Deep Q Networks

Easwar Subramanian

TCS Innovation Labs, Hyderabad

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

October 11, 2021

Overview

1. Review

2. Convergence of Approximation Methods

3. Towards a Stable Deep Q Network Algorithm

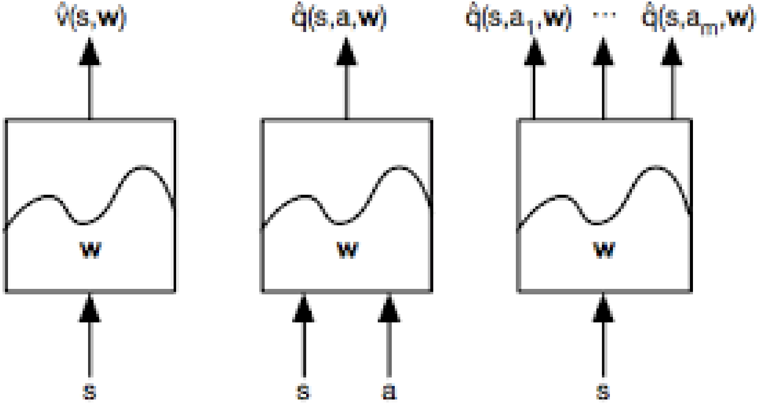
4. Efficacy of DQN Algorithm

Review

Value Function Approximators

- Value function evaluation in tablular RL methods 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



Slide Credit: David Silver : UCL

Policy Evaluation Using Neural Networks

The value of a policy π is given by

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

We collect training data by evaluating the experiences using samples

- Value function fitting using Monte Carlo
- Fitted V Iteration

Optimal Value Fuction : Control

- For transitions (s, a, r, s') we can compute targets as $r + \gamma \max_{a'} Q(s', a')$
- 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

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

Fitted Q Iteration : Algorithm

Algorithm Fitted Q Iteration

```
Initialize number of iterations $N$
for $j = 1$ to $N$ do
    Sample $K$ transitions $(s, a, r, s')$ using any behaviour policy $\mu$
    for $i = 1$ to $K$ do
        Calculate targets $y_i$ using one step TD approximation
            $y_i = \left[ r + \gamma \max_{a'} Q_{\phi_j}(s'_i, a') \right]$
        Form input-output pairs $(s_i, , y_i)$ ($K$ Datapoints in total)
    end for
    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 $\phi_{j+1}$
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)

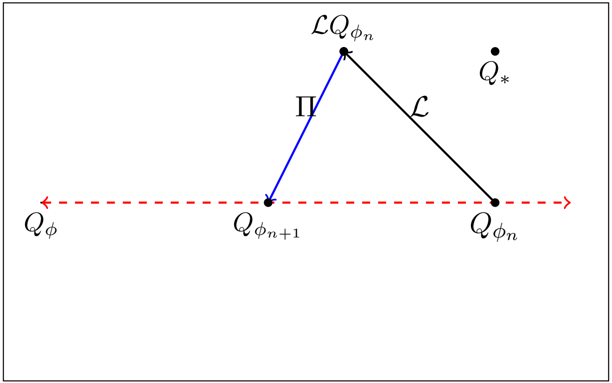
$$Q_*(s,a) \leftarrow \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q_*(s',a') \right) \right]$$

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

Space of Q Functions



Convergence Guarantee For Fitted Iteration Methods

- Define operator $\mathcal{L} : \mathcal{Q} \to \mathcal{Q}$ 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

Convergence of Monte Carlo Based Algorithm

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

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**

Fitted Q Iteration

Algorithm Fitted Q Iteration

```
Initialize number of iterations $N$
for $j = 1$ to $N$ do
    Sample $K$ transitions $(s, a, r, s')$ using any behaviour policy $\mu$
    for $i = 1$ to $K$ do
        Calculate targets $y_i$ using one step TD approximation
            $y_i = \left[ r + \gamma \max_{a'} Q_{\phi_j}(s'_i, a') \right]$
        Form input-output pairs $(s_i, , y_i)$ ($K$ Datapoints in total)
    end for
    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 $\phi_{j+1}$
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

```
for $n = 1$ to $N$ do
    Take an action $a$ and obtain the transition $(s, a, r, s')$ using $\epsilon$-greedy policy
    Calculate target $y$ using one step TD approximation
        $y = \left[ r + \gamma \max_{a'} Q_{\phi_n}(s', a') \right]$
    Compute $g^{(n)} = \nabla_{\phi} (Q_{\phi_n}(s, a) - y)^2$
    Set $\phi_{n+1} = \phi_n - \alpha g^{(n)}$
end for
```

Convergence Guarantee on Online Q learning

Algorithm Online Q Learning

```
for $n = 1$ to $N$ do
    Take an action $a$ and obtain the transition $(s, a, r, s')$ using $\epsilon$-greedy policy
    Calculate target $y$ using one step TD approximation
        $y = \left[ r + \gamma \max_{a'} Q_{\phi_n}(s', a') \right]$
    Compute $g^{(n)} = \nabla_\phi (Q_{\phi_n}(s, a) - y)$
    Set $\phi_{n+1} \leftarrow \underbrace{\phi_n - \alpha g^{(n)}}_{\text{Is this GD ?}}$
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

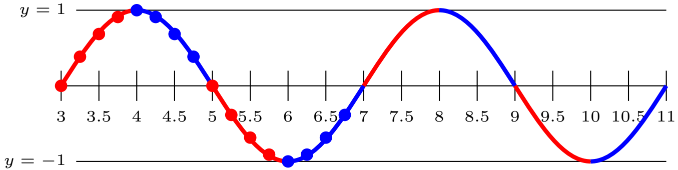
- Projection (Π) of the backup operator ($\mathcal{L}$) of optimal Q function need not be a contraction in any norm
- Fitted V iteration or fitted Q iteration need not converge because of the moving target problem
- In online Q learning algorithm,
  - ★ Samples obtained are sequentially correlated
 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

On Correlated Samples

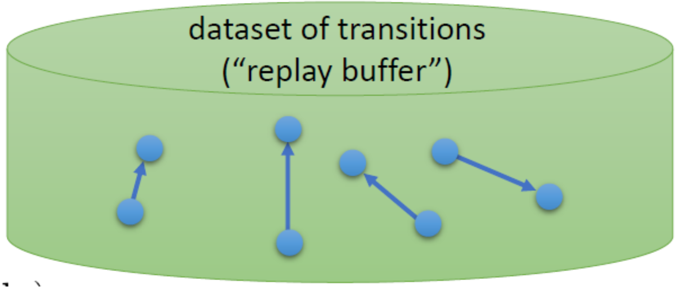


- Correlated samples will overestimate segments and will eventually be a bad fit to the regression problem

Figure Source: Sergey Levine : UCB

Replay Buffers

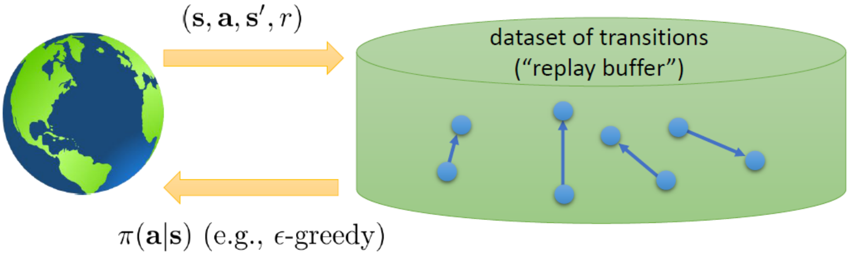
- Use the idea from fitted Q-iteration to collect and store transitions $(s, a.s', r)$



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

Figure Source: Sergey Levine : UCB

Replay Buffers


$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
dataset of transitions
("replay buffer")
$\pi(\mathbf{a}|\mathbf{s})$ (e.g., ϵ-greedy)

- 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

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

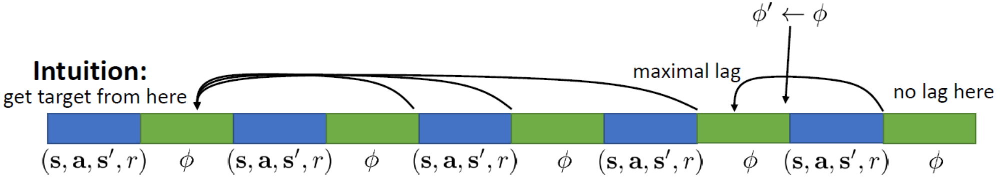
- Target network is kept constant for a while (every C steps) before being changed
  - ★ Every C steps the weights of the original network is copied to target network

DQN Algorithm

Algorithm DQN Algorithm

```
Intialize replay memory D to capacity N
Initialize action value function $Q$ with parameters $\phi$
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 $\epsilon$-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 $\phi$
        Every $C$ steps set $\widehat{Q} = Q$
    end for
end for
```

Alternative Target Network


$\phi' \leftarrow \phi$
Intuition:
get target from here
maximal lag
no lag here
$(\mathbf{s},\mathbf{a},\mathbf{s}',r)$
ϕ
$(\mathbf{s},\mathbf{a},\mathbf{s}',r)$
ϕ
$(\mathbf{s},\mathbf{a},\mathbf{s}',r)$
ϕ
$(\mathbf{s},\mathbf{a},\mathbf{s}',r)$
ϕ
$(\mathbf{s},\mathbf{a},\mathbf{s}',r)$
ϕ

Polyak Averaging

- Replace target network update step (Step 14) by

$$\phi' : \phi' \leftarrow \tau\phi' + (1-\tau)\phi$$

- Typical value for $\tau = 0.99$

Figure Source: Sergey Levine : UCB

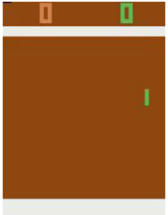
Efficacy of DQN Algorithm

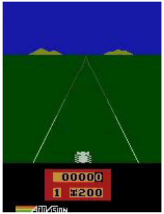
Historical Notes [2]

- 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

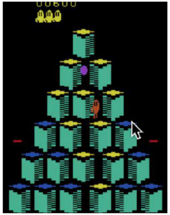
DQN on Atari [2]


Pong



Enduro

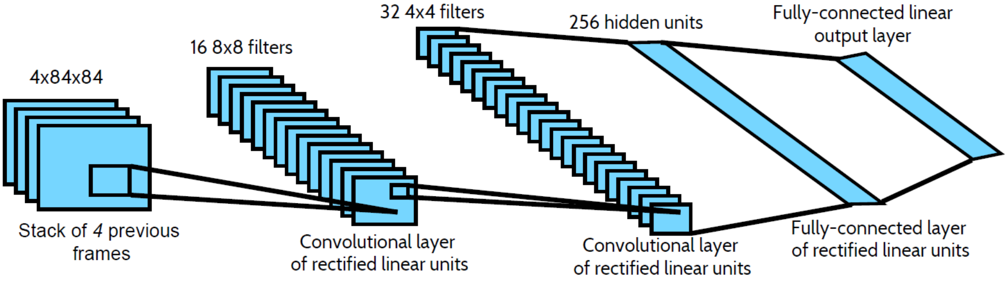
Beamrider


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

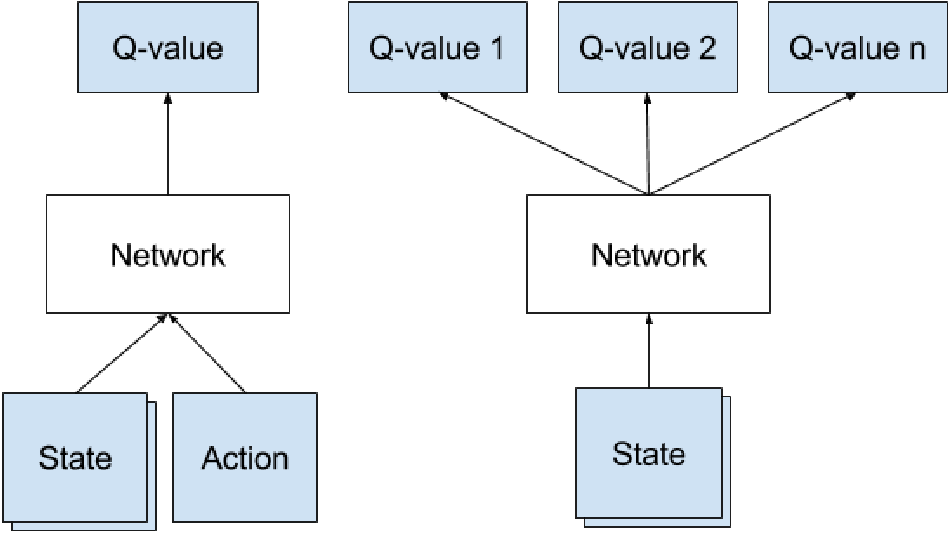
[2] Slide content from Minh

Atari DQN Architecture

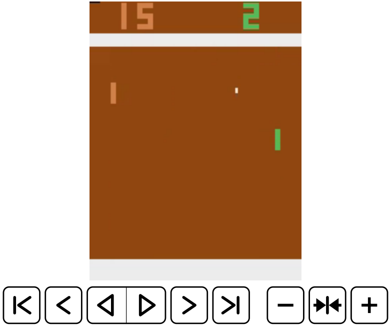

4x84x84
16 8x8 filters
32 4x4 filters
256 hidden units
Fully-connected linear output layer
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

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

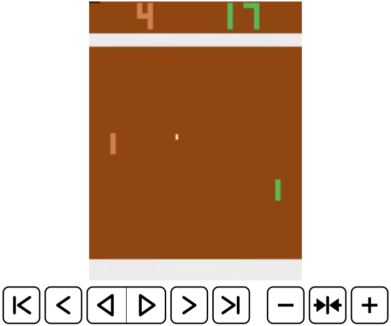
Profile of Q Function Approximator



Demonstration - Ping Pong

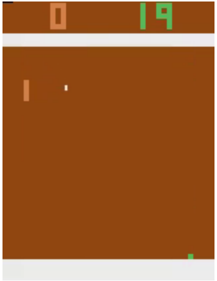


Random Policy



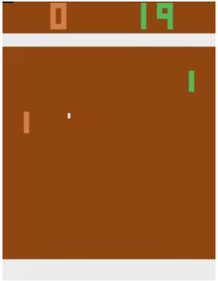
After 5.2 Millon Epochs

Demonstration - Ping Pong



⏮ < ◁ ▷ > ⏭ − ⏸ +

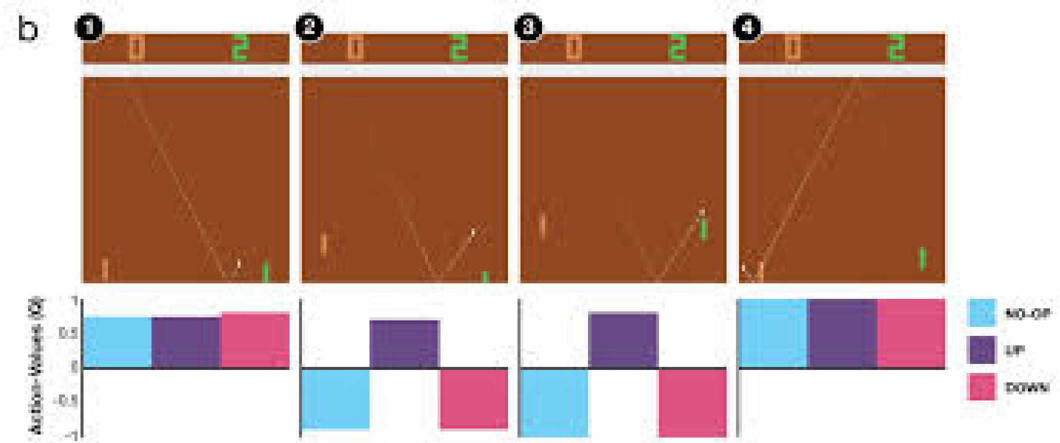
After 8 Million Epochs



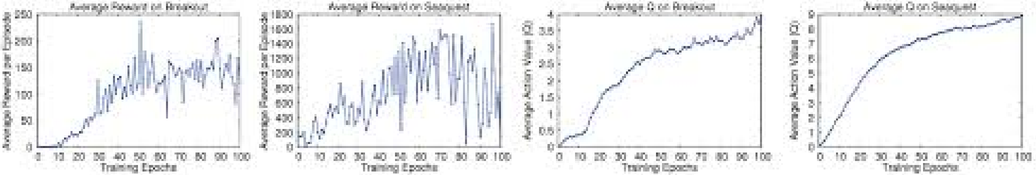
⏮ < ◁ ▷ > ⏭ − ⏸ +

After 9.5 Millon Epochs

Are the Q-Values Meaningful ?



On Tracking the Training Process


Average Reward on Breakout
Average Reward per Episode
Training Epochs
Average Reward on Seaquest
Average Reward per Episode
Training Epochs
Average Q on Breakout
Average Action Value (Q)
Training Epochs
Average Q on Seaquest
Average Action Value (Q)
Training Epochs