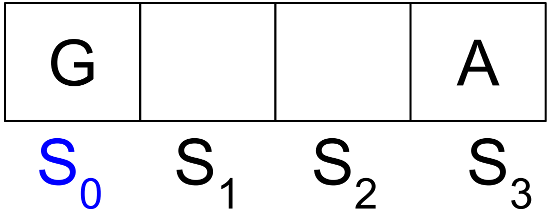
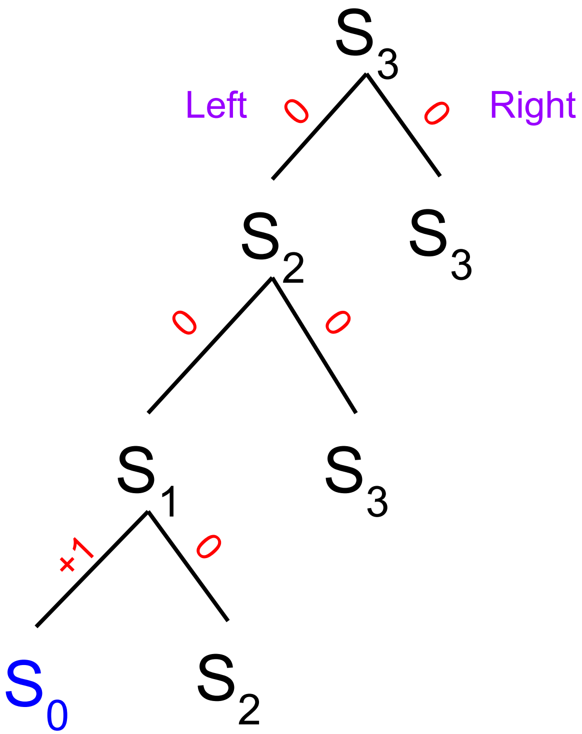
Deep RL

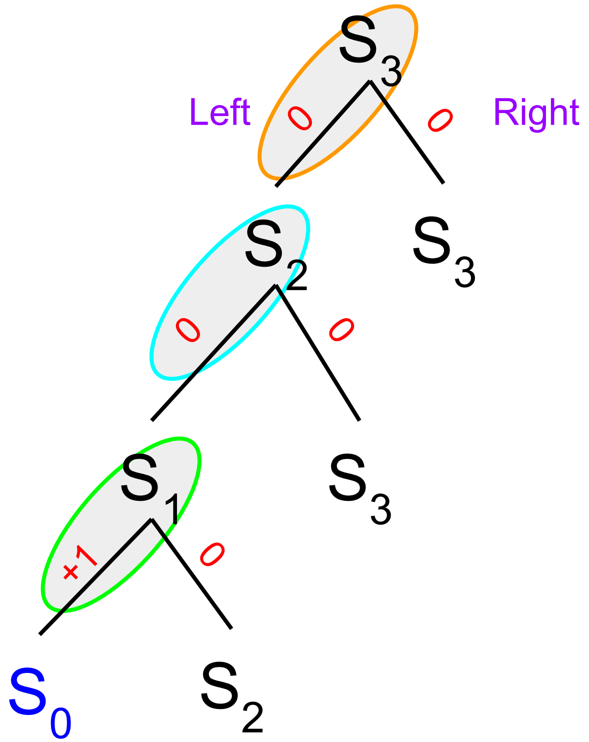


Actions: Left / Right
Rewards: +1 for Goal

Goal: Find policy $\pi(s) \rightarrow a$



G			A
S_0	S_1	S_2	S_3



Goal: Find policy $\pi(s) \rightarrow a$

Q-Value Function: $Q(s,a) = \sum_{t=0} \gamma^t r_t$

$Q(S_3, L) = 0 + 0 + \gamma^2 1$ $\quad Q(S_3, L) = .9025$

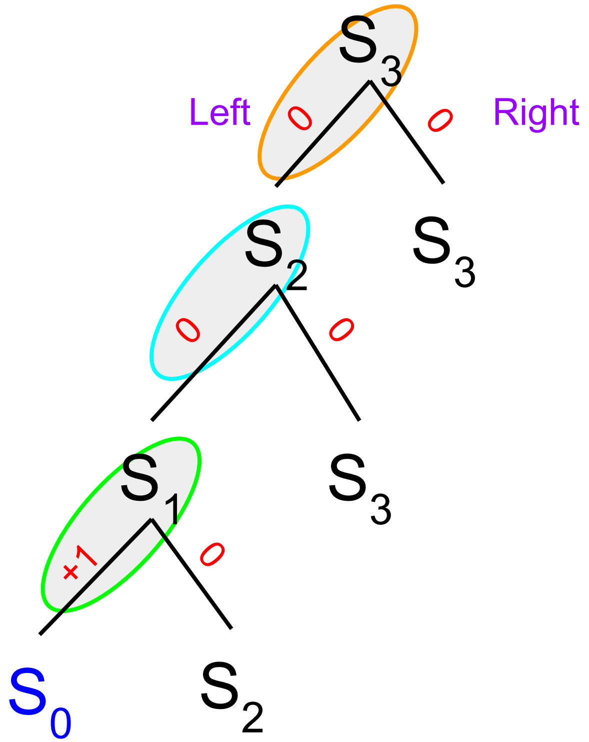
$Q(S_2, L) = 0 + \gamma^1 1$ $\quad Q(S_2, L) = .95$

$Q(S_1, L) = \gamma^0 1$ $\quad Q(S_1, L) = 1$

Q-Value Function yields policy

$\pi(s) = \mathrm{argmax}_a \; Q(s,a)$

G			A
S_0	S_1	S_2	S_3



Goal: Learn Q-Value Function

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

$Q(S_3, L) = 0 + \gamma \max(Q(S_2,L), Q(S_2,R)) = .9025$
$Q(S_2, L) = 0 + \gamma \max(Q(S_1,L), Q(S_1,R)) = .95$
$Q(S_1, L) = 1$

Need to estimate Q-values from experience!

Q-Learning Algorithm

1. Start with uniform Q-Values $Q(*,*) = 0$
2. For Episode 1 … convergence:
   a. Get s
   b. Take action $a = \text{argmax}_a Q(s,a)$
   c. Get reward r
   d. Get next state s’
   e. Target $y = r + \gamma \max_{a'} Q(s',a')$
   f. $Q(s,a) \mathrel{+}= \alpha\ (\ y - Q(s,a)\ )$

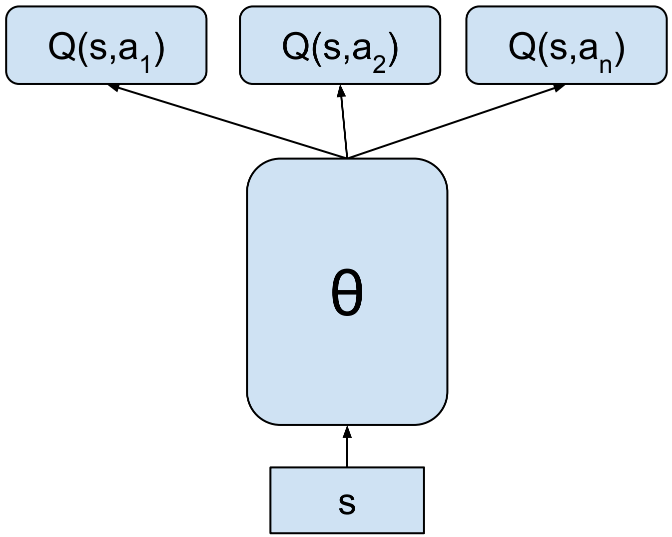
Over time, $Q(*,*)$ becomes more accurate → $\pi(s)$ gets better.
Converges to optimal Q^*, π^* in limit.

Issues

1. Scalability as a function of |S|
   a. |S| <= 10^5 Ok; |S| >= 10^6 maybe not okay

2. Generalization to new states
   a. Need a good estimate for $Q(s_{new}, *)$

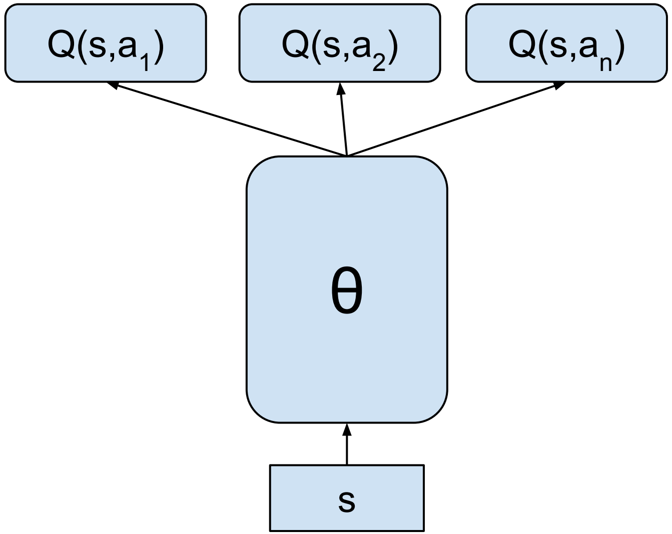
Deep RL - Represent Q(*,*) as a NN



One scalar output node for each action

Each output node estimates Q-Value

Update



Given experience (s,a,r,s'):

Generate target:

$$y = r + \gamma \max_{a'} Q(s',a'|\theta)$$

$$L(s,a,r,s'|\theta) = [\, y - Q(s,a|\theta)\,]^2$$

Minimize loss via SGD

Issues

1. Scalability as a function of |S|
   a. $|S| <= 10^5$ Ok; $|S| >= 10^6$ maybe not okay

      NN is agnostic to |S|.

2. Generalization to new states
   a. Need a good estimate for $Q(s_{new}, *)$

      NN has good generalization.

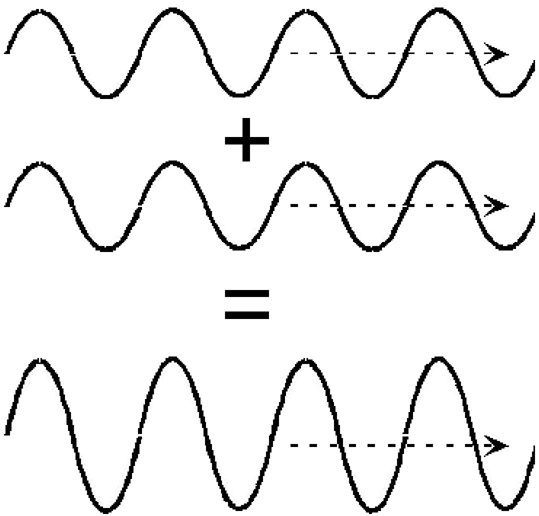
Issue 1: Constructive Interference

$y = r + \gamma \max_{a'} Q(s',a'|\theta)$

$L(s,a,r,s'|\theta) = [\ y - Q(s,a|\theta)\ ]^2$

If $r > 0$ and $s \cong s'$ then repeating this update causes $Q(s,a) \rightarrow \infty$.

Update	Q(s,a)	Q(s',a')	y
0	0	0	1
1	.5	.25	1.25
2	.87	.5	1.5
3	1.25	.75	1.75



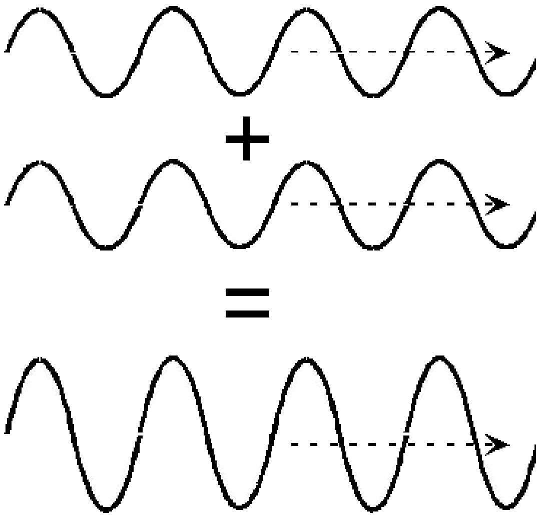
Solution 1: Target Networks

Target Network $Q(s,a|\theta^-)$ slowly tracks $Q(s,a|\theta)$

Revised Update: $y = r + \gamma \max_{a'} Q(s',a'|\theta^-)$
$L(s,a,r,s'|\theta) = [\ y - Q(s,a|\theta)\ ]^2$

Every 10,000 updates: $\theta^- = \theta$

Time	$Q(s,a|\theta)$	$Q(s',a'|\theta^-)$	y
0	0	0	1
1	.5	0	1
2	.75	0	1
3	.875	0	1
10,000	~1	.5	1.5



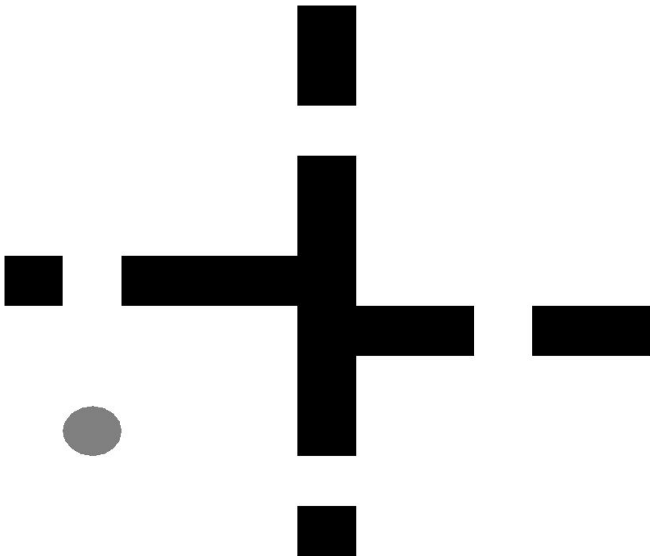
Generalization of NN is a double edged sword!

Issue 2: Policy Influences Data

No fixed dataset; Data generated by interactions using π.

Possible to get to get “stuck” in a portion of the state space and bias the update data.

If π prefers a certain part of the state space, agent can avoid learning anything else by never visiting the rest of the space.



Solution 2: Experience Replay

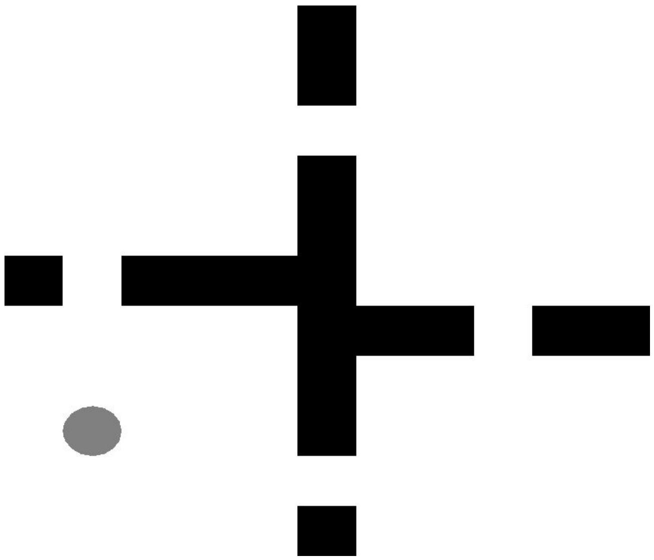
Maintain a Queue of experience tuples:

D = { (s,a,r,s’), … , (s,a,r,s’) }

Updates randomly sample from (s,a,r,s’) ~ D

Benefits:

1. Learn from collected experience more than once
2. Decorrelates (s,a,r,s’) tuples in updates
3. Can learn from states that π won’t currently visit



Issue 3: Growing Rewards

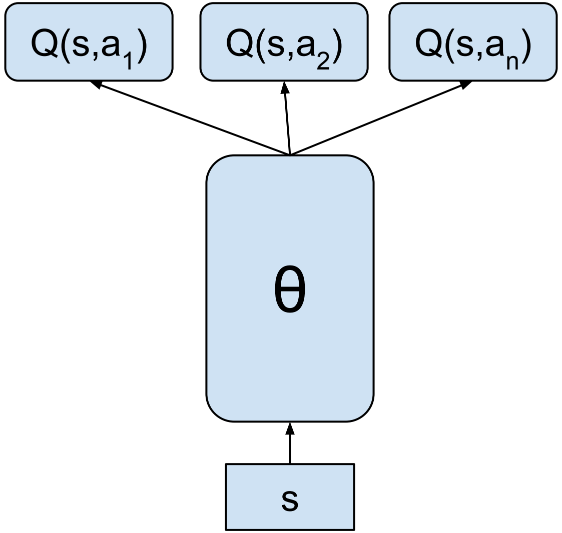
Traditional Deep Learning uses SGD + momentum with learning rate decay.

This is a problem in Deep-RL if the agent discovered a new source of reward, but had a learning rate too far decayed to change the policy to exploit new rewards.

Solution: Adaptive Learning Rate Methods

Adam / RMSProp / AdaDelta / AdaGrad

Deep Q-Learning



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 $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

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

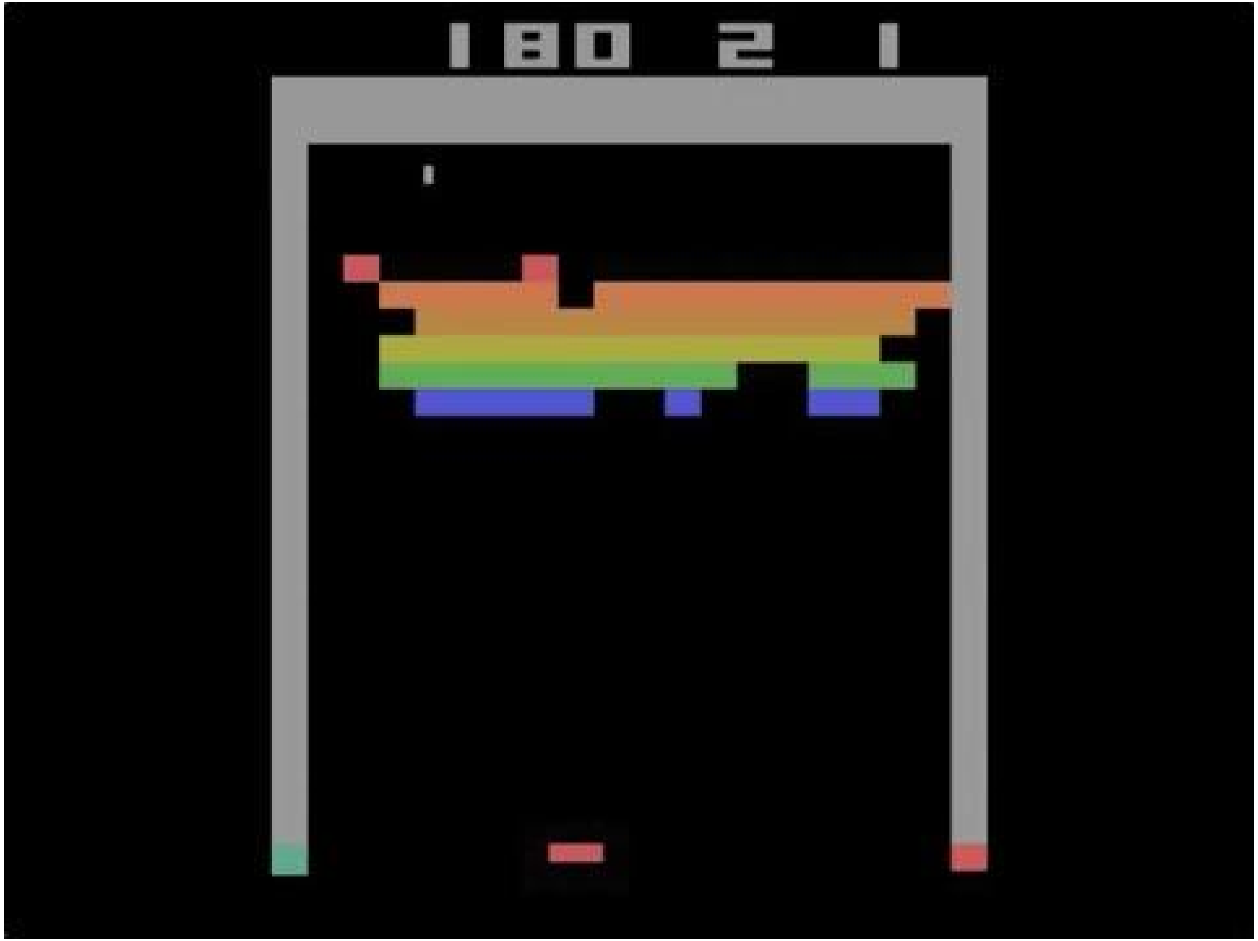
$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For


180 2 1

DQN

1. Approximate Q-Values using NN
2. Follows Basic Q-Learning Algorithm
   a. Target networks and Experience Replay Queue for stability
3. Adaptive Learning Rate Optimizer keeps policy nimble

Questions about 1st paper?

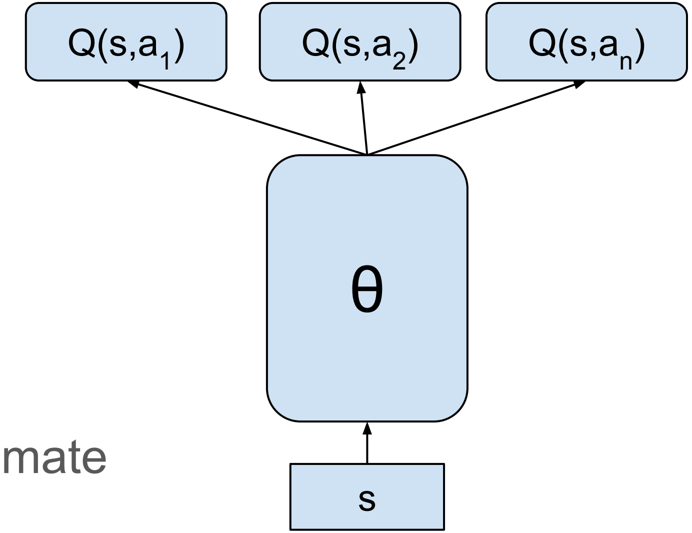
Continuous Action Spaces

Atari has discrete actions but many domains require continuous control: e.g. torque on actuator.

The Good: NN can output continuous actions

The Bad: DQN uses these continuous outputs to estimate Q-Values rather than using them for control.

The Ugly: Need a new architecture!



Actor-Critic Methods

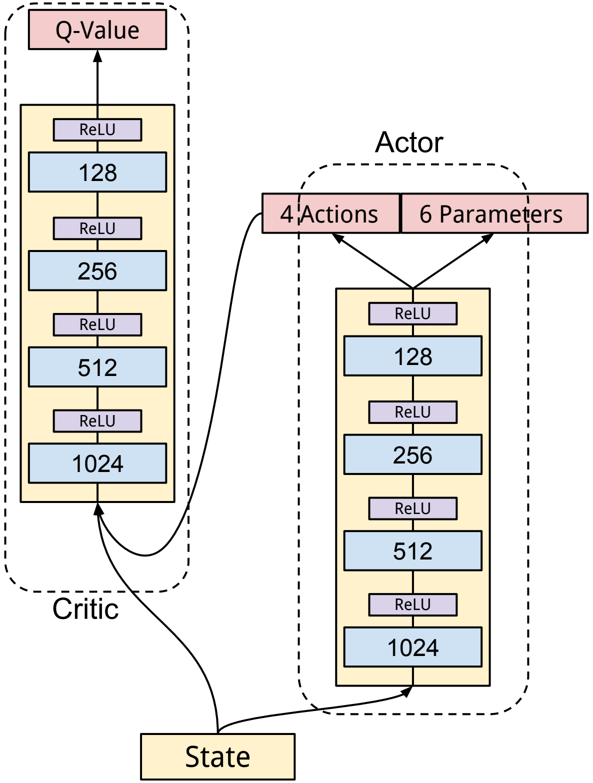
Two network solution:

Actor Network: a = $\mu(s|\theta^{\mu})$

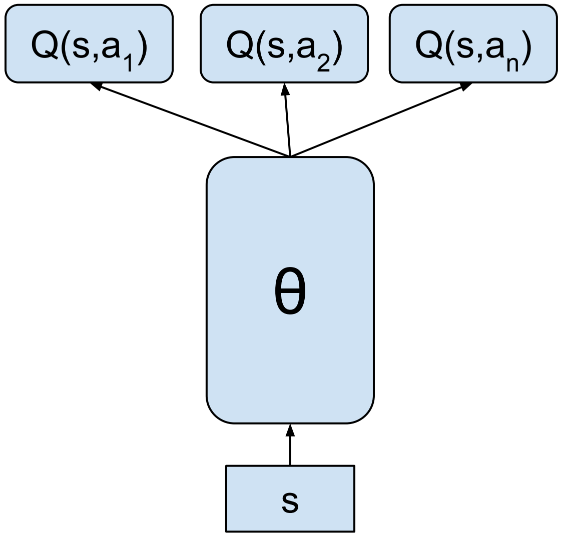
Outputs continuous actions. Actor is π.

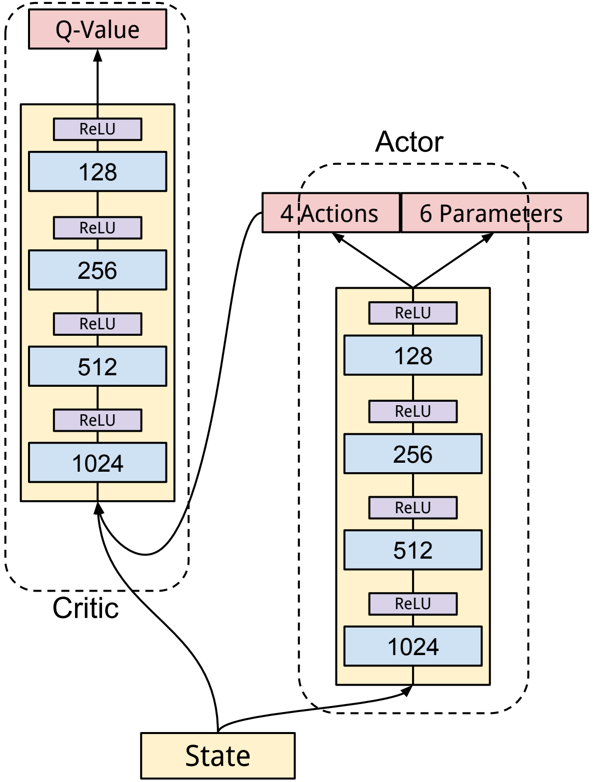
Critic Network: q = $Q(s,a|\theta^{Q})$

Evaluates state, action pairs.



Actor-Critic Methods





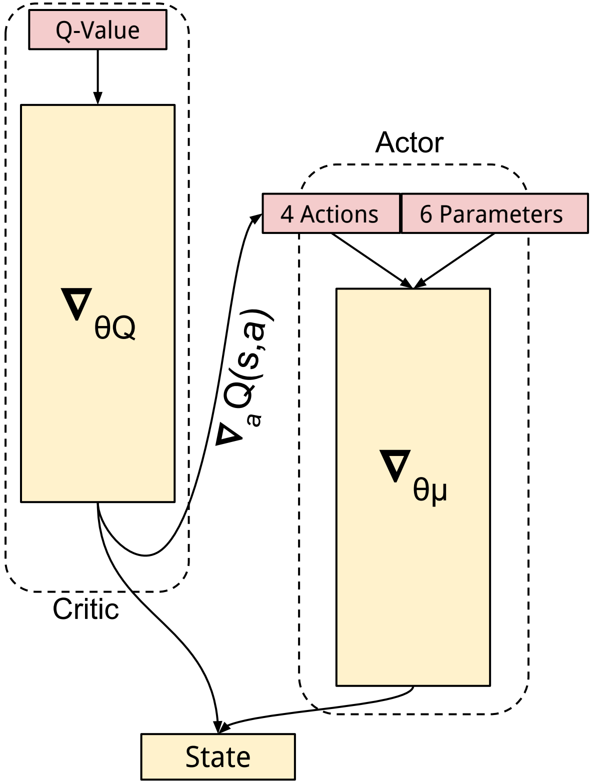
Critic Update

Given (s_i, a_i, r_i, s_{i+1})

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

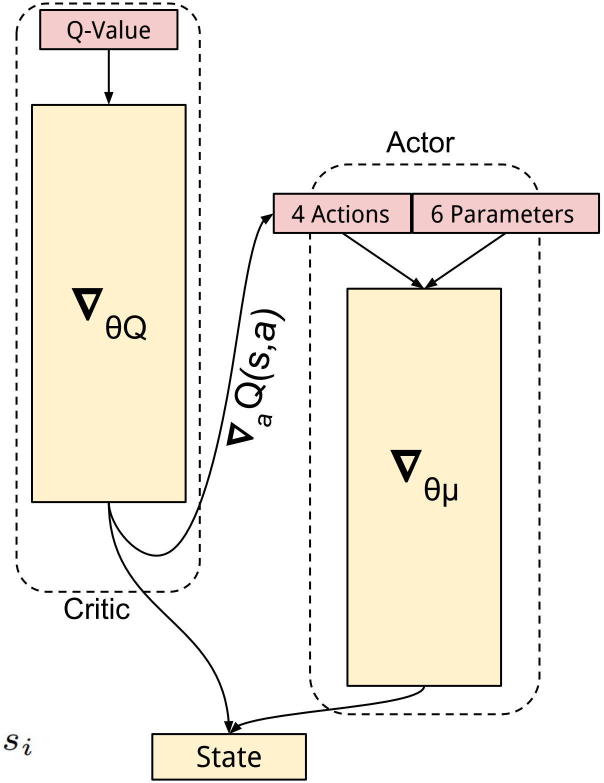
Note: No max over a'!



Actor Update

1. Forward Pass: q = Q(s, μ(s|θ^{μ}) |θ^{Q})
2. Target y = q + ε
3. L = (y - q)^2 = ε
4. Backwards pass through critic (ignore diff) and then actor.
5. Equivalent to linking networks together and backproping through both networks

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$



Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

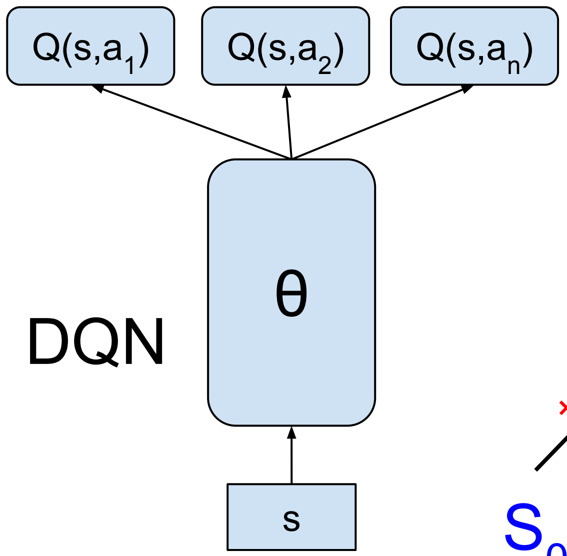
Stability

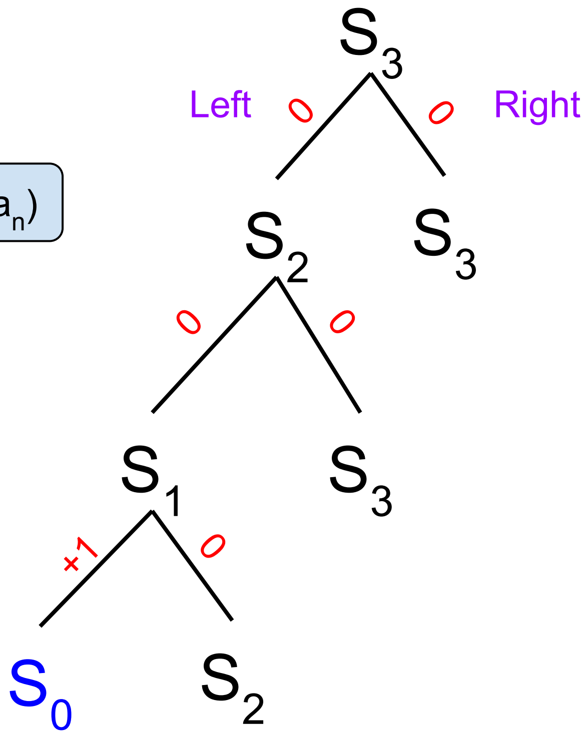
1. Target networks for both actor & critic
2. Experience replay + Adam
3. Batch Normalization + Clipped Gradients

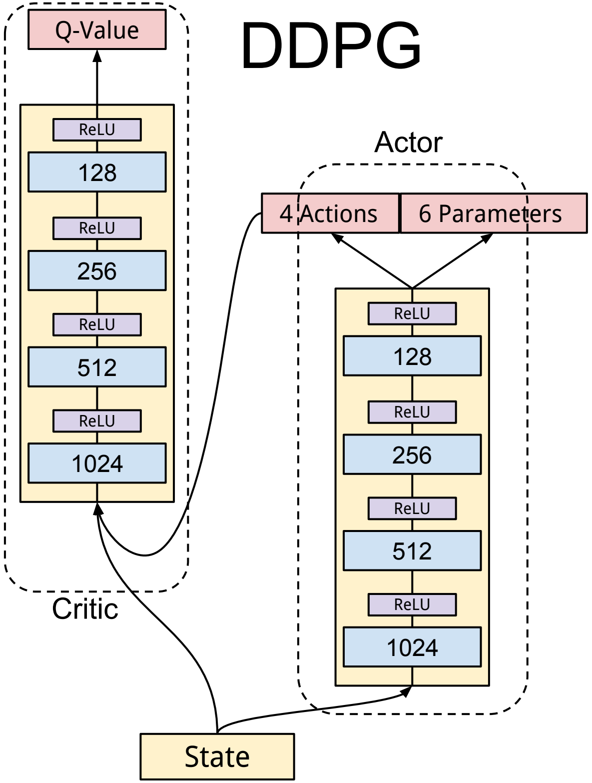
4. Close relationship to GAN training, where Critic = Discriminator and Actor = Generator


Cheetah
Low Dimensional Features

Deep RL







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