Playing Atari with Deep Reinforcement Learning

Overview of Literature

Main DQN Papers

- (2013) “Playing Atari with Deep Reinforcement Learning,” *NIPS Workshop*
- (2015) “Human-Level Control through Deep Reinforcement Learning,” *Nature*
- (2015) “Deep Reinforcement Learning with Double Q-Learning,” *AAAI Conference on AI*
- (2016) “Prioritized Experience Replay,” *ICLR*
- (2016) “Dueling Network Architectures for Deep Reinforcement Learning,” *arXiv*
- (2016) “Asynchronous Methods for Deep Reinforcement Learning,” *ICML*

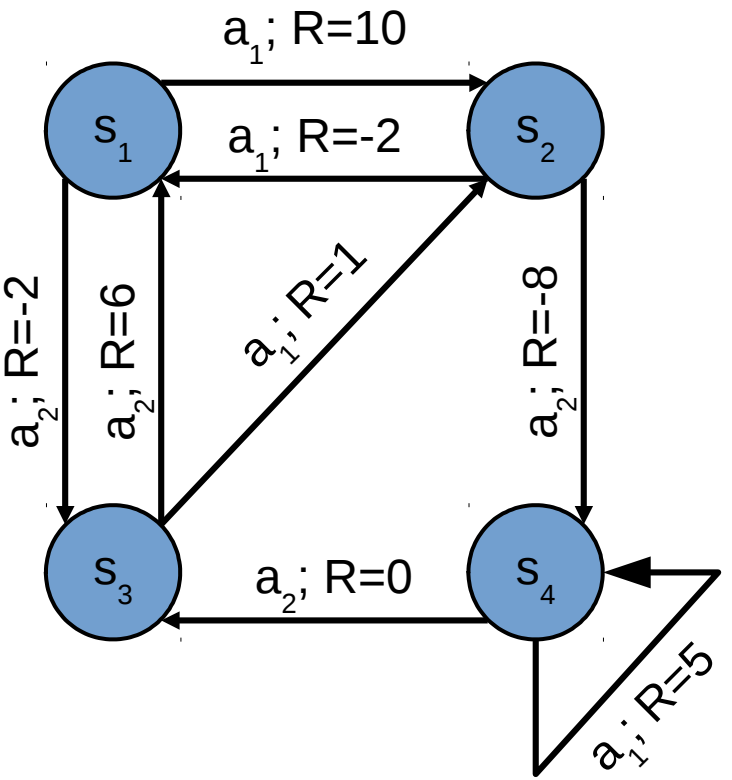
Related Literature

- (2005) “Neural Fitted Q Iteration—First Experiences with a Data Efficient Neural Reinforcement Learning Method,” *ECML*
- (2012) “Imagenet Classification with Deep Convolutional Neural Networks,” *NIPS*
- (2010) “Double Q-Learning,” *NIPS*

Markov Decision Process

Markov Decision Process

- Set of states: $S=\{s_1, s_2, \ldots s_n\}$
- Set of actions: $A=\{a_1, a_2, \ldots a_n\}$
- Transition model: $T=P(s_{t+1}|s_t,a_t)$
  - Markov assumption
  - Not necessarily deterministic
  - May not have (model-free RL)
- Reward function: $R(s_t,a_t)$



Q Learning

Find Optimal Policy

- Assume some *policy*, $a \sim \pi(s) = P(a_t = a \mid s_t = s)$
- Define action-value function $Q^{\pi}(s,a)$ is the expected reward for performing action a in state s, and following policy π in the future

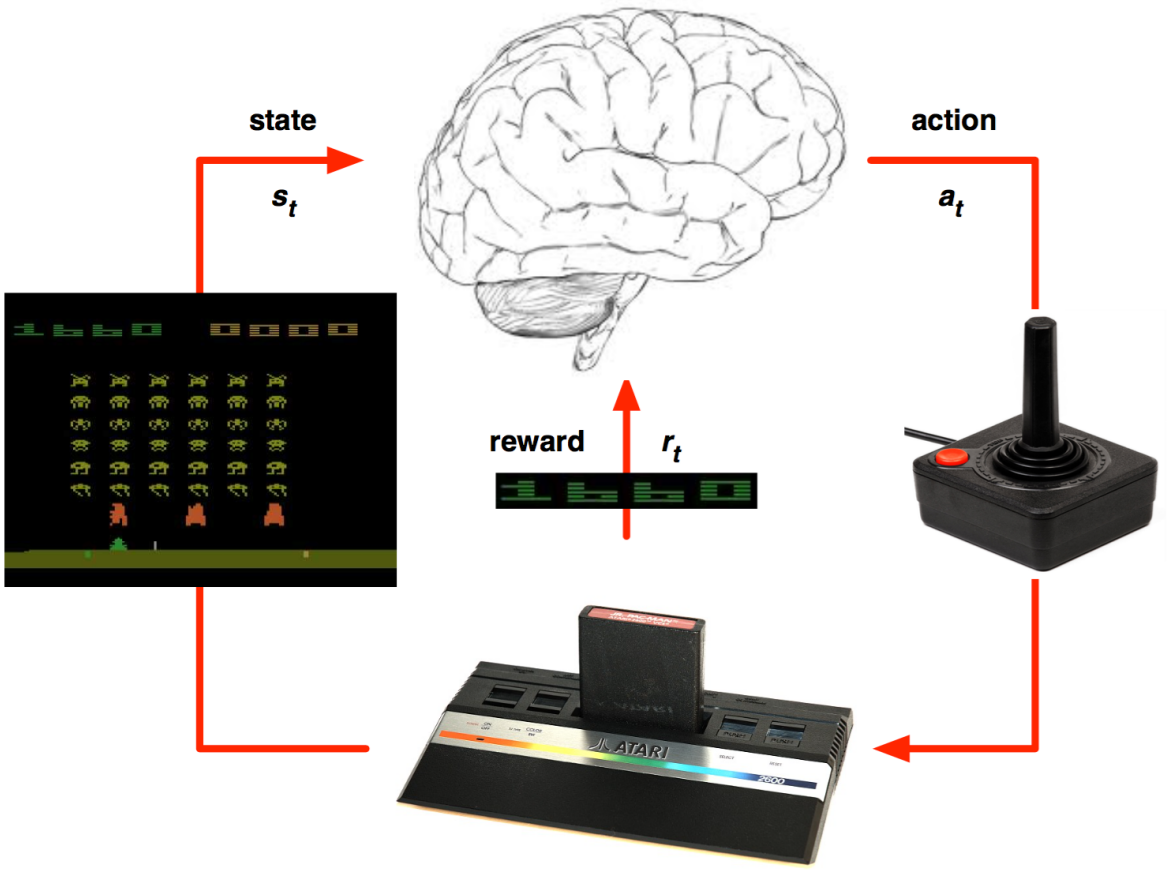
  $Q^{\pi}(s_t,a_t) = E_{\pi}\{\Sigma_t \gamma^t r_t \mid s_t, a_t\} = r_t + \gamma\, Q^{\pi}(s_{t+1},\pi(s))$

  $V^{\pi}(s) = E_{a\sim\pi}\{Q(s,a)\}$
- Optimal action-value function as Bellman equation

  $Q^*(s,a) = R(s,a) + \gamma \max_{a'}(Q^*(s',a'))$
- Can perform value iteration (dynamic programming) to find Q

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Atari Environment



Agent

- State: seq of 4 frames
- Actions:
  - UP, DOWN, FIRE, NOOP, etc.
- Reward: points gained

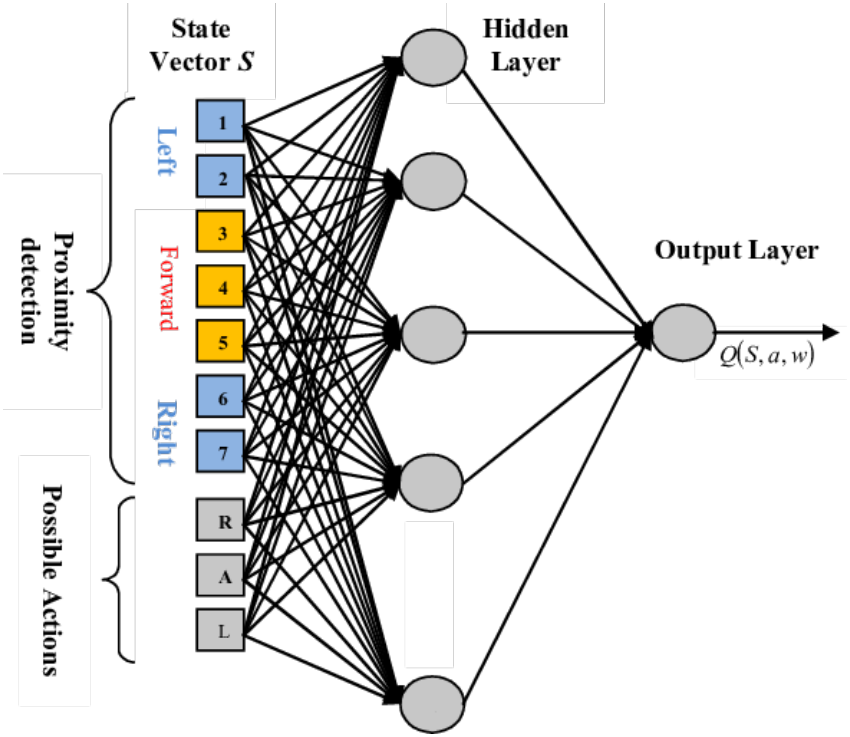
Problem

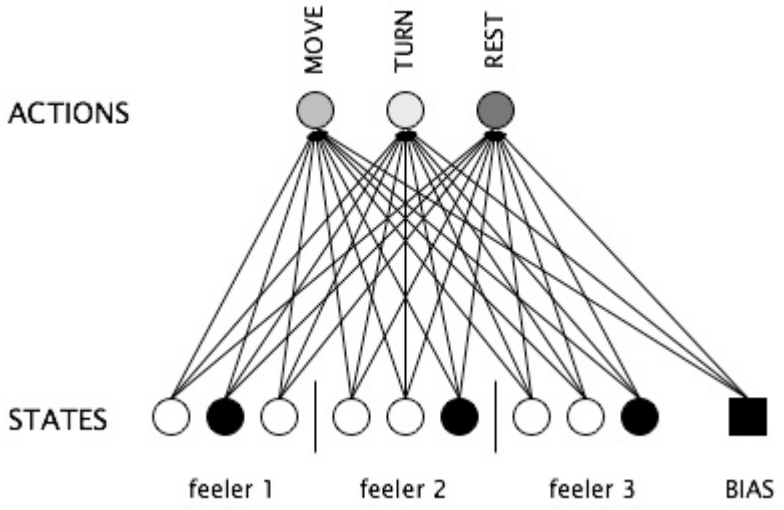
- ***HUGE*** state space
- 84x84 greyscale image, 4 frames
- $|S| = 2^{84*84*4*256}$ = a lot
- value iteration infeasible

Neural Fitted Q Iteration (*Riedmiller, 2005*)

Possible solution:

Train neural network to approximate Q(s,a) *Riedmiller, 2005*





Neural Fitted Q Iteration

Recall the update rule

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Bigg(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Bigg)$$

TD Error

Set loss function of neural network to MSE of TD error

$L = [Q(s_t,a_t) - (r_t + \gamma \max_a Q(s_{t+1},a))]^2$

when $Q(s_t,a_t) = r_t + \gamma \max_a Q(s_{t+1},a)$, loss is zero

Neural Fitted Q Iteration

Algorithm

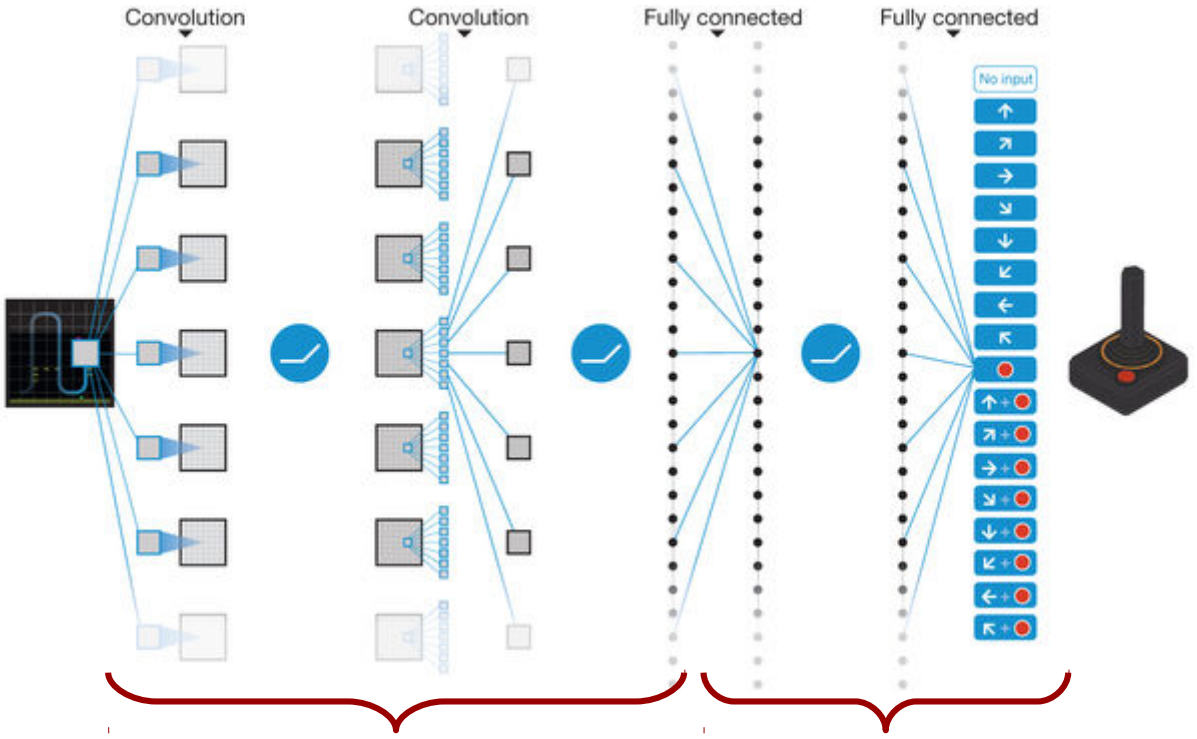
1. Initialize a neural network, Q
2. Initialize empty replay memory, *D*
3. Generate a set of trajectories using Q:
   - Select $a_t = \max_a Q(s_t, a)$
   - Perform a_t on environment, receive reward r_t
   - Store (s_t, a_t, r_t, s_{t+1}) in *D*

   } Assume network provides good approximations of Q to get data
4. For each experience in *D*, calculate target Q value
   - $x = (s_t, a_t)$; $y = r_t + \gamma \max_a Q(s_{t+1}, a)$ ← Make approximation better
5. Train Q with training data (x, y) using backprop
6. Repeat steps 2 – 6

Deep Q Network (*Minh 2013*)

How to learn to play Atari

Use previous 4 frames* as input to the network



CNN – “image” features

FC – “image” features → Q values

* each frame is pixel level max of previous 2 frames; only every 4th frame recorded. This is freaking hard to figure out from the Nature paper if you’re implementing this...

Playing Atari with Deep Reinforcement Learning

Stability Issues with Naive Q Learning

Q-learning can oscillate or diverge

1. Sequential observations → HIGHLY correlated
    - Breaks IID assumption
2. Small changes to Q value can cause rapid change in policy
    - Oscillation in policy
    - Distribution of samples vary rapidly
3. Scale of Q-values unknown
    - Large (and unstable) gradients using backprop

Playing Atari with Deep Reinforcement Learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Fixed Replay Memory size (1 million experiences)

Train using small batch randomly sampled from replay memory (32 samples)

Single gradient descent step (baby step)

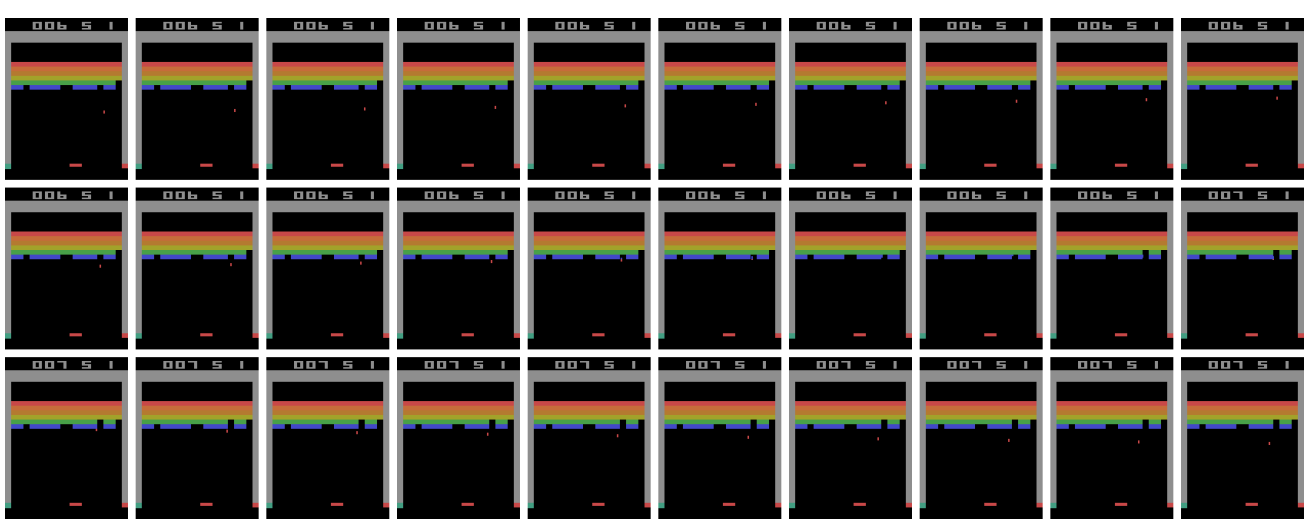
ε-greedy policy: Perform random action with probability (1-ε), anneal from 1.0 to 0.1 over 1 million frames. Allow early exploration

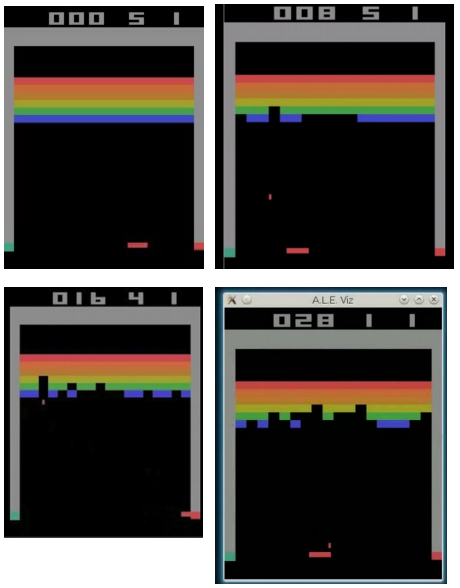
Playing Atari with Deep Reinforcement Learning

Experience Replay

During training, samples are uniformly drawn from the previous 1 million frames (experiences)

Break correlations → restore IID



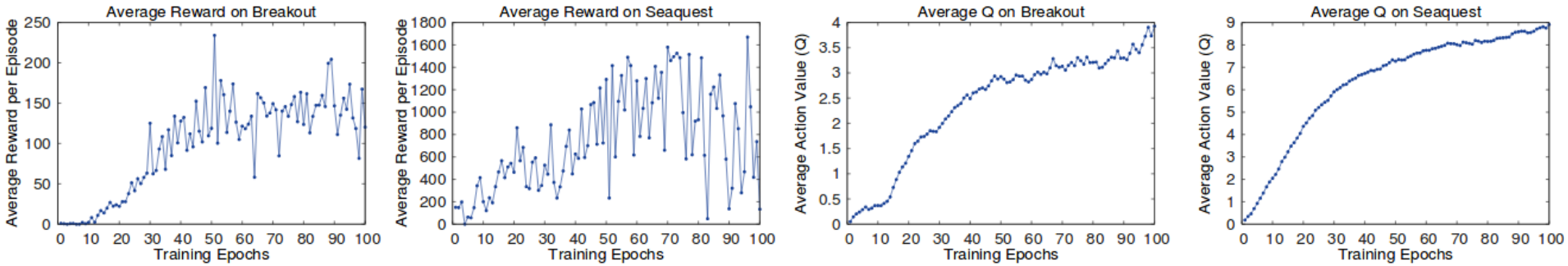


Playing Atari with Deep Reinforcement Learning

Architecture

84x84x4 input image
Layer 1: (16) 8x8 filters, stride 4, ReLU
Layer 2: (32) 4x4 filters, stride 2, ReLU
Layer 3: 256 fully connected units, ReLU

Playing Atari with Deep Reinforcement Learning



Evaluate by running a bunch of games with ε=0.05
Wide variation in average score per game, but stable Q values

Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

Playing Atari with Deep Reinforcement Learning

Results

	Random	Contingency	Human	DQN (NIPS)
Breakout	1.2	6	31	**168**
Seaquest	110	723	**28010**	1705
Space Invaders	179	268	**3690**	581
Pong	-20.4	-17	-3	**20**
Enduro	0	159	368	**470**

Improvements

Major moving parts:

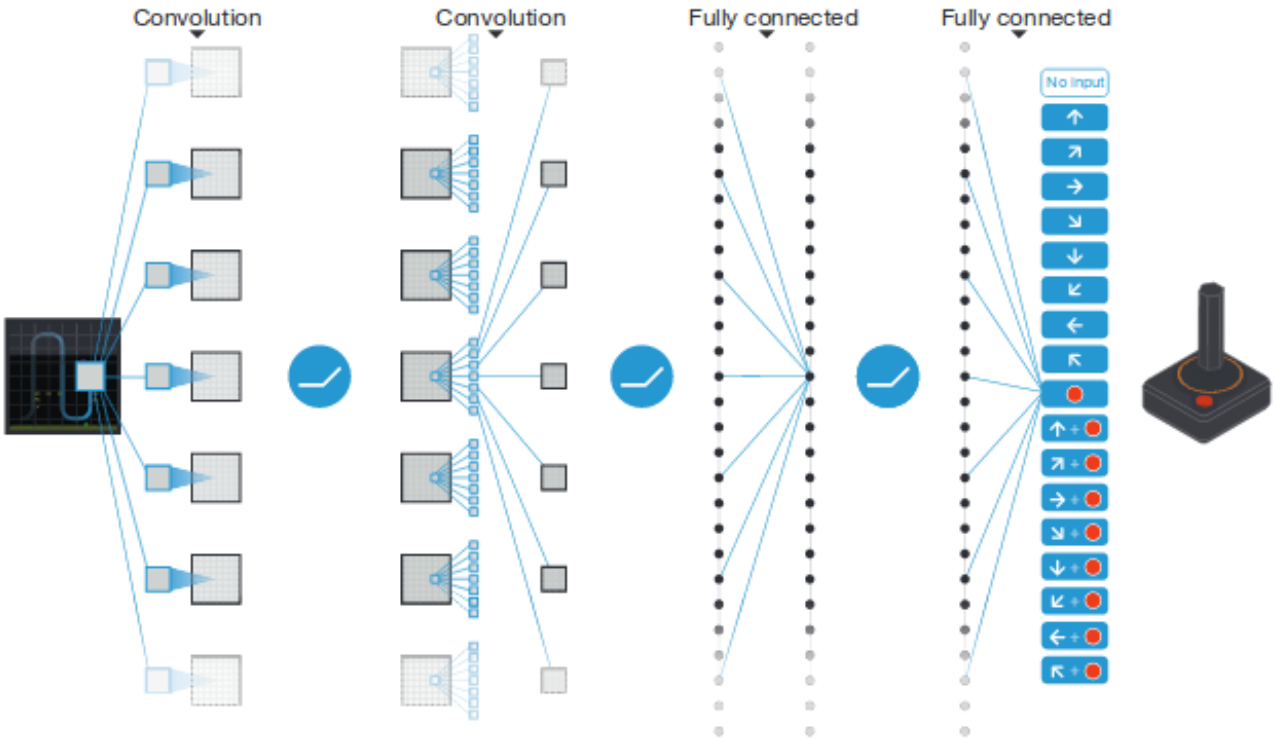
1. Deep-Q Learning Algorithm
   - Human-Level Control through Deep Reinforcement Learning
   - Deep Reinforcement Learning with Double Q-Learning
2. Replay Memory
   - Prioritized Experience Replay
3. Network Architecture
   - Dueling Network Architectures for Deep Reinforcement Learning
4. Agent(s)
   - Asynchronous Methods for Deep Reinforcement Learining

Human-Level Control through Deep Reinforcement Learning

Basically, the same as the NIPS workshop paper, but with:

1. Bigger / deeper neural network
2. Introduce “Target Q Network”
3. Introduce Huber loss to stabilize training
4. More Games
5. Prettier pictures
6. Publish in Nature

Human-Level Control through Deep Reinforcement Learning



Discount Factor
$\gamma = 0.99$

Layer 1: (32) 8x8 filters, stride 4, ReLU
Layer 2: (64) 4x4 filters, stride 2, ReLU
Layer 3: (64) 3x3 filters, stride 1, ReLU
Layer 4: 512 Fully Connected Units, ReLU

RMS-Prop
Learning Rate = 0.00025
Momentum = 0.95

Human-Level Control through Deep Reinforcement Learning

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

Introduce target DQN, same parameters as original DQN

Use DQN to select actions while playing the game

Use target DQN to determine TD error (training targets)

Copy DQN to target DQN every so often

Human-Level Control through Deep Reinforcement Learning

DQN

Target DQN

Start off with two identical DQN

Replay Memory

Human-Level Control through Deep Reinforcement Learning

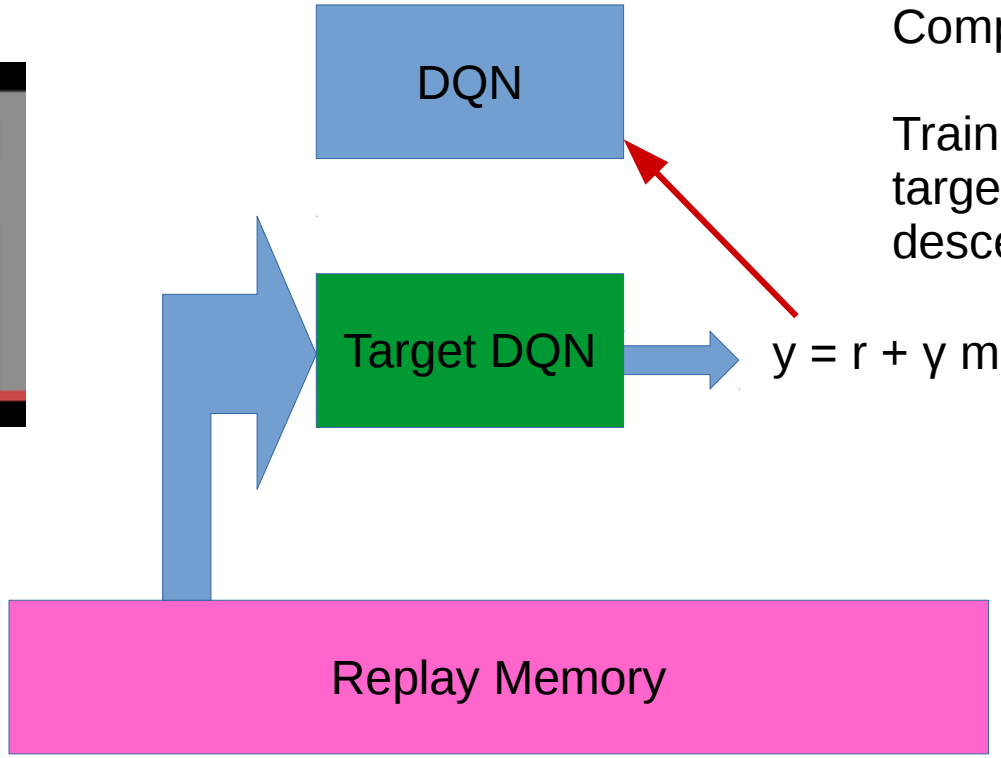
DQN

Target DQN

Use DQN to generate actions each step, Saving experinces to Replay Memory

Replay Memory

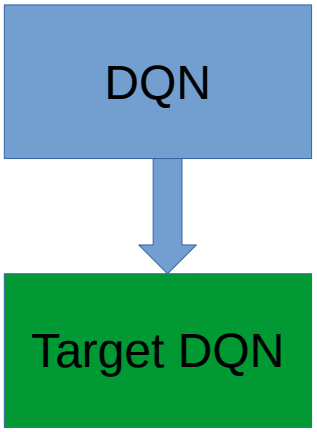
Human-Level Control through Deep Reinforcement Learning



Every 4 steps, extract a batch of 32 experiences. Use Target DQN to Compute target Q value

Train DQN with these targets (one gradient descent update)

Human-Level Control through Deep Reinforcement Learning



Every 10,000 steps, replace target DQN parameters with the DQN parameters

Replay Memory

Human-Level Control through Deep Reinforcement Learning

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Target Q network provides stable Q estimates, while DQN can rapidly change values (and therefore policy)
--Not using Target Q reduces test scores

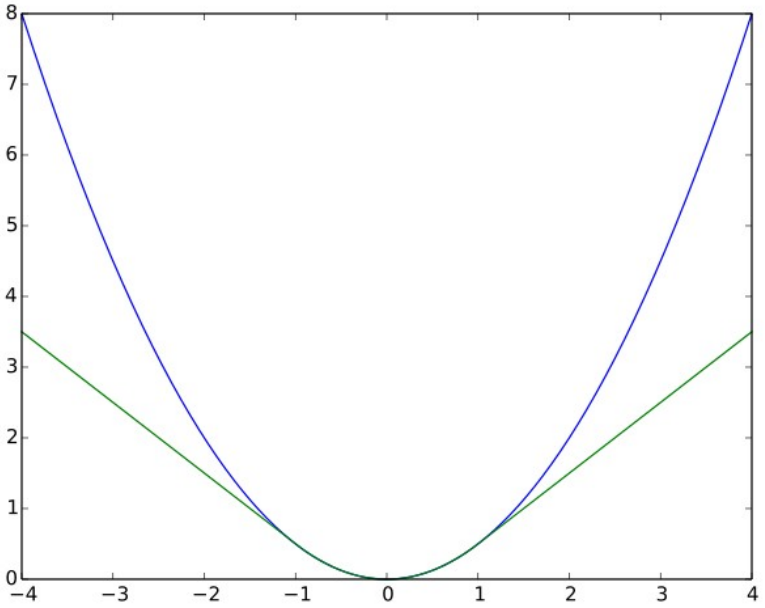
Replay memory de-correlates experiences. Not doing this *greatly* reduces score

Human-Level Control through Deep Reinforcement Learning

Huber Loss

Large gradients could drastically change Q network

Reduce effect of big errors by using L1 norm instead of L2 norm when |err| > 1



Playing Atari with Deep Reinforcement Learning

Results

	Human	DQN (NIPS)	DQN (Nature)
Breakout	31.8	168	**401.2**
Seaquest	**20182**	1705	5286
Space Invaders	1652	581	**1976**
Pong	9.3	**20**	18.9
Enduro	368	**470**	301.8

Playing Atari with Deep Reinforcement Learning

Stability Issues with Naive Q Learning

Q-learning can oscillate or diverge

1. ~~Sequential observations → HIGHLY correlated~~
   - Replay Memory
2. ~~Small changes to Q value can cause rapid change~~ in policy
   - Target Q network
3. ~~Scale of Q-values unknown~~
   - Huber loss / clip gradient

Deep Reinforcement Learning with Double Q Learning

Target Q function

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) .$$

Assuming Q is estimated by the Target Q network, the max operator has the Target Q network both choose what the best action is, as well as evaluate the value of that action

$$Y_t^{\text{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t) .$$

Pick the best action

Evaluate the best action

This can lead to the Q network *overestimating* the value of Q

- Bias the policy towards the selected action

Deep Reinforcement Learning with Double Q Learning

Solution

Have a separate Q network select the action, and the target Q network evaluate the action

- Decouple action selection from evaluation
- Luckily, we have the other DQN (non-target) to select the action

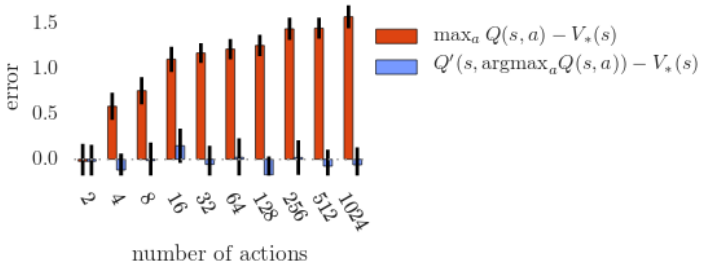


Figure 1: The orange bars show the bias in a single Q-learning update when the action values are $Q(s,a) = V_*(s) + \epsilon_a$ and the errors $\{\epsilon_a\}_{a=1}^m$ are independent standard normal random variables. The second set of action values Q', used for the blue bars, was generated identically and independently. All bars are the average of 100 repetitions.

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) .$$

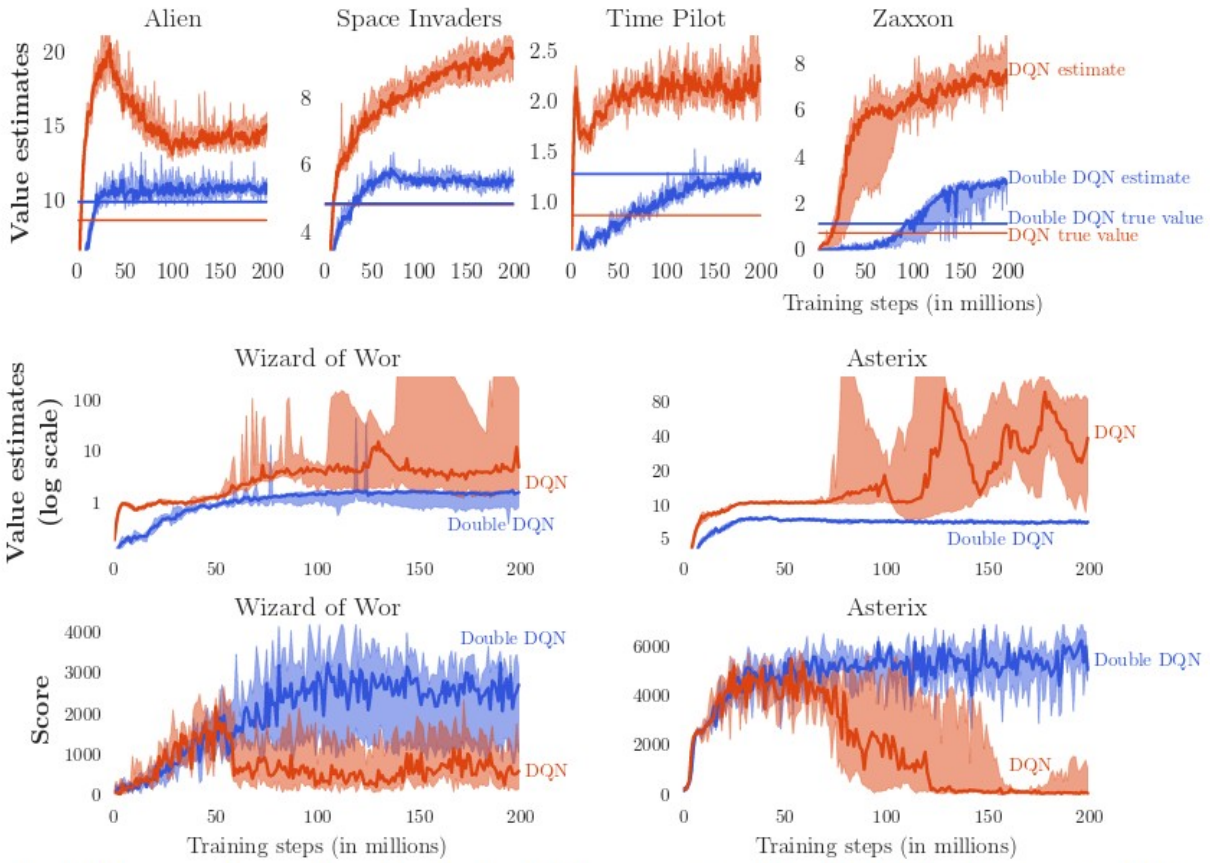
DQN: Pick the best action

Target DQN: Evaluate the best action

- Change 1 line of agent code

Deep Reinforcement Learning with Double Q Learning

Benefit



Deep Reinforcement Learning with Double Q Learning

Results

	Human	DQN (NIPS)	DQN (Nature)	D-DQN
Breakout	31.8	168	**401.2**	375
Seaquest	**20182**	1705	5286	7995
Space Invaders	1652	581	1976	**3156**
Pong	9.3	20	18.9	**21**
Enduro	368	**470**	301.8	319.5

Priority Experience Replay

During training, experiences are sampled uniformly from the Replay Memory

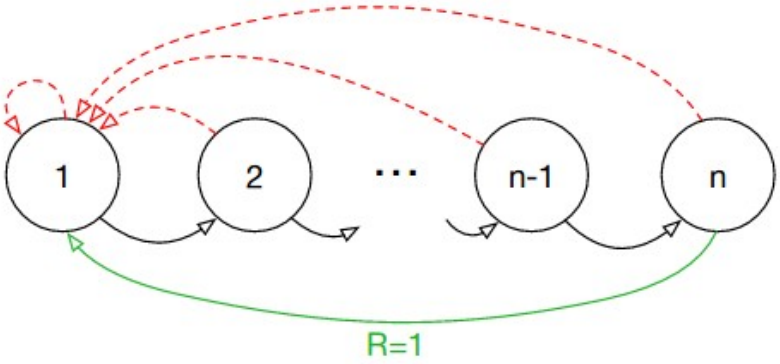
When the Target Q is calculated from these, the TD error for some experiences will be larger than for others…

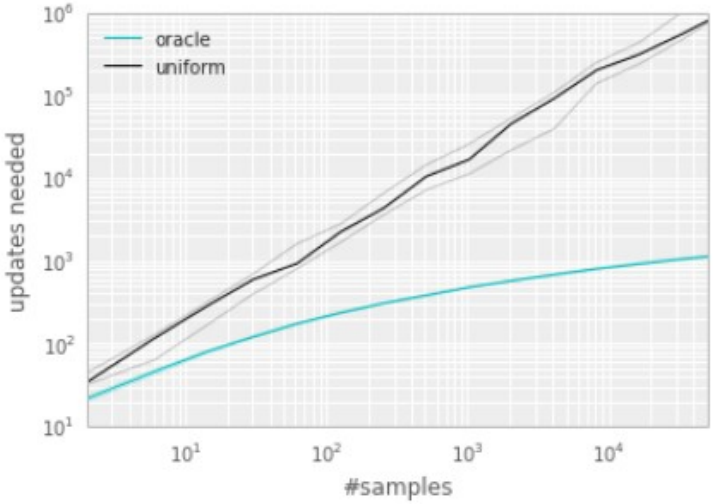
The Q network will learn more from experiences with higher TD errors than lower

- i.e., larger gradients

Priority Experience Replay

Example – Blind Cliffwalk





Sparse Reward

If we know the best examples to learn from (oracle), then can have logarithmic speedup in the number of training updates needed (over uniform

- Can sampling method be adjusted to try to get closer to this?

Priority Experience Replay

Approach

Set a prioritization value for each transition, p_i

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- Initially set this to 1, set to TD-error if selected
- Alternatively, sort replay memory using p_i, and use 1/rank(i) as prioritization value (rank-based)

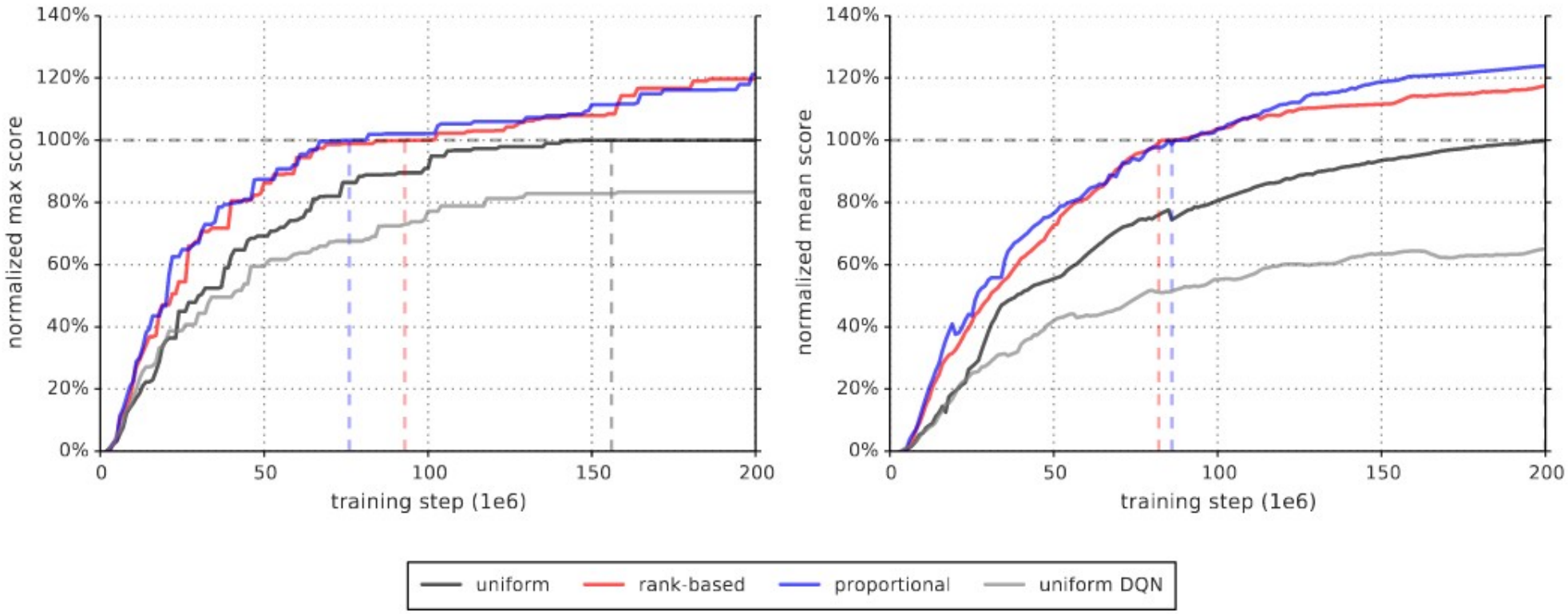
Normalize prioritization values to determine sampling probability when selecting experience

- Introduces biases in the expected value updates, as the sampling distribution is not the same as uniform
- Fix with importance sampling weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

Linearly anneal β from 0.4 to 1
At 1, bias is fully compensated

Priority Experience Replay



Priority Experience Replay

Results

	Human	DQN (NIPS)	DQN (Nature)	D-DQN	PER
Breakout	31.8	168	401.2	375	**481.1**
Seaquest	20182	1705	5286	7995	**39096.7**
Space Invaders	1652	581	1976	3156	**9063**
Pong	9.3	20	18.9	**21**	18.9
Enduro	368	470	301.8	319.5	**1884.4**

Dueling Network Architectures for Deep Reinforcement Learning

So far, no significant changes to network architecture

- Multilayer CNN

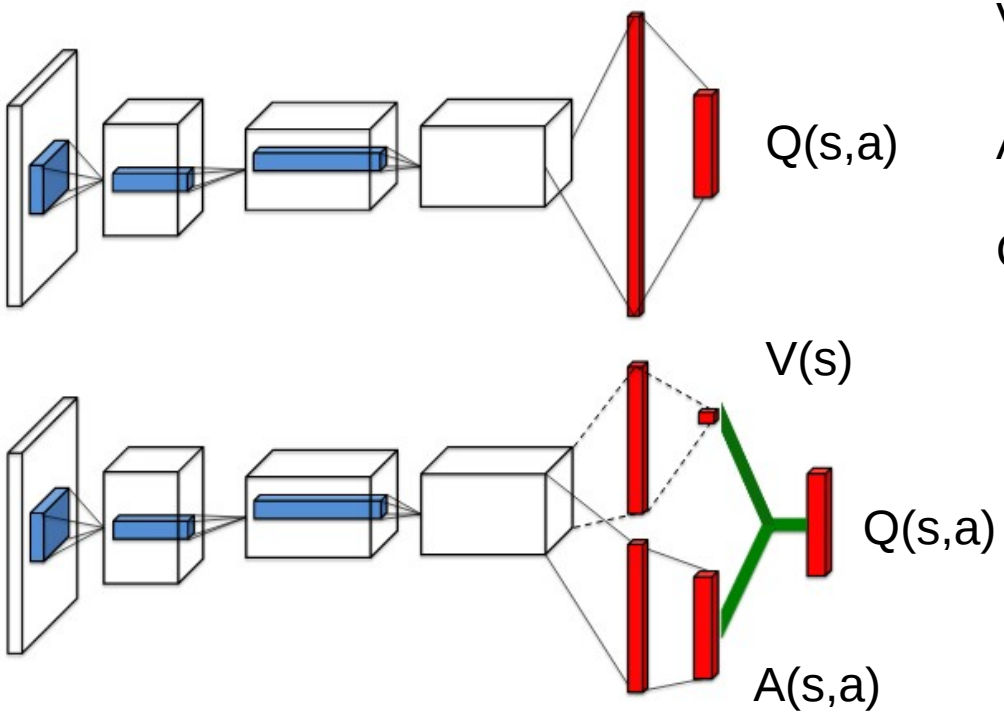
Also, what to do when there is no clear “good” action?

- Updates affect Q value of individual action, Q(s,a)
- Doesn’t consider the value of the state, V(s)
- Forcing update of a single action in this case can bias towards or away from this action in the policy

Dueling Network Architectures for Deep Reinforcement Learning

Approach: Create two streams in the network

- One calculates the value function of the state, V(s)
- One calculates the advantage of the action, A(s,a)

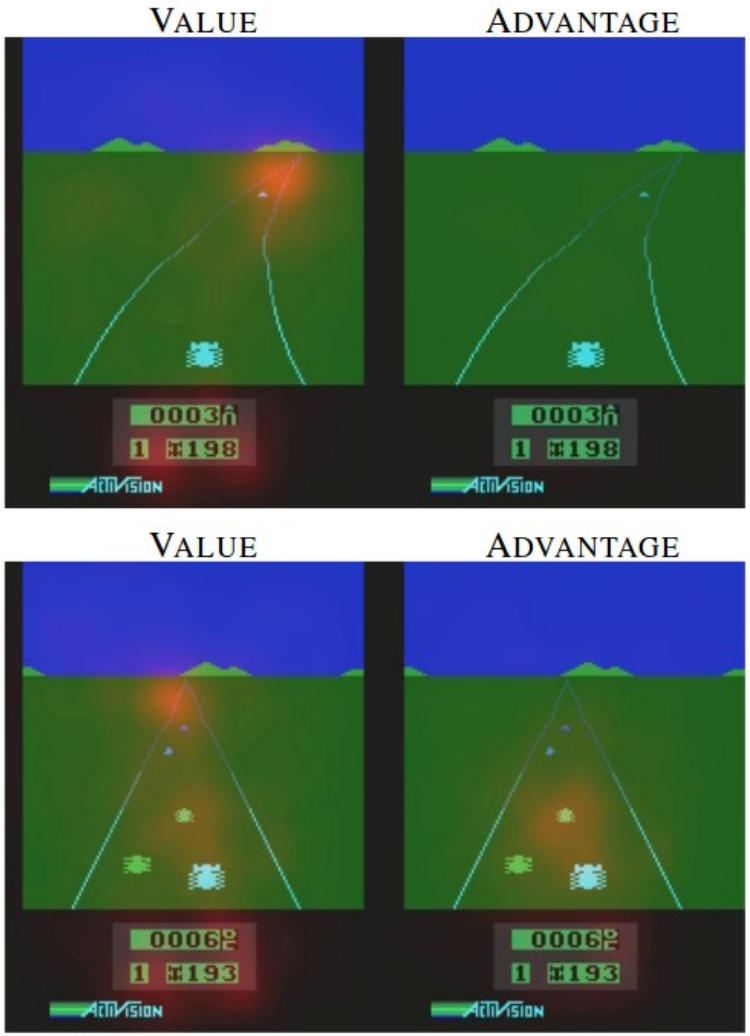


$V(s) = \max_a Q(s,a)$

$A(s,a) = Q(s,a) - V(s,a)$

$Q(s,a) = V(s) + (A(s,a) - (1/|A|)\Sigma_{a'} A(s,a))$

Dueling Network Architectures for Deep Reinforcement Learning



No immediate benefit to moving left or right, rely instead on the state-value

Going right is definitely better than left or straight, rely on advantage function

Priority Experience Replay

Results

	Human	DQN (NIPS)	DQN (Nature)	D-DQN	PER	Duel
Breakout	31.8	168	401.2	375	**481.1**	366
Seaquest	20182	1705	5286	7995	39096.7	**50254.2**
Space Invaders	1652	581	1976	3156	9063	**15311.5**
Pong	9.3	20	18.9	**21**	18.9	**21**
Enduro	368	470	301.8	319.5	1884.4	**2306.4**

Github Repo

http://github.com/danathughes/AtariRL

Implements:
- Deep Q Network (Nature Paper algorithm)
- Double Deep Q Learning
- Dueling Networks

Working on:
- Priority Experience Replay