Announcements

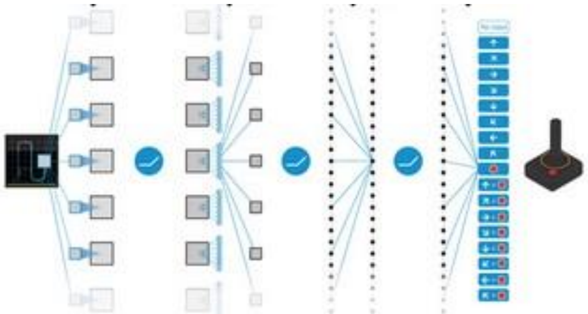
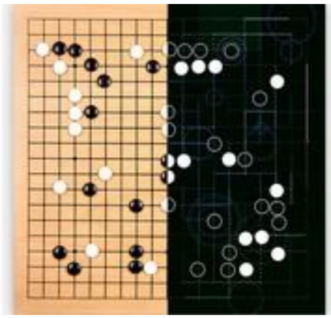
Reminder: pset5 out, due midnight today

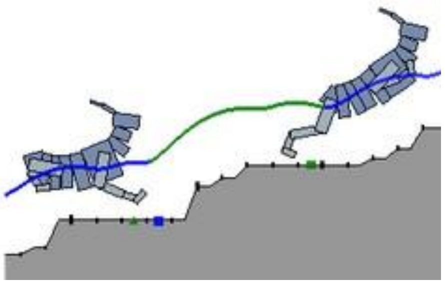
- pset5 self-grading form out Monday, due 11/16 (1 week)
- pset 6 out next week 11/12

Reinforcement Learning

Deep Mind's bot playing Atari Breakout

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



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?

Reinforcement Learning

Types of learning

Supervised

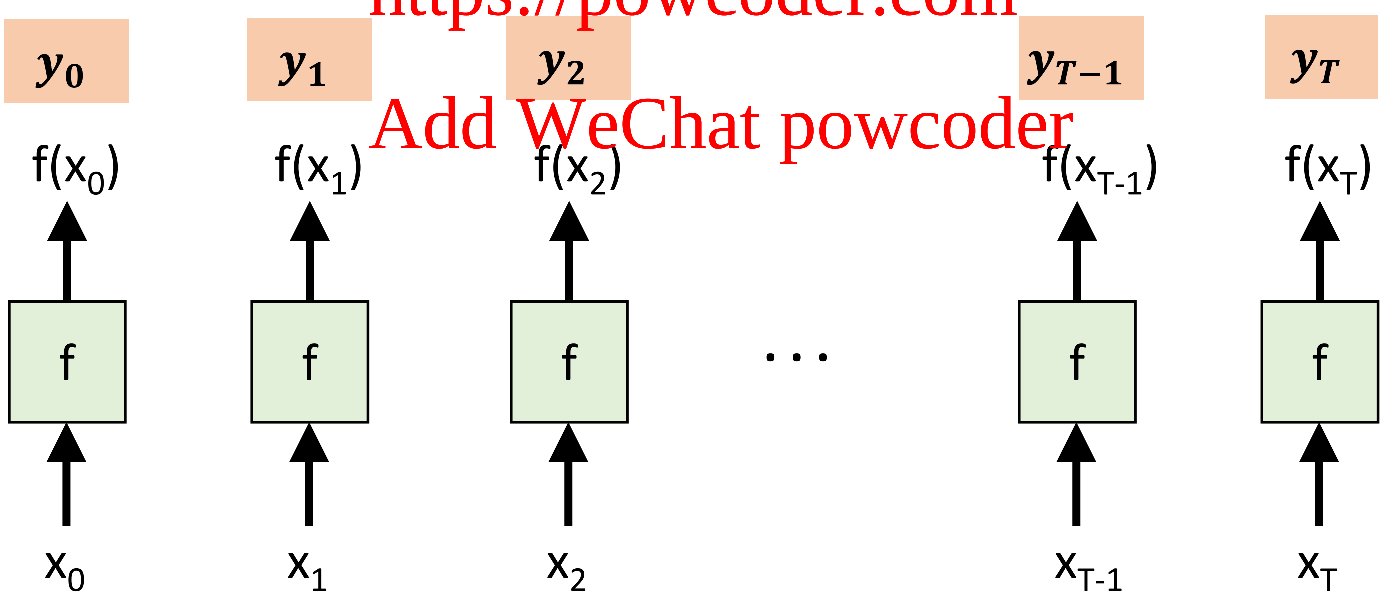
Unsupervised

Reinforcement

Supervised learning

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

Supervised learning: (in arbitrary order of examples)

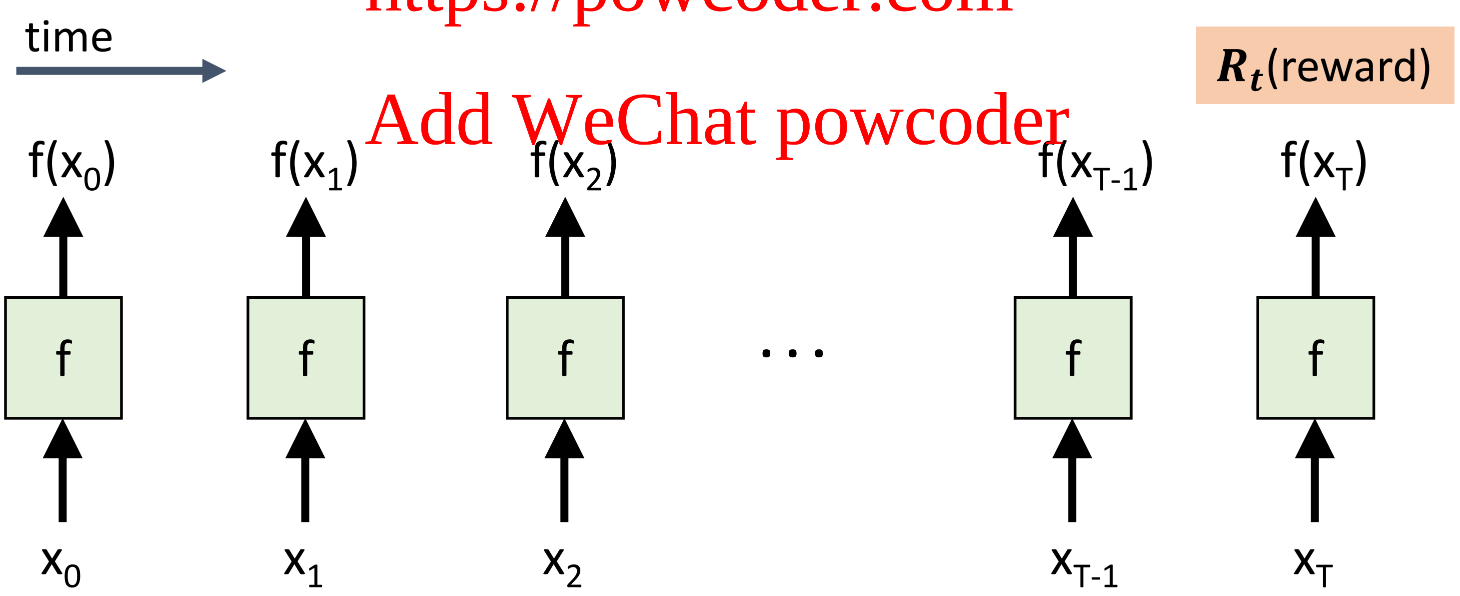


This is not how humans learn!

Reinforcement learning

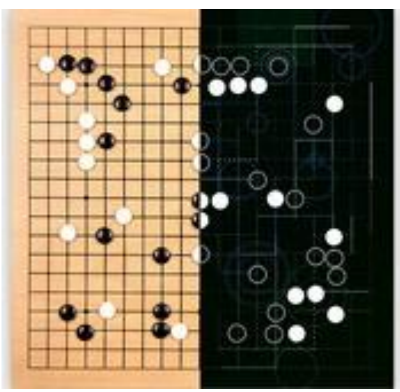
- agent receives input *x*, chooses action
- gets *R* (reward) after *T* time steps
- actions affect the next input (state)

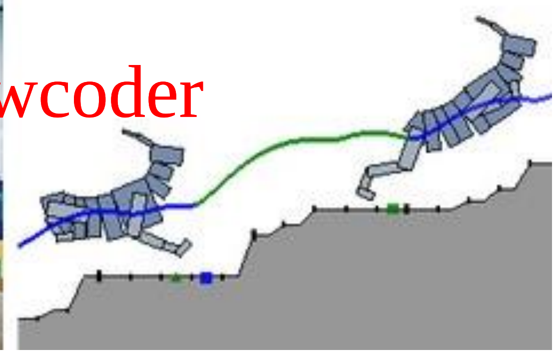
Reinforcement learning:



Input is the "world's" state

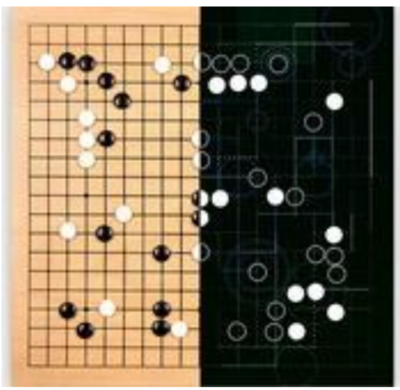
- Current game board layout
- Picture of table with blocks
- Quadriped position and orientation

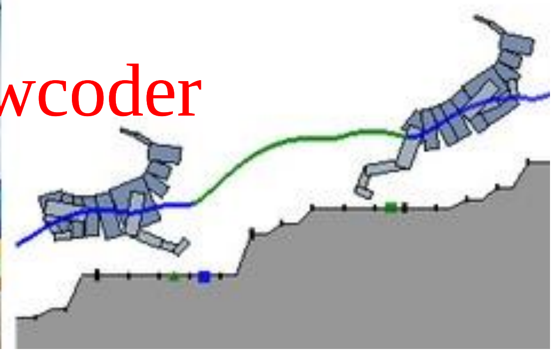




Output is an action

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



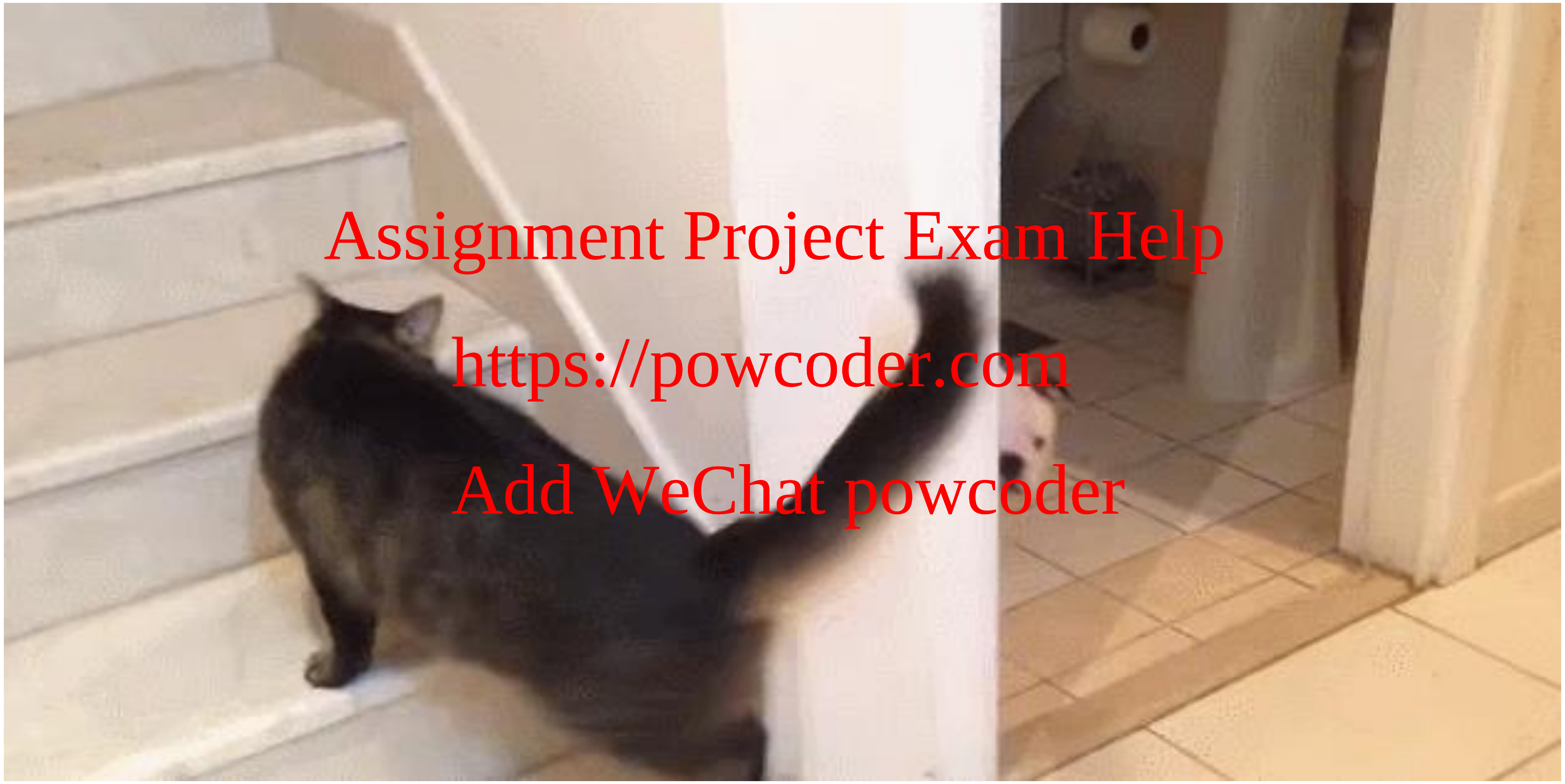


Actions affect state!

action→reward

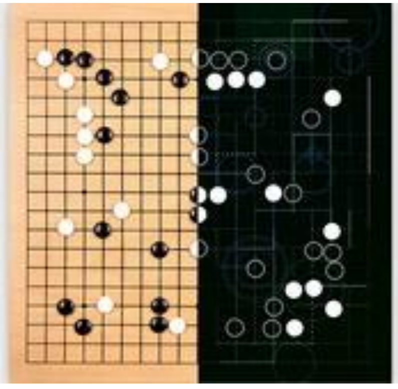
Only some actions lead to rewards

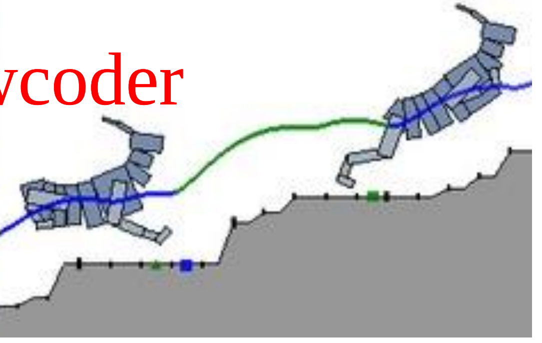
Some rewards are negative



Reward examples

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

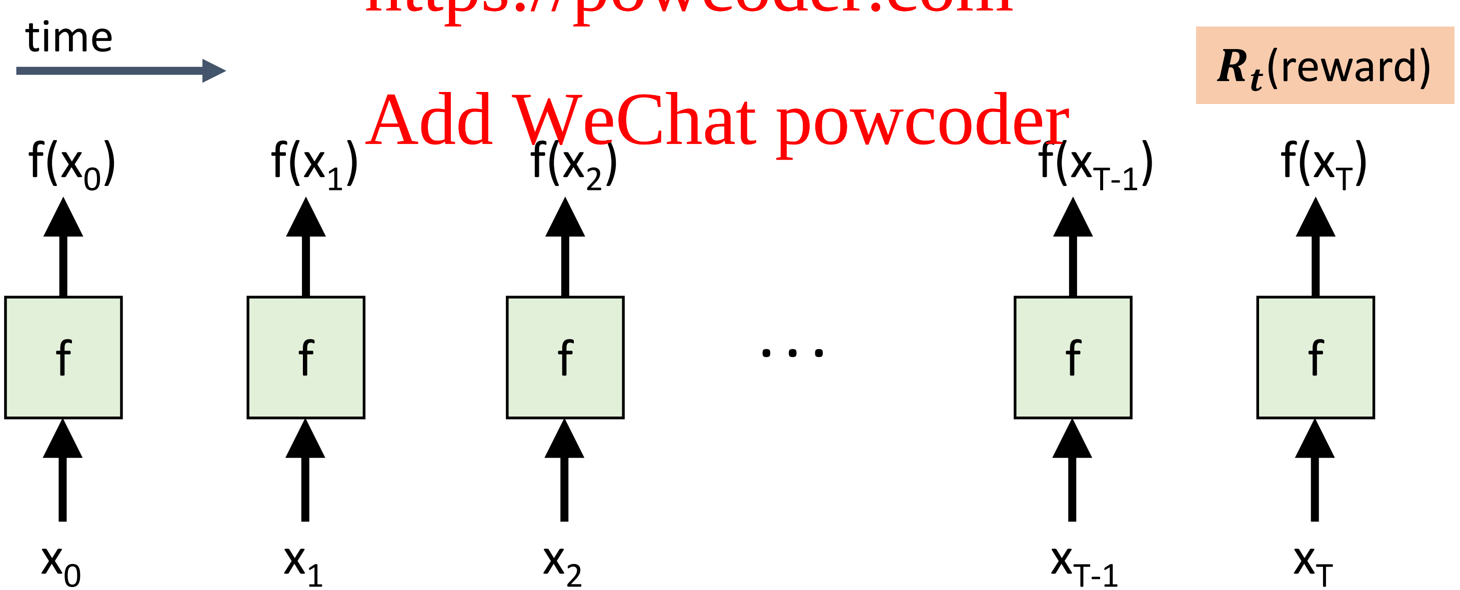




Goal of reinforcement learning

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

Reinforcement learning:

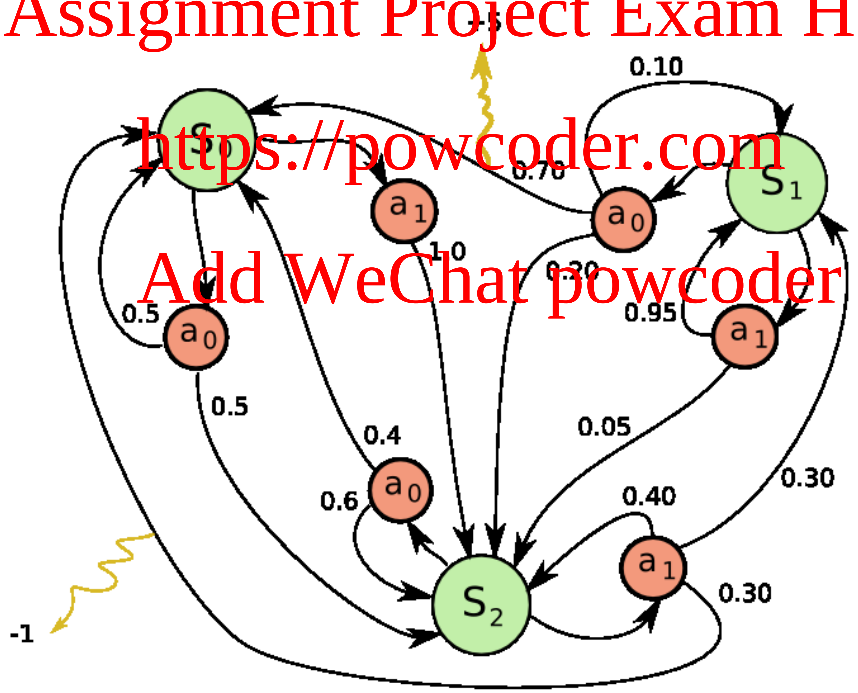


Markov Decision Process

Reinforcement Learning

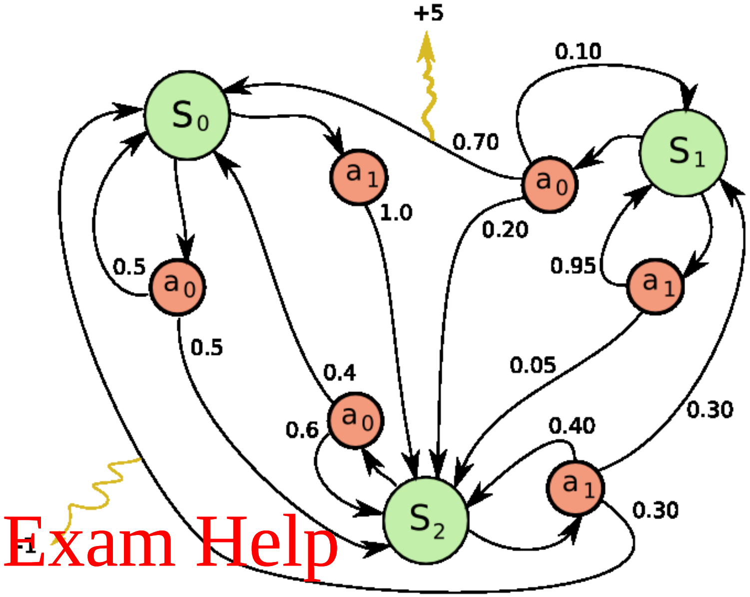
Markov Decision Process (MPD)

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)

Discount factor γ

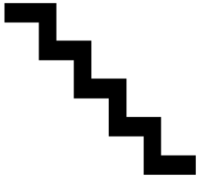
- discount factor prevents the total reward from going to infinity ($0 \leq \gamma \leq 1$)

- makes the agent prefer immediate rewards to rewards that are potentially received far away in the future

- E.g., two paths to the goal state: 1) longer path but gives higher reward 2) shorter path with smaller reward; the γ value controls which the path the agent should prefer

MDP (Simple example)

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

	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

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

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

MDP - Dynamics

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

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

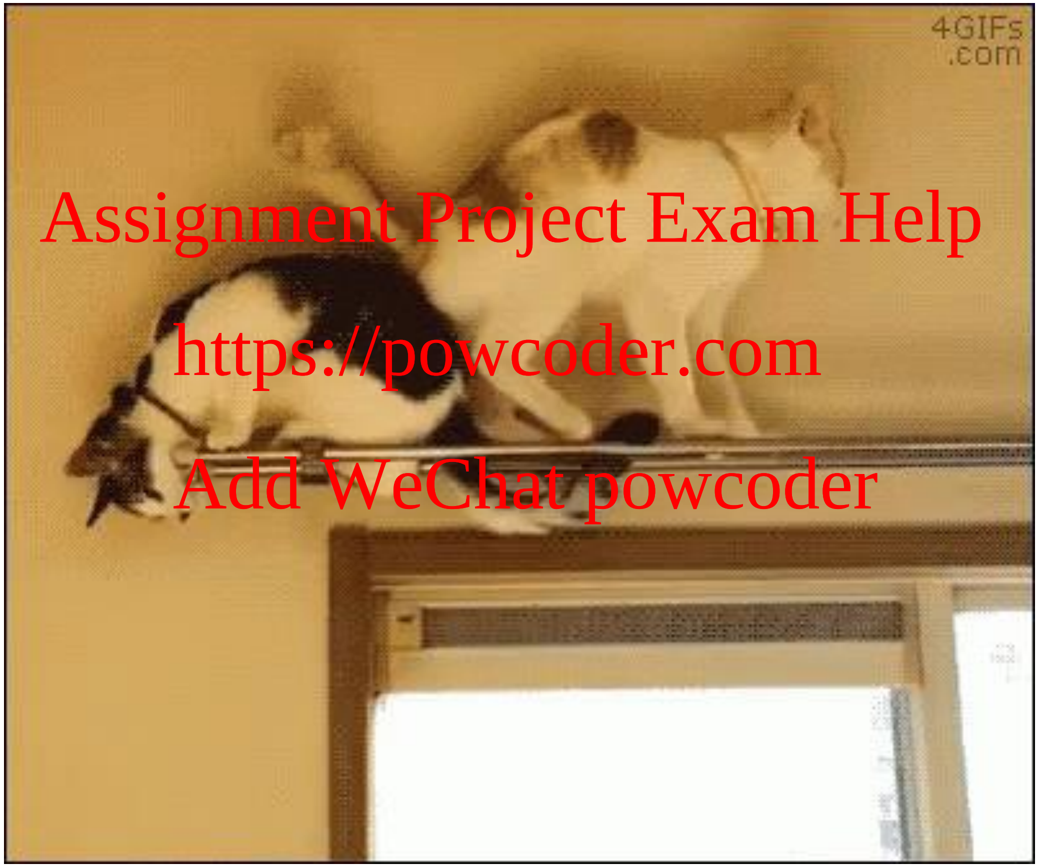
-.02	-.02	-.02

- 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 P_{sa}

	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, how much to delay reward

MDP – Policy π

States S = locations

Actions A = {↑ ,→ ,←,↓ }

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

Transition P_{sa}

	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 is a function $\pi: S \rightarrow A$

Policy Value and Q functions

Reinforcement Learning

MDP – Policy value function

States S = locations

Actions A = {↑ ,→ ,← ,↓ }

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

Transition P_{sa}

Policy $\pi: S \to A$

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

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

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

Expected sum of discounted rewards

MDP – Policy value function

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

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

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

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

solve $|S|$ eqs.

$$\begin{cases} V_1 = R(1) + \gamma P_{1,\downarrow}(2)\, V_2 + \ldots \\ V_2 = \ldots \\ \ldots \\ V_{10} = R(10) + \gamma\big(P_{10,\uparrow}(6) V_6 + P_{10,\uparrow}(9) V_9 + P_{10,\uparrow}(11) V_{11}\big) \\ \ldots \end{cases}$$

	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

Optimal value function

If the agent uses a given policy **π** to select actions, the corresponding value function is given by:

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

There exists an optimal value function that has higher value than other functions for all states

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathbb{S}$$

The optimal policy **π*** is the policy that corresponds to the optimal value function

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in \mathbb{S}$$

Value function vs. Q-function

For convenience, RL algorithms introduce the Q-function, which takes a state-action pair and returns a real value

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

The optimal Q-function $Q^*(s, a)$ means the highest expected total reward received by an agent starting in s and choosing action a which maximizes value over

$$V^*(s) = \max_a Q^*(s, a) \quad \forall s \in \mathbb{S}$$

$Q^*(s, a)$ is an indication for how good it is for an agent to pick action a while being in state s

Optimal Q-function

If we know the optimal Q-function $Q^*(s,a)$, the optimal policy π^* can be easily extracted by choosing the action a that gives maximum $Q^*(s,a)$ for state s:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \quad \forall s \in \mathbb{S}$$

RL Approaches

Reinforcement Learning

Reinforcement learning approaches

Value-based RL

- Estimate the optimal value function
- *i.e.*, the maximum value achievable under any policy
- Guaranteed to converge to optimum

Policy-based RL

- Search directly for the optimal policy
- re-define the policy at each step and compute the value according to this new policy until the policy converges
- Guaranteed to converge, often faster than value

Q-learning

- Search for the optimal Q-function
- No prior knowledge of MDP required

Value Iteration Algorithm

Given $\mathrm{P}_{s,a}(s') = p(s'|s,a)$, iteratively compute the Q and value functions using Bellman's equation.

Initialize $V(s)$ to arbitrary values

Repeat

For all $s \in S$

For all $a \in \mathcal{A}$

$Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$

$V(s) \leftarrow \max_a Q(s,a)$

Until $V(s)$ converge

Policy Iteration Algorithm

Given $\mathrm{P}_{s,a}(s') = p(s'|s,a), \pi$, iteratively compute the policy's value function and improve the policy to maximize it

Initialize a policy π' arbitrarily

Repeat

$\pi \leftarrow \pi'$

Compute the values using π by

solving the linear equations

$$V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V^{\pi}(s')$$

Improve the policy at each state

$$\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$$

Until $\pi = \pi'$

Reinforcement learning approaches

Optimal value function — need P_{sa}

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

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

Optimal policy — need π, 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: $Q^*(s,a) = R(s) + \gamma \max_{a \in A} Q^*(s', a)$

Q-Learning (discrete)

Reinforcement Learning

Q-value function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal Q-value function

- An optimal value function is the maximum achievable value

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) = Q^{\pi^*}(s, a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s, a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

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

Q-learning algorithm

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

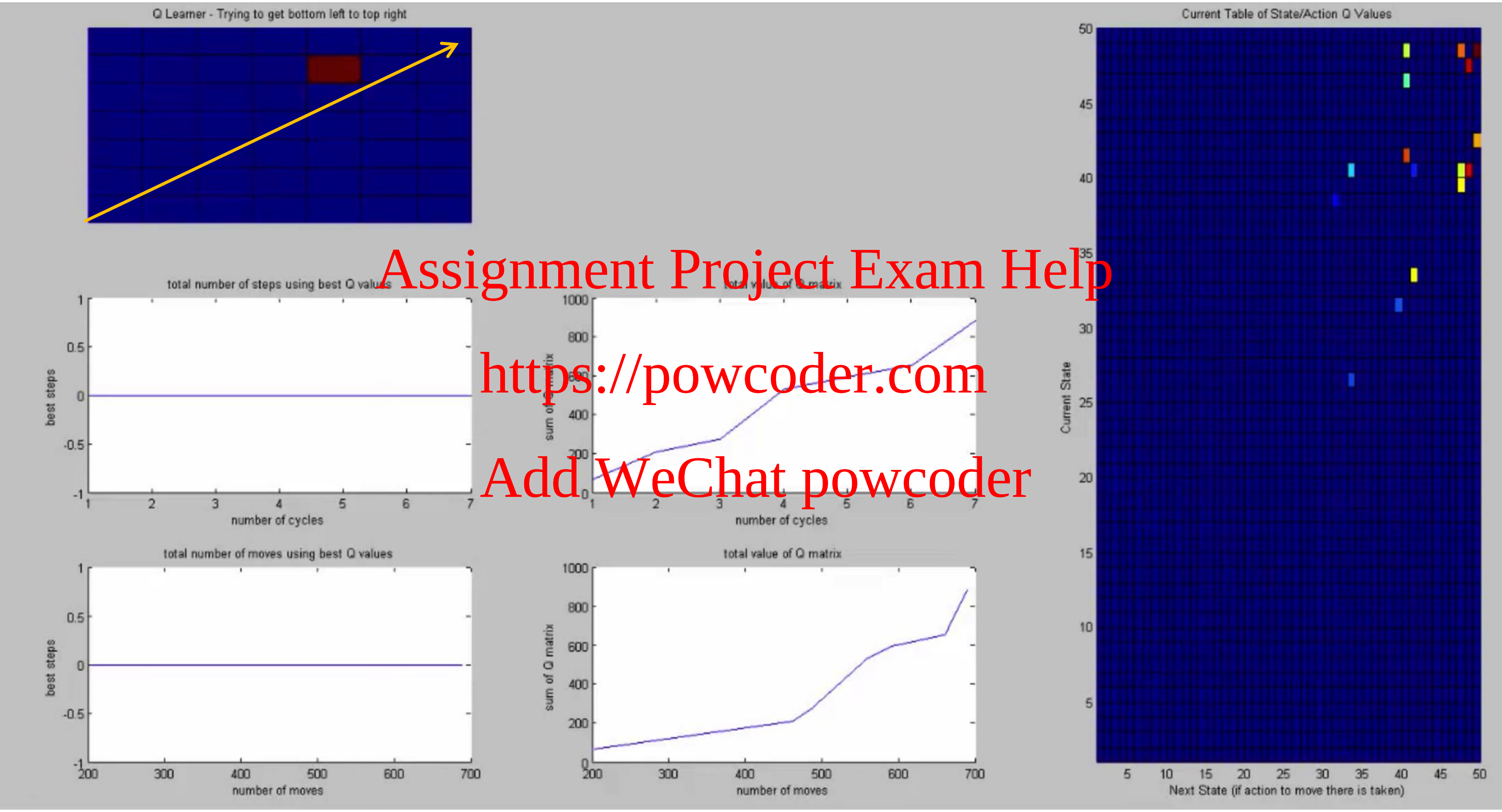
learning rate

discount

largest increase over all possible actions in new state

Q-learning example

Goal: get from bottom left to top right



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

Exploration vs exploitation

- How does the agent select actions during learning? Should it trust the learned values of Q(s, a) to select actions based on it? or try other actions hoping this may give it a better reward?

- This is known as the exploration vs exploitation dilemma

- Simple ε-greedy approach: at each step with small probability ϵ, the agent will pick a random action (explore) or with probability (1-ϵ) the agent will select an action according to the current estimate of Q-values

- The ϵ value can be decreased overtime as the agent becomes more confident with its estimate of Q-values

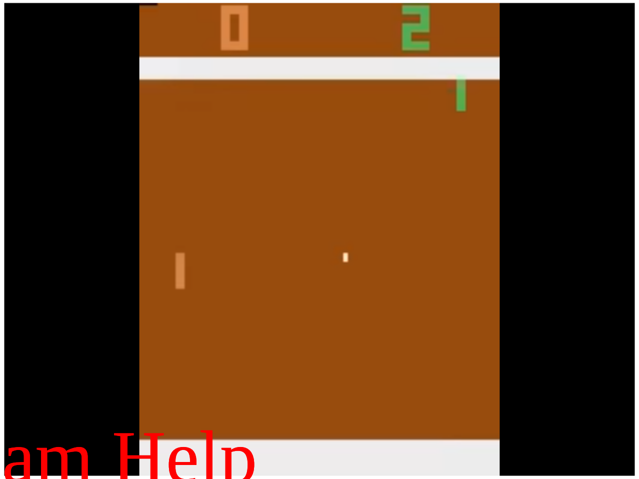
Continuous state

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 of the state!

Deep RL

Reinforcement Learning

Deep RL playing DOTA

https://www.youtube.com/watch?v=eHipy_j29Xw

Deep RL

- V, Q or π can be approximated with deep network
- Deep Q-Learning
  - Input: state, action
  - Output: Q-value

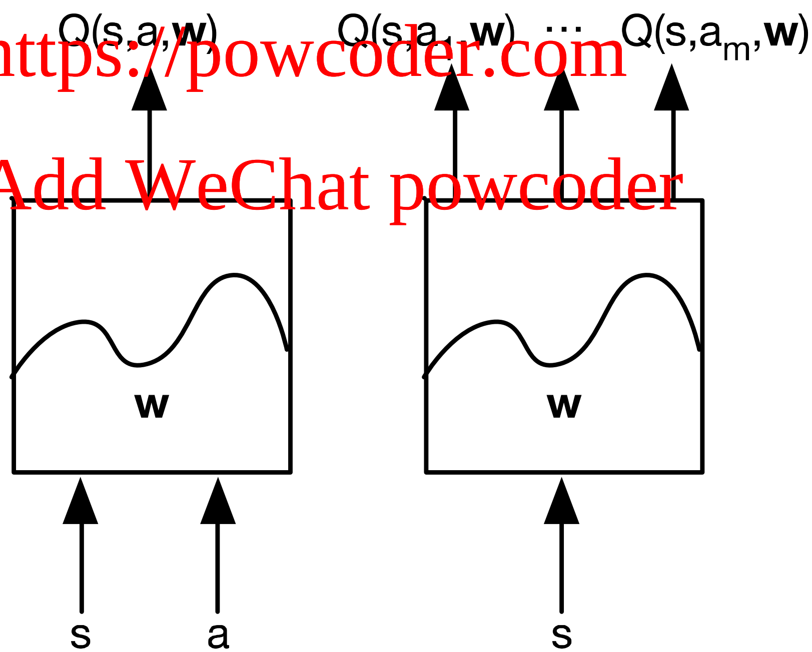
Cover today

- Alternative: learn a Policy Network
  - Input: state
  - Output: distribution over actions

Q-value network

Represent value function by Q-network with weights $\mathbf{w}$

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



Q-value network

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand side $r + \gamma \max_{a} Q(s',a',\mathbf{w})$ as a target

- Minimise MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w}) \right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Deep Q-network (DQN)

To remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3 $\rightarrow$ s, a, r, s'
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

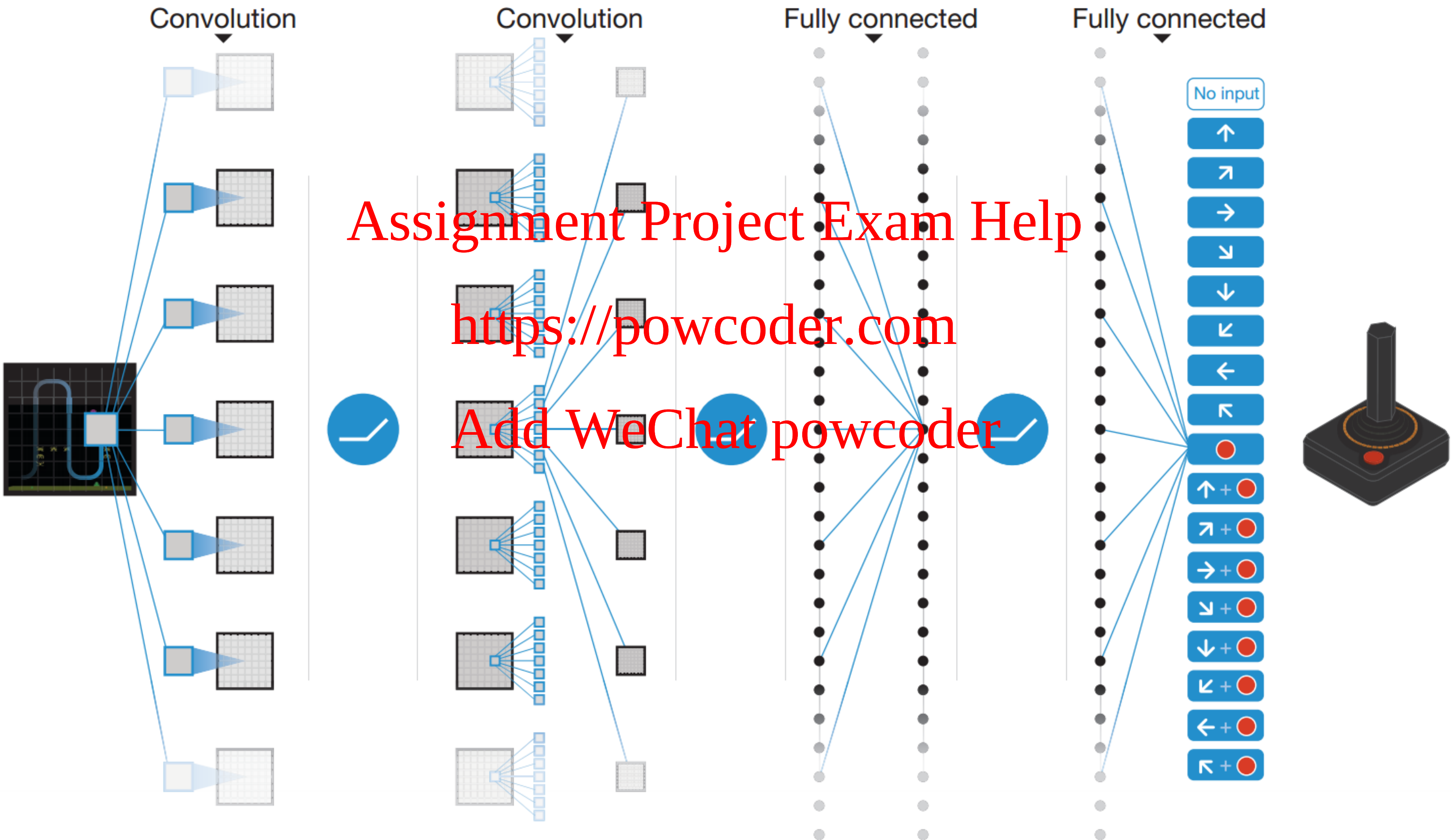
$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$

Sample experiences from data-set and apply update

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

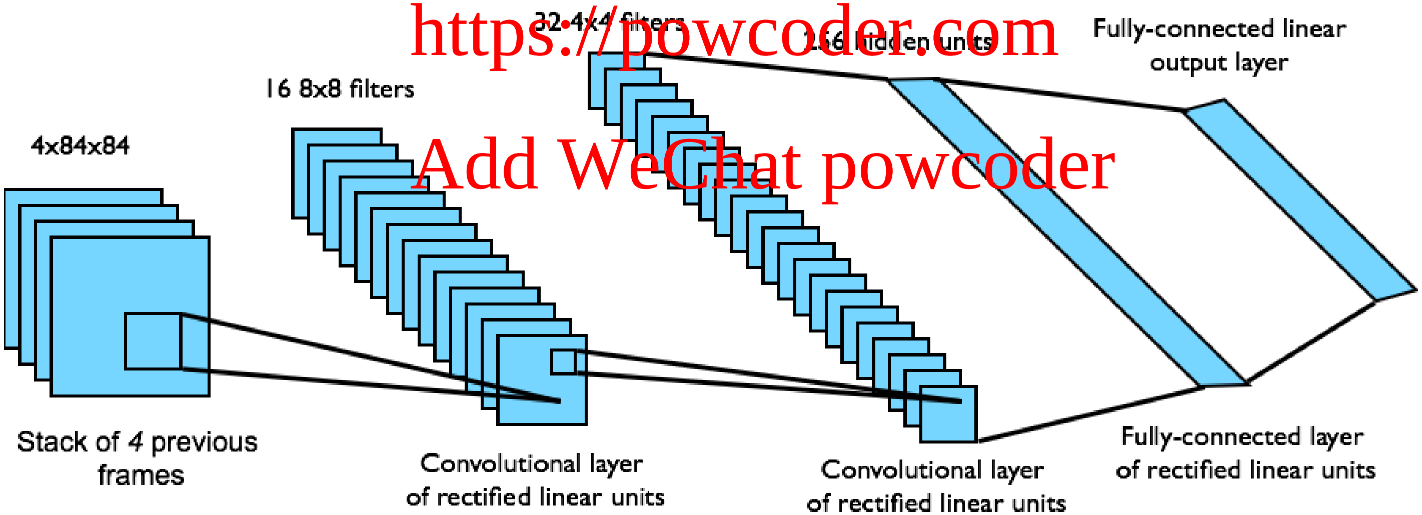
To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

DQN - Playing Atari



DQN - Playing Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

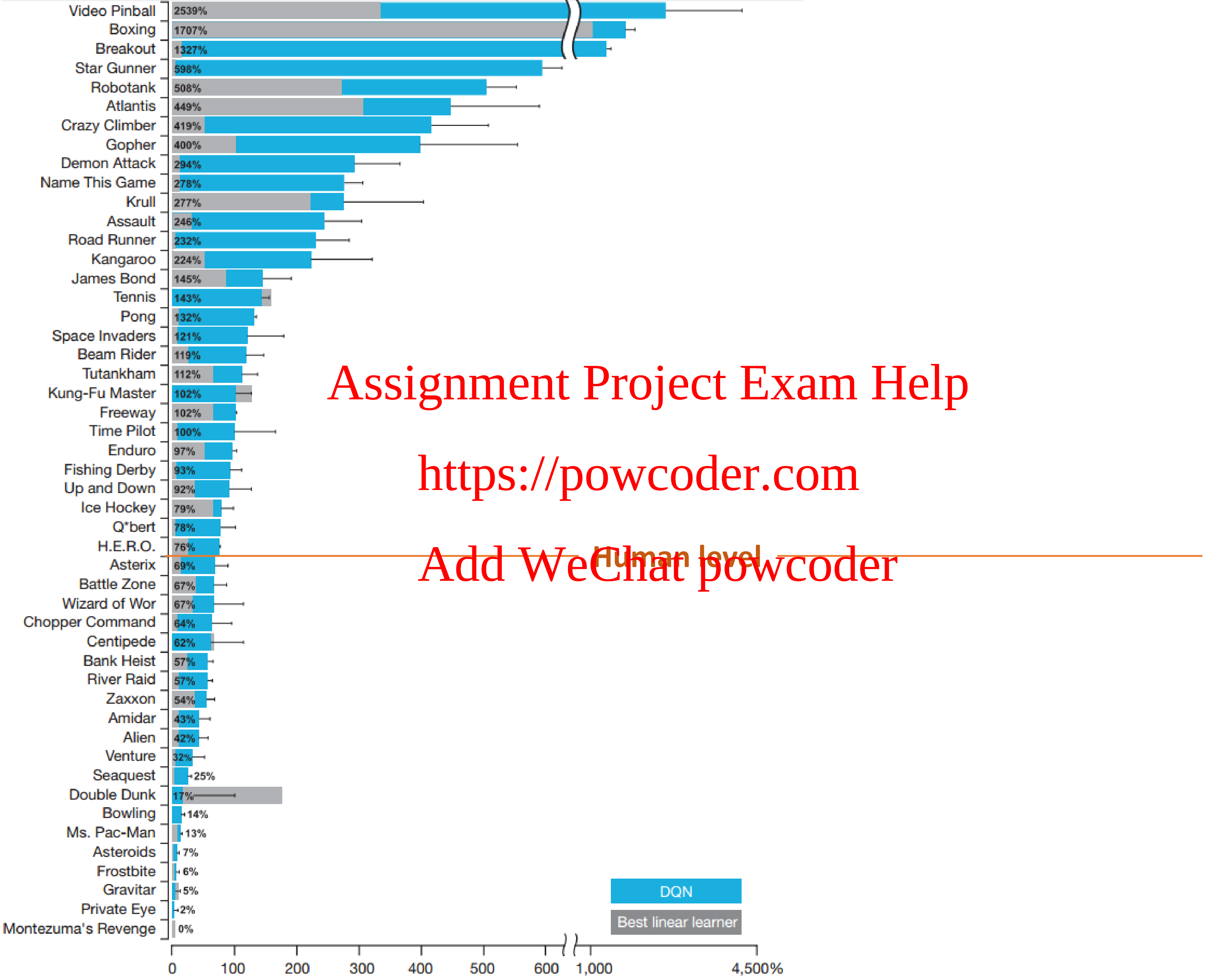


Network architecture and hyperparameters fixed across all games

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%
Human level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

DQN for Atari

DQN paper:

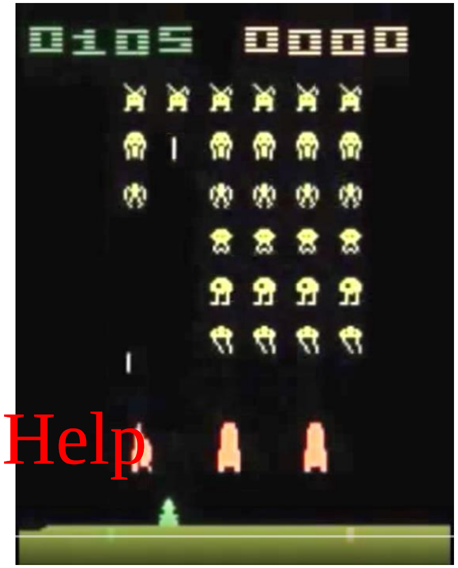
www.nature.com/articles/nature14236

DQN demo:

https://www.youtube.com/watch?v=iqXKQf2BOSE

DQN source code:

www.sites.google.com/a/deepmind.com/dqn/



Downsides of RL

- RL is less sampling efficient than supervised learning because it involves bootstrapping, which uses an estimate of the Q-value to update the Q-value predictor
- Rewards are usually sparse and learning requires to reach the goal by chance
- Therefore, RL might not find a solution at all if the state space is large or if the task is difficult

Summary

- The goal of Reinforcement learning:
  - learn to predict actions that maximize future rewards
- Markov Decision Process:
  - Formalizes the RL framework as
  - MDP = (S, A, R, P_{sa}, γ)
- Approaches to reinforcement learning:
  - Learn value function (offline)
  - Learn optimal policy (offline)
  - Learn Q-function (online)

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/

Next Class

Reinforcement Learning II

Q-learning cont'd; deep Q-learning (DQN)