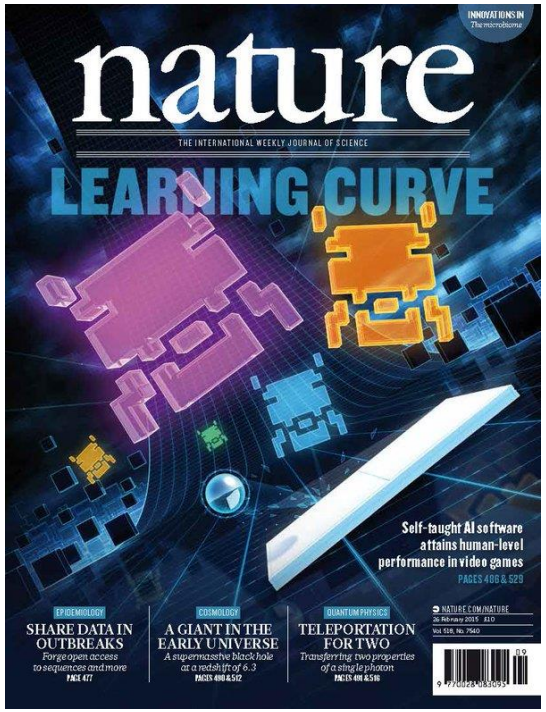
CSCE-642 Reinforcement Learning

Deep Q-learning



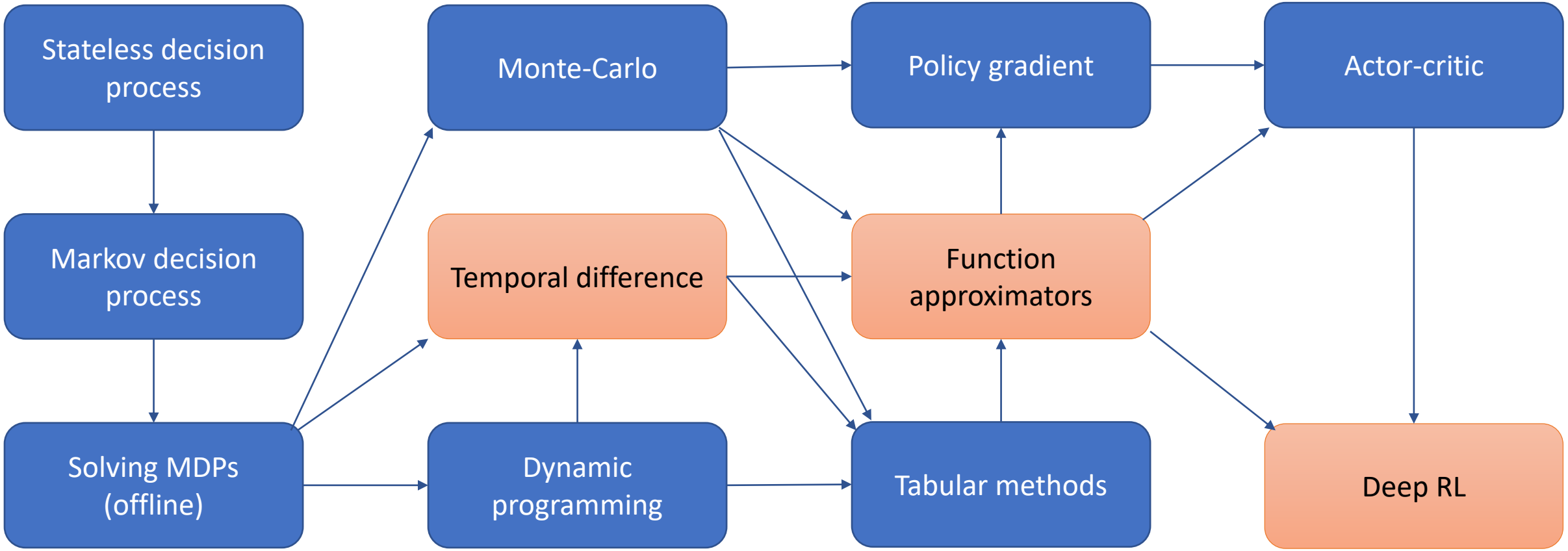
Instructor: Guni Sharon

Successfully address the deadly triad

- Danger of instability and divergence arises whenever we combine all of the following three elements
- **Function approximation**: A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources
- **Bootstrapping**: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods)
- **Off-policy training**: Training on a distribution of transitions other than that produced by the target policy.

* Divergence can always occur if the learning rate is set too high

CSCE-689, Reinforcement Learning

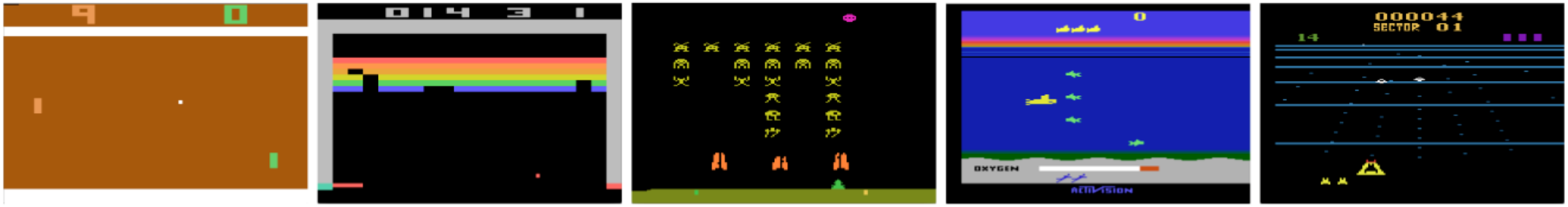


Mnih et al. 2015

- First deep learning algorithm to successfully learn control policies directly from high-dimensional sensory input using RL
- The algorithm is a convolutional neural network, trained with a variant of Q-learning
- Input is raw pixels and output is an action-value function estimating future rewards
- Surpassed a human expert on various Atari video games

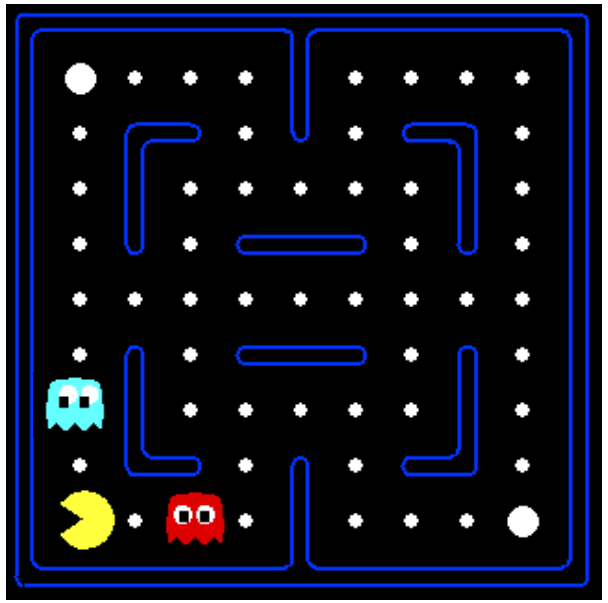
The age of deep learning

- Previous models relied on hand-crafted features combined with linear value functions
  - The performance of such systems heavily relies on the quality of the feature representation
- Advances in deep learning have made it possible to automatically extract high-level features from raw sensory data

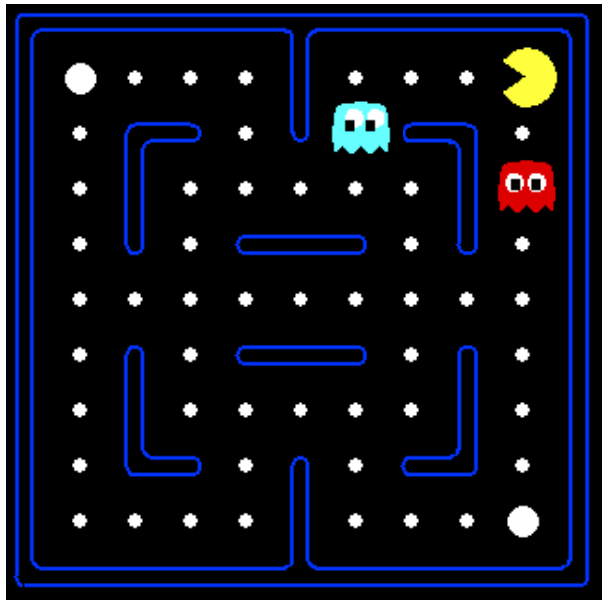


Example: Pacman

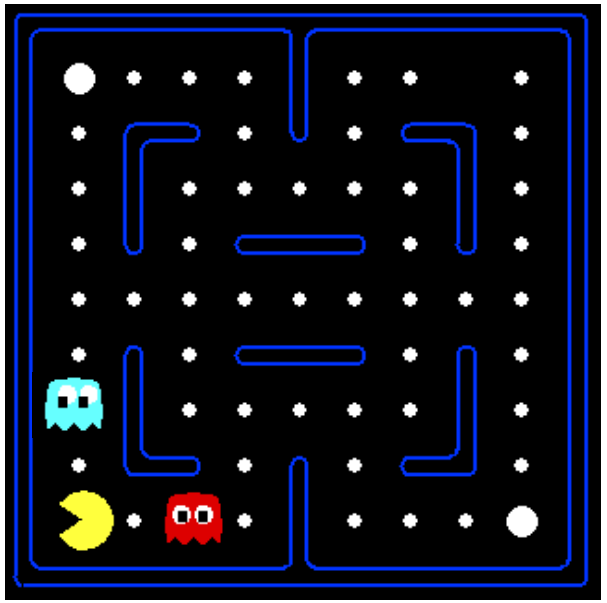
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:

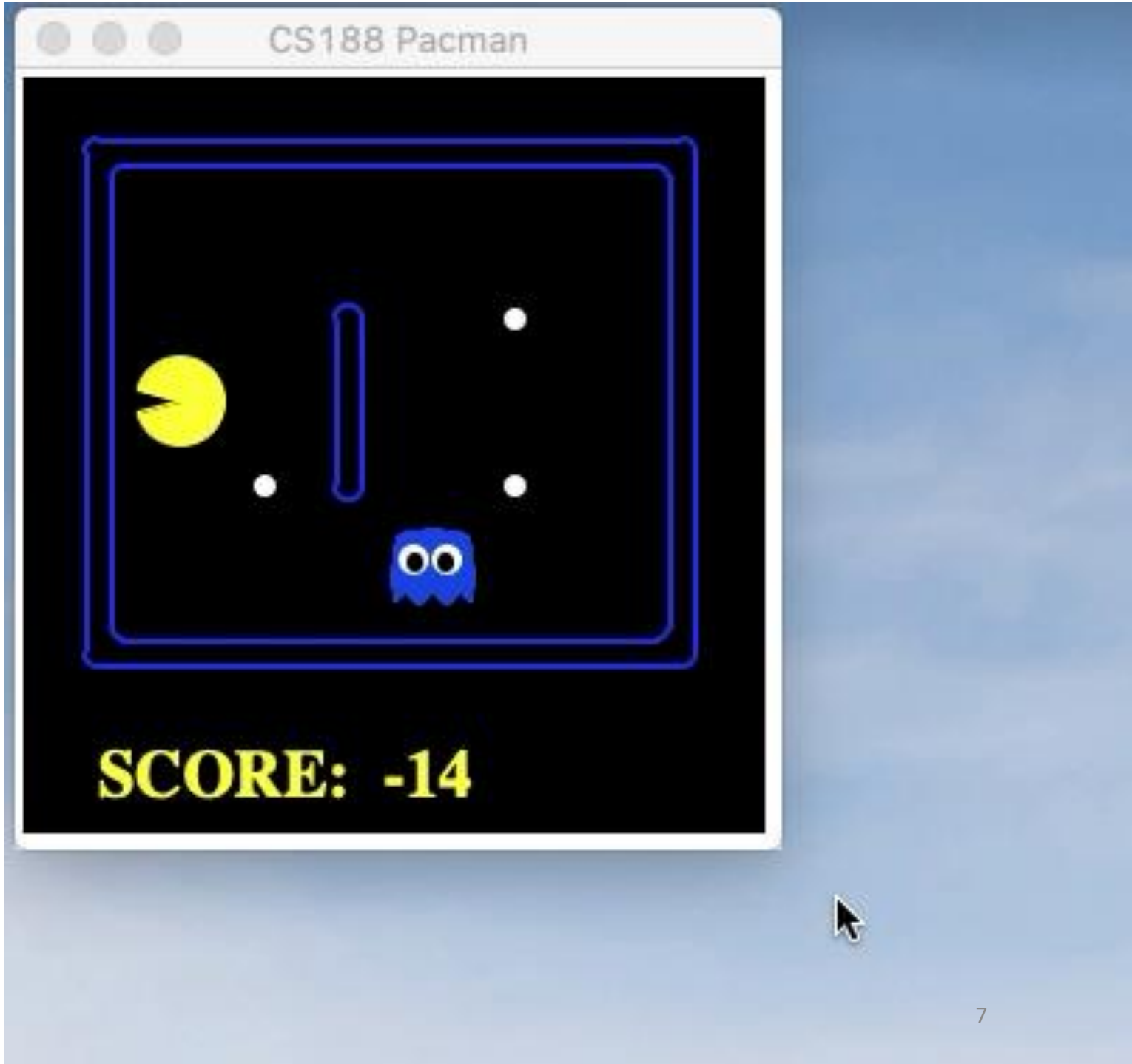


Or even this one!



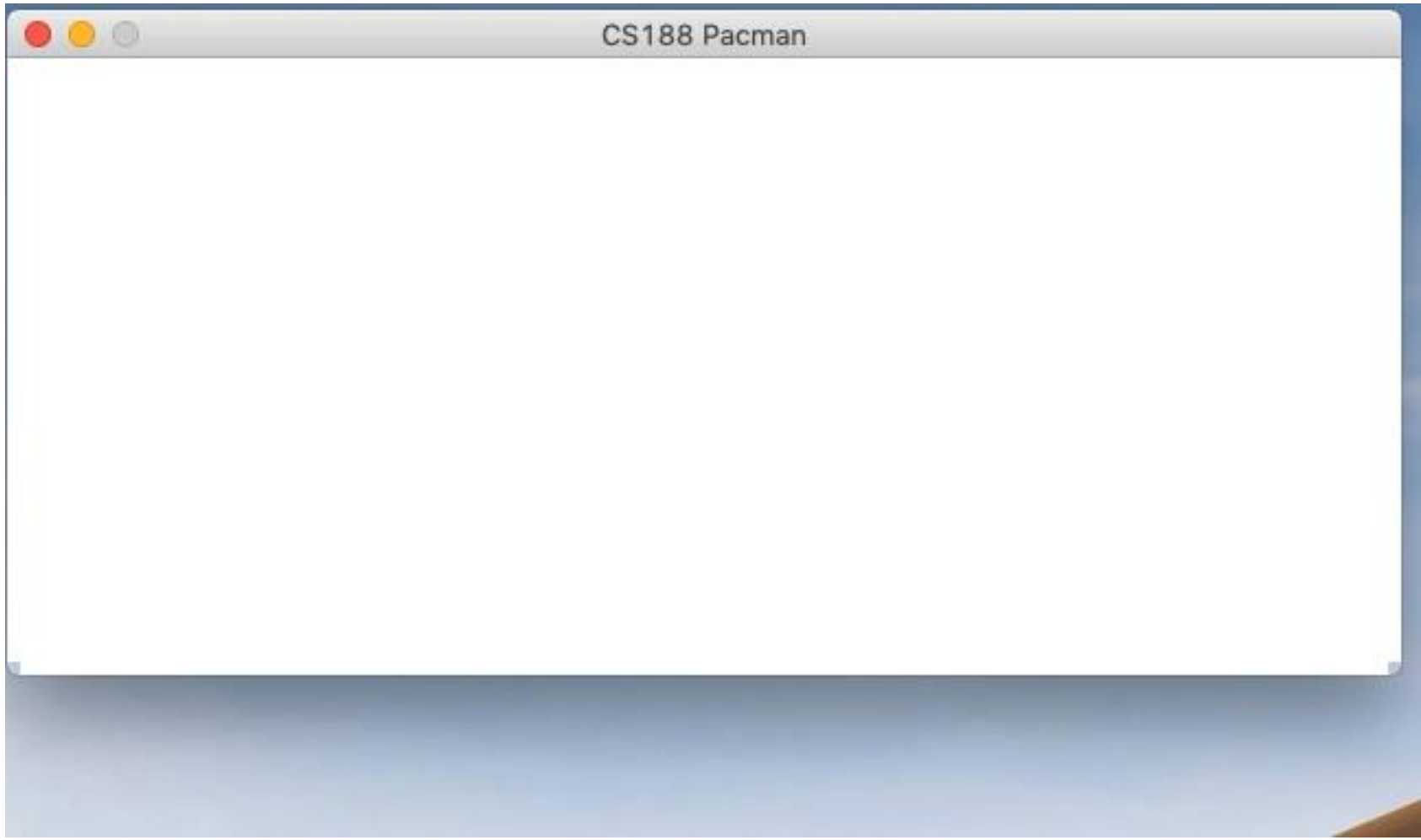
We must generalize our knowledge!

- Naïve Q-learning
- After 50 training episodes



- Generalize Q-learning with function approximator

- Generalizing knowledge results in efficient learning
- E.g., learn to avoid the ghosts



Generalizing with Deep learning

- **Supervised**: Require large amounts of hand-labelled training data
  - **RL** on the other hand, learns from a scalar reward signal that is frequently sparse, noisy, and delayed
- **Supervised**: Assume the data samples are independent
  - In **RL** one typically encounters sequences of highly correlated state
- **Supervised**: Assume a fixed underlying distribution
  - In **RL** the data distribution changes as the algorithm learns new behaviors
- DQN was first to demonstrate that a convolutional neural network can overcome these challenges to learn successful control policies from raw video data in complex RL environments

Deep Q learning [Mnih et al. 2015]

- Trains a generic neural network-based agent that successfully learns to operate in many diverse domains
- The network is not provided with any domain-specific information or hand-designed features
- Must learn from nothing but the raw input (pixels), the reward, terminal signals, and the set of possible actions

Original Q-learning

- What's the problem with using DNNs in Basic Q-learning?

1. Train on correlated (successive states) samples
2. Train on sparse data, i.e., only a small portion of the state space
3. Policy change -> sample distribution shifts

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $\theta = \theta + \alpha\left(R + \gamma \max_a \hat{Q}(S', a; \theta) - \hat{Q}(S, A; \theta)\right)\nabla_\theta \hat{Q}(S, A; \theta)$
    $S \leftarrow S'$
  until S is terminal

Deep Q learning [Mnih et al. 2015]

- DQN addresses problems of correlated data and non-stationary distributions
  - Use an experience replay mechanism
  - Randomly samples and trains on previous transitions
  - Results in a smoother training distribution over many past behaviors

Deep Q learning [Mnih et al. 2015]

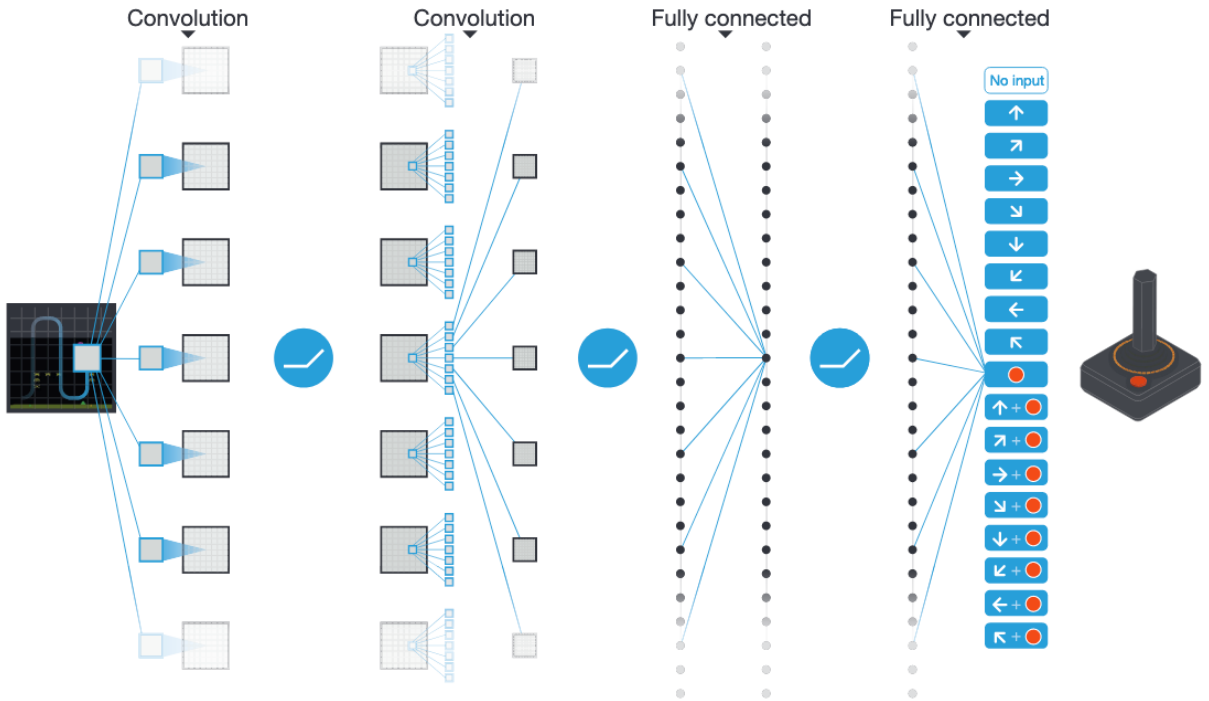
- Learn a function approximator (Q network), $\hat{Q}(s, a; \theta) \approx Q^*(s, a)$
- Value propagation: $Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a')\right]$
  - $\mathcal{E}$ represents the environment (underlying MDP)
- Update $\hat{Q}(s, a; \theta)$ at each step, i, using SGD with squared loss:
- $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[\left(y_i - \hat{Q}(s, a; \theta_i)\right)^2\right]$
  - $y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{i-1})\right]$
  - $\rho(s, a)$ is the behavior distribution, e.g., epsilon greedy
  - θ_{i-1} is considered fix when optimizing the loss function (helps when taking the derivative with respect to θ and with stability)

Deep Q learning [Mnih et al. 2015]

- $L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - \hat{Q}(s,a;\theta_i)\right)^2\right]$

Independent of θ_i (because θ_{i-1} is considered fix)

- $\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim\mathcal{E}}\left[-\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_{i-1}) - \hat{Q}(s,a;\theta_i)\right)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)\right]$



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 $\theta^{-} = \theta$

Initialize the replay memory and two identical Q approximators (CNN). $\hat{Q}$ is our target approximator.

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

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** ← Play *m* episodes (full games)
  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

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

Start episode from x_1 (pixels at the starting screen).
Preprocess the state (include 4 last frames, RGB to grayscale conversion, downsampling, cropping)

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** ← For each time step during the episode
    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

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

With small probability select a random action (explore), otherwise select the, currently known, best action (exploit).

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Execute the chosen action and store the (processed) observed transition in the replay memory

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Experience replay:
Sample a random minibatch of transitions from replay memory and perform gradient decent step on Q (not on $\hat{Q}$)

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Once every several steps set the target function, $\hat{Q}$, to equal Q

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Such delayed online learning helps in practice:
"This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases
the target y_j, possibly leading to oscillations or divergence of the policy" [Human-level control through deep reinforcement learning. Nature 518.7540 (2015): 529.]

Deep Q learning

- *model-free*: solves the reinforcement learning task directly using samples from the emulator $\mathcal{E}$, without explicitly learning an estimate of $\mathcal{E}$
- *off-policy*: learns the optimal policy, $a = \underset{a}{arg\max}\, Q(s, a; \theta)$, while following a different behavior policy
  - One that ensures adequate exploration of the state space

Experience replay

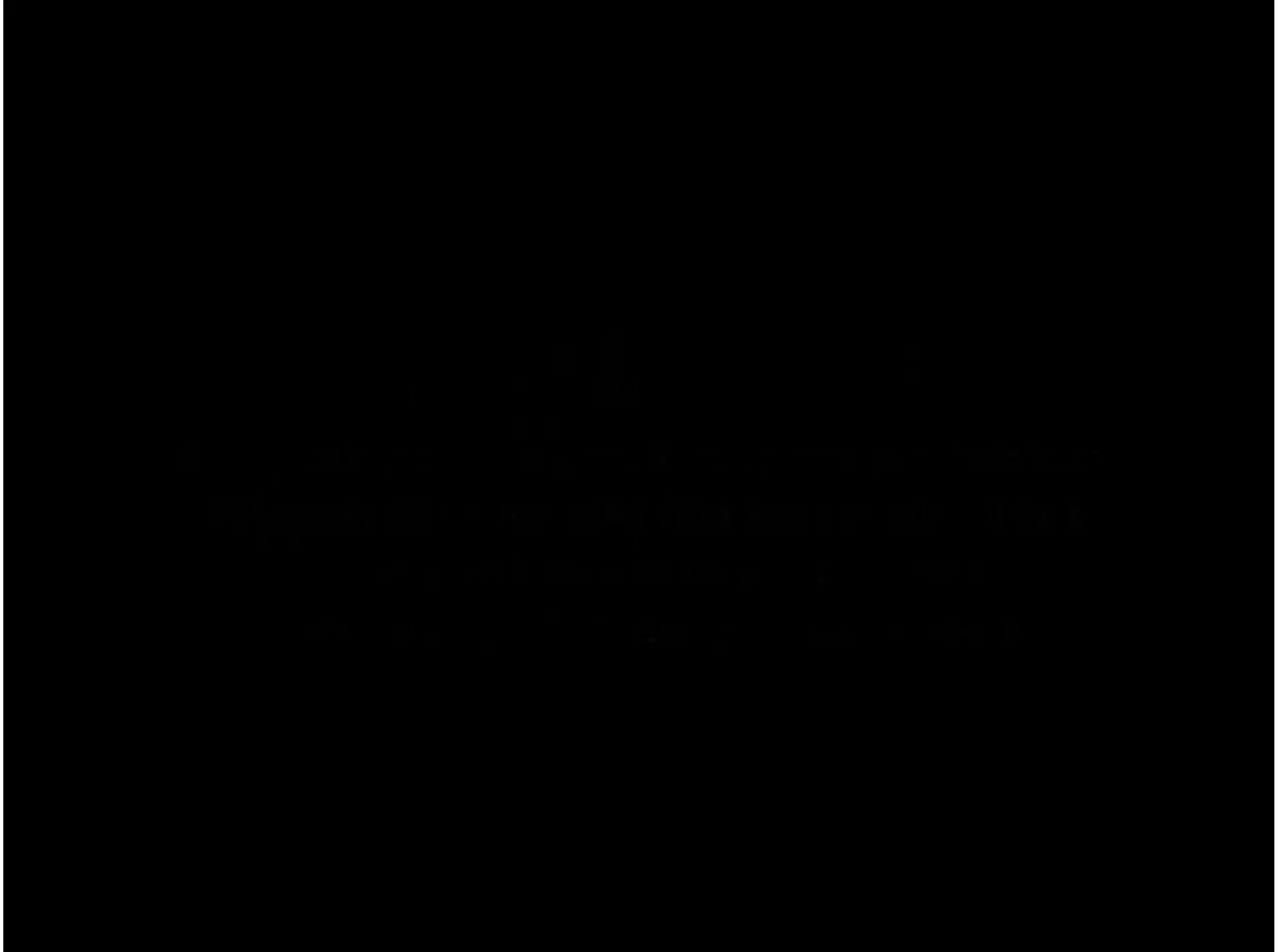
- Utilizing experience replay has several advantages
  - Each step of experience is potentially used in many weight updates, which allows for greater data efficiency
  - Learning directly from consecutive samples is inefficient, due to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates
  - The behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters
- Note that when learning by experience replay, it is necessary to learn off-policy (because our current parameters are different to those used to generate the sample), which motivates the choice of Q-learning

Experience replay

- DQN only stores the last N experience tuples in the replay memory
  - Old transitions are overwritten
- Samples uniformly at random from the buffer when performing updates
- Is there room for improvement?
  - Important transitions?
    - Prioritized replay
  - Prioritize deletions from the replay memory
  - see prioritized experience reply, https://arxiv.org/abs/1511.05952

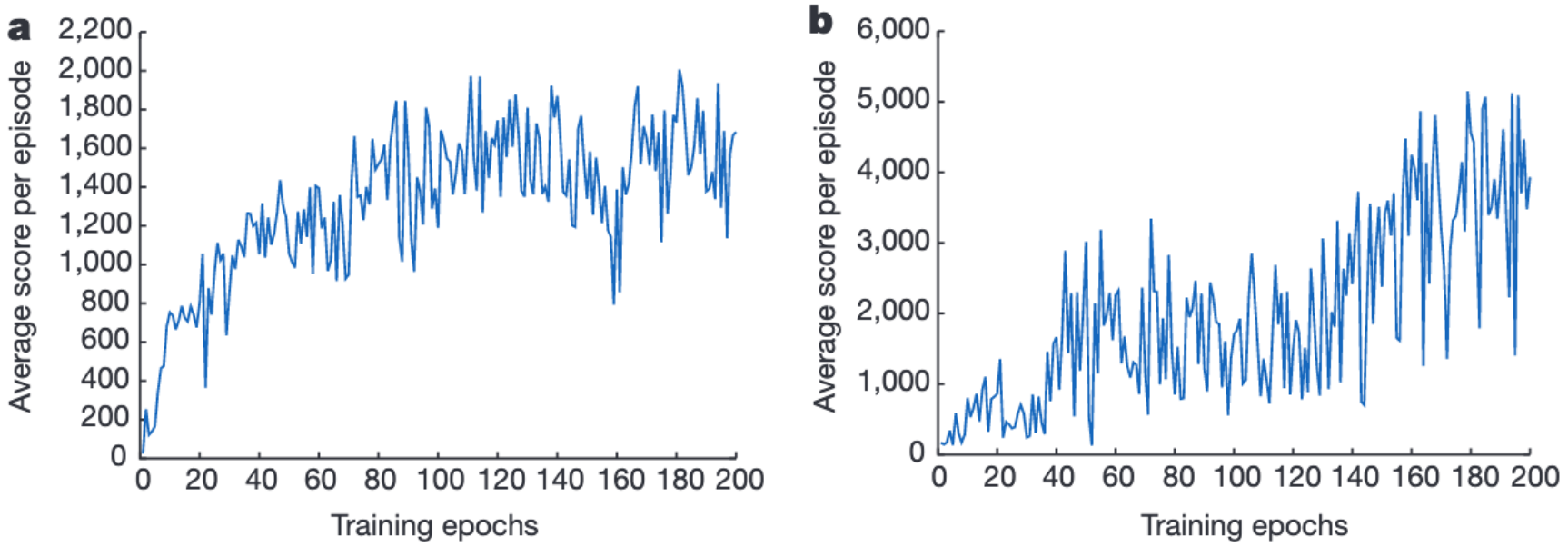
Before training

peaceful swimming



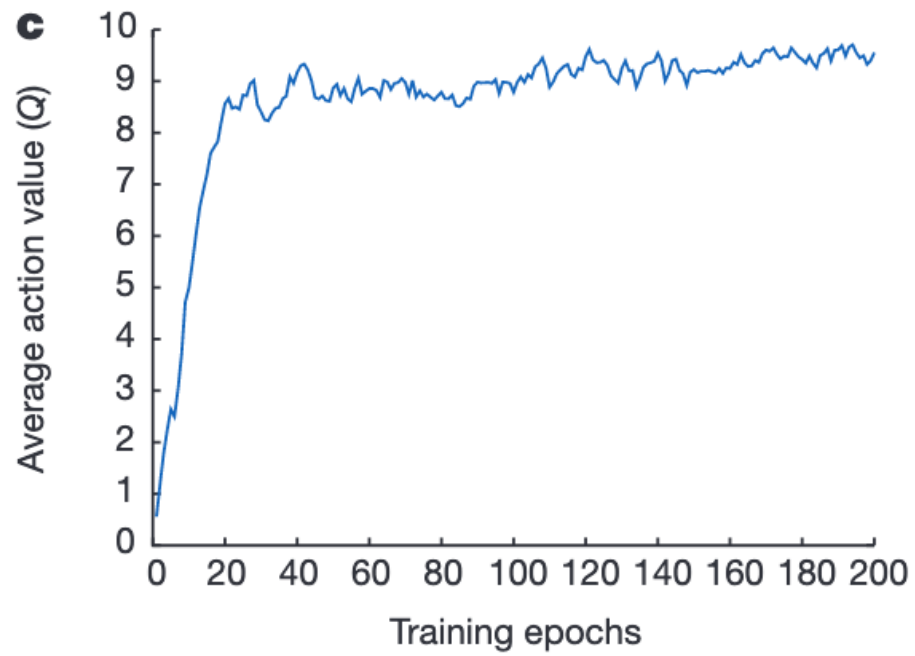
Results: DQN

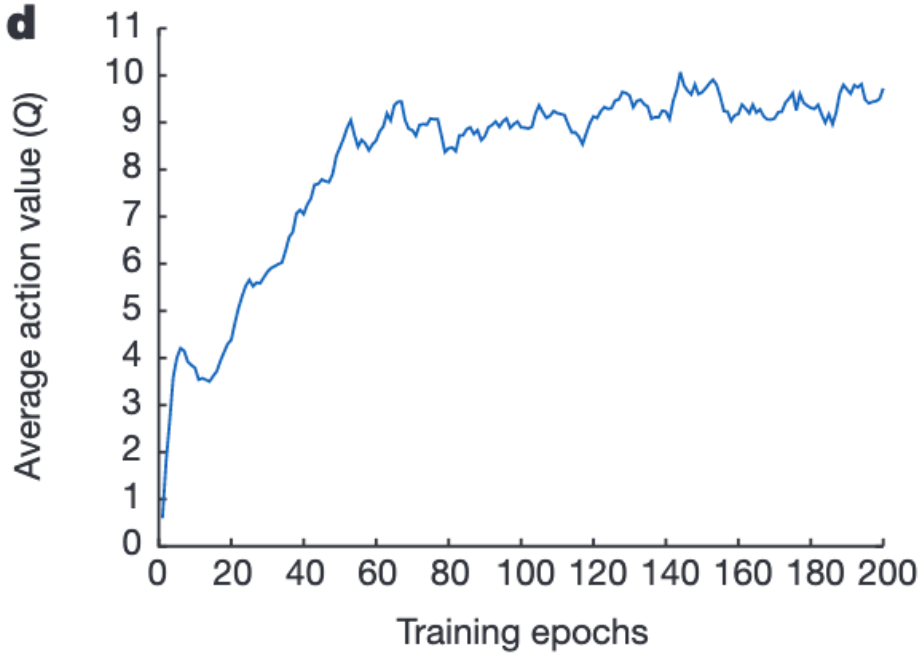
- Each point is the average score achieved per episode after the agent is run with an epsilon-greedy policy ($\varepsilon = 0.05$) for 520k frames on SpaceInvaders (a) and Seaquest (b)

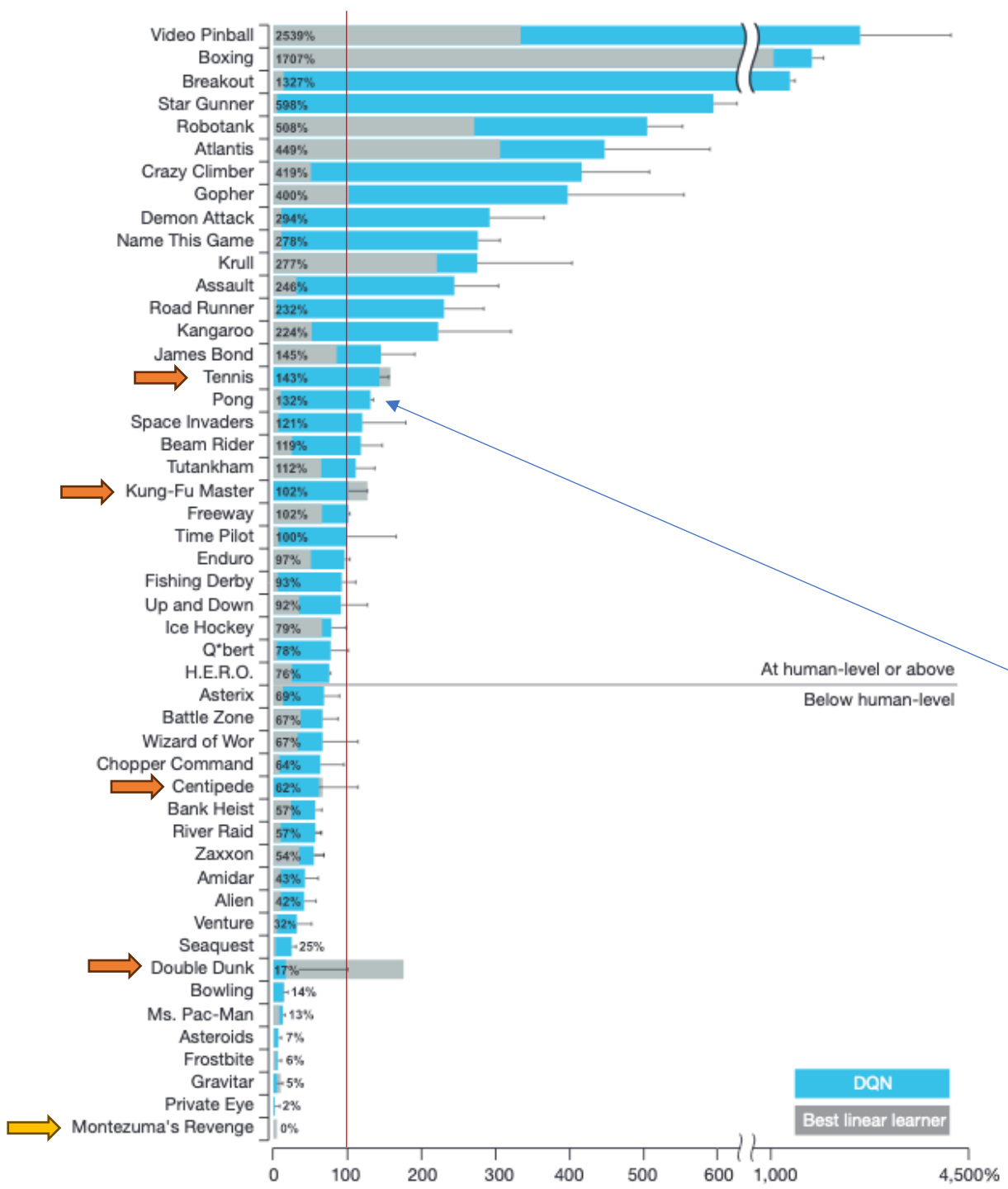


Results: DQN

- Average predicted action-value on a held-out set of states on Space Invaders (c) and Seaquest (d)



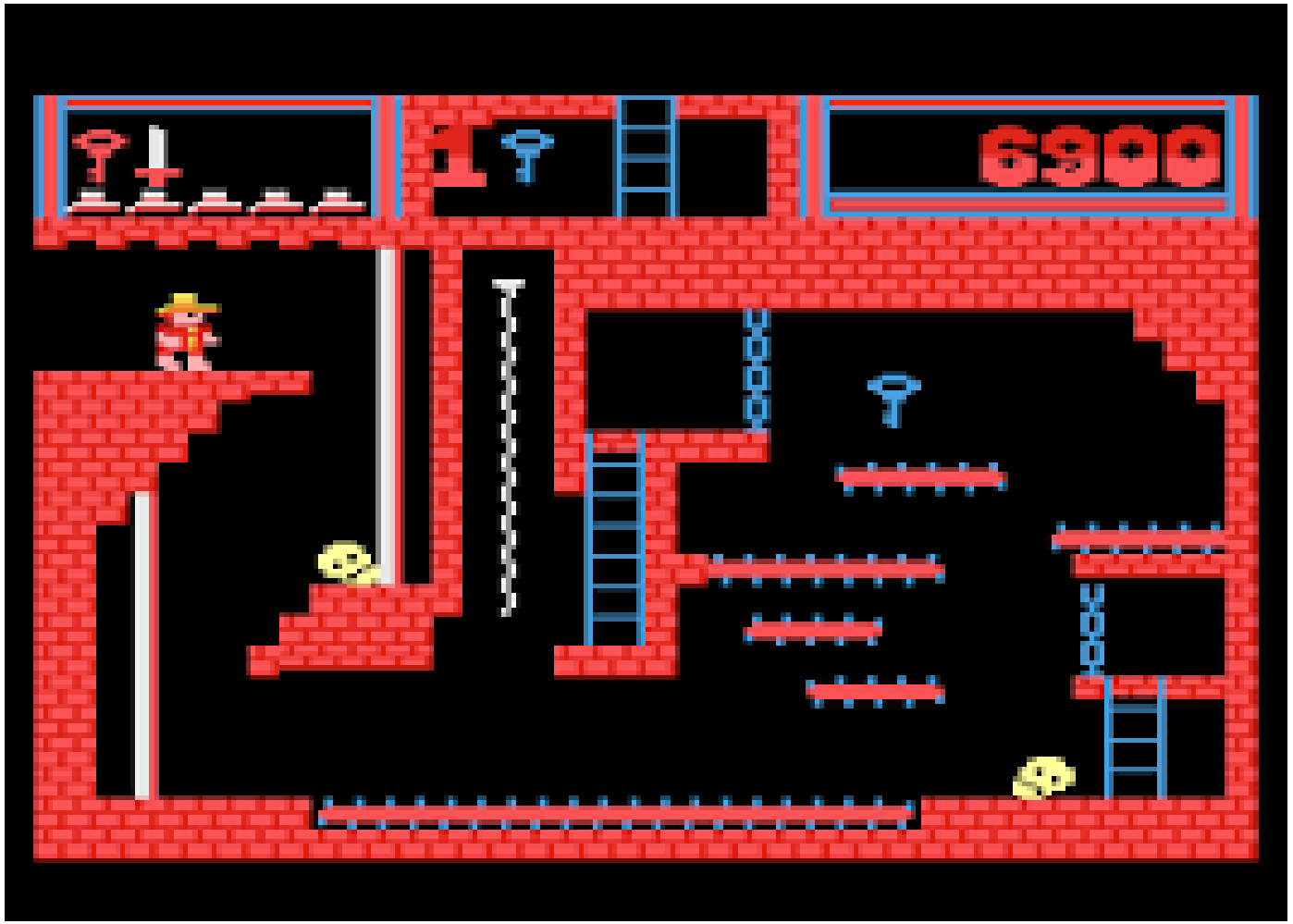




- Normalized between a professional human games tester (100%) and random play (0%)
  - The normalized performance of DQN is: 100x(DQN score-random play score)/(human score-random play score)

Montezuma revenge

- Long-term planning
- Sparse rewards



Maximization bias

- See lecture 5TD_learning.pptx, slide 41
- The max operator in Q-learning uses the same values both to select and to evaluate an action
  - $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{i-1}) \right]$
  - Makes it more likely to select overestimated values, resulting in overoptimistic value estimates
- We can use a technique similar to the previously discussed Double Q-learning (lecture 5TD_learning.pptx, slide 43)
  - Two value functions are trained. Update only one of the two value functions at each step while considering the $\max_a$ from the other

Double Deep Q networks (DDQN)

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 $\theta^{-} = \theta$

We have two available Q networks.
Let's use them for double learning

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

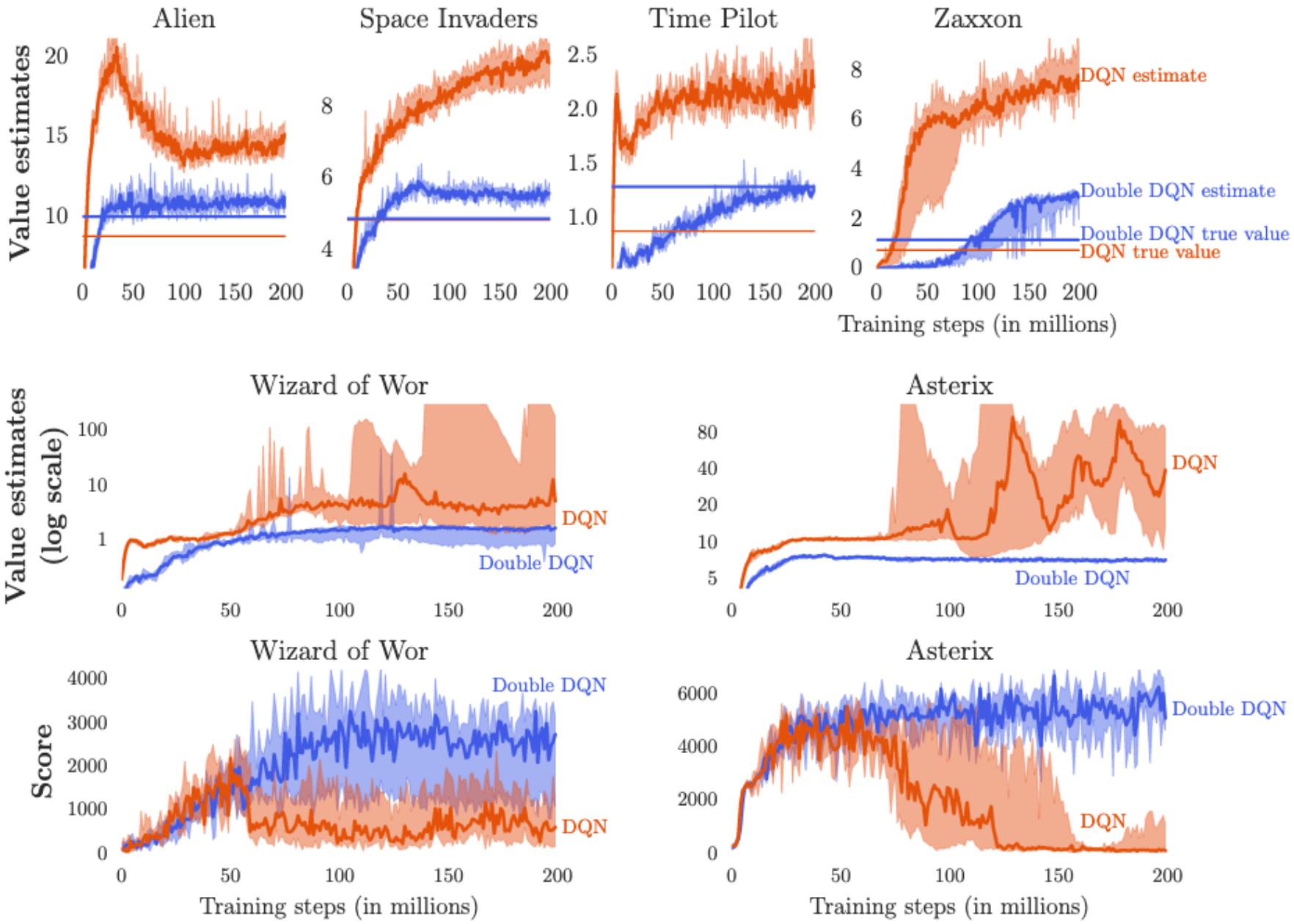
Double Deep Q networks (DDQN)

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

We have two available Q networks.
Let's use them for double learning

$$r_j + \gamma \hat{Q}\left(\phi_{j+1}, \arg\max_{a'} \boldsymbol{Q}(\boldsymbol{\phi_{j+1}}, \boldsymbol{a'}; \boldsymbol{\theta}), \theta^{-}\right)$$

- Hasselt et al. 2015
- The straight horizontal orange (for DQN) and blue (for Double DQN) lines in the top row are computed by running the corresponding agents after learning concluded, and averaging the actual discounted return obtained from each visited state. These straight lines would match the learning curves at the right side of the plots if there is no bias.



What did we learn?

- Using deep neural networks as function approximators in RL is tricky
  - Sparse samples
  - Correlated samples
  - Evolving policy (nonstationary sample distribution)
- DQN attempts to address these issues
  - Reuse previous transitions (reply memory)
  - Randomly sample previous transitions to break correlation
  - Use off-policy, TD(0) learning to allow convergence to the true target values (Q*)
    - No guarantees for non-linear (DNN) approximators

What next?

- **Lecture**: Policy Gradient
- **Assignments**:
  - Deep Q-Learning, by Monday Nov. 4, EOD
- **Quiz (on Canvas)**:
  - Deep Q-Learning, by Nov. 28, EOD
  - Eligibility Traces, by Nov. 21, EOD
- **Project**:
  - Literature survey, by Monday, November 4 EOD