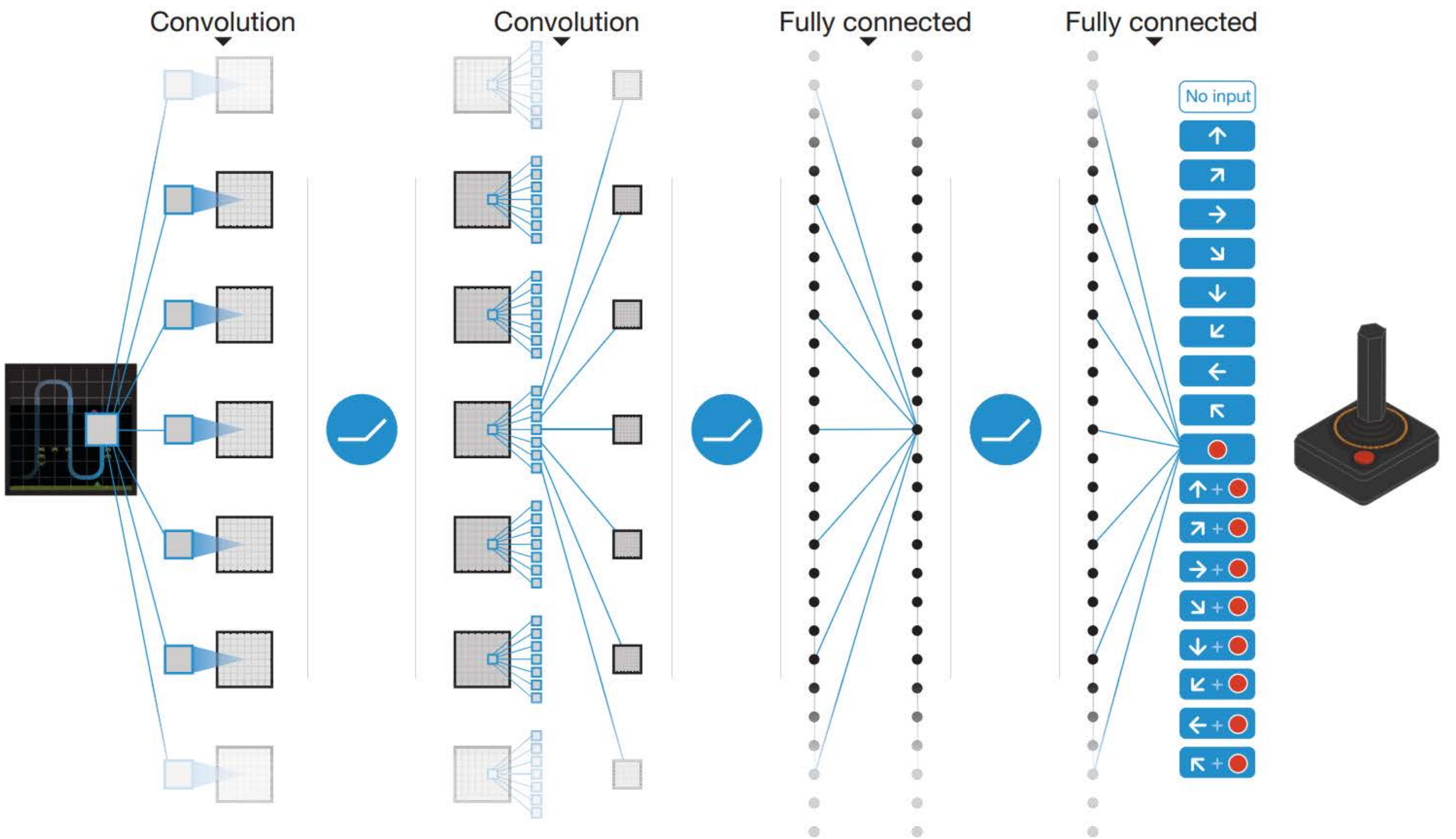
- Action-value: $$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$
- Loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}\left[\left(\mathbb{E}_{s'}\left[y \mid s,a\right] - Q(s,a;\theta_i)\right)^2\right]$$
$$= \mathbb{E}_{s,a,r,s'}\left[\left(y - Q(s,a;\theta_i)\right)^2\right] + \mathbb{E}_{s,a,r}\left[\mathbb{V}_{s'}\left[y\right]\right].$$

- Gradient:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

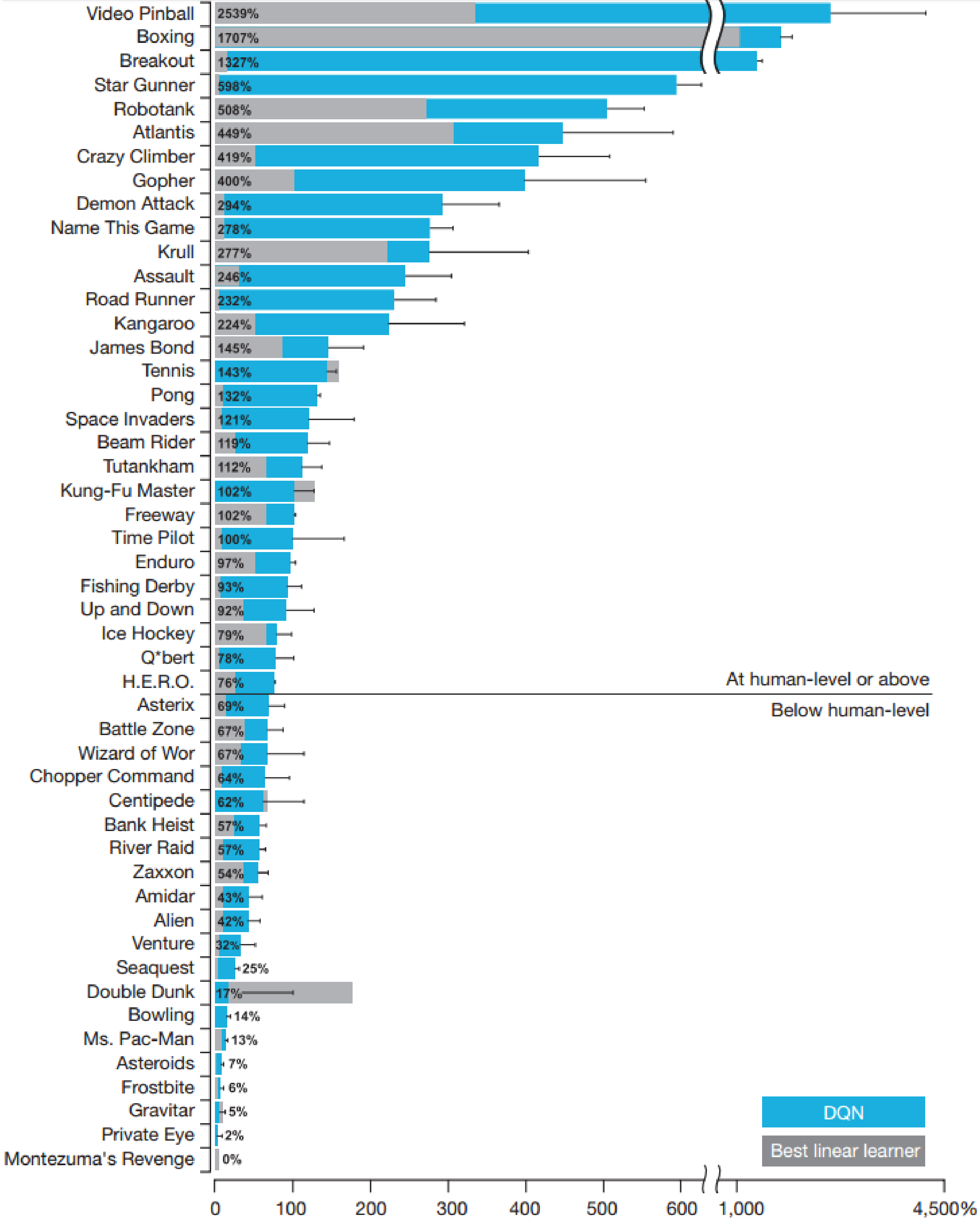
DQN - Playing Atari



DQN - Playing Atari

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


Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
119%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%