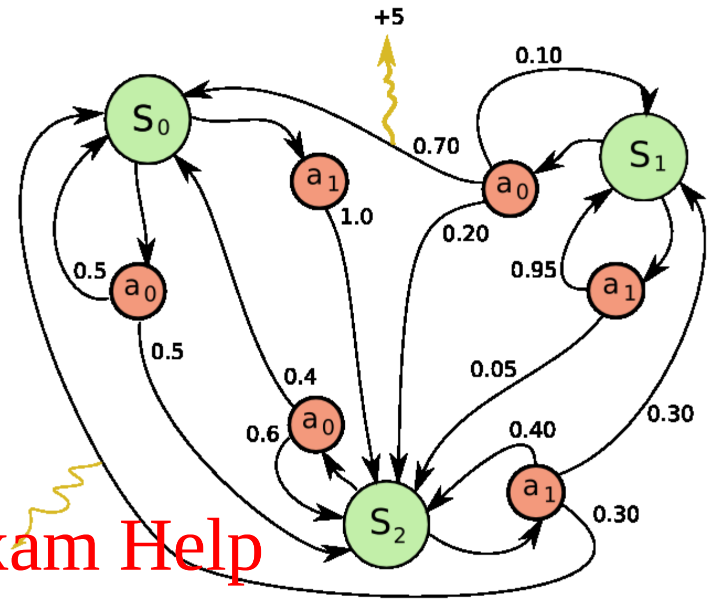
Announcements

Reminder: pset5 self-grading form and pset6 out Thursday, due 11/19 (1 week)

- No lab this week!

Reinforcement Learning II

Recall: 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)

MDP (Simple example)

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

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

-.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

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

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

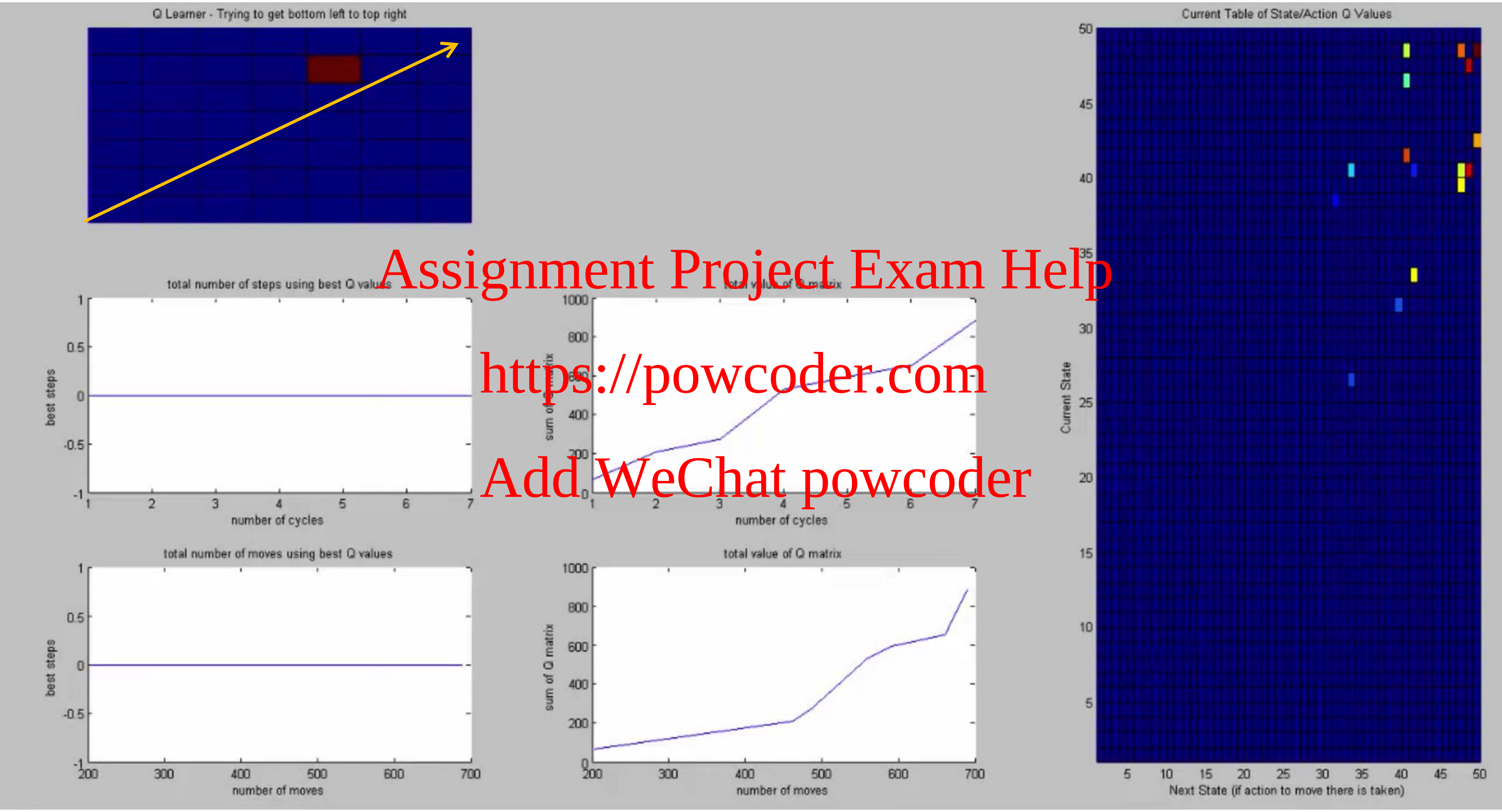
discount

largest increase over all possible actions in new state

learning rate

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 $(1-\epsilon)$ 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

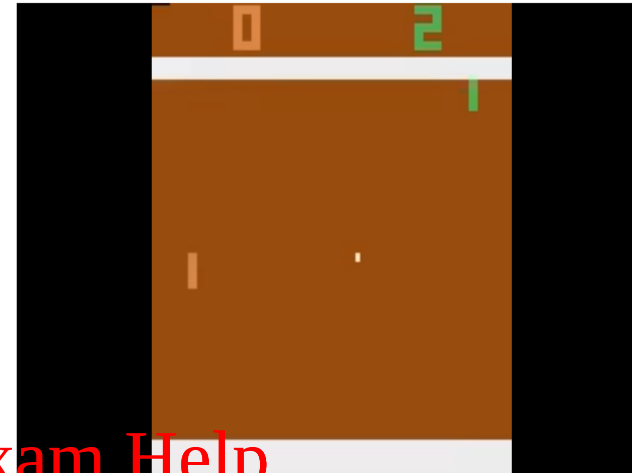
Continuous state

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)
- P_{sa} – 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

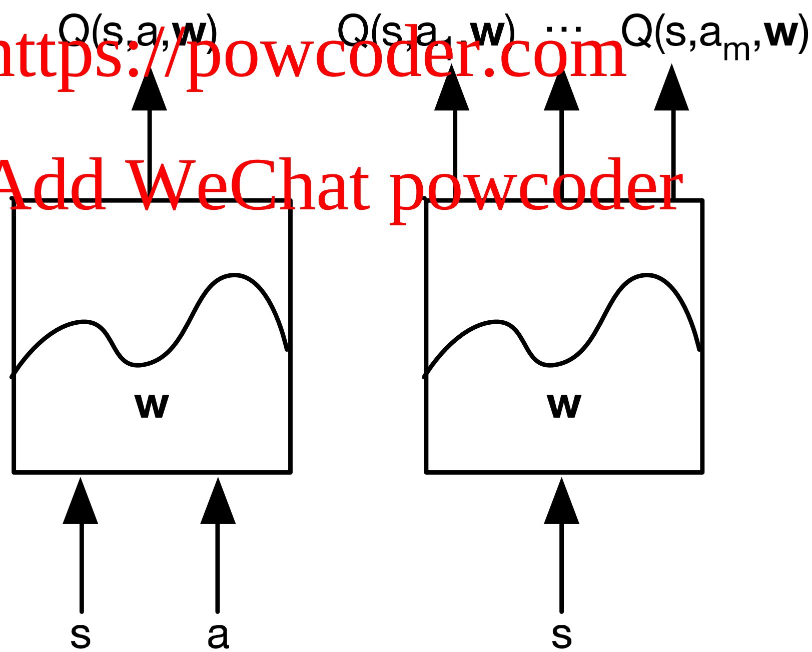
Deep RL

- V, Q or π can be approximated with deep network
- Deep Q-Learning
  - Input: state, action
  - Output: Q-value
- 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
...
$s_t, a_t, r_{t+1}, s_{t+1}$

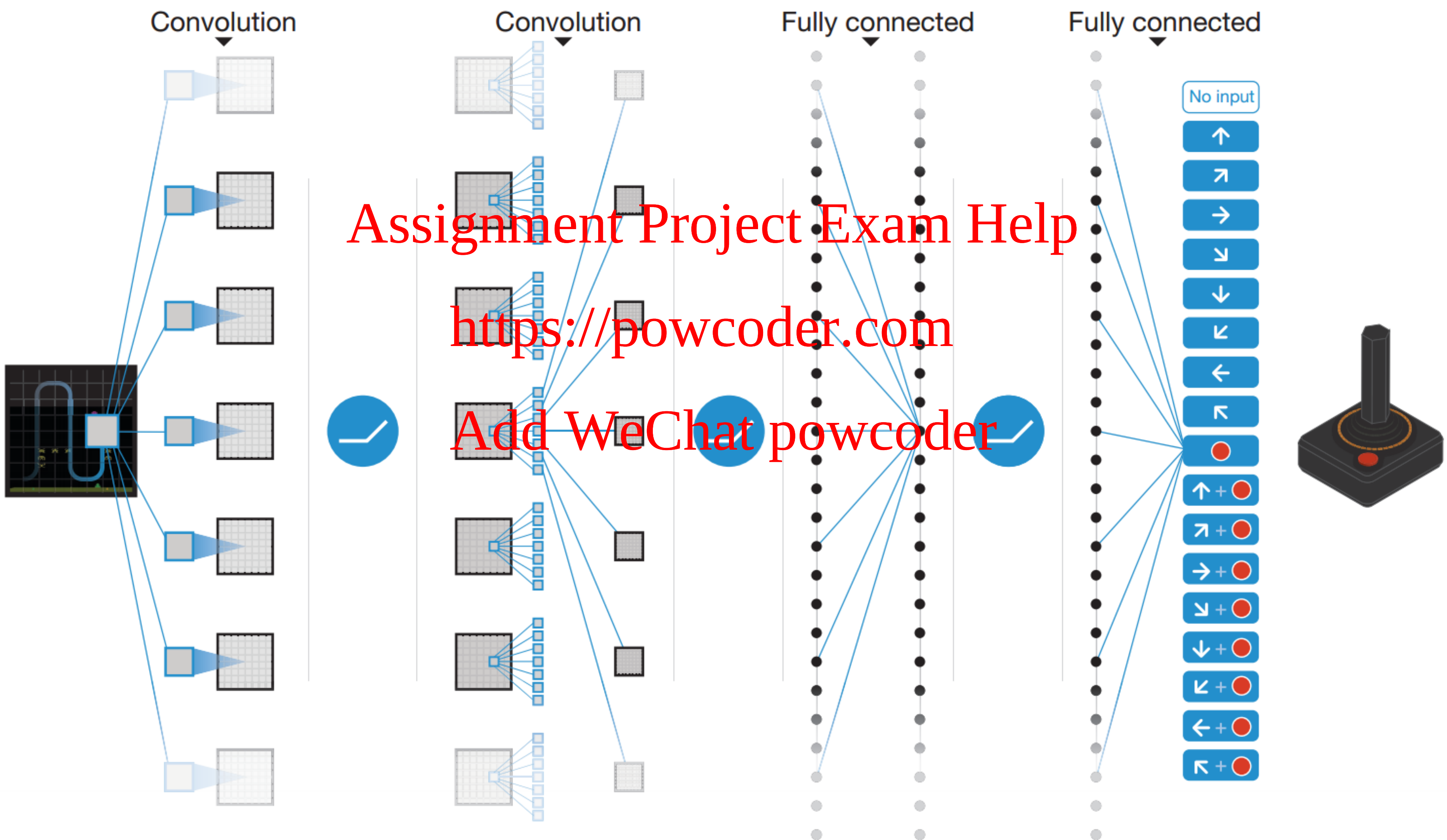
$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

Sample experiences from data-set and apply update

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

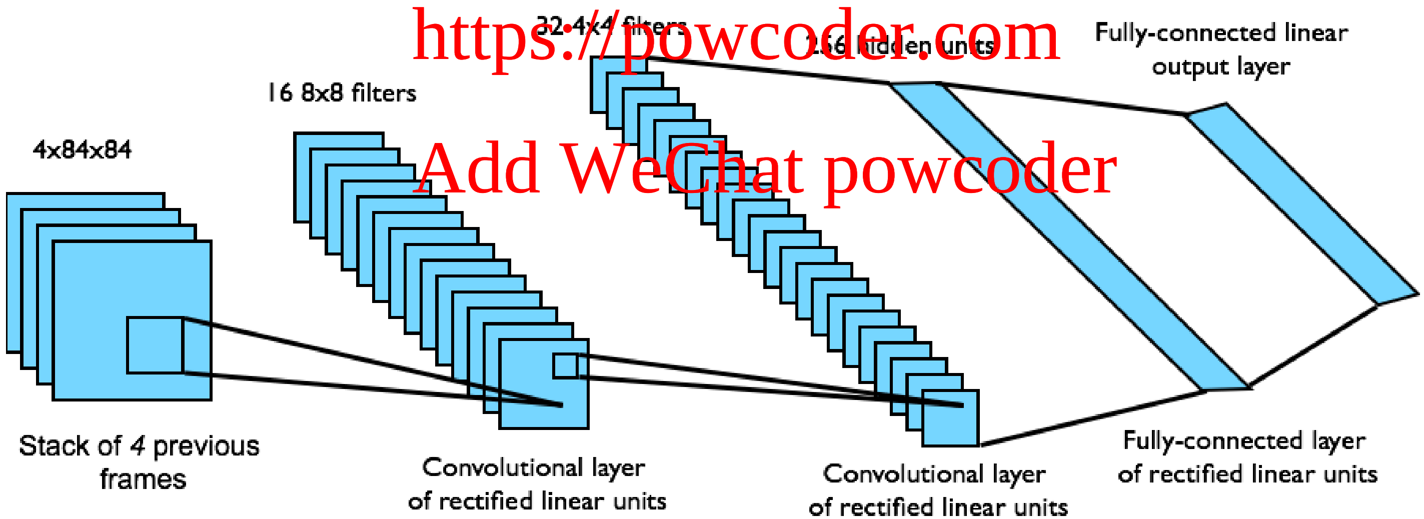
To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

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

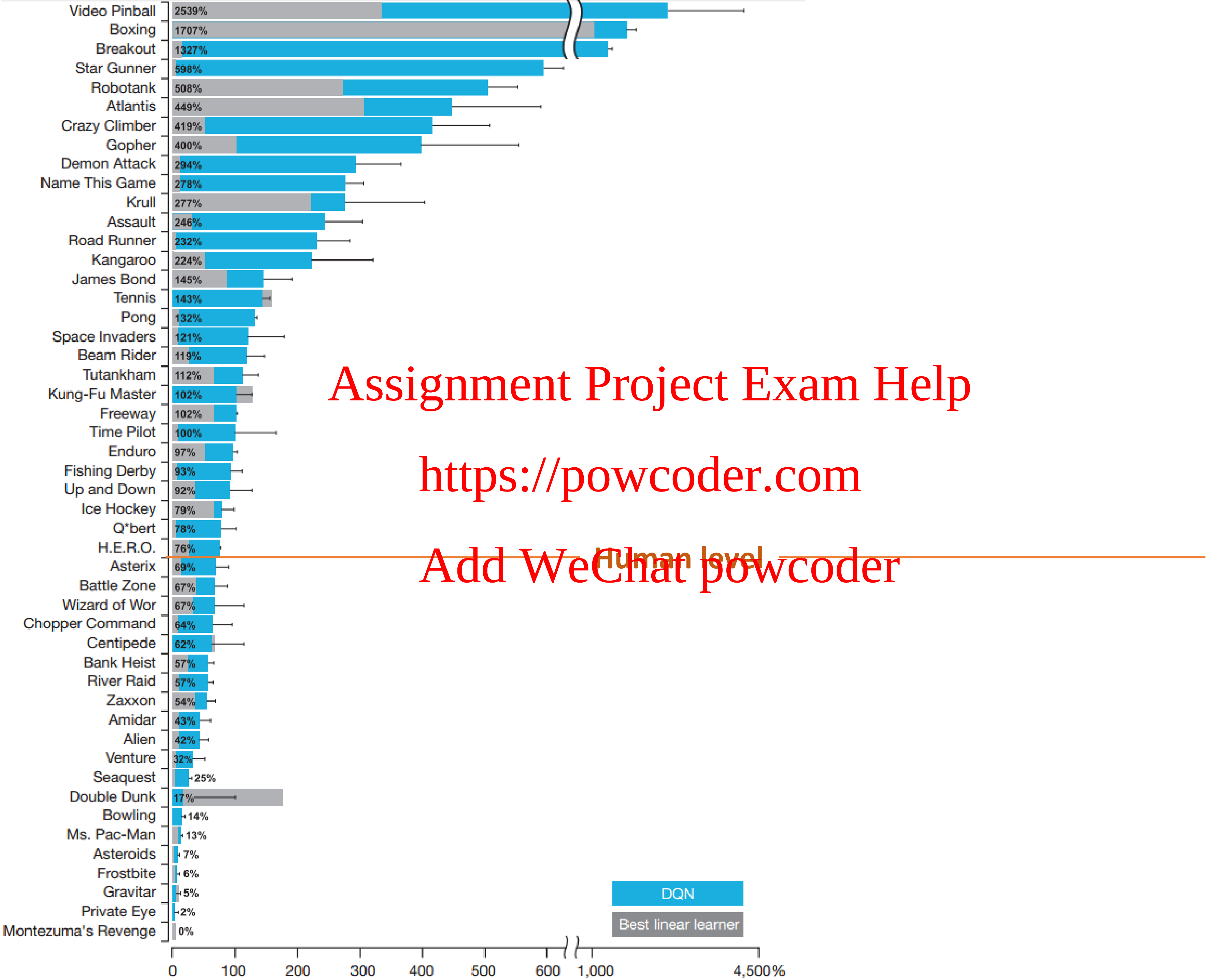


Network architecture and hyperparameters fixed across all games

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

DQN for Atari

DQN paper:

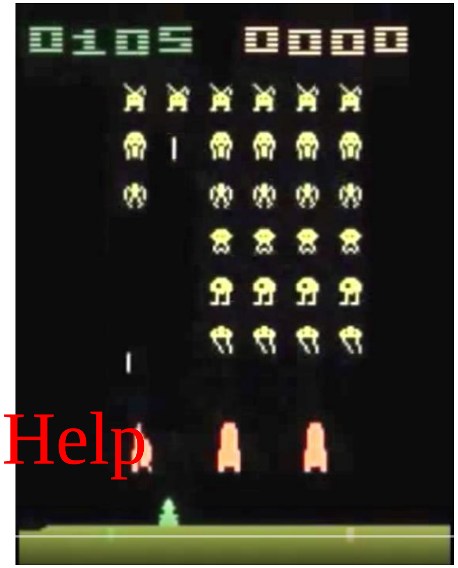
www.nature.com/articles/nature14236

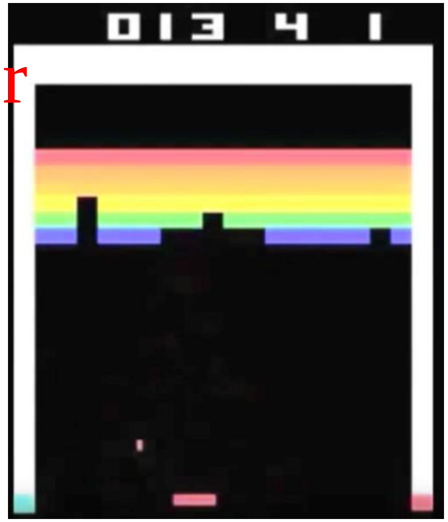
DQN demo:

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

DQN source code:

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

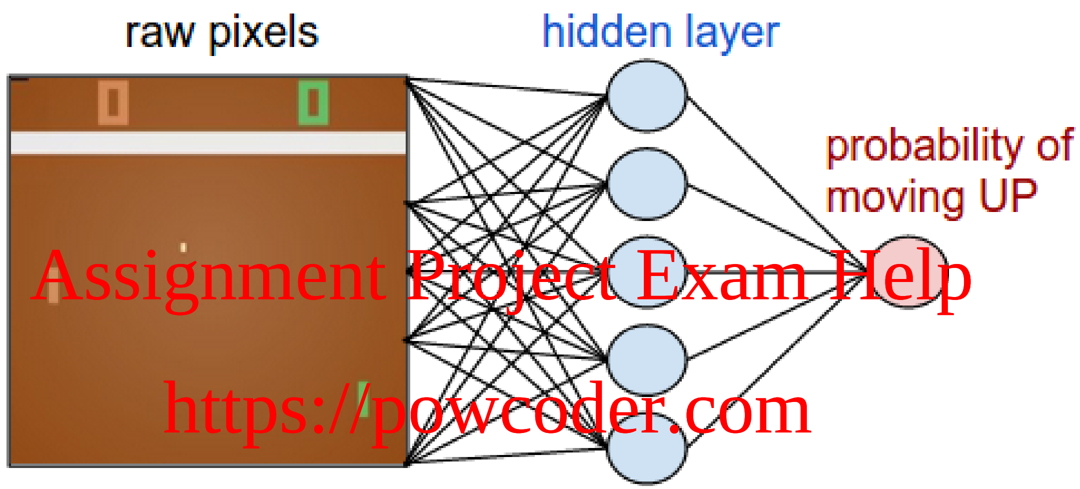




Deep RL

- 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

Policy network for pong

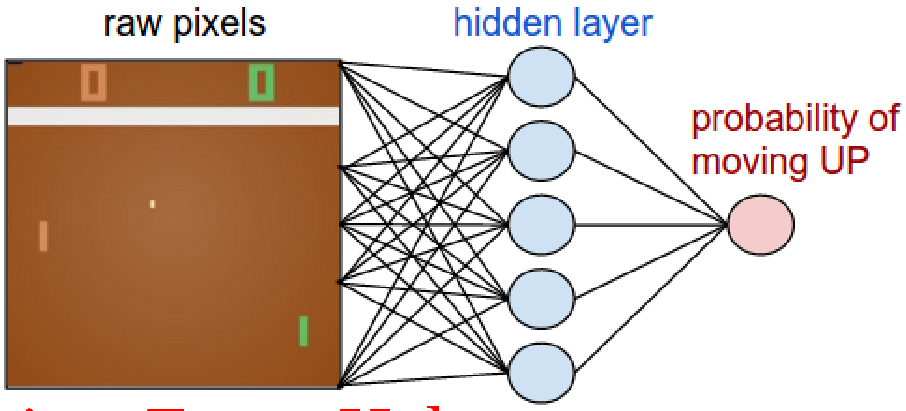


- define a *policy network* that implements the player
- takes the state of the game and decides what to do (move UP or DOWN)
- 2-layer neural network that takes the raw image pixels* (100,800 = 210x160x3), outputs the probability of going UP

*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%

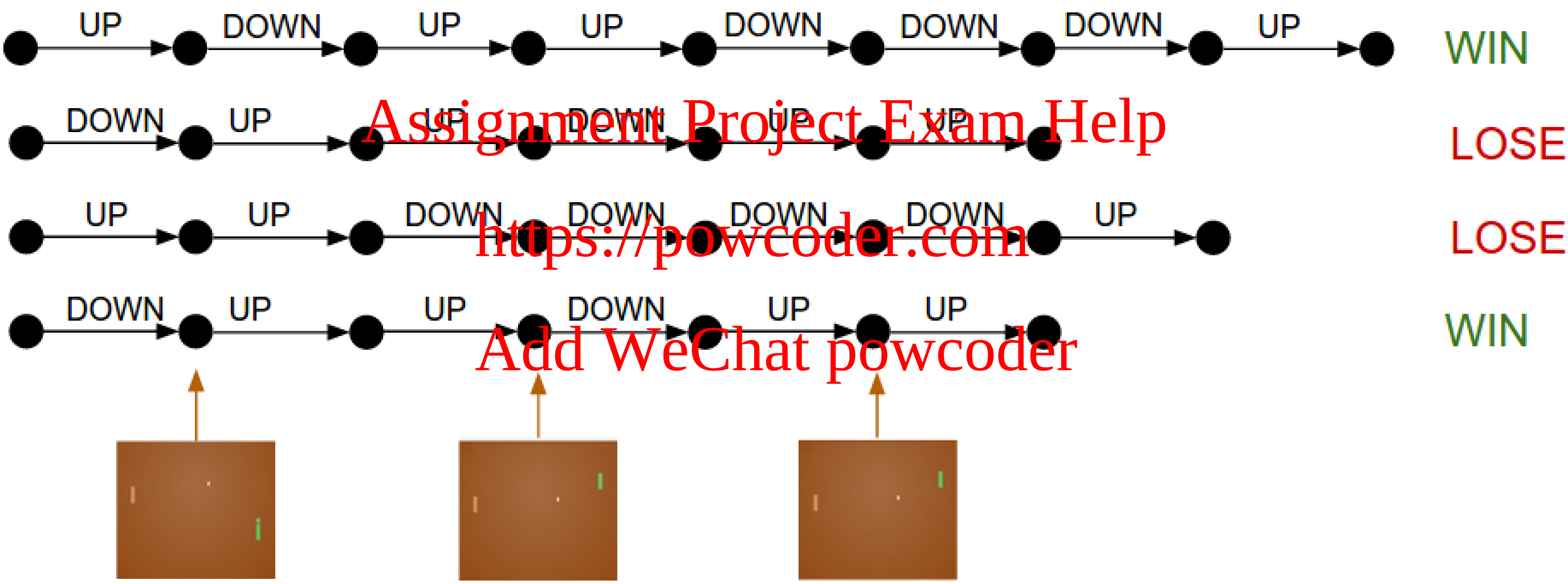
- Can sample an action from this distribution and execute it
- Can immediately use gradient of 1.0 for DOWN and backprop to find the gradient vector that would encourage the network to predict DOWN

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/

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/

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/

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

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

Unsupervised Learning III: Anomaly Detection

Anomaly detection methods: density estimation, reconstruction based method, One Class SVM; evaluating anomaly detection