Lecture 6: CNNs and Deep Q Learning [2]

Emma Brunskill

CS234 Reinforcement Learning.

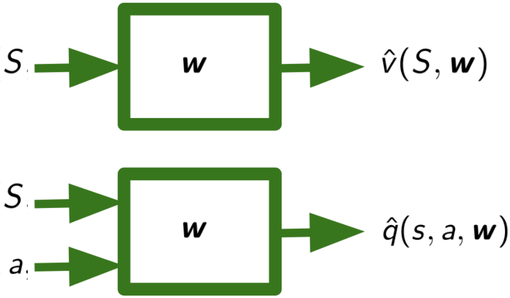
Winter 2018

[2] With many slides for DQN from David Silver and Ruslan Salakhutdinov and some vision slides from Gianni Di Caro and images from Stanford CS231n, http://cs231n.github.io/convolutional-networks/

Table of Contents

Generalization

- Want to be able use reinforcement learning to tackle self-driving cars, Atari, consumer marketing, healthcare, education
- Most of these domains have enormous state and/or action spaces
- Requires representations (of models / state-action values / values / policies) that can generalize across states and/or actions
- Represent a (state-action/state) value function with a parameterized function instead of a table



Recall: The Benefit of Deep Neural Network Approximators

- Linear value function approximators assume value function is a weighted combination of a set of features, where each feature a function of the state
- Linear VFA often work well given the right set of features
- But can require carefully hand designing that feature set
- An alternative is to use a much richer function approximation class that is able to directly go from states without requiring an explicit specification of features
- Local representations including Kernel based approaches have some appealing properties (including convergence results under certain cases) but can't typically scale well to enormous spaces and datasets
- Alternative: use deep neural networks
  - Uses distributed representations instead of local representations
  - Universal function approximator
  - Can potentially need exponentially less nodes/parameters (compared to a shallow net) to represent the same function
- Last time discussed basic feedforward deep networks

Generalization

- Using function approximation to help scale up to making decisions in really large domains

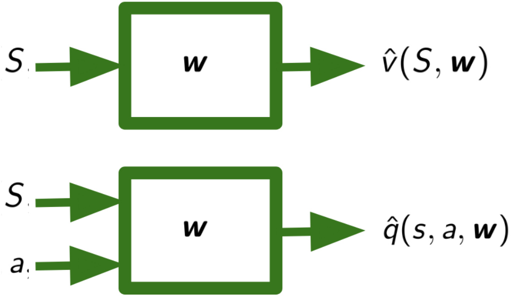
Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent (SGD)

Deep Q-Networks (DQNs)

- Represent value function by Q-network with weights $\boldsymbol{w}$

$$\hat{q}(s, a, \boldsymbol{w}) \approx q(s, a) \tag{1}$$



Recall: Action-Value Function Approximation with an Oracle

- $\hat{q}^{\pi}(s, a, w) \approx q^{\pi}$
- Minimize the mean-squared error between the true action-value function $q^{\pi}(s, a)$ and the approximate action-value function:

$$J(w) = \mathbb{E}_{\pi}[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, w))^2] \tag{2}$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_W J(w) = \mathbb{E}\left[(q^{\pi}(s, a) - \hat{q}^{\pi}(s, a, w)) \nabla_w \hat{q}^{\pi}(s, a, w)\right] \tag{3}$$

$$\Delta(w) = -\frac{1}{2}\alpha \nabla_w J(w) \tag{4}$$

- Stochastic gradient descent (SGD) samples the gradient

Recall: Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta w = \alpha(G_t - \hat{q}(s_t, a_t, w)) \nabla_w \hat{q}(s_t, a_t, w) \tag{5}$$

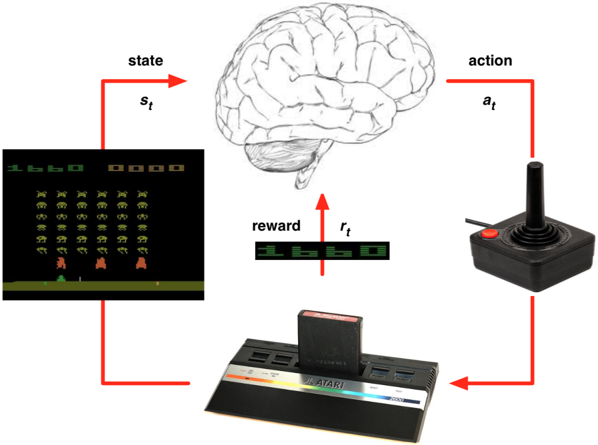
- For SARSA instead use a TD target $r + \gamma \hat{q}(s', a', w)$ which leverages the current function approximation value

$$\Delta w = \alpha(r + \gamma \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s, a, w) \tag{6}$$

- For Q-learning instead use a TD target $r + \gamma \max_a \hat{q}(s', a', w)$ which leverages the max of the current function approximation value

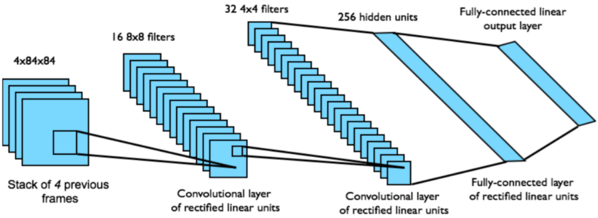
$$\Delta w = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s, a, w) \tag{7}$$

Using these ideas to do Deep RL in Atari



DQNs in 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

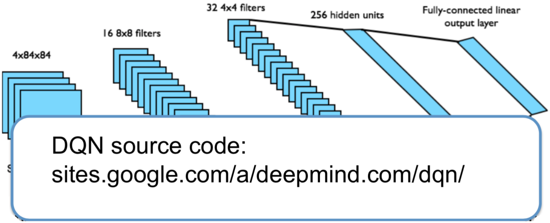


Mnih et.al., Nature, 2014

- Network architecture and hyperparameters fixed across all games

DQNs in 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



Mnih et.al., Nature, 2014

- Network architecture and hyperparameters fixed across all games

Q-Learning with Value Function Approximation

- Minimize MSE loss by stochastic gradient descent
- Converges to optimal q using table lookup representation
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both of these challenges by
  - Experience replay
  - Fixed Q-targets

DQNs: Experience Replay

- To help remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \quad (8)$$

DQNs: Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', w)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(s, a, \mathbf{w}) \qquad (9)$$

- Can treat the target as a scalar, but the weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

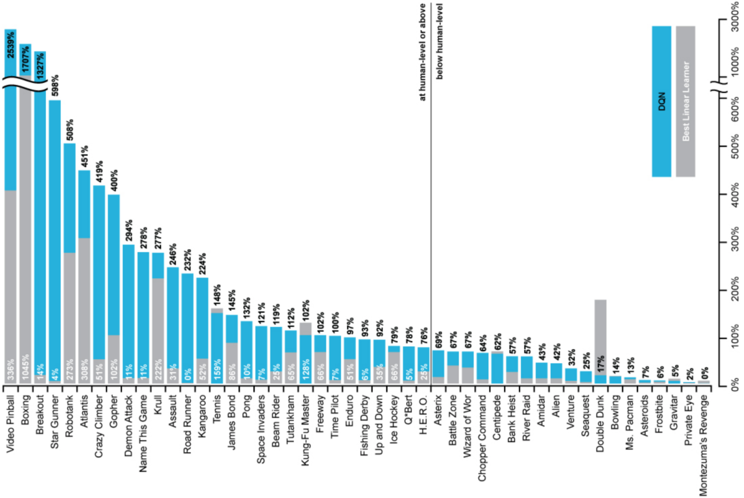
- To help improve stability, fix the **target network** weights used in the target calculation for multiple updates
- Use a different set of weights to compute target than is being updated
- Let parameters $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}^-)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}^-) - \hat{q}(s, a, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) \tag{10}$$

DQNs Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

DQN Results in Atari



Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q	DQN w/ replay	DQN w/replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Seaquest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

- Replay is **hugely** important item Why? Beyond helping with correlation between samples, what does replaying do?

Deep RL

- Success in Atari has lead to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - Double DQN
  - Dueling DQN (best paper ICML 2016)

Double DQN

- Recall maximization bias challenge
  - Max of the estimated state-action values can be a biased estimate of the max
- Double Q-learning

Recall: Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a)$,$\forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
  Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
  Observe (r_t, s_{t+1})
  if (with 0.5 probability) **then**

$$Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha(r_t + Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1},a')) - Q_1(s_t,a_t)) \tag{11}$$

  else

$$Q_2(s_t,a_t) \leftarrow Q_2(s_t,a_t) + \alpha(r_t + Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1},a')) - Q_2(s_t,a_t))$$

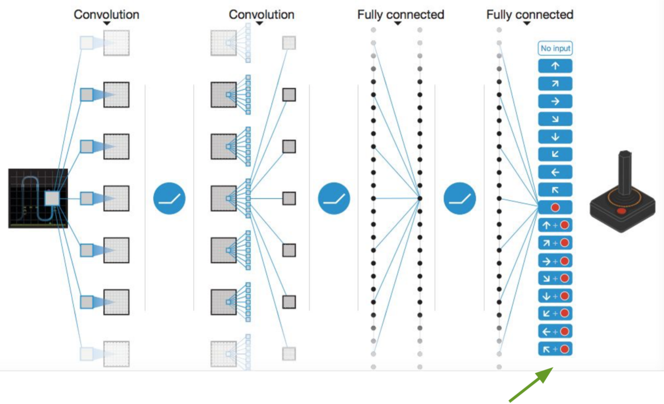
  end if
  $t = t + 1$
end loop

Double DQN

- Extend this idea to DQN
- Current Q-network $\boldsymbol{w}$ is used to select actions
- Older Q-network $\boldsymbol{w}^-$ is used to evaluate actions

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{q}(\underbrace{\arg\max_{a'} \hat{q}(s', a', \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}, \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{q}(s, a, \boldsymbol{w})) \tag{12}$$

Double DQN



1 network, outputs Q value for each action

Double DQN

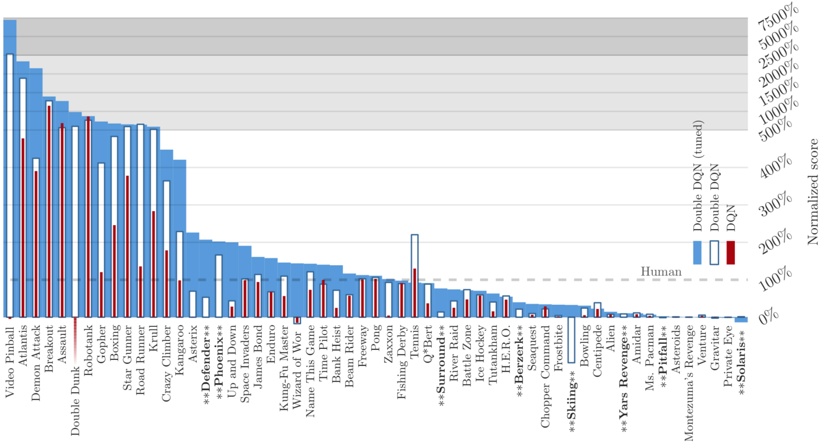


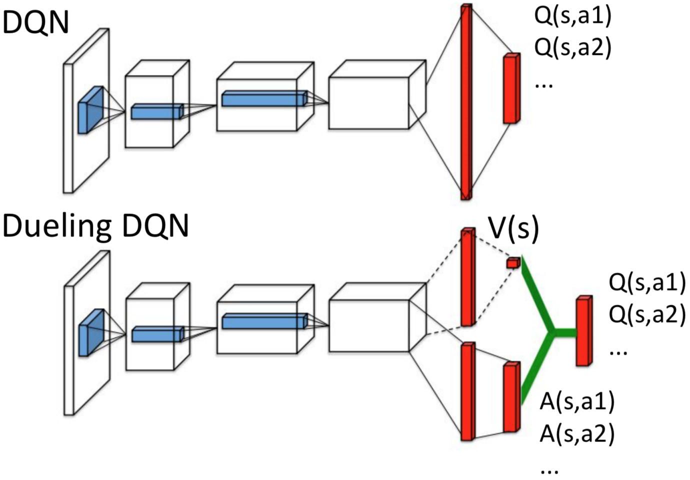
Figure: van Hasselt, Guez, Silver, 2015

Value & Advantage Function

- Intuition: Features need to pay attention to determine value may be different than those need to determine action benefit
- E.g.
  - Game score may be relevant to predicting $V(s)$
  - But not necessarily in indicating relative action values
- Advantage function (Baird 1993)

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

Dueling DQN



Wang et.al., ICML, 2016

Identifiability

- Advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

- Identifiable?

Identifiability

- Advantage function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

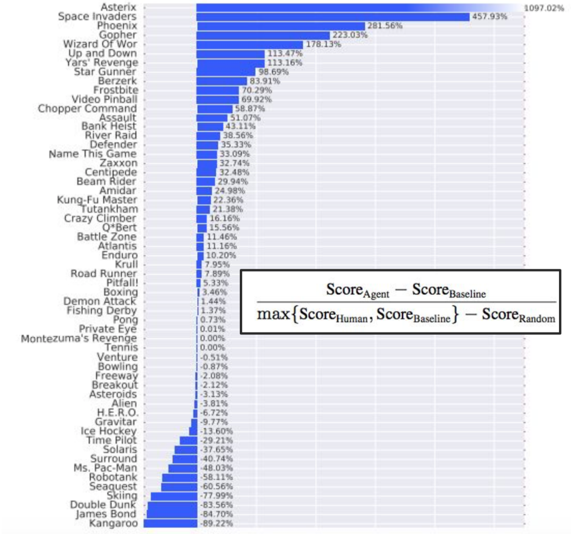
- Unidentifiable
- Option 1: Force $A(s,a) = 0$ if a is action taken

$$\hat{q}(s,a;\boldsymbol{w}) = \hat{v}(s;\boldsymbol{w}) + \left(A(s,a;\boldsymbol{w}) - \max_{a' \in \mathcal{A}} A(s,a';\boldsymbol{w})\right)$$

- Option 2: Use mean as baseline (more stable)

$$\hat{q}(s,a;\boldsymbol{w}) = \hat{v}(s;\boldsymbol{w}) + \left(A(s,a;\boldsymbol{w}) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\boldsymbol{w})\right)$$

V.S. DDQN with Prioritized Replay



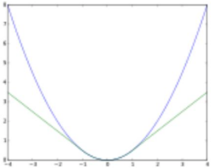
Practical Tips for DQN on Atari (from J. Schulman)

- DQN is more reliable on some Atari tasks than others. Pong is a reliable task: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is key
  - Use uint8 images, don't duplicate data
- Be patient. DQN converges slowly—for ATARI it's often necessary to wait for 10-40M frames (couple of hours to a day of training on GPU) to see results significantly better than random policy
- In our Stanford class: Debug implementation on small test environment

Practical Tips for DQN on Atari (from J. Schulman) cont.

- Try Huber loss on Bellman error

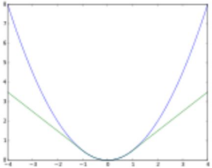
$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$



Practical Tips for DQN on Atari (from J. Schulman) cont.

- Try Huber loss on Bellman error

$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \le \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$



- Consider trying Double DQN—significant improvement from 3-line change in Tensorflow.
- To test out your data pre-processing, try your own skills at navigating the environment based on processed frames
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period
- Try non-standard exploration schedules

Table of Contents

Class Structure

- Last time: Value function approximation and deep learning
- This time: Convolutional neural networks and deep RL
- Next time: Imitation learning