Function Approximation Methods

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : easwar.subramanian@tcs.com / cs5500.2020@iith.ac.in

September 23, 2022

Overview

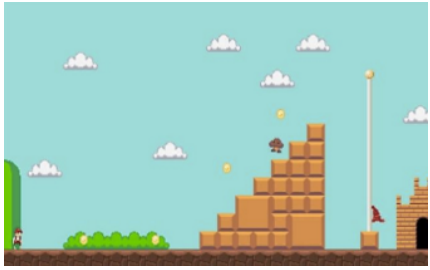
TCS Research & Innovation

1. Function Approximation Methods
2. Convergence of Approximation Methods
3. Towards a Stable Deep Q Network Algorithm
4. Efficacy of DQN Algorithm

Function Approximation Methods

On the need for Function Approximators

- To solve large scale RL problems
  - ★ Game of Backgammon : 10^{20} states
  - ★ Game of Go : 10^{170} states
  - ★ Even Atari games have large state space



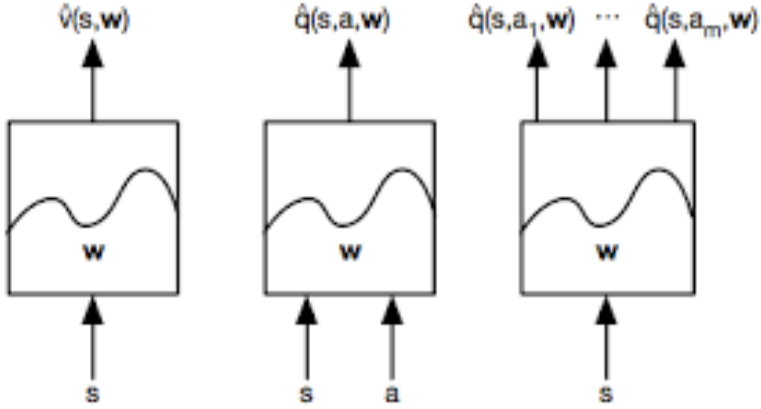
$|\mathcal{S}|$ is very large : Curse of Dimensionality

Slide Credit: David Silver : UCL lectures

Value Function Approximators

- Value function have been basically lookup tables.
- Solution for large MDP's is to use function approximators
  - Generalize from seen to unseen states
- Function approximators could be
  - Linear function approximator
  - Neural networks
  - Decision tree
  - ...

Neural Network Approximators

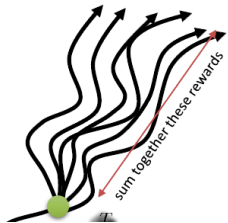


Policy Evaluation Using Neural Networks

The value of a policy π is given by

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} | s_t = s\right) \\ &= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right] \end{aligned}$$

Question : How do we compute the above expectations using neural networks ?



- Roll-out m trajectories from state s and observe rewards

Figure Source: Sergey Levine : UCB Course

Value Function Fitting using Monte Carlo

- Consider a MDP with a finite horizon H

$$V^{\pi}(s) \approx \frac{1}{m}\left[\sum_{j=1}^{m}\left[\sum_{k=0}^{H}\left(\gamma^k r^i_{t+k+1}|s_t = s\right)\right]\right]$$

  Need to reset the simulator back to state s (Not always possible)

- Alternative : Roll-out single sample estimate (high variance, but OK)
- Collect training data for as many states as possible and regress thereafter

$$\left(s_i, \underbrace{\left[\sum_{k=t}^{H}\left(\gamma^k r_{t+k+1}|s_t = s\right)\right]}_{=y_i}\right)$$

Algorithm Monte Carlo Based Value Function Fitting

Initialize number of iterations N

for $i = 1$ to N **do**

Perform a roll-out from an initial state s_i (could be any state from $\mathcal{S}$)

Calculate targets y_i using Monte-Carlo roll outs

$$y_i = \left[\sum_{k=0}^{H} \left(\gamma^k r_{t+k+1}^i | s_t = s_i \right) \right]$$

Form input-output pairs (s_i, y_i) (N datapoints in total)

end for

Perform supervised regression with loss function

$$L(\phi) = \frac{1}{2} \sum_{i=1}^{N} \left[V_\phi^\pi(s_i) - y_i \right]^2$$

Policy Evaluation : MC Based Algorithm

- Needs complete sequences, suitable only for episodic tasks

Fitted V Iteration

We observe transition (s, a, r, s') at time t; Using one step look-ahead,

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[r + \gamma V^{\pi}(s') | s_t = s\right] \\ &\approx r + \gamma V^{\pi}(s') \text{ (Bootstrap } V^{\pi}) \end{aligned}$$

Using function approximators, we get,

$$V_{\phi}^{\pi}(s) \approx r + \gamma V_{\phi}^{\pi}(s')$$

- Directly use the previous fitted value function V_{ϕ}^{π}
- Collect training data,

$$\left(s_i, \underbrace{r + V_{\phi}^{\pi}(s_i')}_{=y_i}\right)$$

- Perform supervised regression

$$L(\phi) = \frac{1}{2}\sum_{i=1}^{N}\left[V_{\phi}^{\pi}(s_i) - y_i\right]^2$$

Algorithm Fitted V Iteration

1: Initialize number of iterations N
2: **for** $j = 1$ to N **do**
3: Sample K transitions (s, a, r, s') using policy π
4: **for** $i = 1$ to K **do**
5: Calculate targets y_i using one step TD approximation

$$y_i = \left[r + V^{\pi}_{\phi_j}(s'_i) \right]$$

6: Form input-output pairs (s_i, y_i) (K datapoints in total)
7: **end for**
8: Perform supervised regression (Optimizer : RProp) using loss function

$$L(\phi_j) = \frac{1}{2} \sum_{i=1}^{K} \left[V^{\pi}_{\phi_j}(s_i) - y_i \right]^2$$

and get a new function approximator with new weights ϕ_{j+1}
9: **end for**

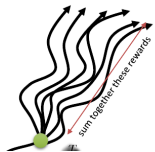
Optimal Value Function : Control

Bellman optimality equation for V_* is given by,

$$V_*(s) \leftarrow \max_a \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_*(s') \right) \right] \approx \max_a E \left[r_{t+1} + \gamma V_*(s_{t+1}) | s_t = s \right]$$

Question : How do we get a sample estimate for transition (s, a, r, s') for V_* ?

$$V(s) \approx \max_a [r + \gamma V(s')]$$



- To compute max over a, we need to know the outcome of all actions starting from s. Mostly not possible and costly as well.
- For model free control, we use approximators for Q and not V

Fitted Q Iteration

Bellman optimality equation for Q_*

$$Q_*(s,a) = \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q_*(s', a')\right)\right] \approx \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q_*(s_{t+1}, a') | s_t = s, a_t = a\right]$$

- Max is inside the expectation; that's ok
- For transitions (s, a, r, s') we can compute $r + \gamma \max_{a'} Q(s', a')$
- Does not require simulating over actions
- Use the previous fitted optimal Q function Q^*_ϕ like in fitted V iteration
- Collect training data,

$$\left(s_i, \underbrace{r + \gamma \max_{a'} Q_\phi(s'_i, a'_i)}_{=y_i}\right)$$

- Perform supervised regression

$$L(\phi) = \frac{1}{2} \sum_{i=1}^{N} \left[Q_\phi(s_i, a_i) - y_i\right]^2$$

Algorithm Fitted Q Iteration

1: Initialize number of iterations N
2: **for** $j = 1$ to N **do**
3: Sample K transitions (s, a, r, s') using any behaviour policy μ
4: **for** $i = 1$ to K **do**
5: Calculate targets y_i using one step TD approximation

$$y_i = \left[r + \gamma \max_{a'} Q_{\phi_j}(s'_i, a') \right]$$

6: Form input-output pairs $(s_i, , y_i)$ (K Datapoints in total)
7: **end for**
8: Perform supervised regression (Optimizer : RProp) using loss function

$$L(\phi_j) = \frac{1}{2} \sum_{i=1}^{K} \left[Q_{\phi_j}(s_i, a_i) - y_i \right]^2$$

and get a new function approximator with new weights ϕ_{j+1}
9: **end for**

Convergence of Approximation Methods

On the Convergence of Fitted Iterations

Question : What can we say about the convergence of fitted iteration methods ?

- Does fitted V iteration converge to V^π ?
- Does neural fitted iteration converge to Q_* ?

Convergence in DP setup

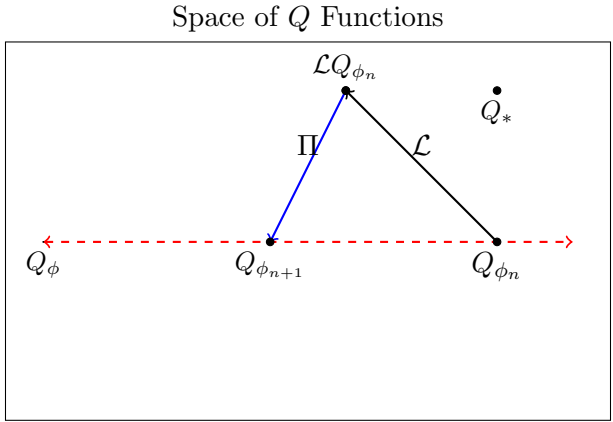
- Use the fixed point equation below to define a **contraction** operator $\mathcal{L}$ (contraction in L_∞ norm)

$$Q_*(s,a) \leftarrow \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q_*(s', a') \right) \right]$$

Convergence in TD setup

- State and action spaces are finite
- All state-action pairs are visited infinitely often
- Robbins-Monroe condition: $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$

Projections and Convergence



Convergence Guarantee For Fitted Iteration Methods

- Define operator $\mathcal{L} : \mathcal{Q} \rightarrow \mathcal{Q}$ such that

$$\mathcal{L}Q = r + \gamma \max_{a'} Q(s', a')$$

- Backup operator $\mathcal{L}$ is a contraction in L_∞ norm
- Projection operator (Π) are contractions in L_2 norm
- What about the composition $(\Pi \circ \mathcal{L})Q$?
  - ★ Need not be a contraction with respect to any norm

Sad Corollary

No guarantees on convergence to optimal value functions (on the manifold) exist for fitted iteration methods

Algorithm Monte Carlo Based Value Function Fitting

1: Initialize number of iterations N
2: **for** $i = 1$ to N **do**
3: Perform a roll-out from an initial state s_i (could be any state from $\mathcal{S}$)
4: Calculate targets y_i using Monte-Carlo roll outs

$$y_i = \left[\sum_{k=0}^{H} \left(\gamma^k r^i_{t+k+1} | s_t = s_i \right) \right]$$

5: Form input-output pairs (s_i, y_i) (N datapoints in total)
6: **end for**
7: Perform supervised regression with loss function

$$L(\phi) = \frac{1}{2} \sum_{i=1}^{N} \left[V^{\pi}_{\phi}(s_i) - y_i \right]^2$$

Convergence of Monte Carlo Based Algorithm

- Step 7 is gradient descent and it will converge at least local optimum
- Important : **Convergence guarantee is in the parameter space (ϕ) and not in value function space**

Algorithm Fitted Q Iteration

1: Initialize number of iterations N
2: **for** $j = 1$ to N **do**
3: Sample K transitions (s, a, r, s') using any behaviour policy μ
4: **for** $i = 1$ to K **do**
5: Calculate targets y_i using one step TD approximation

$$y_i = \left[r + \gamma \max_{a'} Q_{\phi_j}(s'_i, a') \right]$$

6: Form input-output pairs $(s_i, , y_i)$ (K Datapoints in total)
7: **end for**
8: Perform supervised regression (Optimizer : RProp) using loss function

$$L(\phi_j) = \frac{1}{2} \sum_{i=1}^{K} \left[Q_{\phi_j}(s_i, a_i) - y_i \right]^2$$

and get a new function approximator with new weights ϕ_{j+1}
9: **end for**

Online Q learning / Incremental Q learning

Question : Can we do the gradient update for every transition (s, a, r, s') ?

- We use the fitted Q iteration and set $K = 1$
- This is also the Watkins Q-learning update (used with function approximators)

Algorithm Online Q Learning

1: **for** $n = 1$ to N **do**
2: Take an action a and obtain the transition (s, a, r, s') using ϵ-greedy policy
3: Calculate target y using one step TD approximation

$$y = \left[r + \gamma \max_{a'} Q_{\phi_n}(s', a') \right]$$

4: Compute $g^{(n)} = \nabla_{\phi}(Q_{\phi_n}(s, a) - y)^2$
5: Set $\phi_{n+1} = \phi_n - \alpha g^{(n)}$
6: **end for**

Convergence Guarantee on Online Q learning

Algorithm Online Q Learning

1: **for** $n = 1$ to N **do**
2: Take an action a and obtain the transition (s, a, r, s') using ϵ-greedy policy
3: Calculate target y using one step TD approximation

$$y = \left[r + \gamma \max_{a'} Q_{\phi_n}(s', a') \right]$$

4: Compute $g^{(n)} = \nabla_\phi (Q_{\phi_n}(s, a) - y)$
5: Set $\phi_{n+1} \leftarrow \underbrace{\phi_n - \alpha g^{(n)}}_{\text{Is this GD ?}}$
6: **end for**

- Take a closer look at the one step gradient

$$g^{(n)} \leftarrow \phi_n - \alpha \nabla_\phi (Q_\phi(s, a) - \underbrace{\boxed{r + \gamma \max_{a'} Q_\phi(s', a')}}_{\text{moving target}})$$

Summary : Convergence Discussion

- Projection (Π) of the backup operator ($\mathcal{L}$) of optimal Q function need not be a contraction in any norm
- Fitted V iteration or fitted Q iteration need not converge because of the moving target problem
- In online Q learning algorithm,
  - ★ Samples obtained are sequentially correlated
  - ★ Moving target problem
- **Convergence guarantees exist only in tabular case**

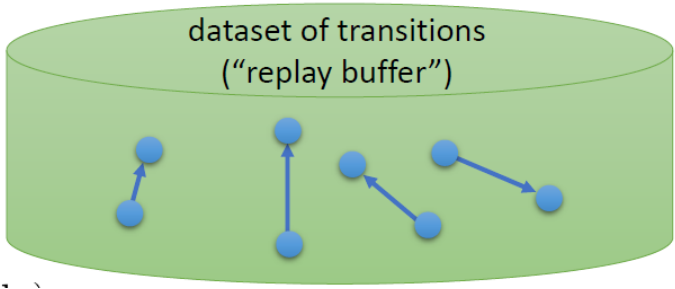
Towards a Stable Deep Q Network Algorithm

Desiredata

- Online algorithm like Q-learning in tabular case
- No sequential correlation in data samples
- Some stability with respect to gradient updates

Replay Buffers

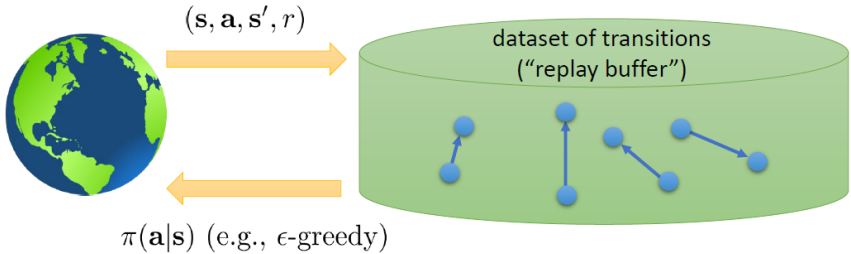
- Use the idea from fitted Q-iteration to collect and store transitions $(s, a.s', r)$



- Stored transition dataset is called **Replay Buffer** denoted by D
- Replay buffers are of fixed size (N)

Figure Source: Sergey Levine : UCB

Replay Buffers



- In an online setting, use ϵ-greedy policy to periodically feed the buffer with newer experiences
- Use FIFO like mechanism to maintain size
- Sample a random minibatch of transitions (B transitions) to perform gradient descent (random sampling ensure samples for SGD are no longer correlated)
- Variance of the gradient estimate is also low compared to gradient computed using one sample

Figure Source: Sergey Levine : UCB

Moving Target Problem - Target Networks

- Use an older set of weights to compute the targets
- Called **Target Network**
- Loss term is given by

$$L_i(\phi_i) = \left[\mathbb{E}_{(s,a,r,s') \in D} \left(Q_{\phi_i}(s,a) - \underbrace{r + \max_{a'} Q_{\phi_i'}(s',a')}_{\text{target}} \right)^2 \right]$$

- Target network is kept constant for a while (every C steps) before being changed
  - ★ Every C steps the weights of the original network is copied to target network

DQN Algorithm

Algorithm DQN Algorithm

Intialize replay memory D to capacity N
Initialize action value function Q with parameters ϕ
Initialize target action value function $\widehat{Q}$ with parameters $\phi' = \phi$
for episodes $= 1$ to M **do**
    Initialize start state s_1
    for steps $t = 1$ to T **do**
        Select action a_t using ϵ-greedy policy
        Execute action a_t and store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample random minibatch (size B) of transitions from D
        for b $= 1$ to B **do**
            Calculate targets for each transitions (Bellman backup or reward)
        end for
        Perform a gradient descent step on $(y_i - Q_\phi(s_t, a_t))^2$ w.r.t ϕ
        Every C steps set $\widehat{Q} = Q$
    end for
end for

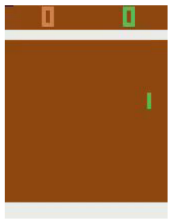
Efficacy of DQN Algorithm

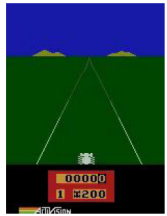
Historical Notes [2]

- Mnih et al. introduced Deep Q-Network (DQN) algorithm, applied it to ATARI games
- Used deep learning / ConvNets, published in early stages of deep learning craze (one year after AlexNet)
- Popularized ATARI (Bellemare et al., 2013) as RL benchmark
- Outperformed baseline methods, which used hand-crafted features

[2] Slide content from Schulman

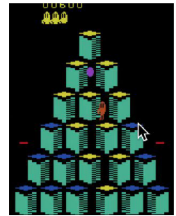
DQN on Atari [2]


Pong



Enduro

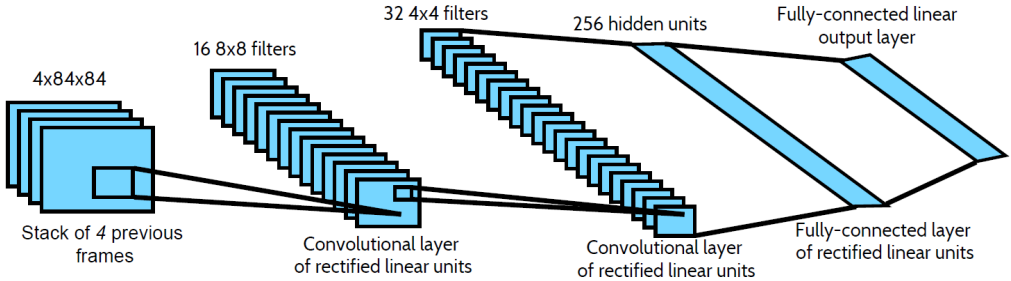
Beamrider


Q*bert

- 49 ATARI 2600 games
- From pixels to actions
- The change in score is the reward
- Same algorithm
- Same function approximator
- Same hyperparameters
- Roughly human-level performance on 29 out of 49 games

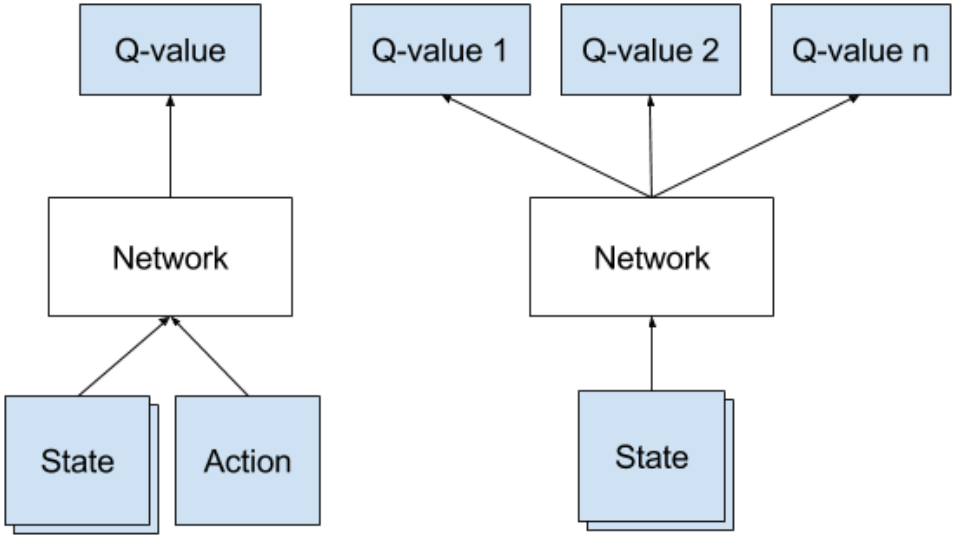
[2] Slide content from Minh

Atari DQN Architecture

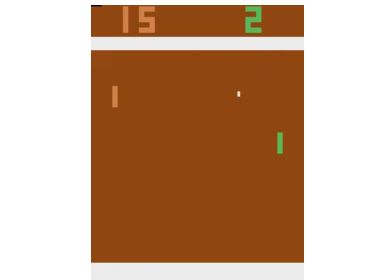


- Convolutional neural network architecture
- History of 4 frames as input
- One output per action ($Q(s, a)$) – expected reward for action a

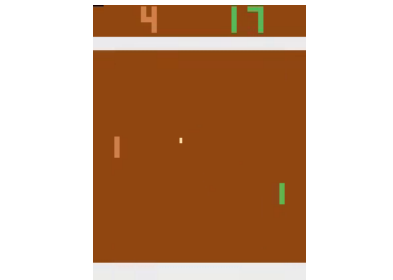
Profile of Q Function Approximator



Demonstration - Ping Pong

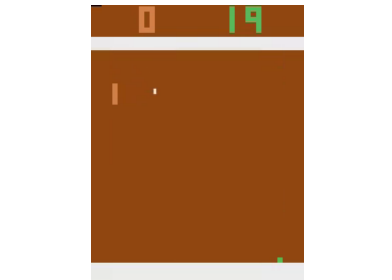


Random Policy

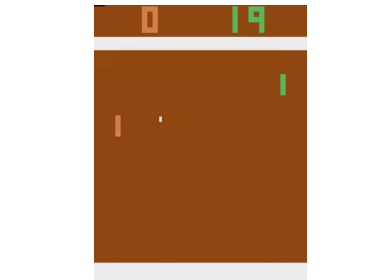


After 5.2 Millon Epochs

Demonstration - Ping Pong

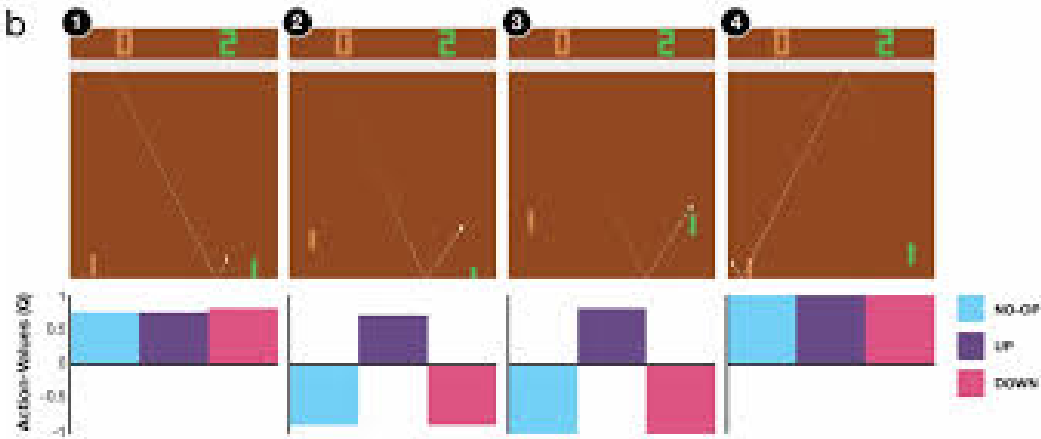


After 8 Million Epochs

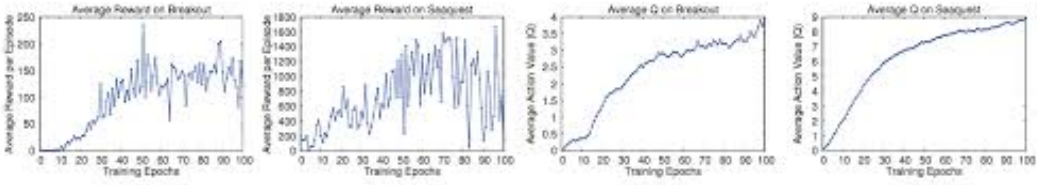


After 9.5 Millon Epochs

Are the Q-Values Meaningful ?



On Tracking the Training Process



Practical Tips for DQN [2]

- DQN is more reliable on some tasks than others. Test your impementation on reliable tasks like Pong and Breakout: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is important
- DQN converges slowly - for ATARI it is often necessary to wait for 10-40 million frames (couple of hours to a day of training on GPU) to see results significantly better than random policy. **Be Patient**
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period

[2] Slide content from Schulman