

Advanced Topics in Reinforcement Learning

Lecture 12: Deep Reinforcement Learning II

Josiah Hanna

University of Wisconsin — Madison

Announcements

- Read Chapter 13 for next week. Policy-based RL!
- Upcoming dates:
 - Literature survey due next week: October 23
 - Exam: October 29

Learning Outcomes

After this week, you will be able to:

1. Describe key benefits and challenges of using neural networks as function approximators in RL.
2. Describe how deep learning techniques are used within RL environments.
3. Implement key deep RL techniques such as target networks and experience replay.

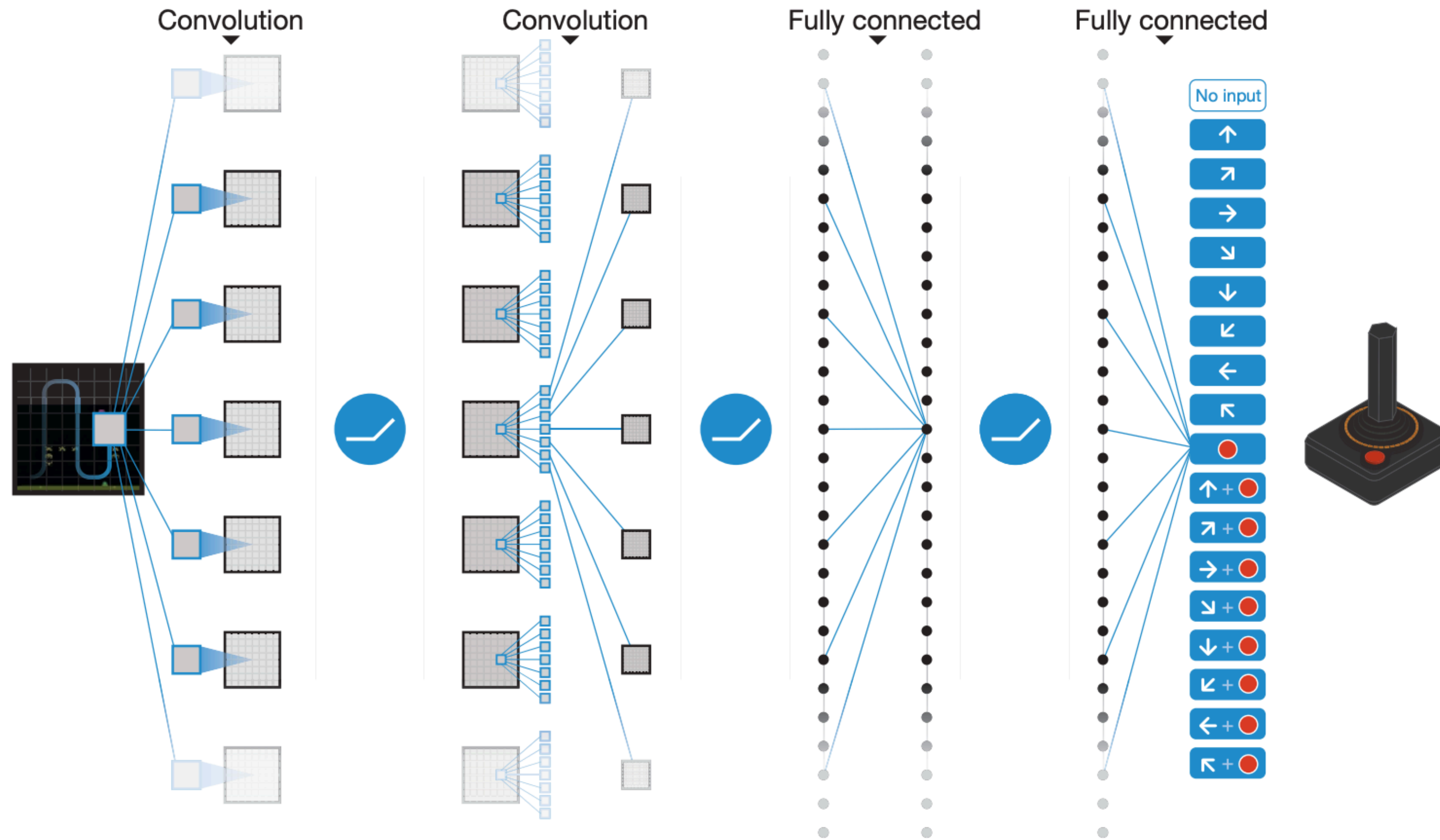
Neural Network Training in RL

- Semi-Gradient Q-learning:
 - $$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)) \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$
- The parameter, $\mathbf{w}_t$, is all weights and biases of the neural network.
- Backpropagation algorithm: use chain rule of calculus to derive gradient of network outputs with respect to each weights or bias of the network.
- Adjust each weight in proportion to gradient of output times TD-error.

DQN Overview

- Single algorithm, neural network architecture, and hyper-parameter setting that played 49 Atari video games at human-level.
- Training and evaluation is independent for each game.
 - The final neural network from training on “Breakout” cannot play “Pong.”
- Landmark result for deep reinforcement learning.

DQN Architecture

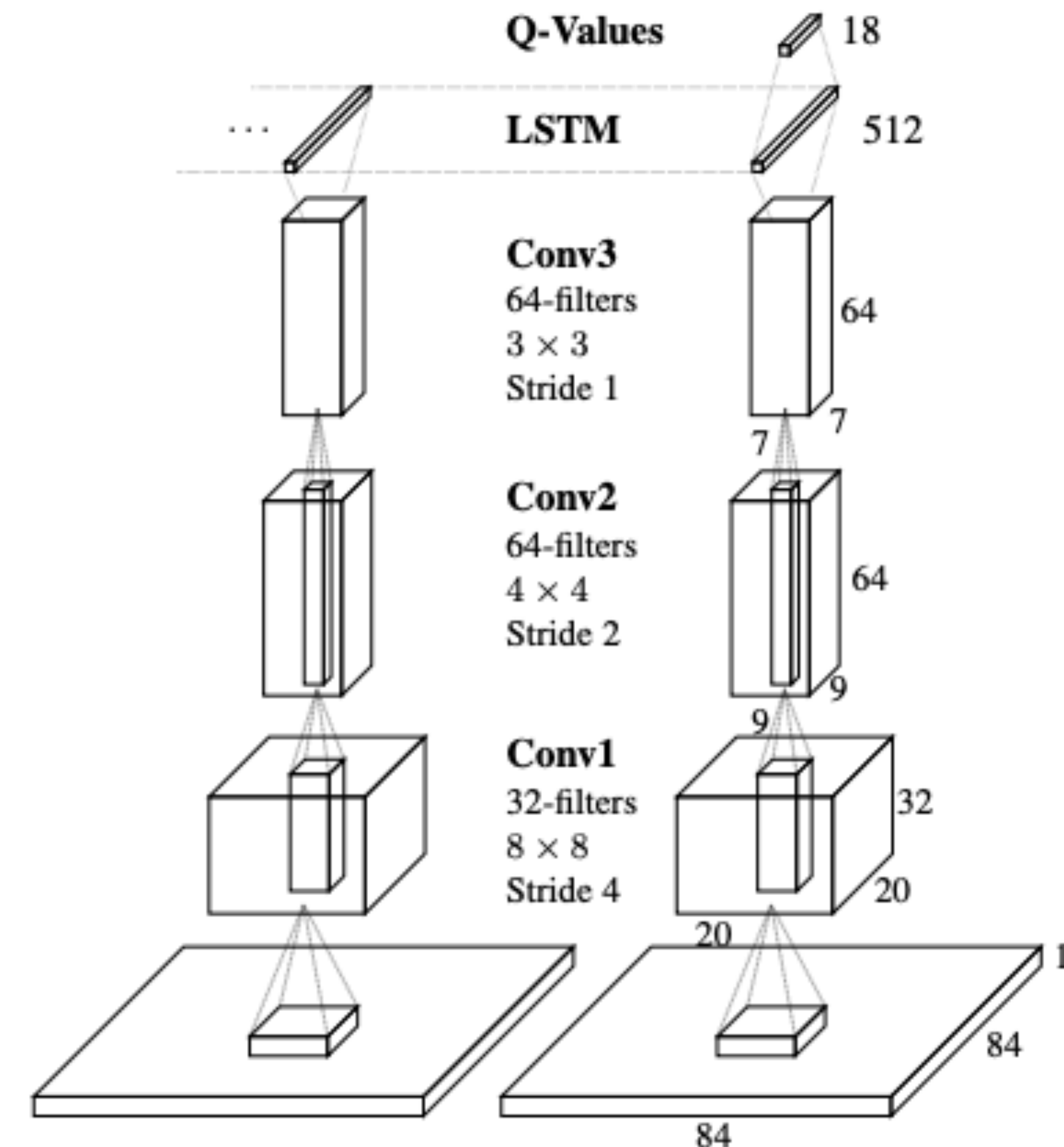


Pre-processing

- Large RGB images take a lot of memory.
 - Solution: downsample and turn the image to greyscale.
- Images are non-Markovian observations of state.
 - Solution: frame-stacking, i.e., concatenate past four frames together.
- The agent repeats the same action for four consecutive frames and then can choose a new action.

Recurrent Neural Networks

- Recurrent neural networks update a hidden state that is an input to computations at the next time-step.
- Allows networks to remember previously seen inputs when computing future outputs.
- In RL, provides a method to learn a Markov state.
- Alternative to frame-stacking.



Experience Replay

- The basic semi-gradient Q-learning algorithm processes (s, a, s', r) transitions as they are experienced and then discards them.
- Experience replay: keep around the most recent transitions (in DQN, the past 1 million) and use a random subset to update the action-value function.
 - Uses samples multiple times.
 - Reduces correlation between samples.
- Other choices besides random subset can improve performance.

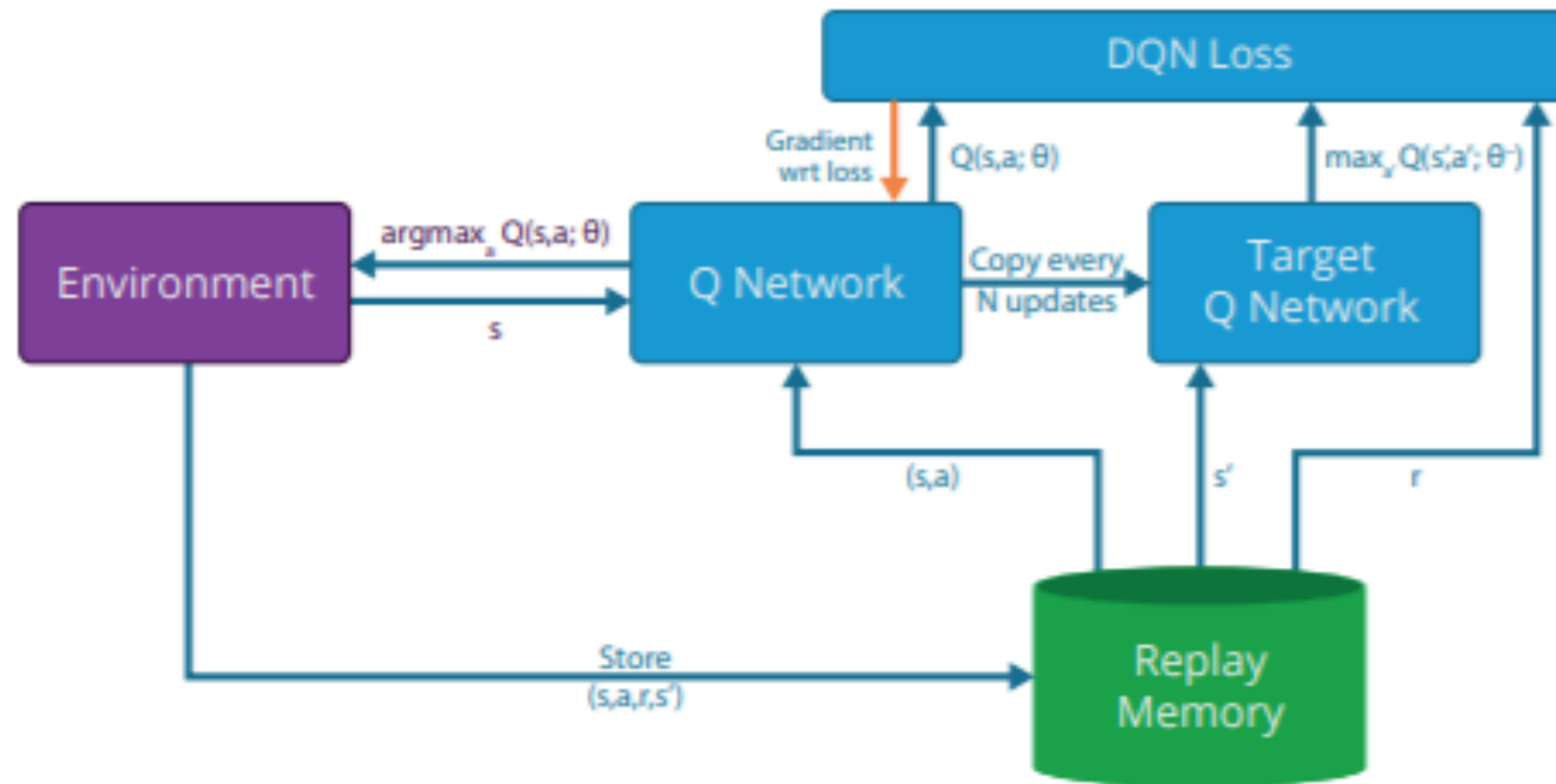
Target Networks

- The basic semi-gradient Q-learning algorithm always uses the most recent parameters to form the training target $R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w})$
- DQN uses a separate **target network** to compute $\gamma \max_{a'} \hat{q}(S_{t+1}, a', \tilde{\mathbf{w}})$.
- The target network is infrequently updated by setting the target network parameters to be the same as the main network's parameters.
- Makes the learning target more stable as in supervised learning.

Reward clipping

- Different Atari games have different reward magnitudes.
- Why is this a challenge?
 - Hard to tune a step-size that works across all games.
- Solution: clip rewards to be between -1 and 1.

DQN Architecture



Overall 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 $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 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

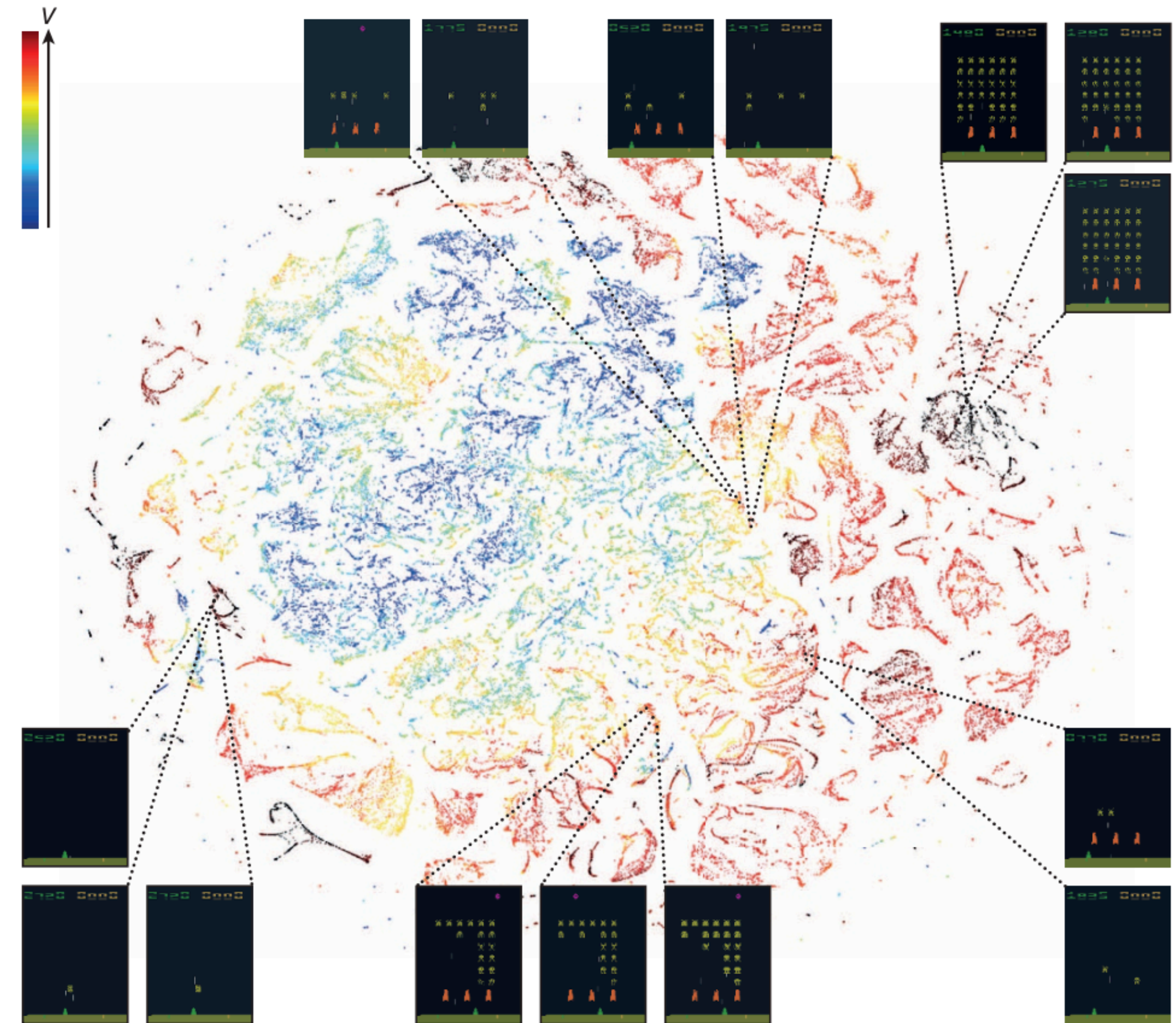
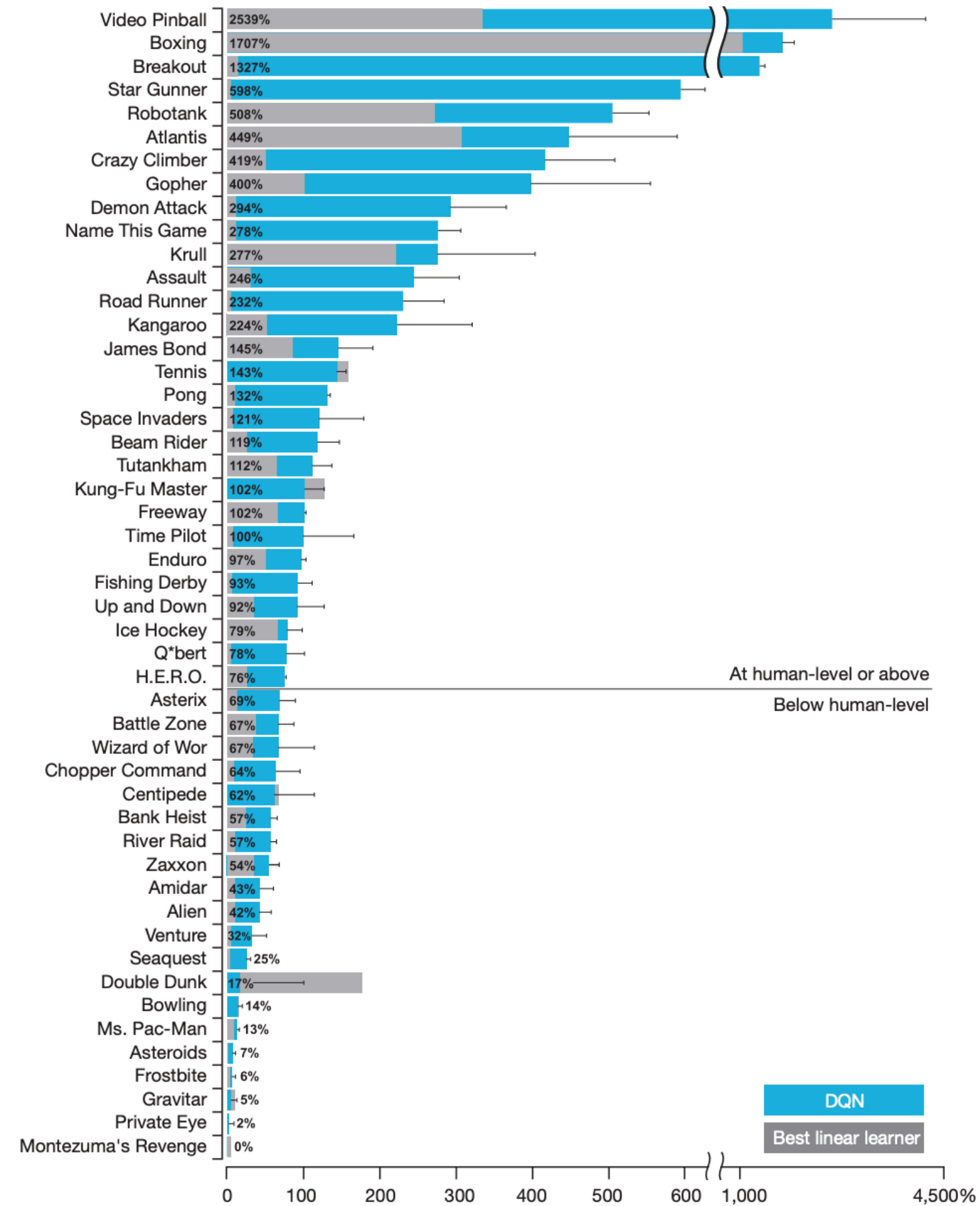
End For

Simple exploration

**Background
planning**

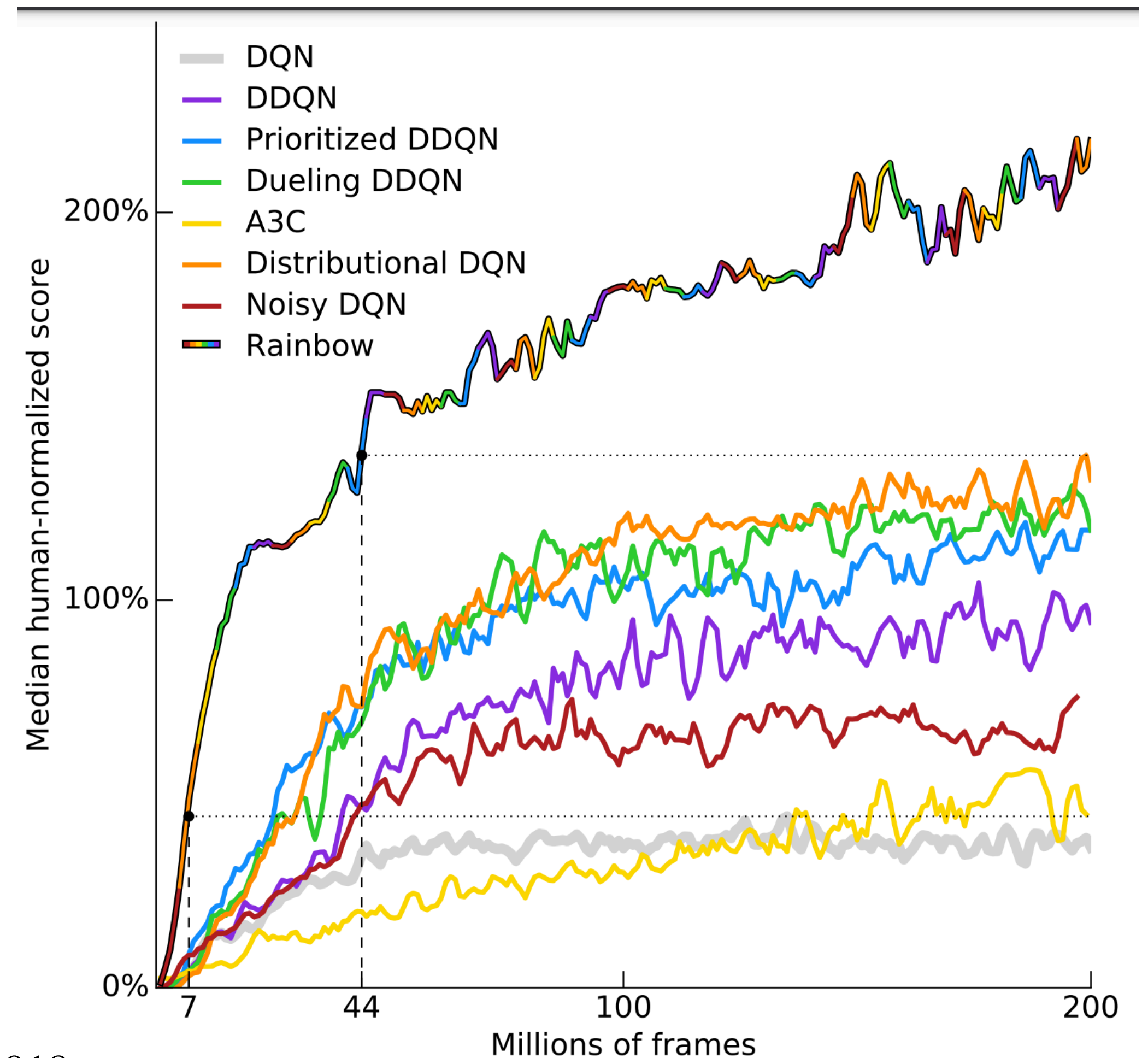
**Semi-gradient
learning**

DQN Results



Looking Forward

- DQN launched a surge of interest in deep reinforcement learning that has led to many exciting new applications and RL developments.
- DQN is widely used in practice though many improvements have been made.



Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018.

<https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark>

Dinesh's Presentation

A Distributional Perspective on Deep RL

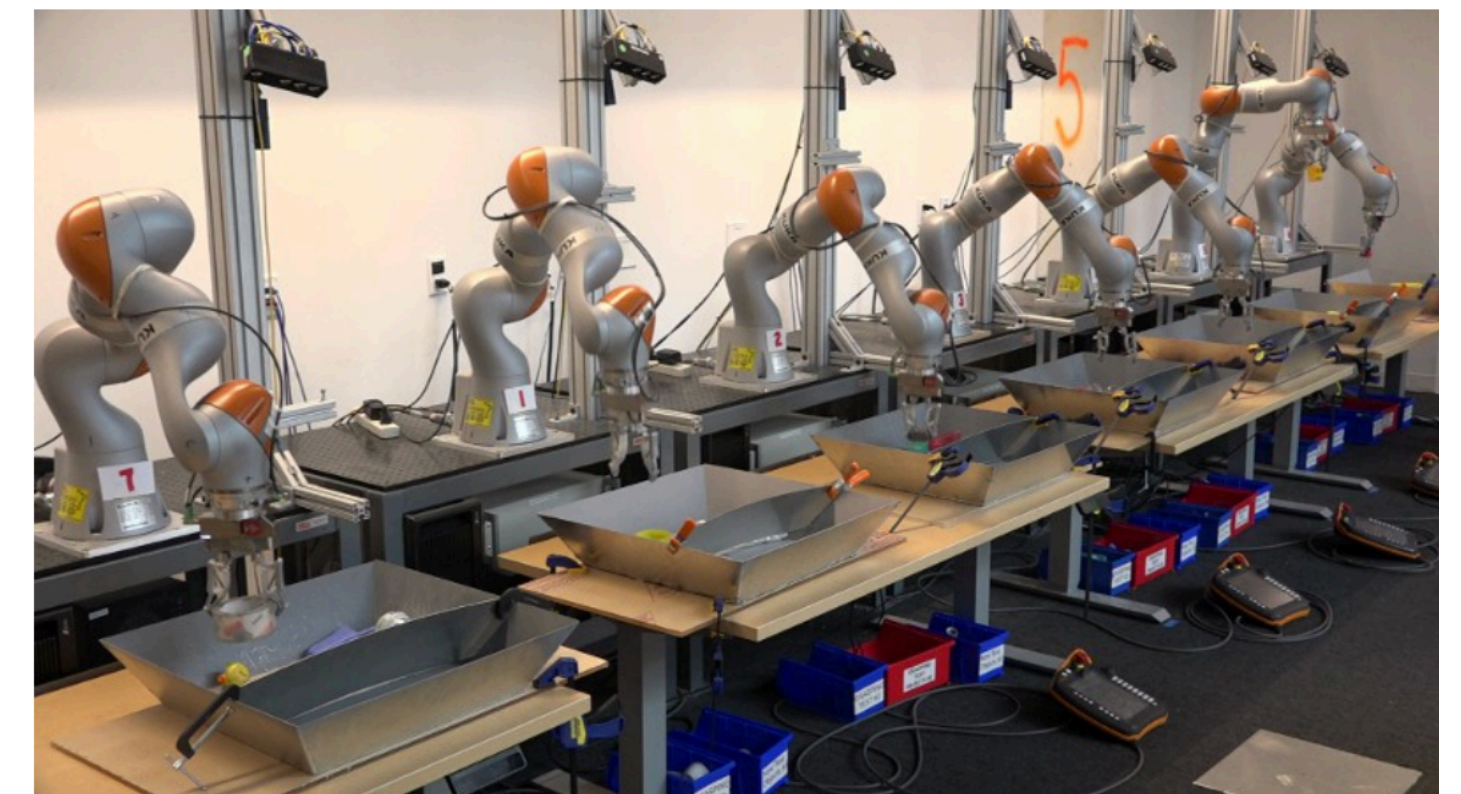
Bellemare et al. 2017

- [Slides](#)

Q-Learning for Continuous Actions

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') - q_k(S_t, A_t)]$$

- The max over the action space is difficult to compute when actions take on real-values. Why?
- One idea: use optimization to find the best action.
 - Examples: cross-entropy method, gradient ascent
 - But can be slow.
- Another idea: discretize the action space.
 - Sometimes works but loses precision in control.



QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

Deterministic Actor-Critic

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') - q_k(S_t, A_t)]$$

- Final idea: learn a policy (often a neural network) that outputs the maximizing action.
- $\mu_\theta(s) \rightarrow a$.
 - Learn θ such that $q_k(s, \mu_\theta(s)) \approx \max_a q_k(s, a)$.
 - $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta q_k(s, \mu_\theta(s)) \quad \nabla_\theta q_k(s, \mu_\theta(s)) = \nabla_a q_k(s, a) \nabla_\theta \mu_\theta(s) |_{a=\mu_\theta(s)}$
- Actor: the policy μ_θ .
- Critic: the action-value function, trained with TD-learning to estimate q_{μ_θ} .

Deep Deterministic Policy Gradient

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process $\mathcal{N}$ for action exploration
 Receive initial observation state s_1
 for t = 1, T **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))|\theta^{Q'}$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

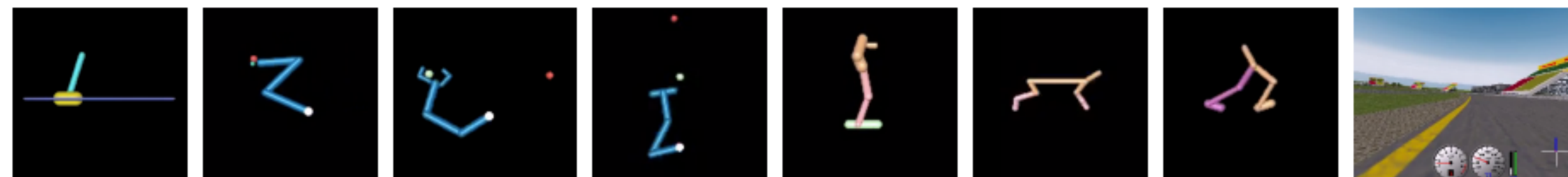
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for
end for



Scaling with Loss Representation

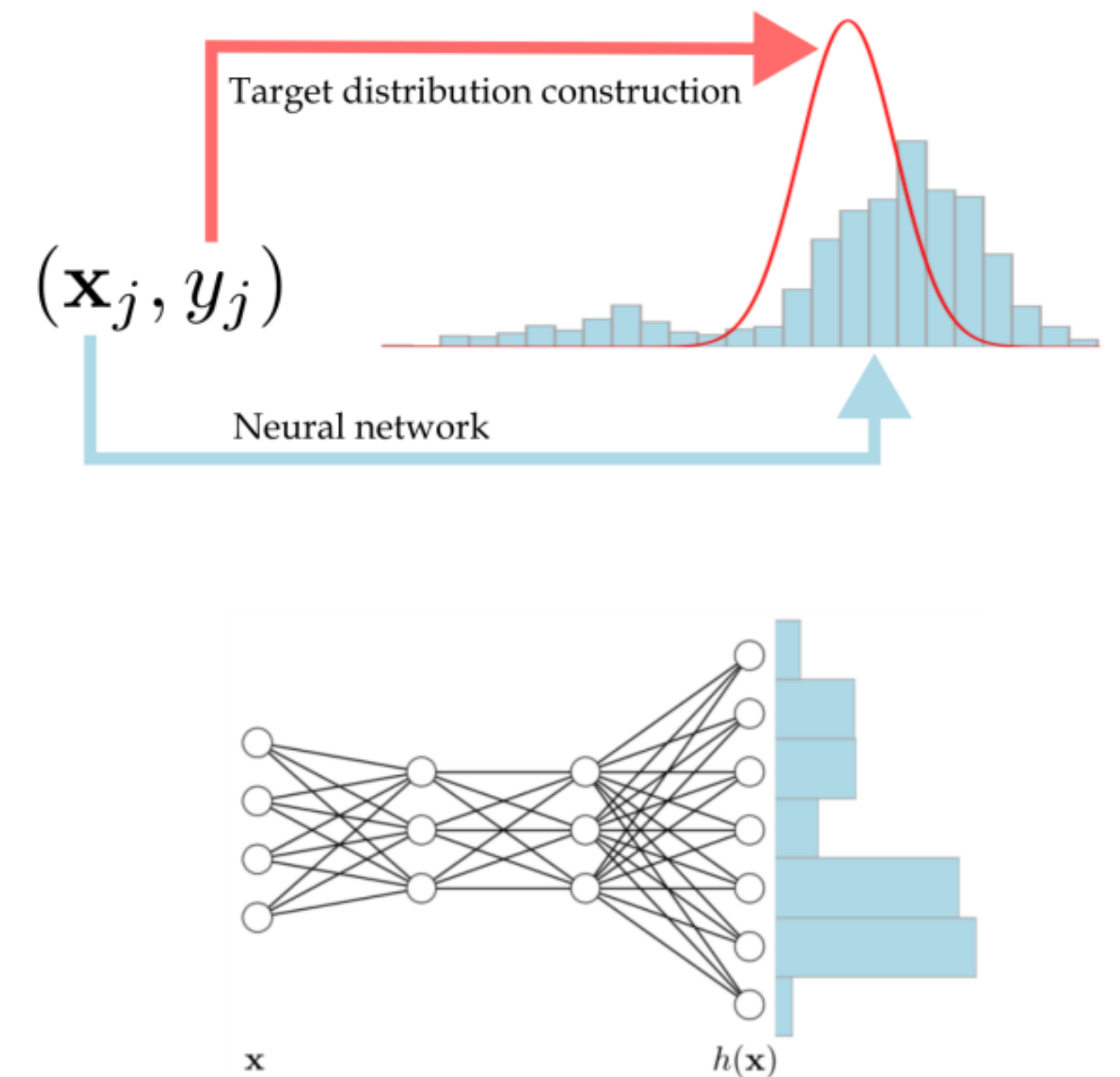
- DQN used an MSE loss — essentially regression against the Bellman target, $R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')$.

- Alternative classification formulation:

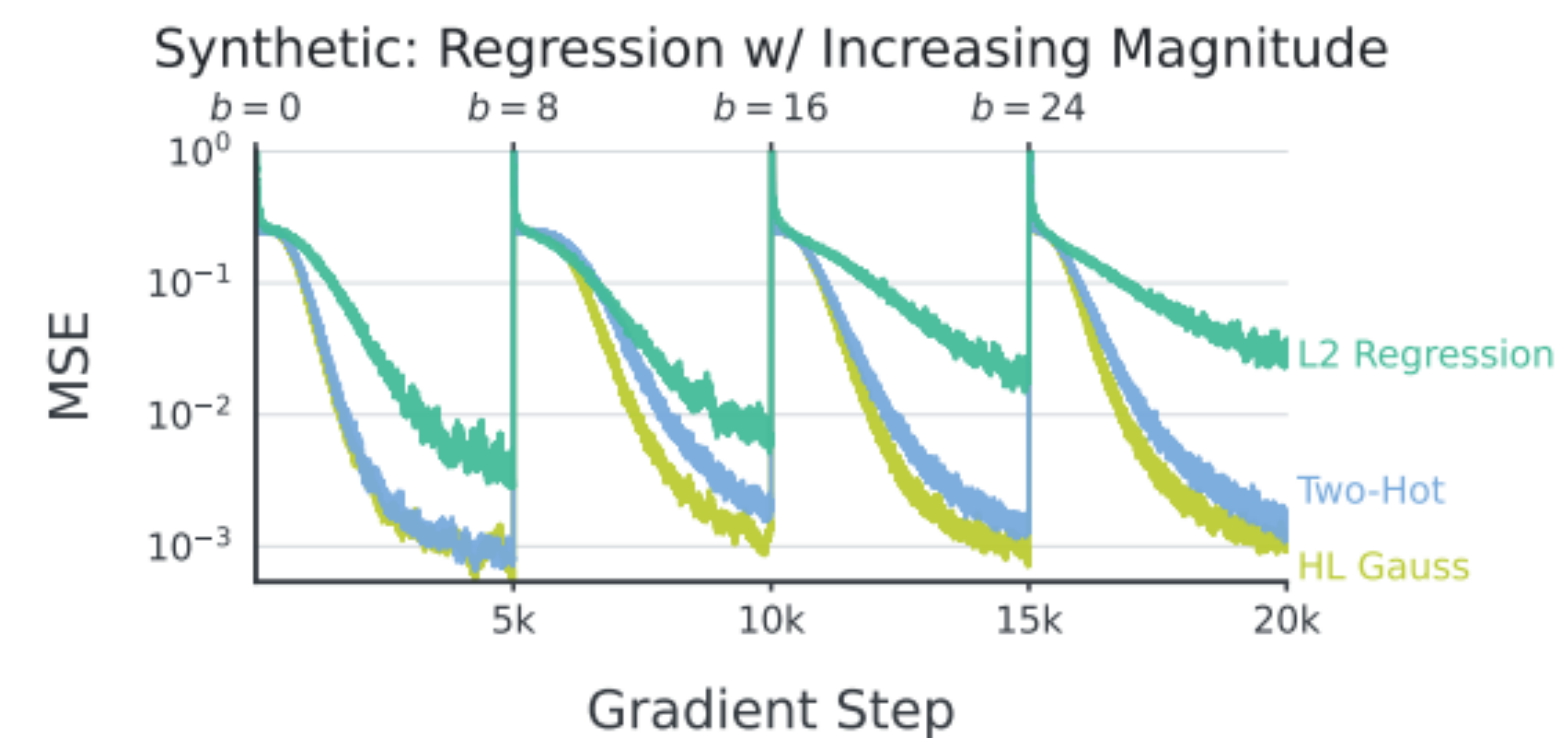
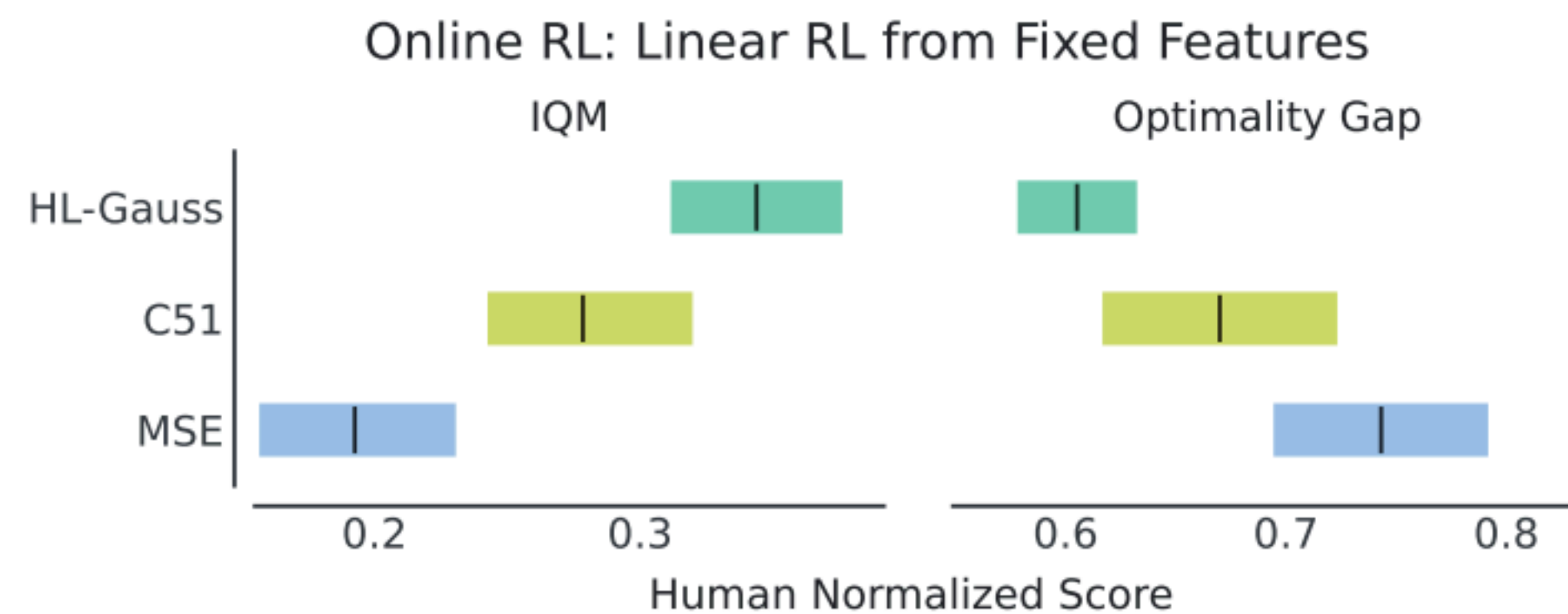
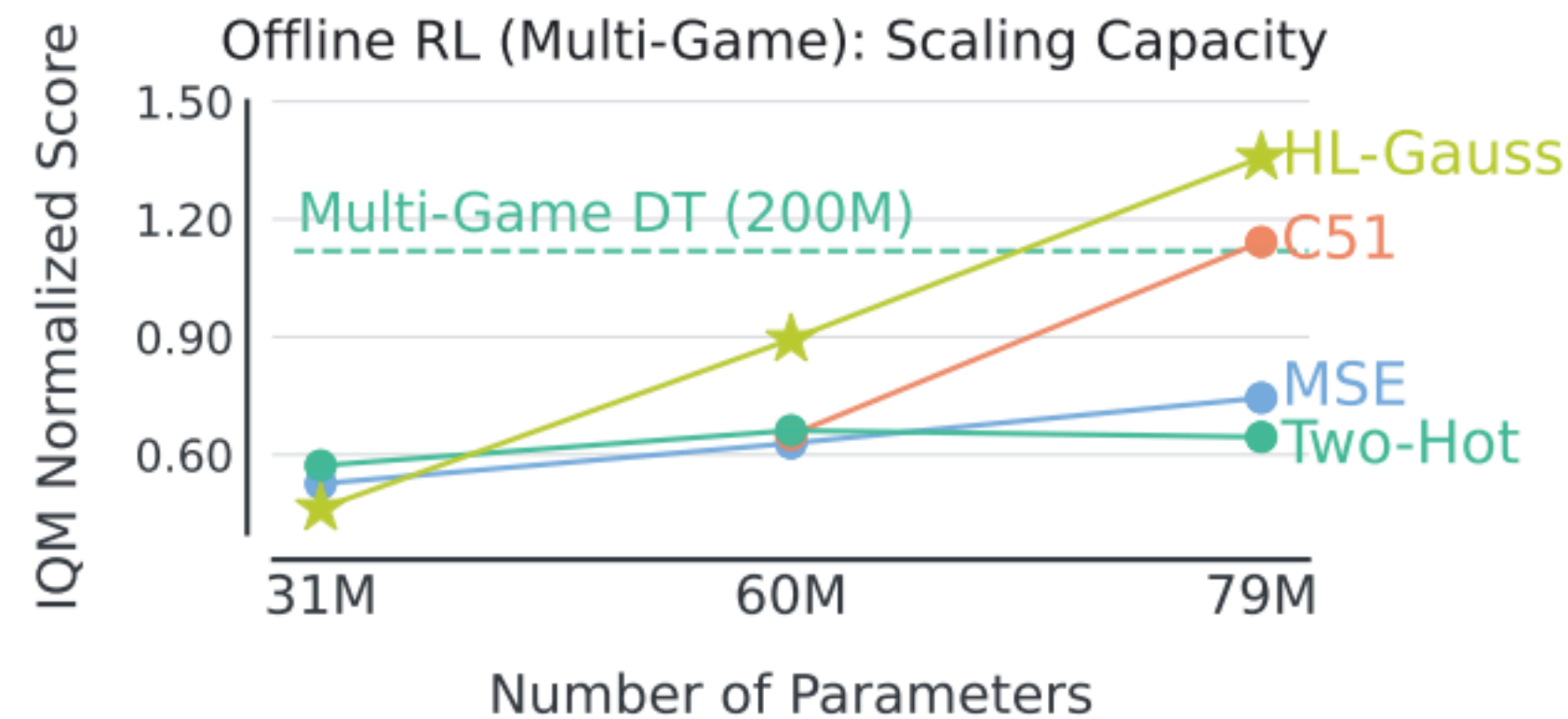
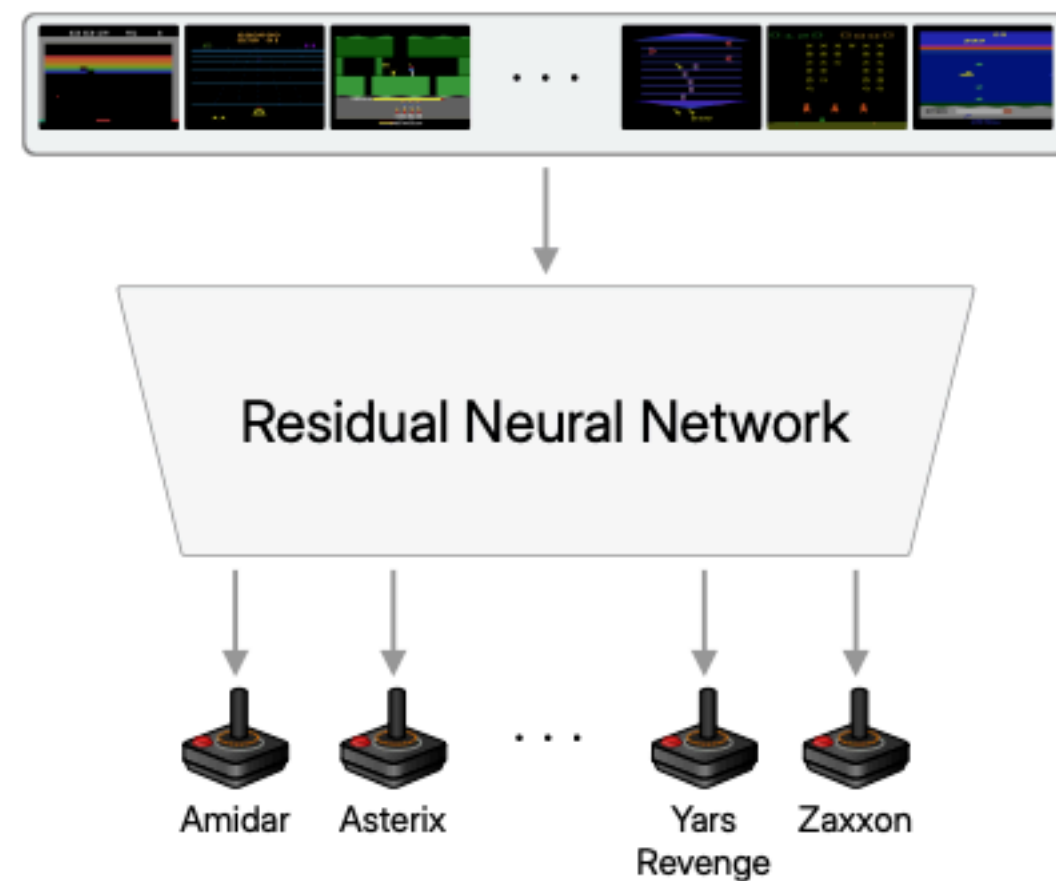
- $L_{\text{MSE}}(x, y, \theta) = (y_{\theta}(x) - y)^2$ vs.

$$L_{\text{CE}}(x, y, \theta) = \sum_{k=1}^n p_y(z_k | x) \log p_{\theta}(z_k | x)$$

- Scalar prediction: $\hat{y} = \sum_{k=1}^n z_k p_{\theta}(z_k | x)$



Scaling with Loss Representation



Summary

- Deep reinforcement learning is not just deep learning + RL.
 - It's often deep learning + RL + new techniques and tricks for stability.
- This week focused on deep value-based RL.
 - Deep networks can be used for model-based RL.
 - Deep networks can represent policies in policy gradient RL.

Action Items

- Complete literature review.
- Read Chapter 13 (skip 13.5).