

Advanced Topics in Reinforcement Learning

Lecture 11: Deep Reinforcement Learning I

Josiah Hanna

University of Wisconsin — Madison

Announcements

- Homework done!
- Read Chapter 13 for next week. Policy-based RL!
- Upcoming dates:
 - Literature survey due: 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.

Linear Function Approximation Review

- Assume value estimate is a linear function of state-action features.

$$\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s, a) = \sum_{i=1}^d w_i x_i(s, a)$$

- 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 features, $x_i(s, a)$, can be non-linear functions of state variables and actions.
 - Expressive choices for $\mathbf{x}(s, a)$ make linear methods more powerful than they first appear.
 - What if we instead learn $\mathbf{x}$ while learning $\hat{q}$? $\rightarrow$ Deep RL!!!

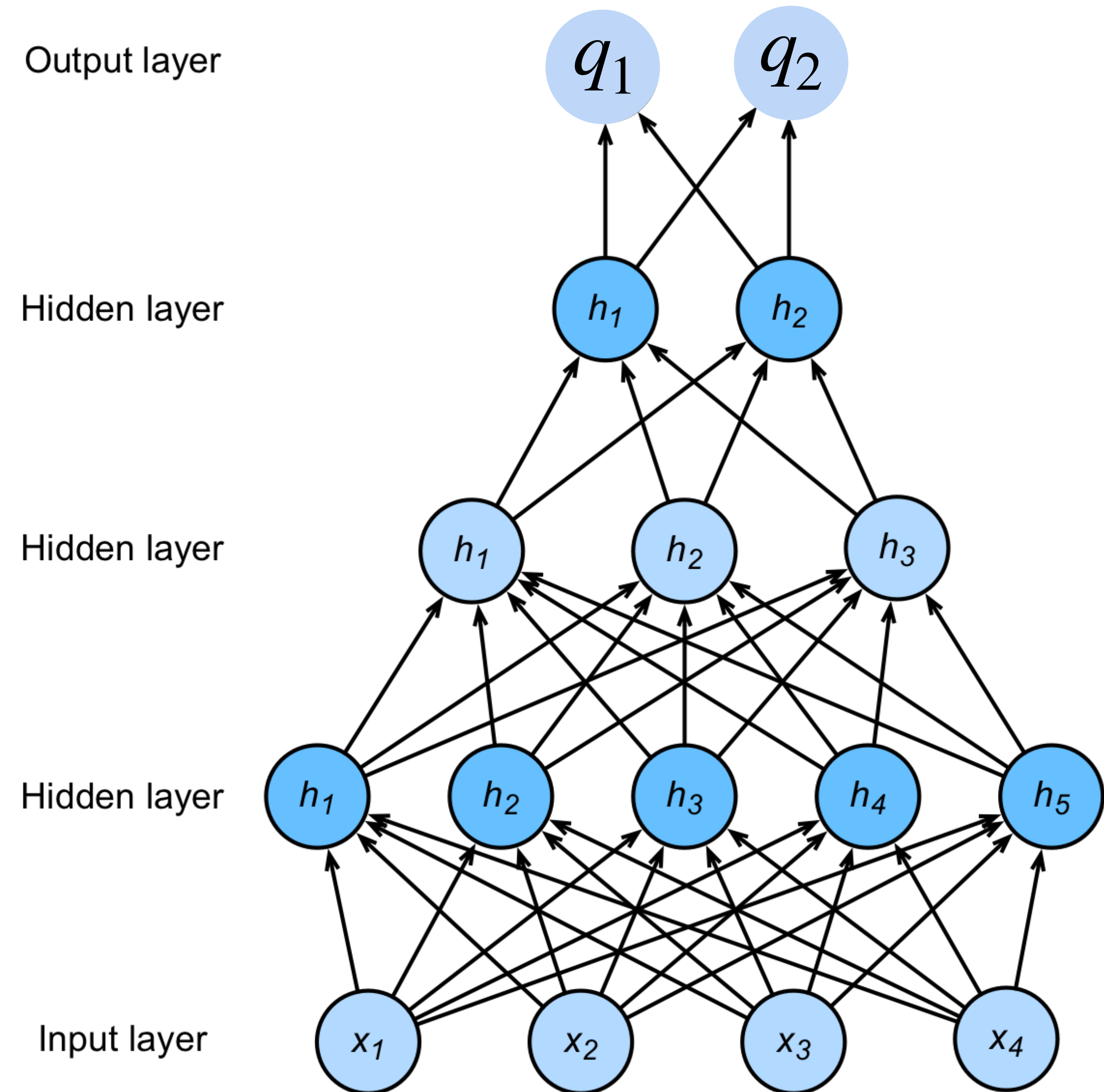
Neural Network Function approximations

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{q} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$



Hierarchical Representations

- Each layer of a neural network transforms the output of the layer before it (the first layer transforms the input).
- We can say that each layer is producing a new representation of the values at the previous layer.
- Chaining layers together allows the network to learn progressively more complex representations of the data.
- Pixels —> Edge detectors —> Shape detectors —> object detectors

Neural Network Training in RL

- Semi-Gradient Q-learning:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \underbrace{(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t))}_{\delta_t} \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- 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 weight or bias of the network.
- Adjust each weight in proportion to gradient of output times TD-error.

Problems with Backpropagation

- Backprop: using chain rule to find gradient of a function with respect to input variables.

- $q(w) = f_1(f_2(f_3(\cdots f_k(w)\cdots)))$ means that $\frac{\partial q}{\partial w} = \underbrace{f'_1(f_2(f_3\cdots f_k(w)\cdots)) \cdot f'_2(f_3\cdots f_k(w)\cdots)) \cdots f'_k(w)}_{\text{long product}}$

- With many layers, gradients for parameters in the initial layers are either very large or very small. Why?
 - Chain rule means the gradient is a product of many factors. Gradient magnitude can grow or decay exponentially in network depth.
- Alternative #1: Find a different learning rule.
 - Evolution; Reinforcement Learning.
- Alternative #2: Make our networks more backprop-friendly.

Training Improvements

- Regularization: methods that encourage learning simple models over more complex ones.
 - Add $||\mathbf{w}||_2^2$ as an additional objective; drop-out
- Better initializations: start gradient descent at a good location in weight space.
 - Pre-training; special initialization rules (e.g., Glorot/Xavier initializations)
- Architecture improvements
 - Residual connections, normalize inputs and layer activations.
- Not all of these ideas work well for reinforcement learning! (E.g., scaling)

DQN Overview

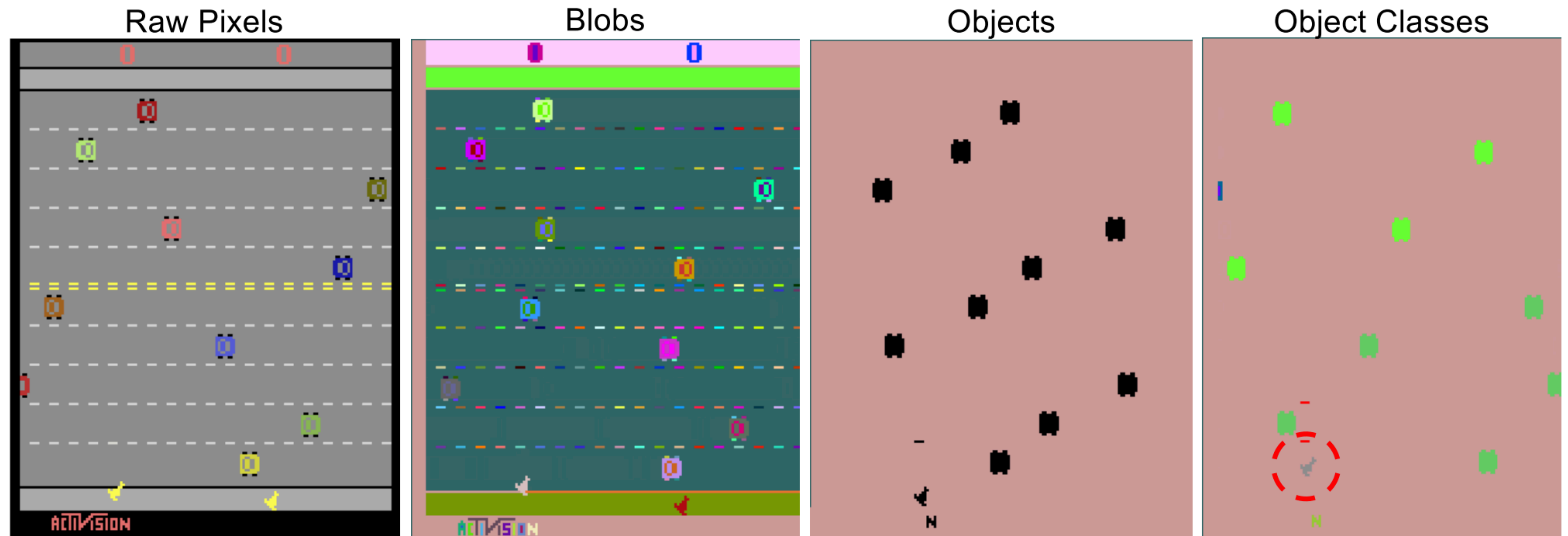
- Single algorithm, neural network architecture, and hyper-parameter setting that played 49 Atari video games at human-level.
- Training and evaluation is independent for each game.
 - The final neural network from training on “Breakout” cannot play “Pong.”
- Landmark result for deep reinforcement learning.

The Atari 57 Benchmark

- 57 Atari video games turned into RL benchmarks
- Why were Atari games hard for reinforcement learning algorithms?
 - Representation learning; hyper-parameter robustness
 - Prior state-of-the-art: neuroevolution and then Deepmind's predecessor to deep Q-learning.
- With a suitable representation, some games are simpler than others.

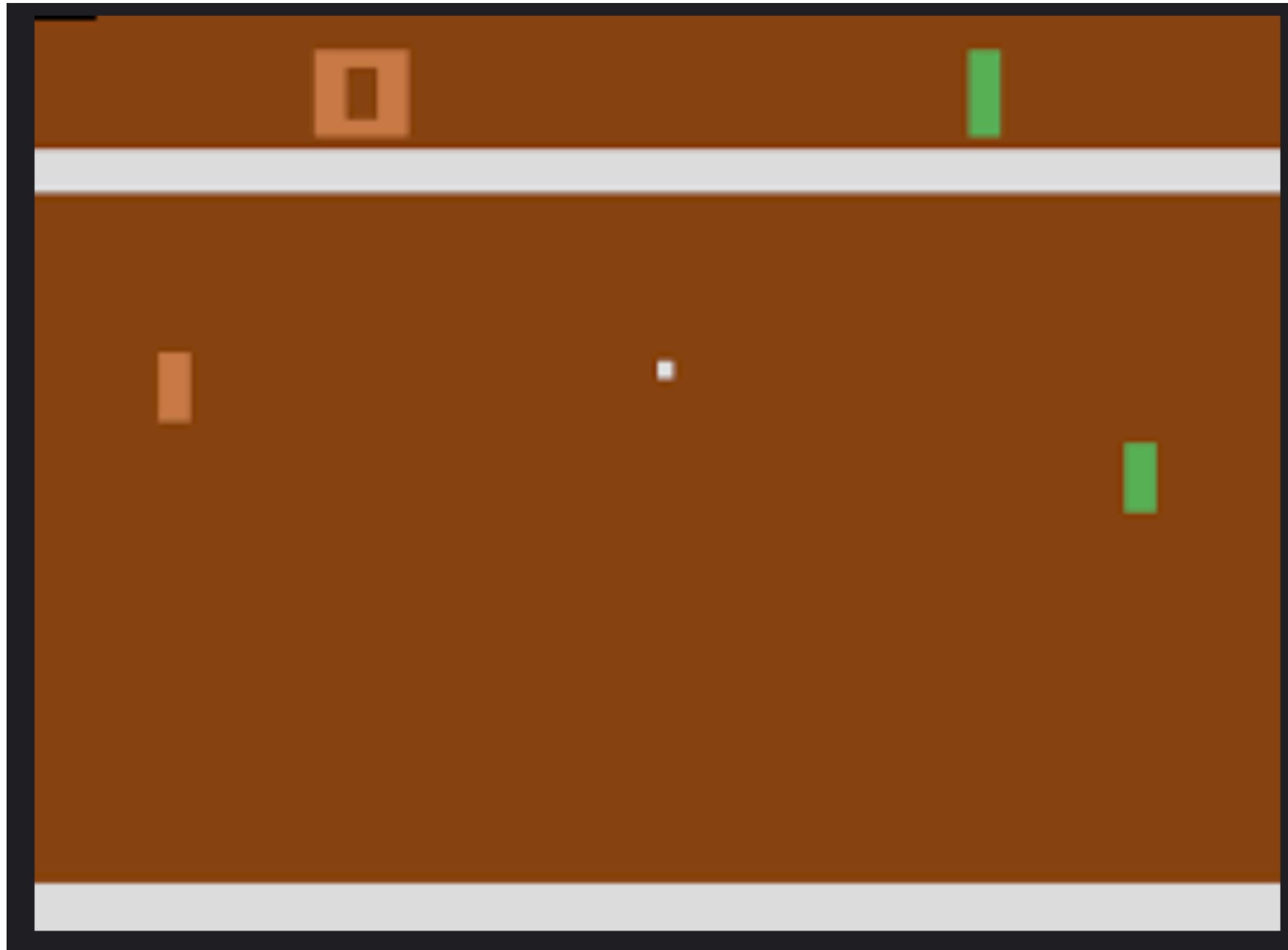
“HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player.” Hausknecht, Khandelwal, Miikkulainen, and Stone. 2012.

Feature Engineering



“HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player.” Hausknecht, Khandelwal, Miikkulainen, and Stone. 2012.

Easy and Hard Games in Atari



Pong



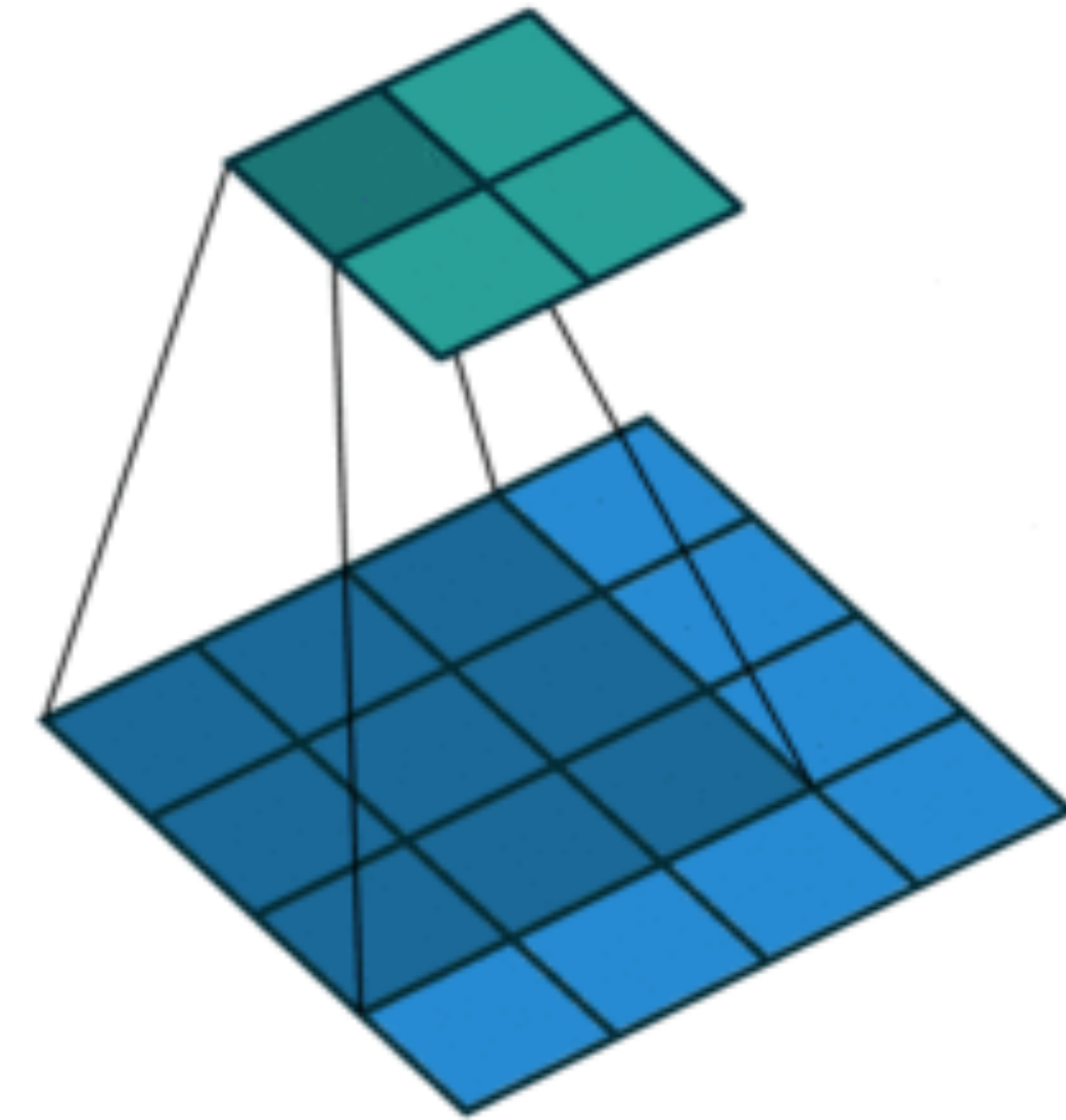
Montezuma's Revenge

DQN Architecture

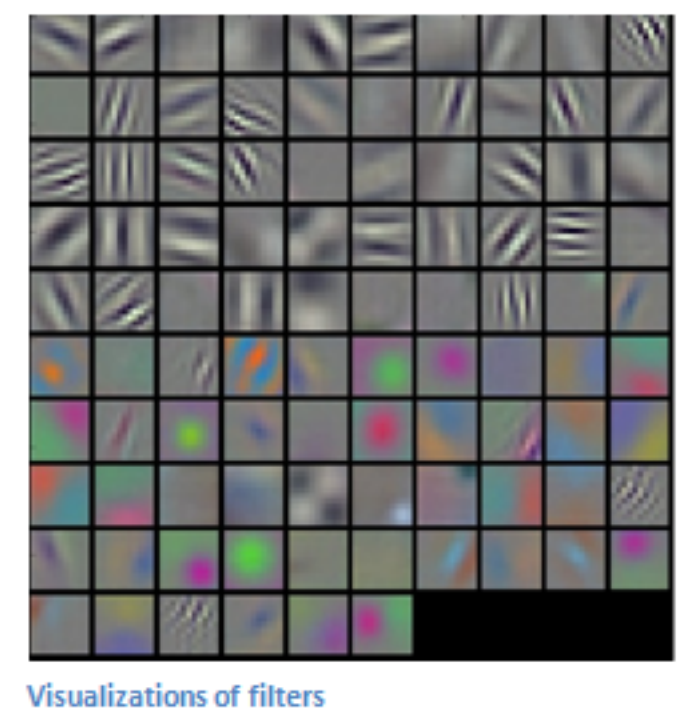
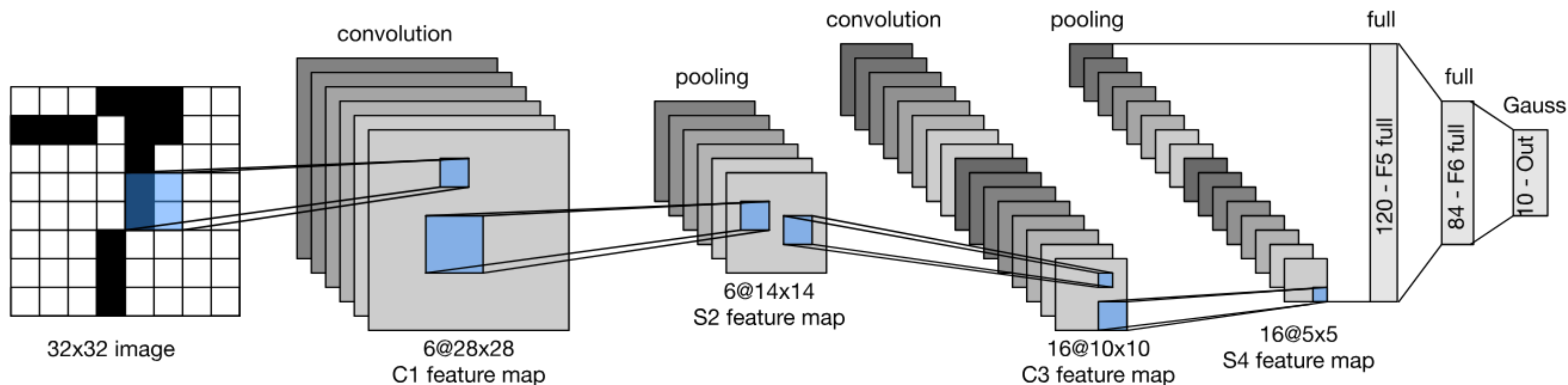
- Core algorithm is semi-gradient Q-learning with a convolutional neural network as the function approximator.
- Key techniques for effective training across tasks:
 - Pre-processing
 - Convolutional neural network
 - Experience replay
 - Target networks
 - Reward clipping

Convolutional Neural Networks

- Special architecture primarily for processing visual inputs.
- Local connections between inputs and outputs at next layer.
- Learn “filters” that have the same weights no matter where applied on an image.

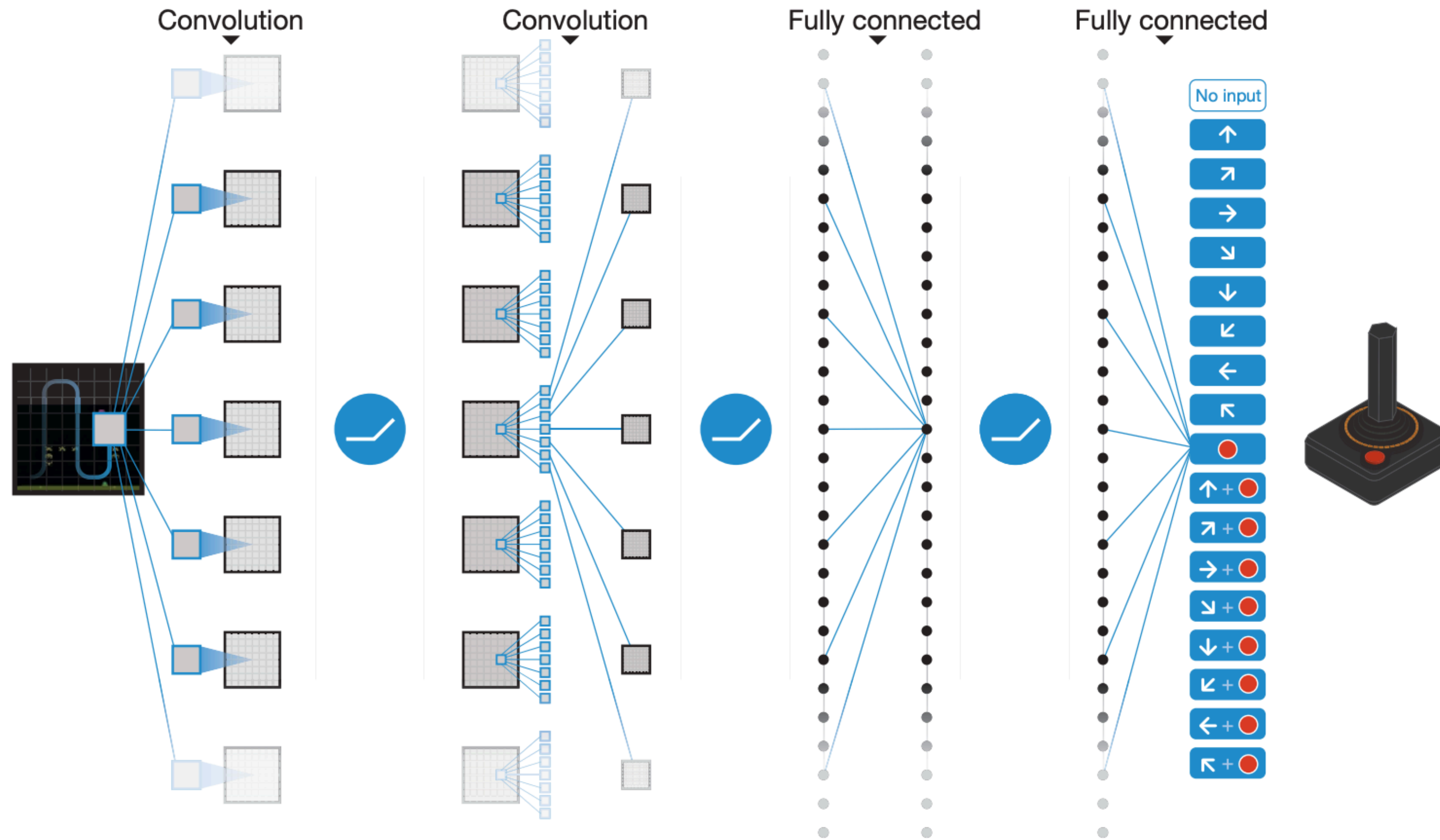


(vdumoulin@ Github)



Visualizations of filters

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.

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.

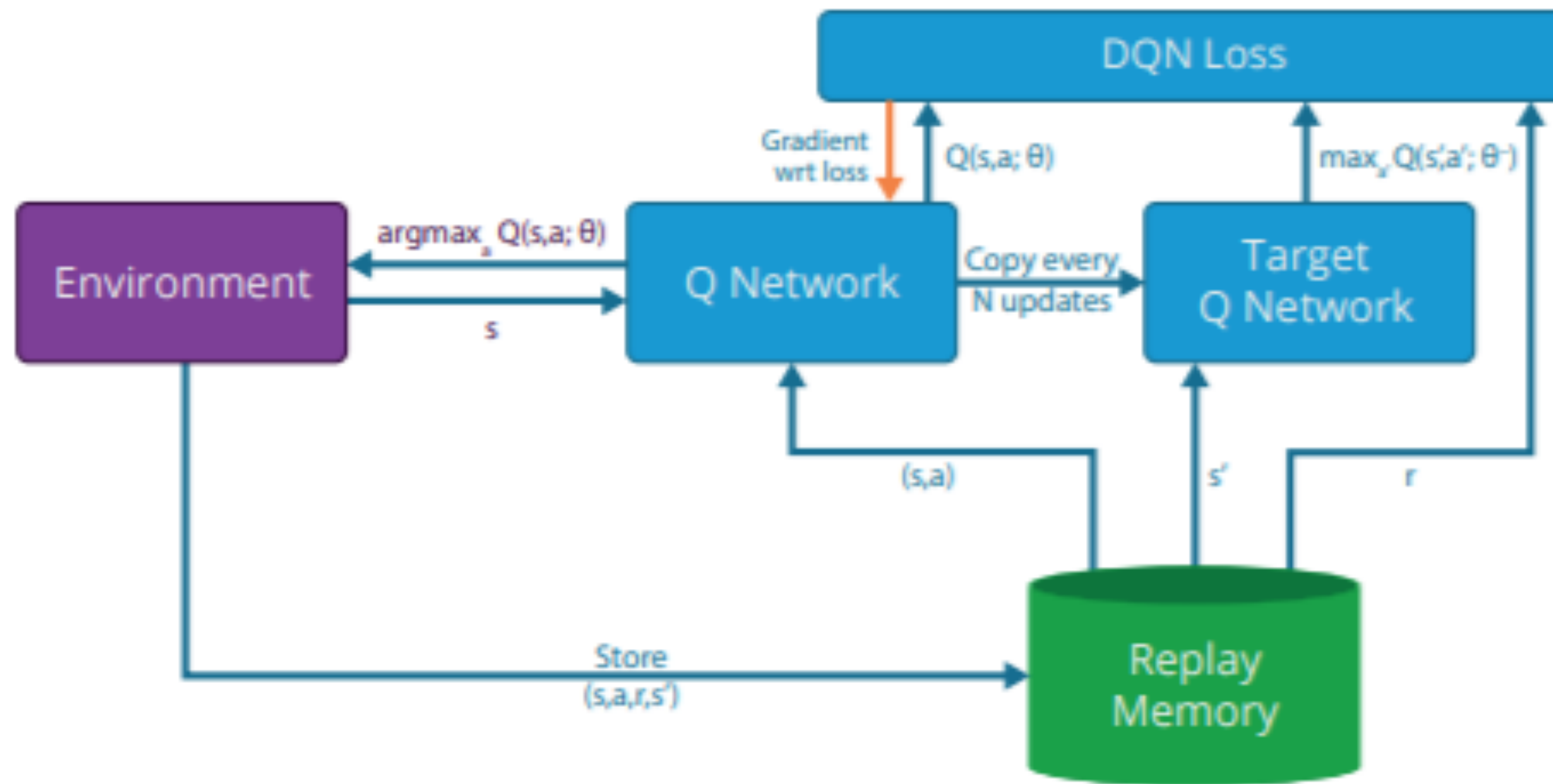
Target Networks

- 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

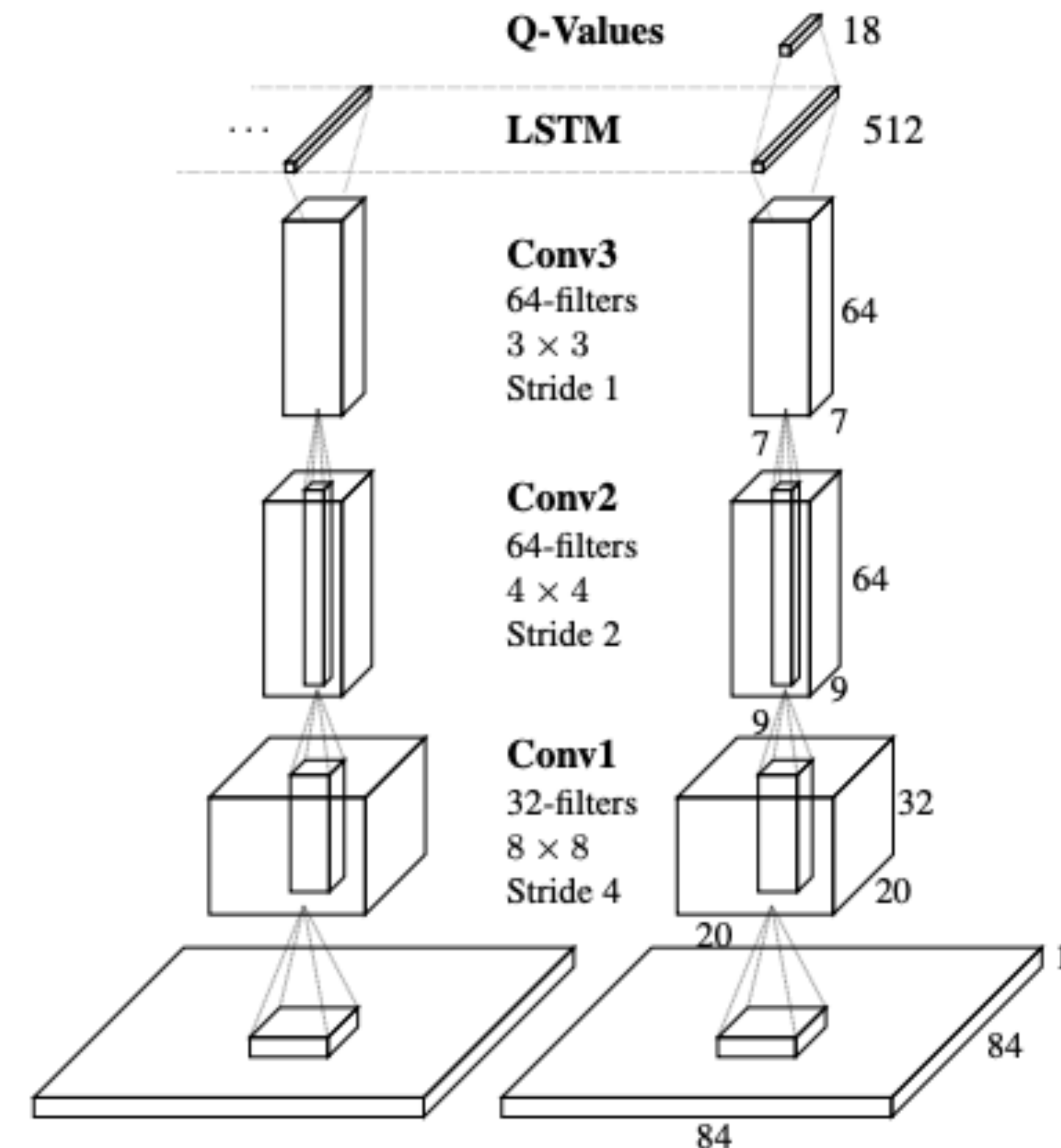
Simple exploration

**Background
planning**

**Semi-gradient
learning**

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.



Joseph's Presentation

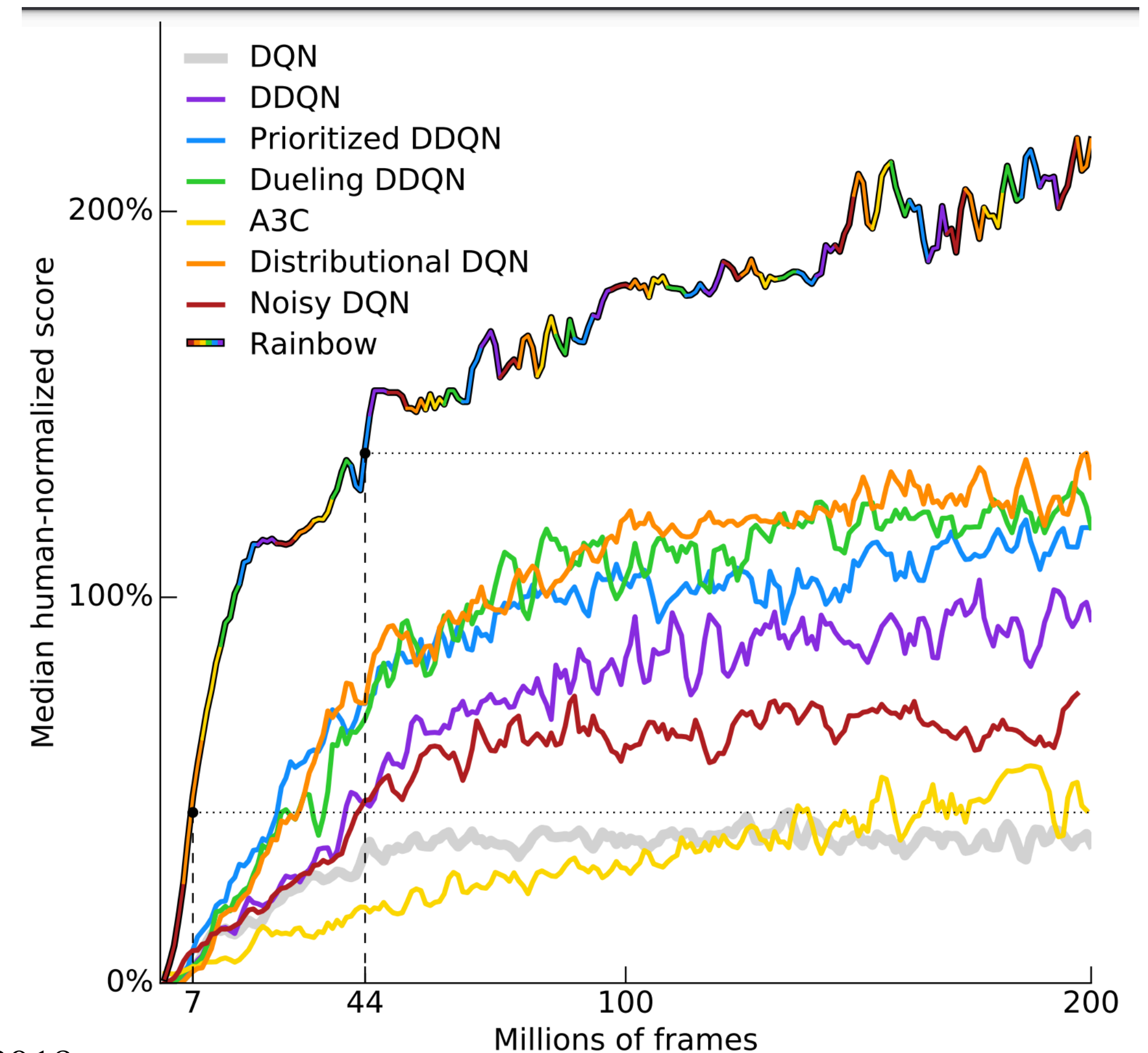
The Value Improvement Path

Dabney et al. 2021

- Slides

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.



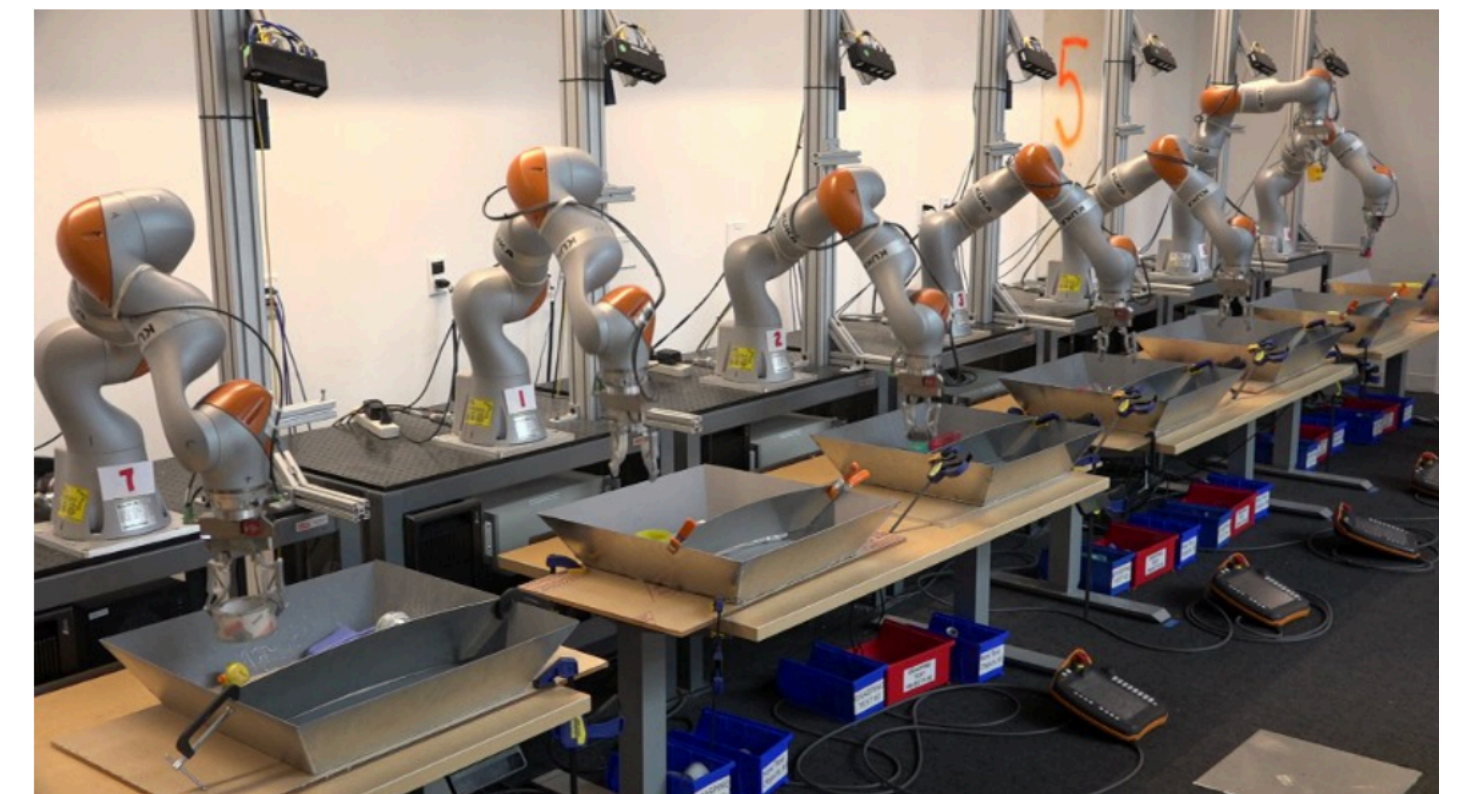
Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018.

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

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 $q_k(s, \mu_\theta(s)) \approx \max_a q_k(s, a)$.
 - $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta q_k(s, \mu_\theta(s))$
- Actor: the policy μ_θ .
- Critic: the action-value function, trained to estimate q_{μ_θ} .

Summary

- Neural networks are differentiable, non-linear function approximations that can be easily (in principle) used with semi-gradient reinforcement learning.
- Instead of learning a linear function of fixed, non-linear features, neural networks also learn the non-linear features.
- Removes the need for feature engineering though may require careful architecture and training considerations.

Action Items

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