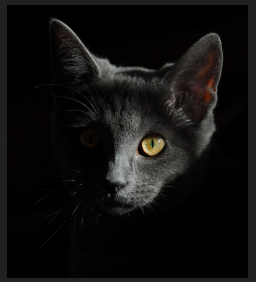
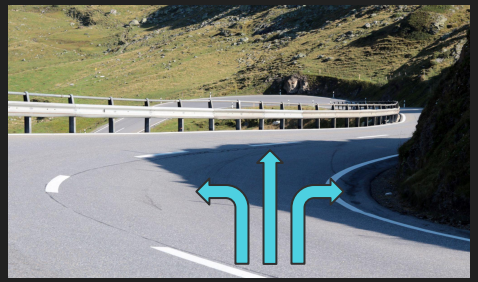
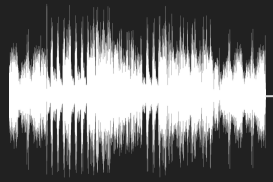
Lecture 10
Reinforcement learning - DQN



Dog or cat?



Left, straight or right?



/ˌgʊd ˈmɔː.nɪŋ/

Speech to phonemes

Yellow paperclip

Given a sentence create an image

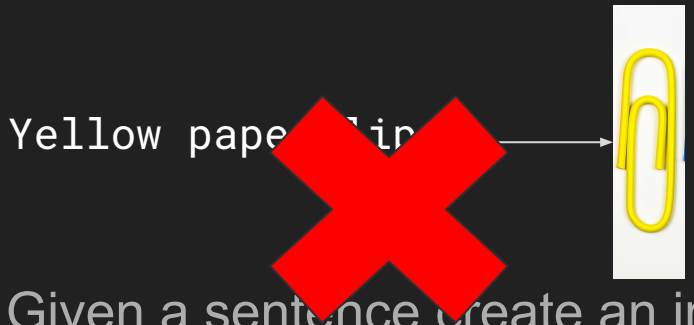
Supervised learning:

- Unique output (correct answer aka label) for each input.
- Input samples are independent.

Current output influences future inputs.

Left, straight or right?

Non-unique output.



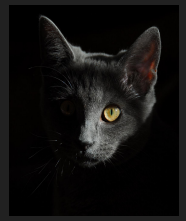
Given a sentence create an image

Supervised learning:

- Unique output (correct answer aka label) for each input.
- Input samples are independent.

Supervised learning

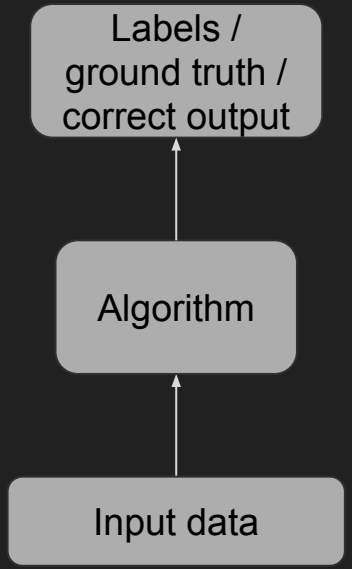
Goal: learn a mapping from data to labels.



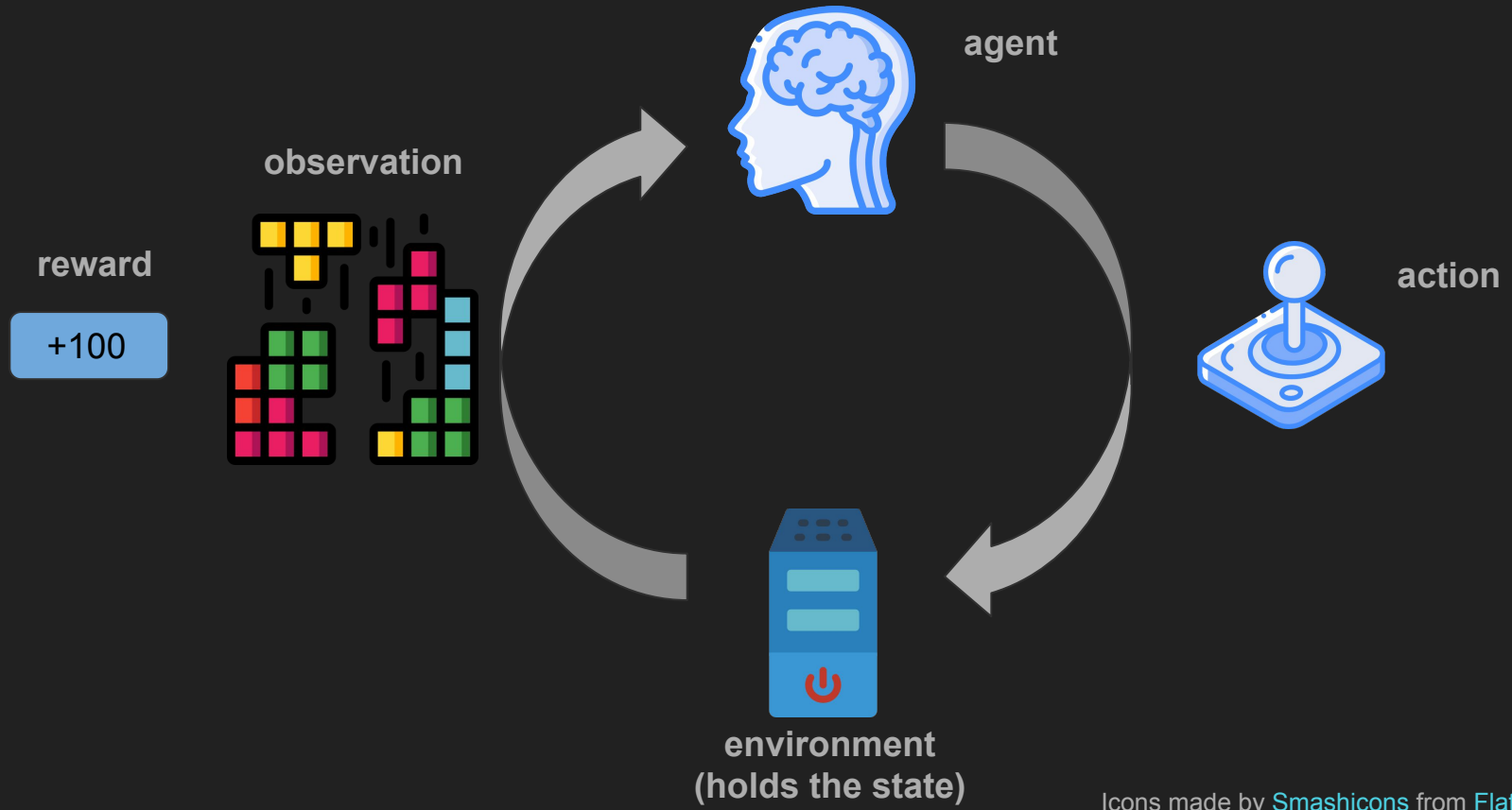
Dog or cat?

/ˌgʊd ˈmɔː.nɪŋ/

Speech to phonemes



Reinforcement learning



Icons made by Smashicons from Flaticon

Reinforcement learning success stories

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE

At last — a computer program that can beat a champion Go player PAGE 484

ALL SYSTEMS GO

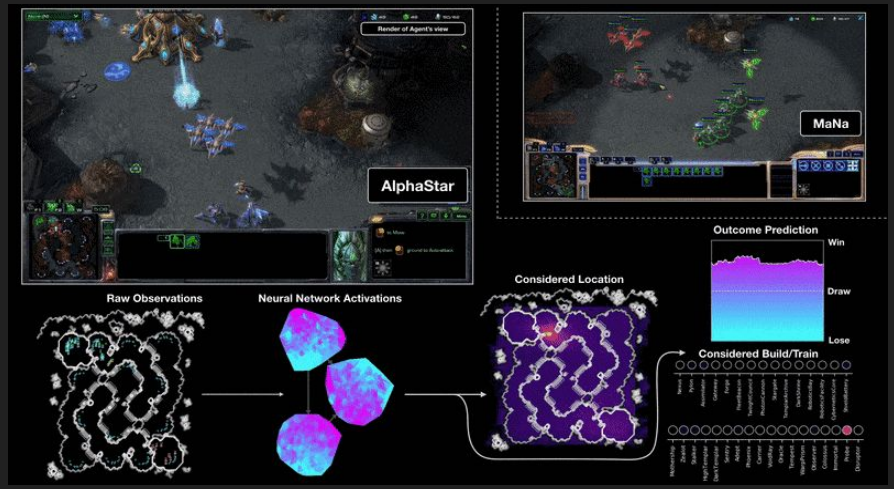
© AP Photo/Lee Jin-man

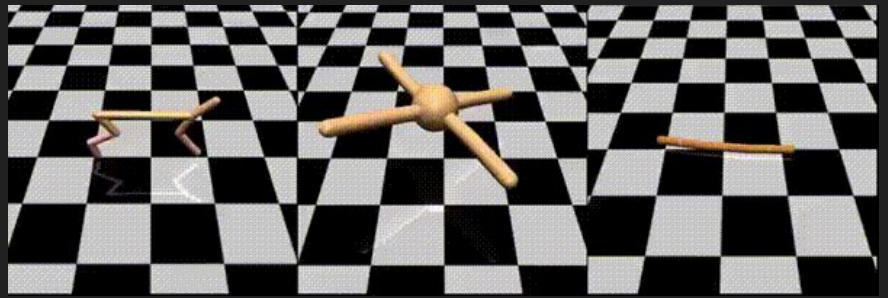
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452

RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459

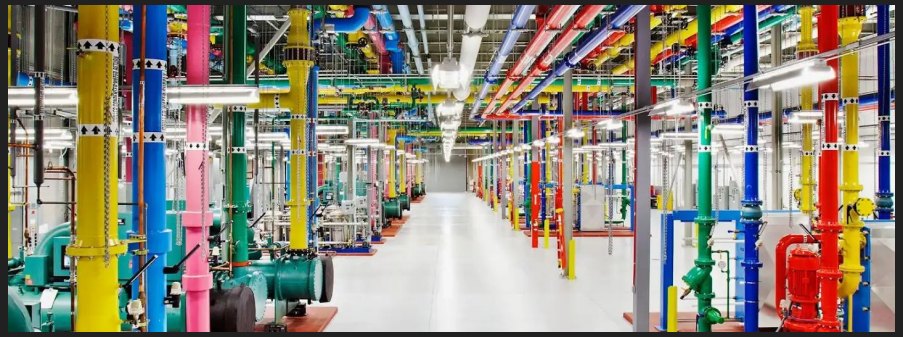
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card forty years on
PAGE 462

NATURE.COM/NATURE
28 January 2016 £10
Vol. 529, No. 7587
9 770028 083095 04



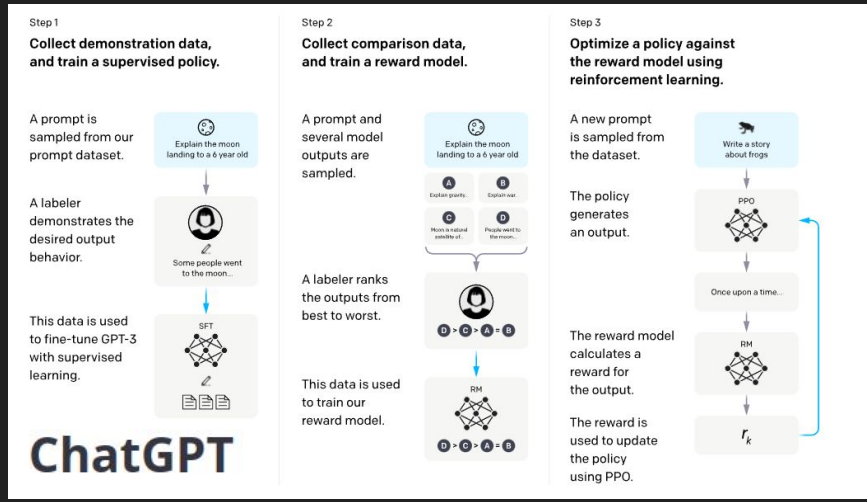


Images from Atari games, Starcraft, Dota 2, Mujoco



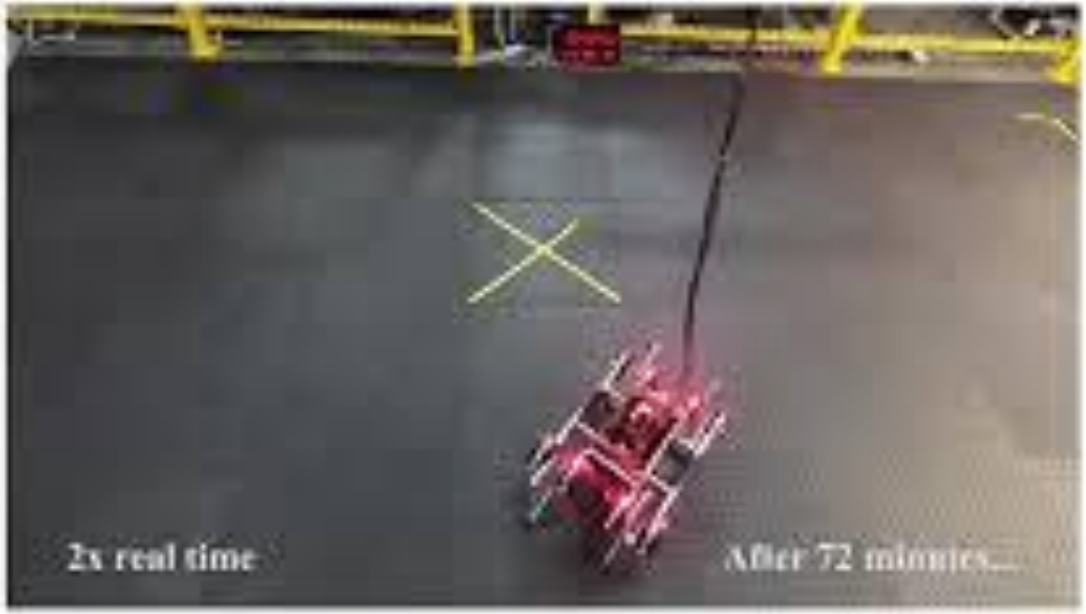
Datacenter cooling

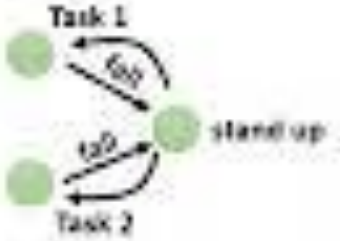
Robot farm



RLHF

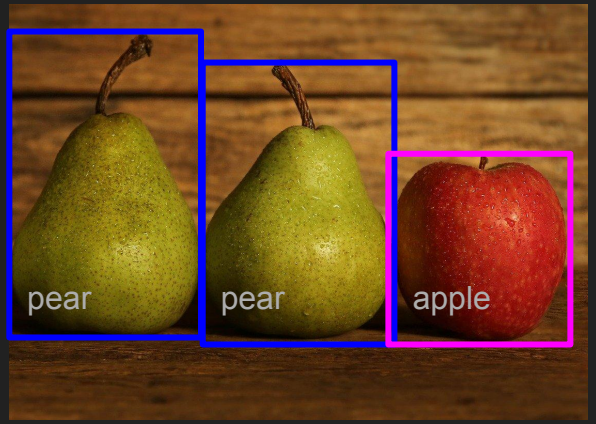
Making it more autonomous…



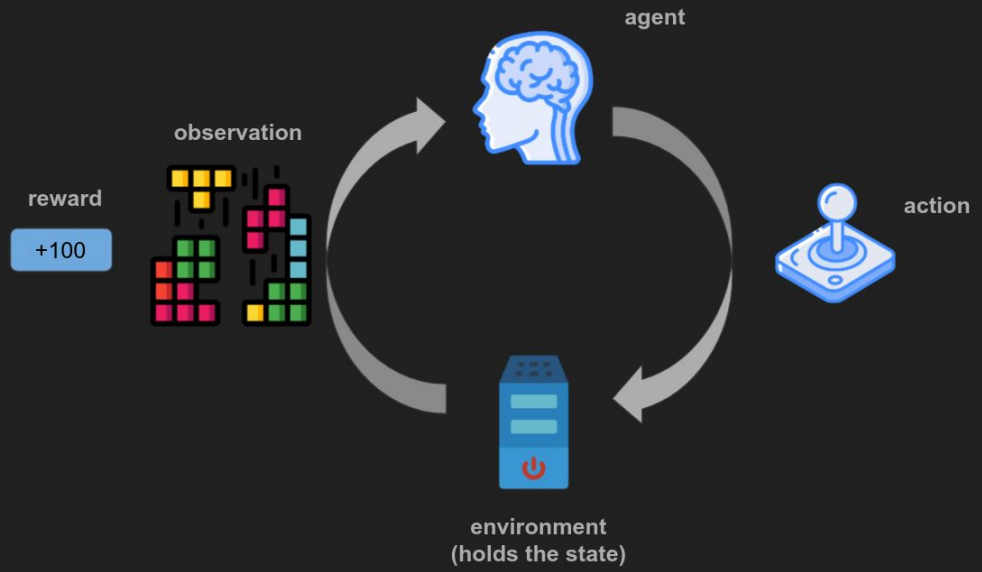


RL vs supervised learning

From single prediction to multi-step control



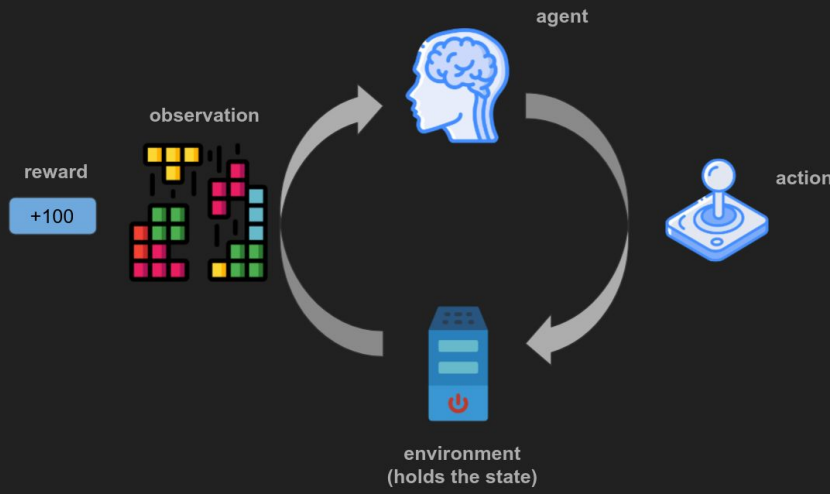
Predictions do not influence future inputs.

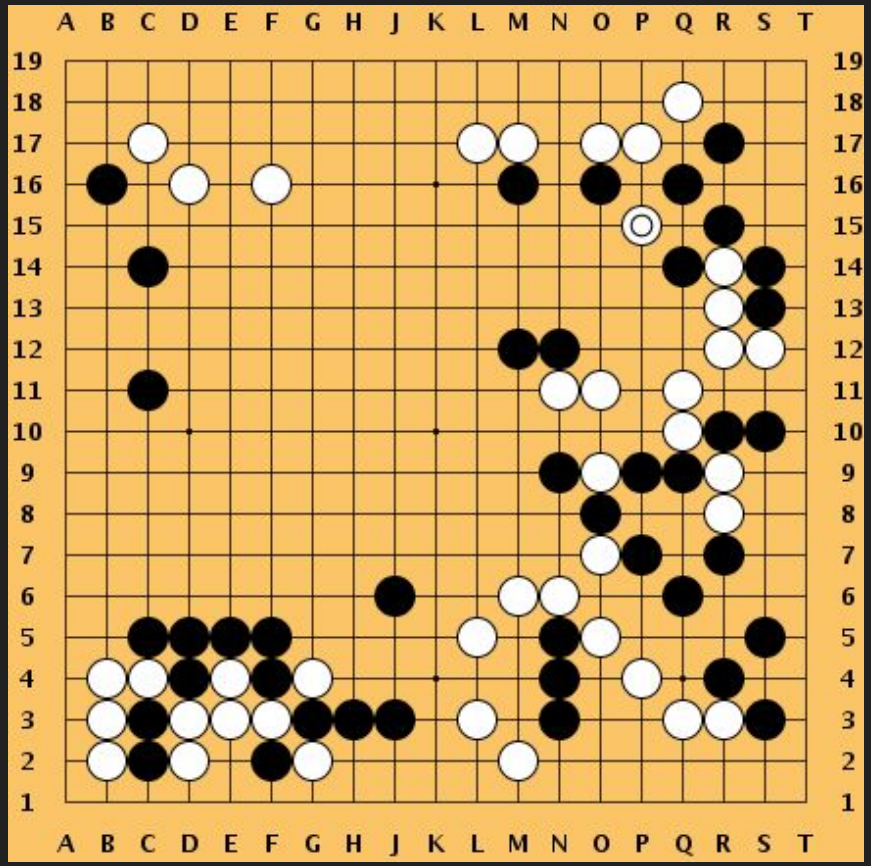


Predictions do influence future inputs.

What is the problem?

- Data is non-i.i.d.
  (current actions affect future observations)
- Sequence of actions
  (delayed feedback, credit assignment).
- Gradient of an arbitrary objective function
  (no supervision, only reward signal)
- Environment is stochastic.





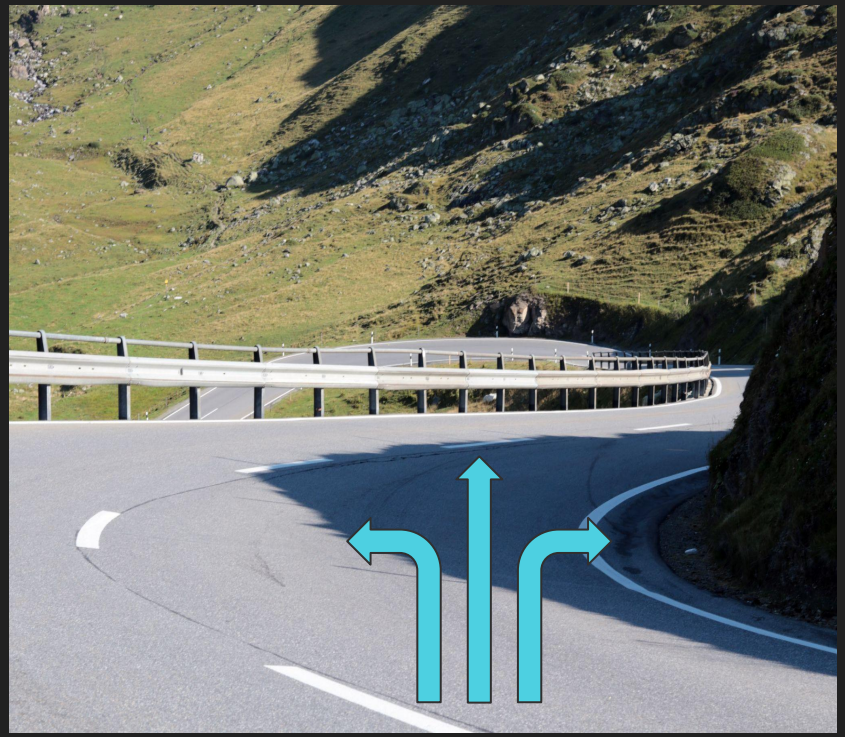
Agent: player.

State: positions of all stones.

Observation: positions of all stones.

Action: placing a stone.

Reward: +1 for winning, -1 for losing.



Agent: driver.

State: state of the world.

Observation: road view, dashboard.

Action: steering wheel, pedals.

Reward: +1 when arriving safely.

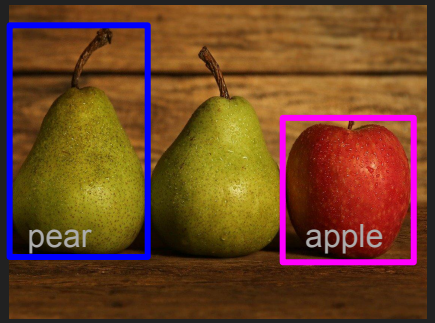
Image by S. Hermann & F. Richter from Pixabay

Supervised learning as a 1-step RL

agent = trained model

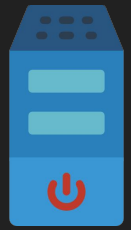
Observation = image

action = prediction



0.75

reward =
Mean average precision



Rest of the course

The next 3 lectures will touch on:

- Q-learning
- Policy gradients
- Alpha zero
- RL from Human Feedback
  - the key step from GPT-3 to ChatGPT

Deep Q-learning (DQN)

Approximate dynamic programming

Mnih et al. *Human-level control through deep reinforcement learning, Nature 518, 529–533 (February 2015, arxiv version Dec 2013)*

Gambler problem (Sutton & Barto, example 4.3)

- Gambler bets on coin flips. Probability of heads is p_h.
- In each round gambler decides on how much to bet (integral number of $) - x.
- Heads → he wins x$, tails → he loses x$.
- The game ends when the gambler has ≥ 100$ (he wins), or when the gambler loses all the money.

Gambler problem (Sutton & Barto, example 4.3)

- Gambler bets on coin flips. Probability of heads is p_h.
- In each round gambler decides on how much to bet (integral number of $) - x.
- Heads → he wins x$, tails → he loses x$.
- The game ends when the gambler has ≥ 100$ (he wins), or when the gambler loses all the money.

Dynamic programming (DP):

$V_k(s)$ - chances of winning (reaching 100$) in at most k turns, having s$ currently

slido

Which recursive formula is correct?

ⓘ Start presenting to display the poll results on this slide.

Gambler problem (Sutton & Barto, example 4.3)

- Gambler bets on coin flips. Probability of heads is p_h.
- In each round gambler decides on how much to bet (integral number of $) - x.
- Heads → he wins x$, tails → he loses x$.
- The game ends when the gambler has ≥ 100$ (he wins), or when the gambler loses all the money.

Dynamic programming (DP):

$V_k(s)$ - chances of winning (reaching 100$) in at most k turns, having s$ currently

$V_k(s) = \max_{i=0,\dots,s} p_h V_{k-1}(s+i) + (1-p_h)V_{k-1}(s-i)$

$V_0(s) = 1$ if $s \geq 100$, 0 otherwise.

Value function

For a state s define:

- $V^*(s)$ - sum of rewards when starting from state s and acting optimally (how many points can I score **from now** till the end of the game)
- $V_k^*(s)$ - as above but for playing k turns

Current score **does not** count towards V!

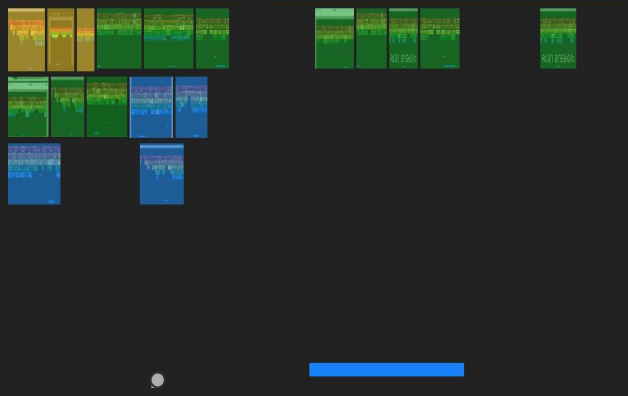
slido

Q: Which state has a higher V-value?

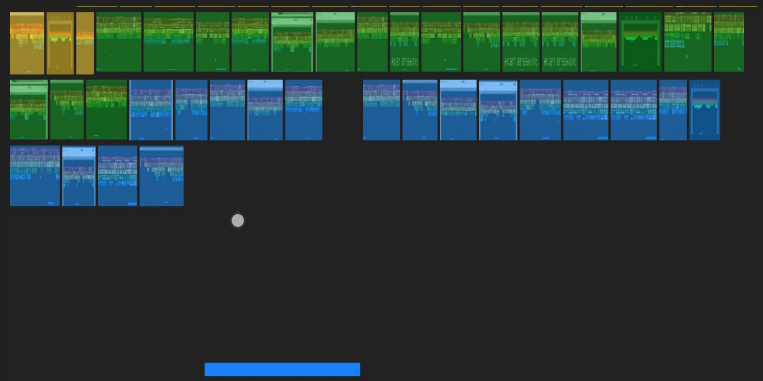
ⓘ Start presenting to display the poll results on this slide.

Value function

Score: 100



Score: 10



Q: Which state has a higher V-value?

Value function

$$V_k^*(s) = max_\pi \mathbb{E}[\sum_{t=0}^{k-1} R(s_t, a_t, s_{t+1}) | \pi, s_0 = s]$$

selecting policy

reward

Conditioning on the policy and initial state

Value iteration (Bellman update)

Start with $V_0(s) = 0$ for each state s.
For k = 1, …, H:
    For each state s:

$$V_k^*(s) = max_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + V_{k-1}^*(s'))$$

selecting action

Immediate reward

Look-up for the remaining k-1 steps

Discount factor

- Immediate reward of x is worth more than reward of x in the future
- Without discounting infinite total reward is possible
- $\gamma < 1$ (think 0.99)

$$V_k^*(s) = max_\pi \mathbb{E}[\sum_{t=0}^{k-1} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s)]$$

$$V^*(s) = max_\pi \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s)]$$

discounting

Value iteration (with discounts)

Start with $V_0(s) = 0$ for each state s.
For k = 1, …, H:
    For each state s:

$$V_k^*(s) = max_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + \gamma V_{k-1}^*(s'))$$

discounting

Theorem. *Value iteration converges. At convergence, we have found the optimal value function V* for the discounted infinite horizon problem, which satisfies the Bellman equations*

$$\forall S \in S: \quad V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$\lim_{k\to\infty} V_k^*(s) = V^*(s)$$

source

Policy

Assume we have V*, but how to play the game?

$$\pi^*(s) = argmax_a \sum_{s'} P(s'|s,a) \cdot (R(s,a,s') + \gamma V^*(s'))$$

Requires knowing the game model
(transition probabilities, rewards, etc.)

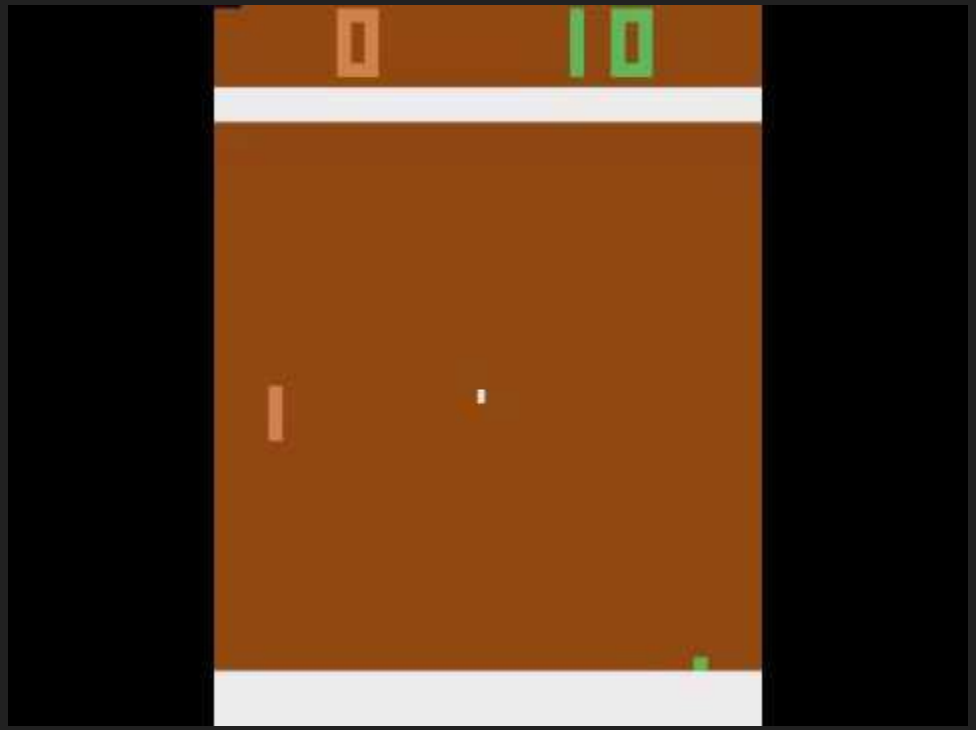
Q-learning

Q-values (action values)

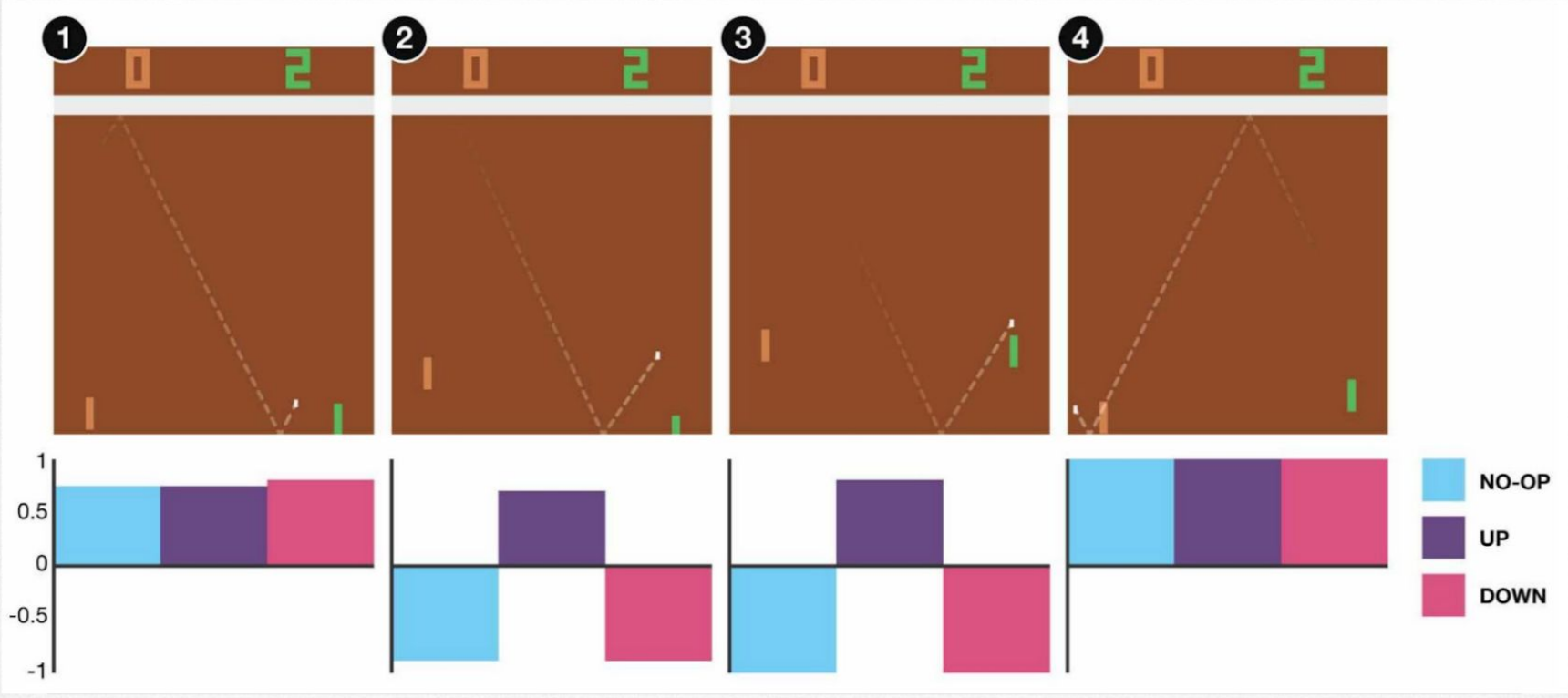
- Q*(s,a) - expected discounted reward, when starting in s, making action a and then playing optimally
- If we have Q-values, we can play the game!

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Pong



Action Values on Pong



source

V*(s')

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

source

Tabular Q-learning - TODO: k no longer means time horizon, just iter number

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

    $$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

  - Incorporate the new estimate into a running average:

    $Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha [\text{target}(s')]$

source

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \cdot \text{target}(s')$$

Q-learning converges, if:

- Each state is visited infinitely often
- Eventually the learning rate gets very small, but not too quickly

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$$

In practice

- Use ε-greedy exploration strategy:
  - In a given state, with probability ε act randomly
  - With probability (1-ε) act greedily (according to the current estimates)
  - Restart after an episode finishes

- Anneal the learning rate

Basic Q-learning algorithm

Initialize $Q(s, a)$ for all s, a.

Get initial state s

Repeat until convergence

  Sample action a, get next state s'

  If s' is terminal:

    target $= R(s, a, s')$

    Sample new initial state s'

  else:

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

  $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \cdot$ target

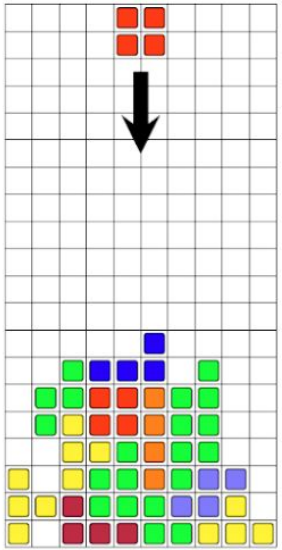
  $s \leftarrow s'$

Basic Q-learning visualization

Link to google drive: https://drive.google.com/file/d/1xFjUAYpsq1GDiI-1WxdPdBFrf0R8wAA2/view?usp=share_link

Approximate Q-learning

Too many states for tabulation



Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

Generalization across states

- Typically we can’t afford keeping a table of all states
- Therefore, the goal is to generalize:
  - Learn on a large enough sample of training states
  - Generalize to unseen states

Approximation by neural networks

Approximate Q-learning

New update rule that updates the current parameters Θ_k to new ones: Θ_{k+1}

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_{\Theta_k}(s', a')$$

$$\Theta_{k+1} = \Theta_k - \alpha \nabla_\Theta [\frac{1}{2}(Q_\Theta(s, a) - \text{target}^2]|_{\Theta=\Theta_k}$$

Connection to tabular Q-learning

- Assume we have a parameter for every entry in the table.
- Analyze the gradient of MSE.

$$\theta \in \mathbb{R}^{|S|\times|A|}, \quad Q_\theta(s,a) = \theta_{sa}$$

$$\nabla_{\theta_{sa}} \left[\frac{1}{2}(Q_\theta(s,a) - \text{target})^2\right]$$

$$= \nabla_{\theta_{sa}} \left[\frac{1}{2}(\theta_{sa} - \text{target})^2\right]$$

$$= \theta_{sa} - \text{target}$$

Connection to tabular Q-learning

- Combine with the update rule.

$$\nabla_{\theta_{sa}} \left[\frac{1}{2}(Q_\theta(s,a) - \text{target})^2\right]$$
$$= \theta_{sa} - \text{target}$$

$$\theta_{sa} \leftarrow \theta_{sa} - \alpha(\theta_{sa} - \text{target})$$
$$= (1-\alpha)\cdot\theta_{sa} + \alpha \cdot \text{target}$$

- Compare to tabular Q-learning update.

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot \text{target}$$

Approximate Q-learning

- Instead of a table, use a parameterized function $Q_\theta(s, a)$
- Can be a linear function in hand-designed features:

$$Q_\theta(s, a) = \Theta_0 f_0(s, a) + \Theta_1 f_1(s, a) + \cdots + \Theta_n f_n(s, a)$$

- Or a large neural network

DQN

DQN

- High-level idea - **make Q-learning look like supervised learning**.
- Two main ideas for stabilizing Q-learning.
- Apply Q-updates on batches of past experience instead of online:
  - **Experience replay** (Lin, 1993).
  - Previously used for better data efficiency.
  - Makes the data distribution more stationary.
- Use an older set of weights to compute the targets (**target network**):
  - Keeps the target function from changing too quickly.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

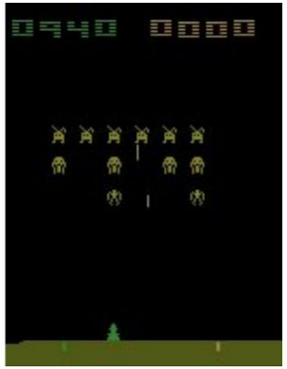
source

Target Network Intuition

- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

s



s'

DeepMind

source

DQN Training Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

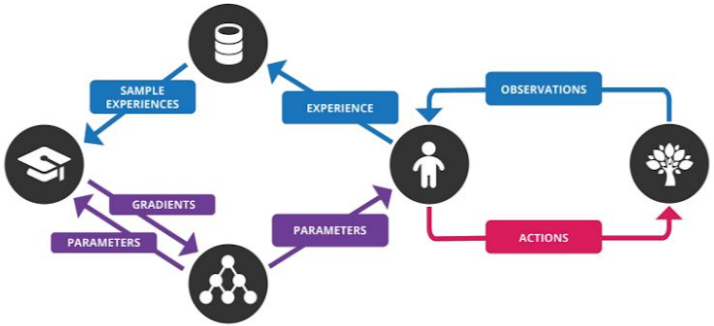
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

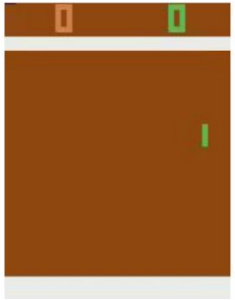
End For



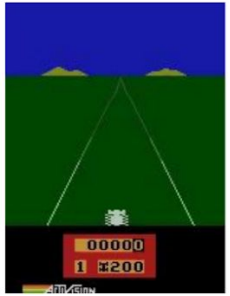
DeepMind

source

DQN on ATARI

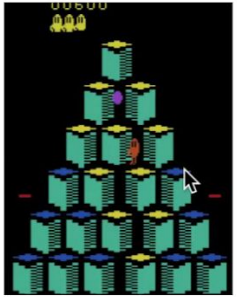


Pong



Enduro

Beamrider



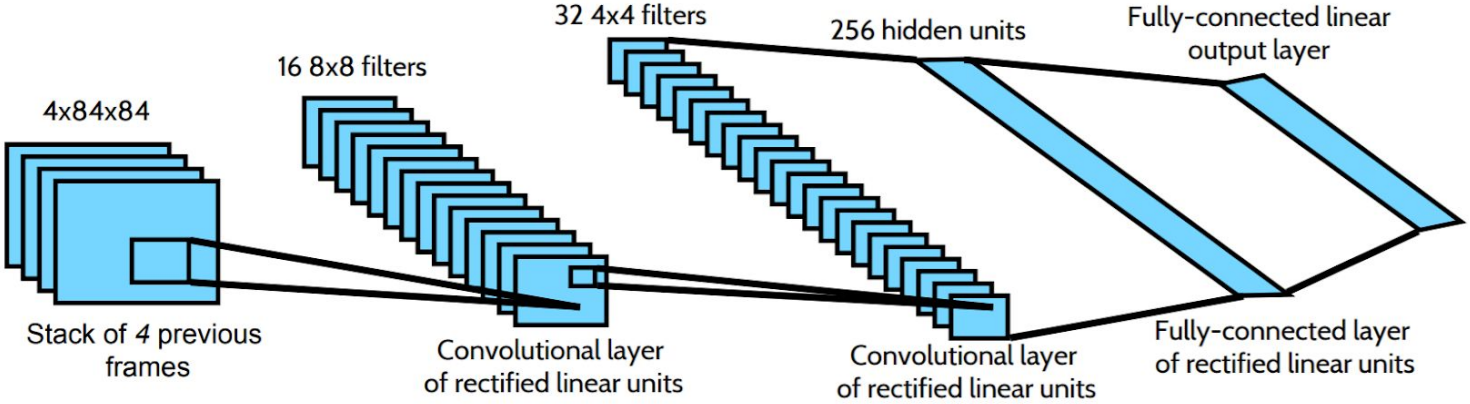
Q*bert

- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

source

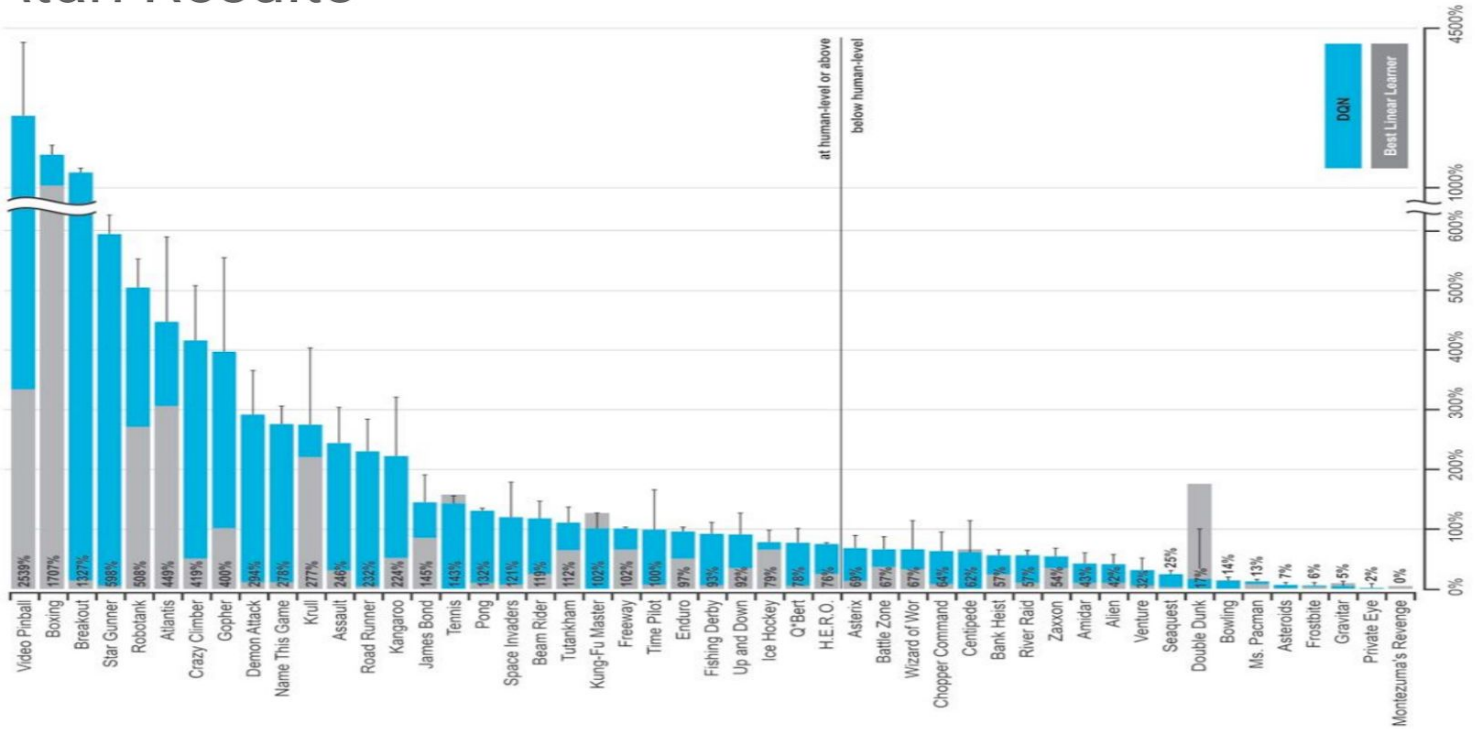
ATARI Network Architecture

- Convolutional neural network architecture:
  - History of frames as input.
  - One output per action - expected reward for that action $Q(s, a)$.
  - Final results used a slightly bigger network (3 convolutional + 1 fully-connected hidden layers).



source

Atari Results



"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

source

Rest of the course

The next 3 lectures will touch on:

- Q-learning (done)
- Policy gradients
- Alpha zero
- RL from Human Feedback
  - the key step from GPT-3 to ChatGPT

Feedback is a gift

https://tinyurl.com/dnn-2025-12-17