Towards Function Approximation Methods

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

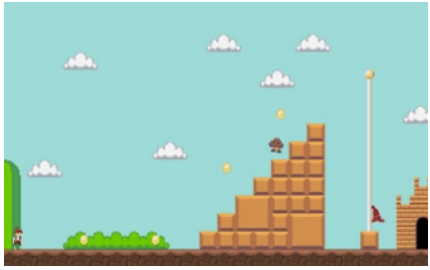
September 21, 2024

Overview

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

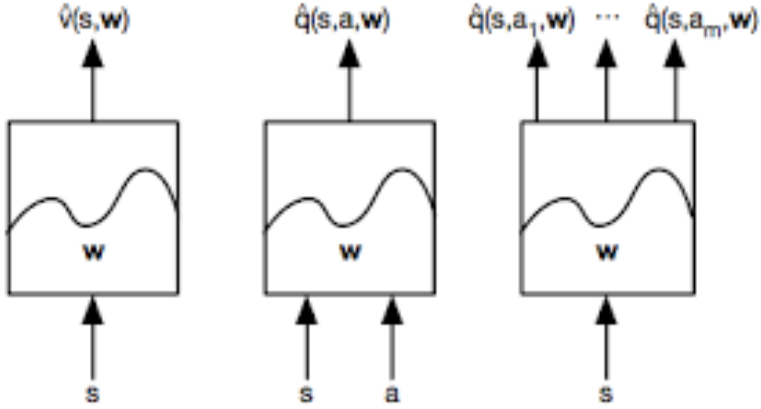
Slide Credit: David Silver : UCL lectures

Value Function Approximators

TCS Research & Innovation

- 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

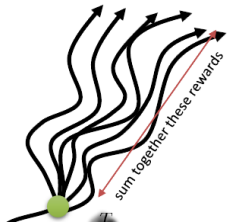


Policy Evaluation Using Neural Networks

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

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

Fitted V Iteration : Algorithm

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**

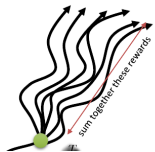
Optimal Value Function : Control

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

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

Fitted Q Iteration : Algorithm

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

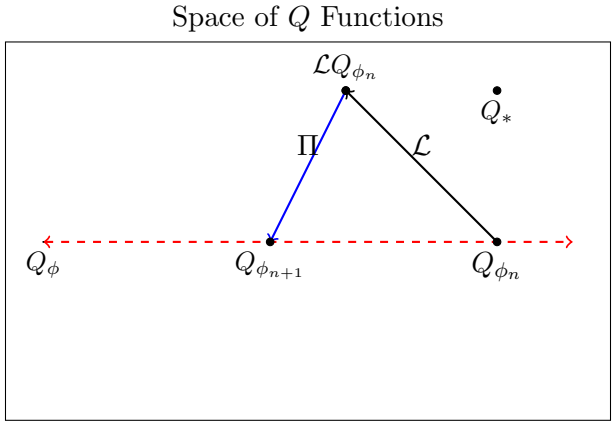
- Use the fixed point equation below to define a **contraction** operator $\mathcal{L}$ (contraction in L_∞ norm)

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

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 $\mathcal{L} : \mathcal{Q} \to \mathcal{Q}$ 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

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

TCS Research & Innovation

- 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**

Deep Q Networks

Easwar Subramanian

TCS Innovation Labs, Hyderabad

cs5500.2020@iith.ac.in

September 28, 2024

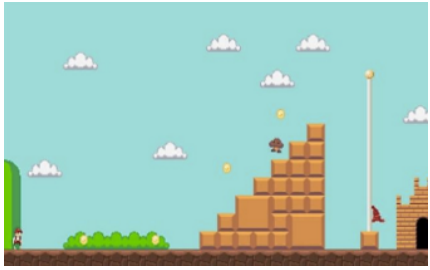
Overview

TCS Research & Innovation

Function Approximation Methods

On the need for Function Approximators

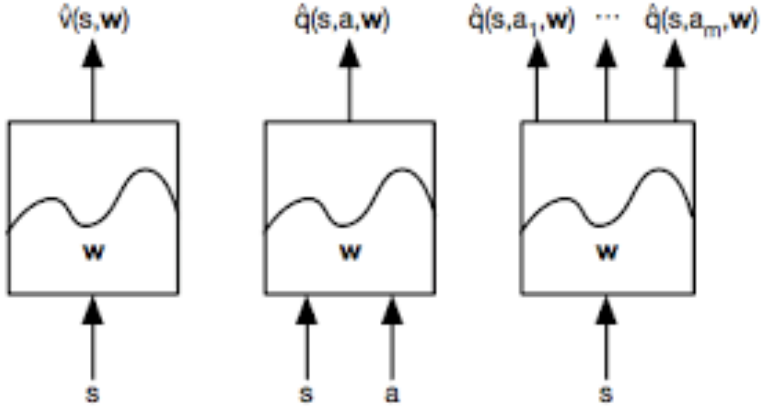
- To solve large scale RL problems
  - ★ Game of Backgammon : 10^{20} states
  - ★ Game of Go : 10^{170} states
  - ★ Even Atari games have large state space



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

Slide Credit: David Silver : UCL lectures

Neural Network Approximators



Policy Evaluation Using Neural Networks

The value of a policy π is given by

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

Training data is collected as,

- MC Update : $\left(s_i, \underbrace{\left[\sum_{k=t}^{H}\left(\gamma^{k} r_{t+k+1} | s_t = s\right)\right]}_{=y_i}\right)$
- Fitted V Iteration : $\left(s_i, \underbrace{r + V_{\phi}^{\pi}(s_i')}_{=y_i}\right)$

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

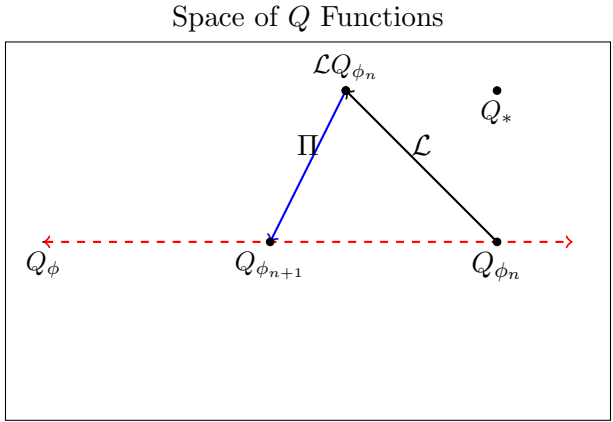
- Use the fixed point equation below to define a **contraction** operator $\mathcal{L}$ (contraction in L_∞ norm)

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

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 s
- 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

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**

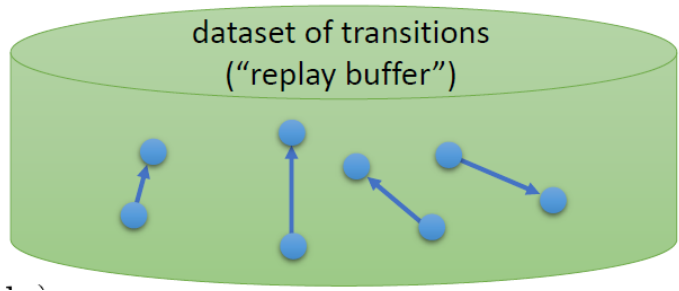
Towards a Stable Deep Q Network Algorithm

Desiredata

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

Replay Buffers

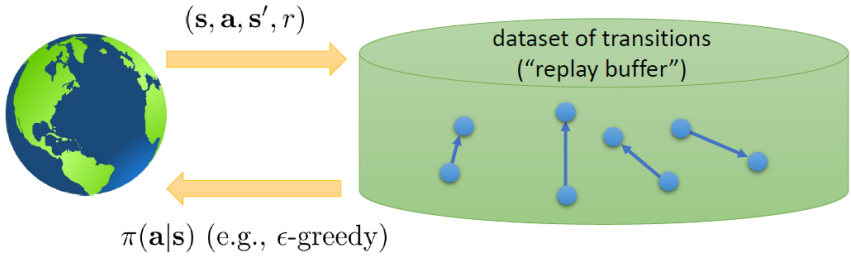
- Use the idea from fitted Q-iteration to collect and store transitions $(s, a.s', r)$



- 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

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

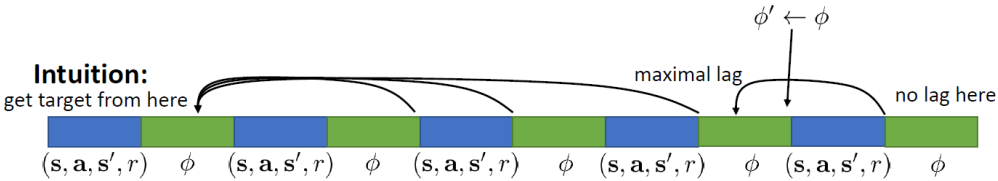
- Target network is kept constant for a while (every C steps) before being changed
  - ★ Every C steps the weights of the original network is copied to target network

DQN Algorithm

Algorithm DQN Algorithm

Initialize 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

Alternative Target Network



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

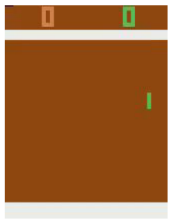
Efficacy of DQN Algorithm

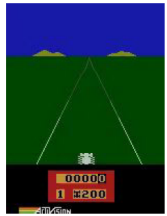
Historical Notes [2]

- 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

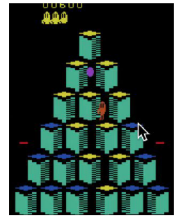
DQN on Atari [2]


Pong



Enduro

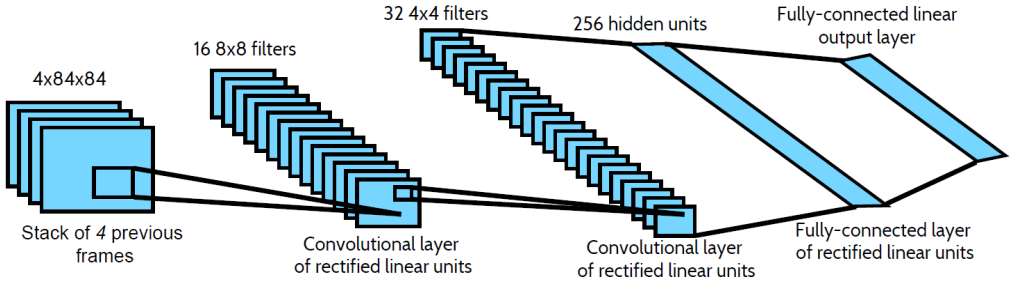
Beamrider


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

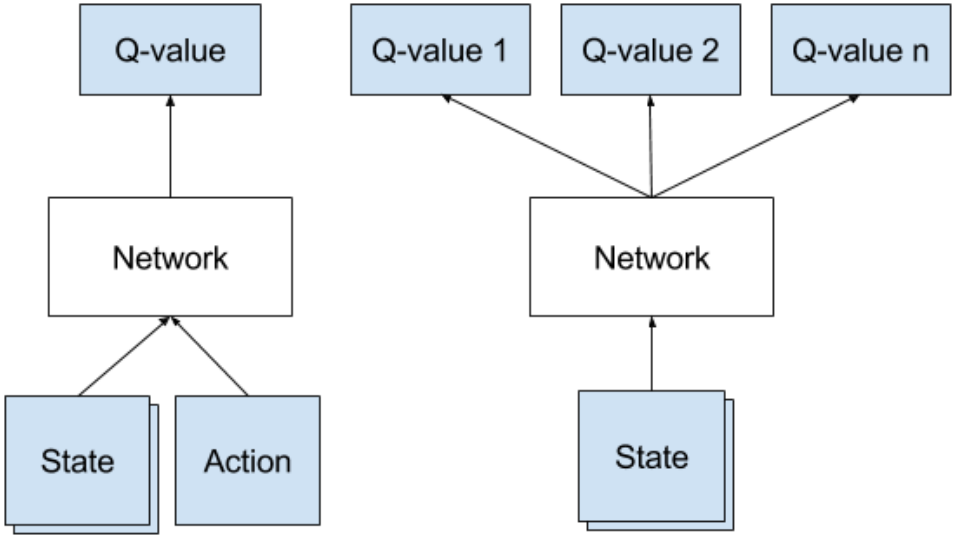
[2] Slide content from Minh

Atari DQN Architecture

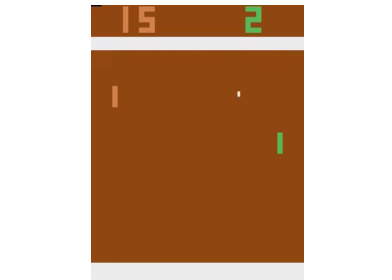


- Convolutional neural network architecture
- History of 4 frames as input
- One output per action ($Q(s, a)$) – expected reward for action a

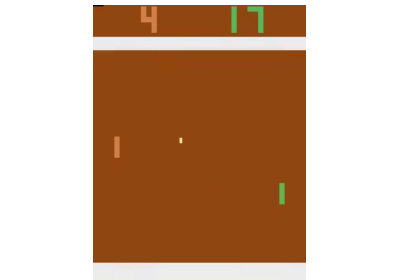
Profile of Q Function Approximator



Demonstration - Ping Pong

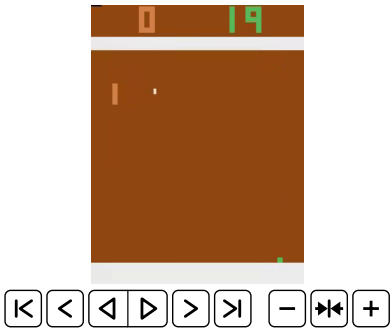


Random Policy

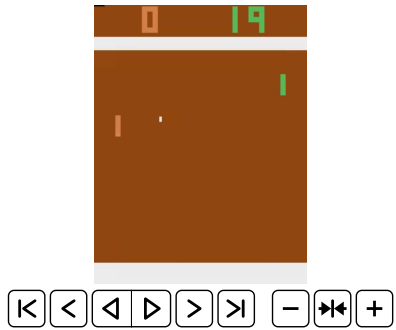


After 5.2 Millon Epochs

Demonstration - Ping Pong

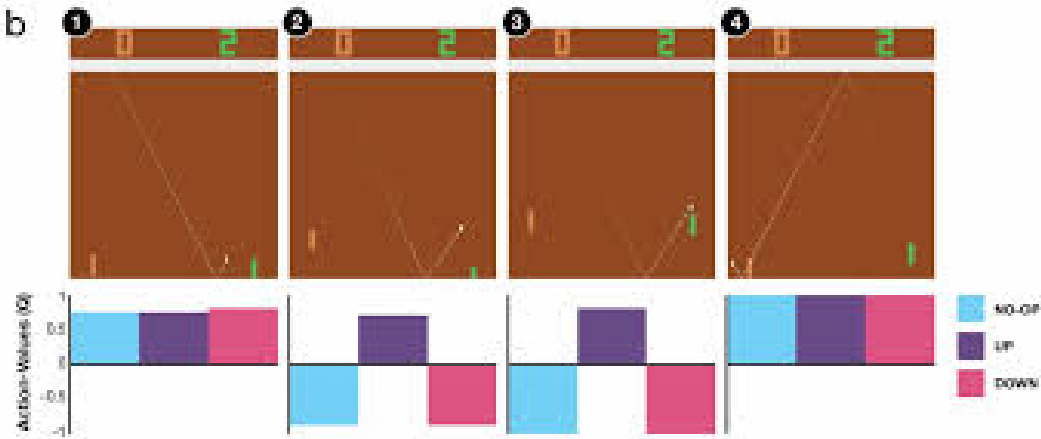


After 8 Million Epochs

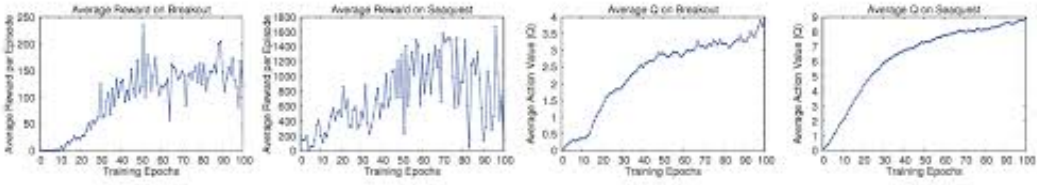


After 9.5 Millon Epochs

Are the Q-Values Meaningful ?



On Tracking the Training Process



Double DQN

Maximization Bias

- Consider single-state MDP with 2 actions.
- Both actions have zero mean rewards (the agent does not know this information)
- Let $\hat{Q}(\cdot, a_1)$ and $\hat{Q}(\cdot, a_2)$ be (unbiased) finite sample estimates of Q for action $a_!$ and a_2 respectively
- The agent will prefer the action which has maximum $\hat{Q}$ based on sample estimates, although both actions have same expected mean reward

Maximization Bias

- Consider single-state MDP with 2 actions.
- One action has $-\epsilon$ (ϵ positive and small) mean reward and the other action has zero mean reward.
- Let $\hat{Q}(\cdot, a_1)$ and $\hat{Q}(\cdot, a_2)$ be (unbiased) finite sample estimates of Q for action a_1 and a_2 respectively
- In this case, the agent might choose action a_1 , depending on how the maximum $\hat{Q}$ values that is based on sample estimates show up, although action a_2 is clearly better in expectation

Extending the analogy, the (tabular or deep) Q-learning algorithm may actiually pick the suboptimal action for target computation and this causes over estimation of Q-values.

Issues in Tabular Q-Learning Approach

- The problem with vanilla tabular Q-Learning is that the same samples are being used to decide which action is the best (highest expected reward), and the same samples are also being used to estimate that action-value
- Break up the Q-learning update rule as follows

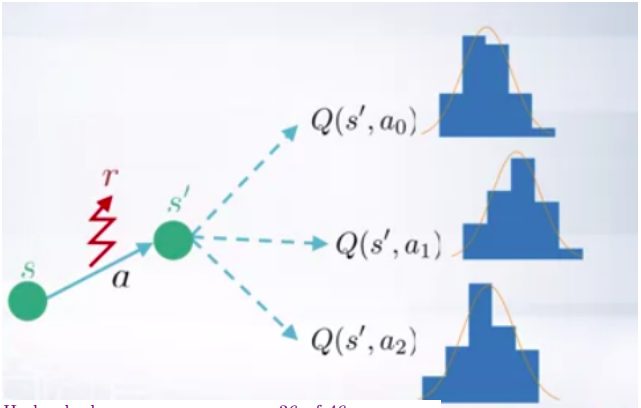
$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, \arg\max Q(s_{t+1}, a)) - Q(s_t, a) \right)$$

- If action value is overestimated, then it is chosen as the best action, and its overestimated value is used as the target

Slide Credit: Medium Blog : Ameet Deshpande

Persistence of the Problem in DQN

- For transition $(s_t, a_t, r_{t+1}, s_{t+1})$ the TD target of the Q-value update is $r_{t+1} + \gamma \arg\max_{a'} Q_\phi(s_{t+1}, a')$
- $Q_\phi(\cdot, \cdot)$ is a noisy estimate during training phase
- Therefore, $\max Q_\phi(\cdot, \cdot)$ would typically be overestimated during training



Evidence from Atari Games

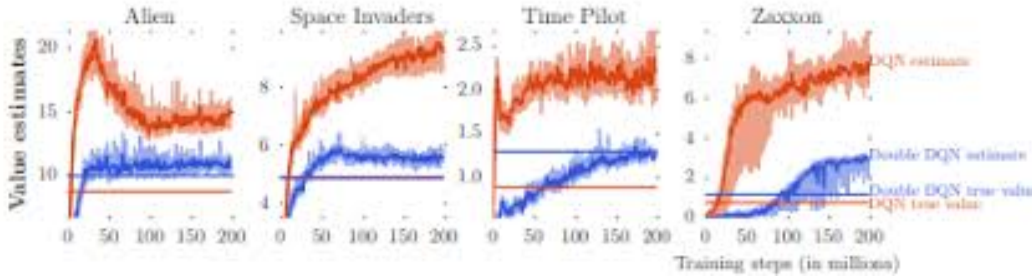


Figure Source: van Hasselt : DDQN paper

Possible Resolution ...

- There are two identical fair coins (we don't know they are fair)
- If a coin lands on head, we get one dollar; otherwise we lose a dollar
- Interested in answering the following questions
  - ★ Which coin will yield will more money in future flips ?
  - ★ How much can we expect to win or lose per flip using the coin from previous question ?
- Two ways to answer
  - ★ Flip each coin n few times and answer both questions
  - ★ Flip each coin n_1 times, answer the first question; collect fresh n_2 samples to answer second question based on the answer to the first question

The idea behind the second method is that we use separate samples to choose the best action and separate samples to use Q-values

Solution to Maximization Bias

- Have two different set of samples to decide the action and to evaluate the target
- The idea is to use two Q functions
- In the tabular Q-learning setting, for each transition quadruple $(s_t, a_t, r_{t+1}, s_{t+1})$ we flip a fair coin to decide any of the two update steps given below,

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q_2(s_{t+1}, \arg\max Q_1(s_{t+1}, a)) - Q_1(s_t, a_t)\right)$$

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q_1(s_{t+1}, \arg\max Q_2(s_{t+1}, a)) - Q_2(s_t, a_t)\right)$$

Double DQN

- In the DQN setting, we already have two Q networks (to address the moving target problem). So, we can take advantage of that by using the following update rule.
- In Double DQN, targets for the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ is computed as follows,

$$Q^{original}(s_t, a_t) \leftarrow r + \gamma Q^{target}(s, \arg\max Q^{original}(s, a))$$

- In the above equation, the $\leftarrow$ actually means assigning targets which can then be picked up while sampling the replay buffer
- The fundamental idea is to use two Q_ϕ's (both are noisy estimates of true Q) in different ways so that the overestimation problem mellows down

Priortized Experience Replay

Priortized Experience Replay

- Replaying all transitions with equal probability is suboptimal
- Replay transitions in proportion to absolute Bellman error

$$\left| r + \gamma \max_{a'} Q_{\phi'}(s', a') - Q_{\phi}(s, a) \right|$$

- Leads to much faster learning

Priortized Experience Replay

- TD error for vanilla DQN is

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\phi'}(s_{t+1}, a) - Q_{\phi}(s_t, a_t)$$

- TD error for DDQN is

$$\delta_i = r_t + \gamma Q_{\phi'}(s_{t+1}, \text{argmax}_{a \in \mathcal{A}} Q_{\phi}(s_{t+1}, a)) - Q_{\phi}(s_t, a_t)$$

- Priority for each entry in replay buffer D is given by $p_i = |\delta_i| + \epsilon$

Priortized Experience Replay

- Sample from replay buffer according to probability distribution

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

with α determining the level of prioritization

- In order to compute the expectation

$$\min_\phi \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}\left[\left(r_t + \gamma \max_{a\in\mathcal{A}} Q_{\phi'}(s_{t+1},a) - Q_\phi(s_t,a_t)\right)^2\right],$$

it is essential to use the importance sampling weights in each mini-batch of the gradient update

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

- The parameter β is an annealing term that is low (0.4 to 0.8) in the beginning of the training and tends to 1 towards the end of the training.
- The question on optimal choices of α and β during various phases of training depends on task at hand

Practical Tips for DQN [2]

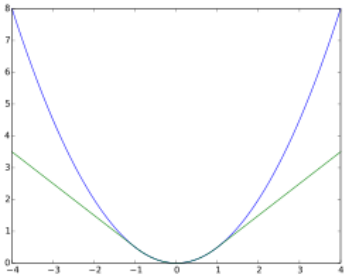
- DQN is more reliable on some tasks than others. Test your impementation on reliable tasks like Pong and Breakout: if it doesn't achieve good scores, something is wrong
- Large replay buffers improve robustness of DQN, and memory efficiency is important
- DQN converges slowly - for ATARI it is often 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

Practical Tips for DQN [2]

- Try non-standard exploration schedules
- Do use Double DQN with priortized experience replay – significant improvement
- Use Huber loss on Bellman error

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$



Policy Gradients Methods

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

October 19, 2024

Overview

TCS Research & Innovation

1. Policy Gradient Formulation
2. Variance Reduction Techniques

Policy Based Reinforcement Learning

TCS Research & Innovation

- Last lecture, we parametrized value functions using parameter ϕ

$$V_\phi^\pi(s) = V^\pi(s)$$

$$Q_\phi^\pi(s,a) = Q^\pi(s,a)$$

- Policy was directly generated from value functions (greedy or ϵ greedy)

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_*(s,a) \\ 0 & \text{Otherwise} \end{cases}$$

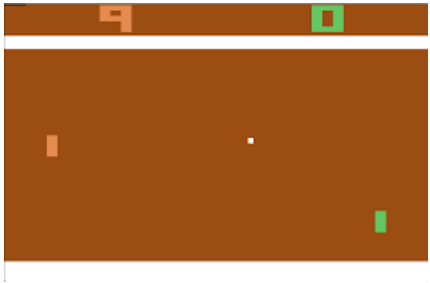
- In the next couple of lectures, we will directly parametrize the policy

$$\pi_\theta(a|s) = P(a|s,\theta)$$

- We will consider model free control with parametrized policies

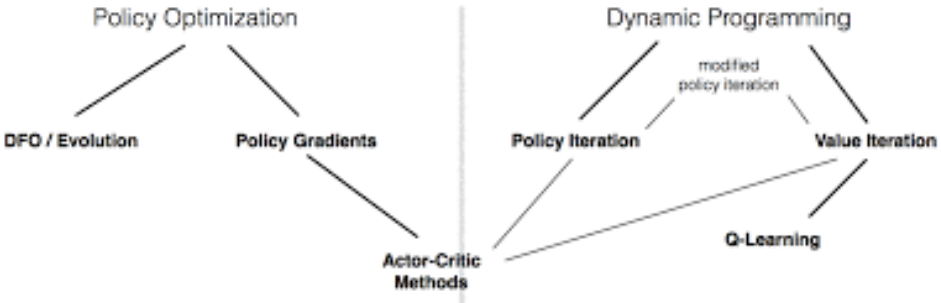
Why Policy Optimization ?

- Often policies (π) are simpler than value functions (V or Q)



- Computing optimal V is bit of problem (we did not see any control algorithms for V)
- With state-value functions Q, computing arg max over actions gets tricky when action space is large or continuous
- Better convergence properties
- Can learn stochastic policies

RL Algorithms Landscape [2]



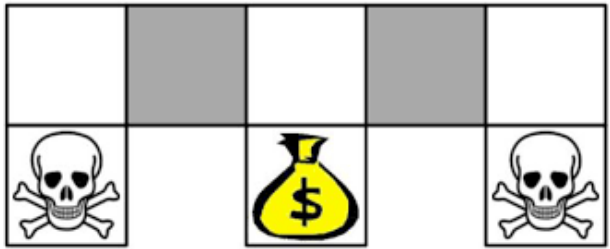
We will now actually look out for the optimal policies in the stochastic policy space !

[2] Slide content from Schulman

Example : Rock-Paper-Scissors

- Two player game of rock-paper-scissors
  - ★ Scissors beats paper
  - ★ Rock beats scissors
  - ★ Paper beats rock
- Consider policies for iterated rock-paper-scissors
  - ★ A deterministic policy is easily exploited
  - ★ A uniform random policy is optimal (i.e. Nash equilibrium)

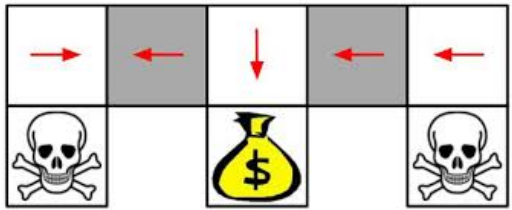
Example : Aliased Grid World



- The agent cannot differentiate the grey states
- For example, state could be represented by features of the following form

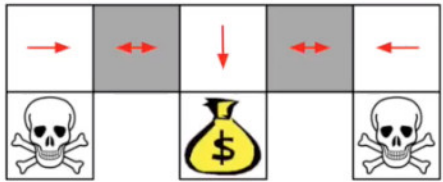
$$\psi(s, a) = 1(\text{wall to } \mathbf{S}, \text{ a=move } \mathbf{E})$$

Example : Aliased Grid World



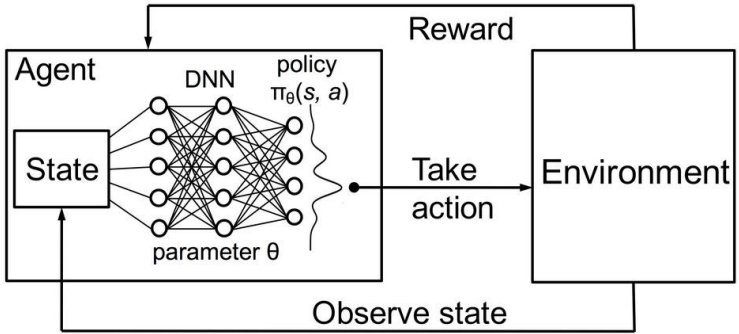
- Under aliasing, an optimal deterministic policy will either
  - ★ move W in both grey states (shown as above)
  - ★ move E in both grey states
- Either way, it can get stuck and never reach the money
- Value based RL learns a near deterministic policy (greedy or ϵ greedy)
- Such a policy will go back and forth on the grid for a long time before hitting money

Example : Aliased Grid World



- An optimal stochastic policy will randomly move E or W in grey states
- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Using Function Approximators



- If action space is discrete
  - ★ Network could output a vector of probabilities (softmax)
- If action space is continuous
  - ★ Network could output the parameters of a distribution (For e.g.. mean and variance of a Gaussian)

Continuous Action Space : Gaussian Policies

- Policy is Gaussian
- The mean (μ) of the Gaussian could be the output of the neural network
- The variance σ of the Gaussian could be constant or can be parametrized.
- One way to operate in continuous action space is to sample an action from the Gaussian distribution. i.e., $a \sim \mathcal{N}(\mu, \sigma)$
- Idea can be extended to any parametrized probability distribution (even multi-variable).

Policy Optimization

A policy $\pi(\cdot)$ is parametrized by parameter θ and denoted by π_θ

Performance of a policy π_θ is given by

$$J(\theta) = V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

Goal of RL is to find a policy

$$\pi_\theta^* = \arg\max_{\pi_\theta} V^{\pi_\theta}(s) = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

We will look for π_θ^* in class of stochastic policies by finding θ that maximizes $J(\theta)$

Policy Gradient

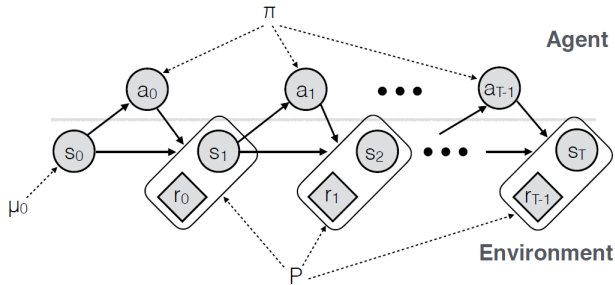
- Let $J(\theta)$ be the policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- $\nabla_\theta J(\theta)$ is the policy gradient and
- α is the step size parameter

Policy Gradient Formulation

Policy Gradient : Notation



- Let policy π be parametrized by θ and denoted by π_θ
- Let $\tau \sim \pi_\theta$ denote the state-action sequence given by $s_0, a_0, s_1, a_1, \cdots, s_t, a_t, \cdots$
- Then, $P(\tau;\theta)$ be the probability of finding a trajectory τ with policy π_θ

$$P(\tau;\theta) = P(s_0)\prod_{t=0}^{\infty}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t,a_t)$$

Policy Gradient : Objective Function

We can define $G(\tau)$ discounted cumulative reward obtained by following trajectory τ

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

Objective function $J(\theta)$ for policy gradient approach is written as,

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi_\theta \right] = \sum_{\tau \sim \pi_\theta} \left[P(\tau; \theta) G(\tau) \right]$$

Goal is to find θ^* such that

$$\theta^* = \arg\max_{\theta} J(\theta)$$

Policy Gradient Derivation

$$J(\theta) = \sum_{\tau} \left[P(\tau;\theta) G(\tau) \right]$$

Taking gradient with respect to θ gives

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \left(\sum_{\tau} \left[P(\tau;\theta) G(\tau) \right] \right) \\
&= \sum_{\tau} \nabla_\theta \left[P(\tau;\theta) G(\tau) \right] \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} [\nabla_\theta P(\tau;\theta)] G(\tau) \\
&= \sum_{\tau} \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} P(\tau;\theta) G(\tau) \\
&= \sum_{\tau} \nabla_\theta \log P(\tau;\theta) P(\tau;\theta) G(\tau) \qquad \left(\because \quad \nabla_\theta \log f(x) = \frac{\nabla_\theta f(x)}{f(x)} \right)
\end{aligned}$$

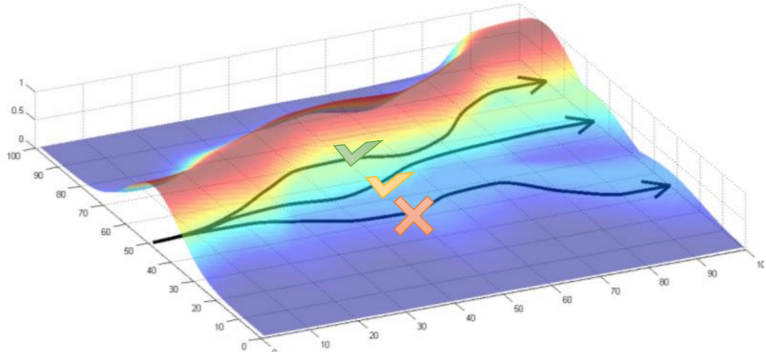
Policy Gradient : Sample Based Estimate

$$\nabla_\theta J(\theta) = \sum_{\tau \sim \pi_\theta} \nabla_\theta \log P(\tau;\theta) P(\tau;\theta) G(\tau) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log P(\tau;\theta) G(\tau) \right]$$

Sample based estimate is given by

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \nabla_\theta \log P(\tau^{(i)};\theta) G(\tau^{(i)})$$

Policy Gradient : Intuition



- Increase the probability of paths with positive $G(\tau)$
- Decrease the probability of paths with negative $G(\tau)$
- Formalize the notion of 'trial and error'

Policy Gradient : Model Free Formulation

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \nabla_\theta \log P(\tau^{(i)};\theta) G(\tau^{(i)})$$

Is the above formula good enough for implementation ?

$$P(\tau;\theta) = P(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) P(s_t|s_{t-1}, a_t)$$

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)};\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{\infty} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi(a_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{\infty} \log P(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)}) + \sum_{t=0}^{\infty} \log \pi(a_t^{(i)}|s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{\infty} \log \pi(a_t^{(i)}|s_t^{(i)}) = \underbrace{\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)}|s_t^{(i)})}
\end{aligned}$$

Policy Gradient : Model Free Formulation

The following formulation provides an unbiased estimate of the policy gradient and we can calculate it without using the dynamics model

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}\left[\nabla_\theta \log P(\tau^{(i)};\theta)\right] G(\tau^{(i)})$$

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t^{(i)}|s_t^{(i)})\right]\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}^{(i)}\right]$$

REINFORCE : Monte-Carlo based Policy Gradient

Algorithm REINFORCE : MC based Policy Gradient

1: Initialize policy network π with parameters θ_1 and learning rate α
2: **for** $n = 1$ to N **do**
3: Sample K trajectories from π_{θ_n}
4: Calculate gradient estimate

$$\nabla_{\theta_n} J(\theta_n) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_{\theta_n} \log \pi(a_t^{(i)} | s_t^{(i)}) \right] \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}^{(i)} \right]$$

5: Perform gradient update

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

6: **end for**

Connections to Maximum Likelihood

Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \right] \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}^{(i)} \right]$$

Maximum Likelihood

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \right]$$

(Supervised Learning : Given s_t find a_t)

Issues with Gradient Estimate

- The gradient estimate, thus calculated, is unbiased but has high variance (reason : we are sampling stochastic paths)
- Hence the gradient descent is slow to converge
- Some variance reduction techniques are required in practice

Variance Reduction Techniques

Discount Factor and Variance Reduction

Gradient estimate is given by,

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \right] \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}^{(i)} \right]$$

One can rewrite the above equation as

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{k=0}^{\infty} \gamma^k r_{k+1}^{(i)} \right] \right]$$

- For infinite horizon MDPs having $\gamma < 1$ not only helps in proving convergence of algorithms but also helps reduce variance of the policy gradient estimate
- Ignoring reward terms 'far' into the future gives us a reasonable approximation to policy gradient but with lower variance

Aside : Score Function

Score function in policy gradient is the term

$$\nabla_\theta \log \pi(a_t|s_t)$$

Expectation of the score function is zero

$$\begin{aligned}
\mathbb{E}_{a_t|s_t}\left[\nabla_\theta \log \pi(a_t|s_t)\right] &= \int_{a_t} \pi(a_t|s_t) \nabla_\theta \log \pi(a_t|s_t)\, da_t \\
&= \int_{a_t} \nabla_\theta \pi(a_t|s_t)\, da_t \\
&= \nabla_\theta \int_{a_t} \pi(a_t|s_t)\, da_t \\
&= \nabla_\theta 1 = 0
\end{aligned}$$

Principle of Causality

Causality : Policy at time t' cannot affect reward at time t when $t < t'$.

- When we take an action at timestep t, it can only affect the rewards from timesteps t and onwards.

Recall that,

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{k=0}^{\infty} \gamma^k r_{k+1}^{(i)} \right] \right]$$

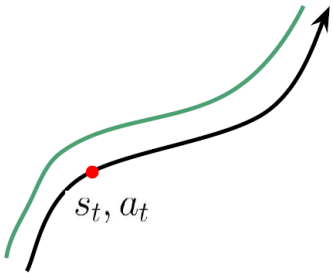
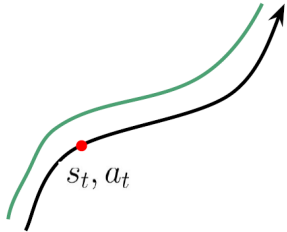


Figure Source: Jie-Han-Chen:SlideShare

Principle of Causality



$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

Let $\tau_{a:b}$ denote the states and actions visited from time a to b and

$$G_{a:b}(\tau) = \sum_{t=a}^{b} \gamma^t r_{t+1}$$

Therefore for any time t, we have,

$$G(\tau) = G_{0:t-1}(\tau) + G_{t-1:\infty}(\tau)$$

Figure Source: Jie-Han-Chen:SlideShare

Temporal Structure

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t)\cdot\left[\sum_{k=0}^{\infty}\gamma^k r_{k+1}\right]\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t) G(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t) G_{0:t-1}(\tau) + \sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t) G_{t:\infty}(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t) G_{0:t-1}(\tau)\right] \\
&\quad + \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi(a_t|s_t) G_{t:\infty}(\tau)\right]
\end{aligned}
$$

Temporal Structure

Consider evaluating the expectation of the first term

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log\pi(a_t|s_t)G_{0:t-1}(\tau)\right] = \left[\sum_{t=0}^{\infty}G_{0:t-1}(\tau)\,\mathbb{E}_{\pi_\theta}\,\nabla_\theta \log\pi(a_t|s_t)\right]$$
$$= \sum_{t=0}^{\infty}G_{0:t-1}\cdot 0 = 0$$

Therefore, the policy gradient estimate with temporal structure is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log\pi(a_t|s_t)G_{t:\infty}(\tau)\right]$$

Temporal Structure

The sample estimate of the gradient expression is given by

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}^{(i)} \right] \right]$$

- The above policy gradient estimate with temporal structure is also an unbiased estimate of the true policy gradient but has **lower variance** since it has '*thrown out*' a few terms

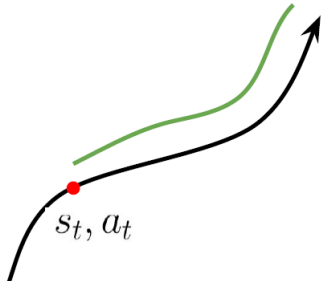
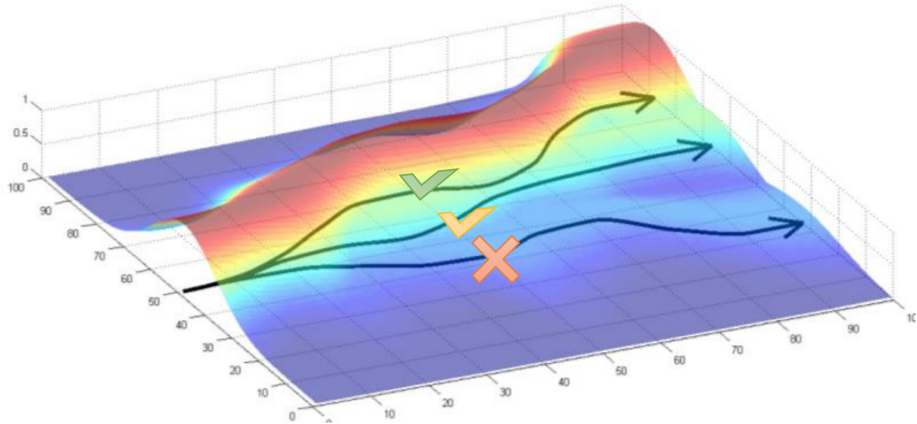


Figure Source: Jie-Han-Chen:SlideShare

Need for a Baseline



What if all paths have positive reward sum ?

Baseline

Can we subtract a baseline without biasing the gradient ?

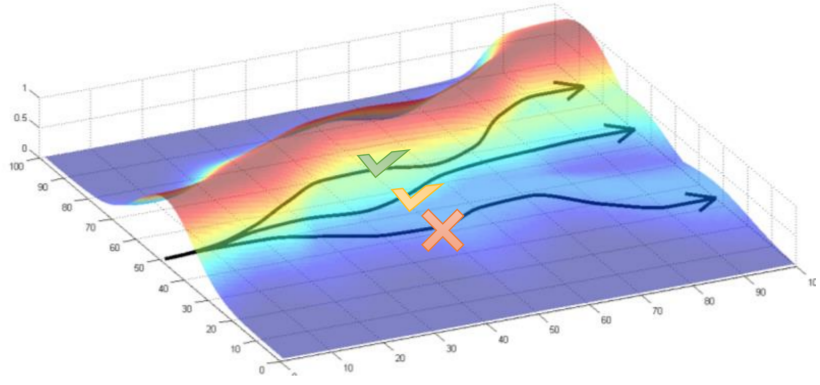
Let $b(s_t)$ be a baseline that is conditioned on s_t. Then,

$$\mathbb{E}_{a_t|s_t}\left[b(s_t)\nabla_\theta \log \pi(a_t|s_t)\right] = b(s_t)\mathbb{E}_{a_t|s_t}\left[\nabla_\theta \log \pi(a_t|s_t)\right] = 0$$

Therefore,

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi(a_t|s_t)\cdot G_{t:\infty}(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi(a_t|s_t)\cdot G_{t:\infty}(\tau)\right] - \mathbb{E}_{\tau\sim\pi_\theta}\left[b(s_t)\nabla_\theta \log \pi(a_t|s_t)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi(a_t|s_t)\cdot \left[G_{t:\infty}(\tau) - b(s_t)\right]\right]
\end{aligned}$$

Need for a Baseline



A good choice for baseline :

$$b = \mathbb{E}(G(\tau)) \approx \frac{1}{K} \sum_{i=1}^{K} G(\tau^{(i)})$$

Slide Credit: Pieter Abeel : Deep RL Boot Camp

Popular choices of Baseline

- Constant Baseline

$$b = \mathbb{E}(G(\tau)) \approx \frac{1}{K}\sum_{i=1}^{K} G(\tau^{(i)})$$

- Time Dependent Baseline

$$b_t = \frac{1}{K}\sum_{i=1}^{K} G_{t:\infty}(\tau^{(i)})$$

- Optimal Baseline

$$b = \frac{\mathbb{E}_\tau(\nabla_\theta \log \pi(a_t|s_t)^2 G_{t:\infty}(\tau))}{\mathbb{E}_\tau(\nabla_\theta \log \pi(a_t|s_t)^2)}$$

- State dependent expected return

$$b(s) = \mathbb{E}_{\pi_\theta}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s] = V^\pi(s)$$

State dependent expected return

$$b(s) = \mathbb{E}_{\pi_\theta}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s] = V^\pi(s)$$

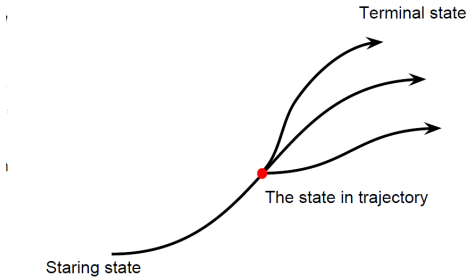


Figure Source: Jie-Han-Chen:SlideShare

Algorithm Vanilla Policy Gradient Algorithm

1: Initialize policy network π with parameters θ_1 learning rate α and baseline b
2: **for** $n = 1$ to N **do**
3: Sample K trajectories by executing the policy π_{θ_n}
4: At each time step of each trajectory compute $G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t+1}$ and advantage estimate $A_t = G_t - b(s_t)$
5: Calculate gradient estimate

$$\nabla_{\theta_n} J(\theta_n) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_{\theta_n} \log \pi(a_t^{(i)} | s_t^{(i)}) A_t \right]$$

6: Perform gradient update

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

7: **end for**

Improvements to Vanilla Policy Gradient

- The REINFORCE and Vanilla policy gradient as described above is on-policy
  - ★ There is an off-policy way to do policy gradient algorithms
- We do learning by Monte-Carlo roll-outs
  - ★ Will be addressed by Actor-Critic method

Actor Critic Methods

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

October 26, 2024

Overview

TCS Research & Innovation

Review

Policy Based Reinforcement Learning

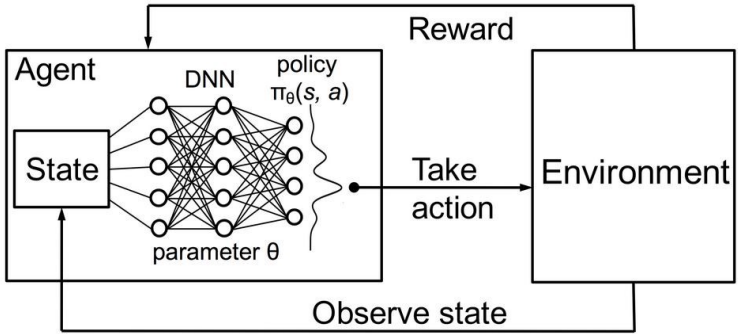
TCS Research & Innovation

- We will directly parametrize the policy

$$\pi_\theta(a|s) = P(a|s, \theta)$$

- We will consider model free control with parametrized policies
  - ★ With state-value functions Q, computing arg max over actions gets tricky when action space is large or continuous
  - ★ Better convergence properties
  - ★ Can learn stochastic policies

Policy Using Function Approximators



- If action space is discrete
  - ★ Network could output a vector of probabilities (softmax)
- If action space is continuous
  - ★ Network could output the parameters of a distribution (For e.g.. mean and variance of a Gaussian)

Policy Optimization

A policy $\pi(\cdot)$ is parametrized by parameter θ and denoted by π_θ

Performance of a policy π_θ is given by

$$J(\theta) = V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

Goal of RL is to find a policy

$$\pi_\theta^* = \arg\max_{\pi_\theta} V^{\pi_\theta}(s) = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

We will look for π_θ^* in class of stochastic policies by finding θ that maximizes $J(\theta)$

Policy Gradient Estimate

- Gradient derivation yields the following estimate

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log P(\tau; \theta) G(\tau) \right]$$

- Sample based estimate is given by

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \nabla_\theta \log P(\tau^{(i)}; \theta) G(\tau^{(i)})$$

Policy Gradient : Model Free Formulation

Model free formulation of the policy gradient is given by

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \right] \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}^{(i)} \right]$$

REINFORCE : Monte-Carlo based Policy Gradient

Algorithm REINFORCE : MC based Policy Gradient

1: Initialize policy network π with parameters θ_1 and learning rate α
2: **for** $n = 1$ to N **do**
3: Sample K trajectories from π_{θ_n}
4: Calculate gradient estimate

$$\nabla_{\theta_n} J(\theta_n) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_{\theta_n} \log \pi(a_t^{(i)} | s_t^{(i)}) \right] \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}^{(i)} \right]$$

5: Perform gradient update

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

6: **end for**

Issues with Gradient Estimate

- The gradient estimate, thus calculated, is unbiased but has high variance (reason : we are sampling stochastic paths)
- Hence the gradient descent is slow to converge
- Some variance reduction techniques are required in practice

Different Policy Gradient Formulations

Gradient of the performance measure is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \Psi_t \right]$$

1. $\Psi_t = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = G_0$, Total reward of the trajectory
2. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} = G_{t:\infty}$,Total reward following action a_t
3. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} - b(s_{t'}) = G_{t:\infty} - b(s_t)$, Baseline version of the previous formula

Vanilla Policy Gradient Algorithm

Algorithm Vanilla Policy Gradient Algorithm

1: Initialize policy network π with parameters θ_1 learning rate α and baseline b
2: **for** $n = 1$ to N **do**
3: Sample K trajectories by executing the policy π_{θ_n}
4: At each time step of each trajectory compute $G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t+1}$ and advantage estimate $A_t = G_t - b(s_t)$
5: Calculate gradient estimate

$$\nabla_{\theta_n} J(\theta_n) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_{\theta_n} \log \pi(a_t^{(i)} | s_t^{(i)}) A_t \right]$$

6: Perform gradient update

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

7: **end for**

Improvements to Vanilla Policy Gradient

- The REINFORCE and Vanilla policy gradient as described above is on-policy
  - ★ There is an off-policy way to do policy gradient algorithms
- We do learning by Monte-Carlo roll-outs
  - ★ Will be addressed by Actor-Critic method

Towards Actor-Critic Formulation

Temporal Structure and Actor-Critic Algorithms

The policy gradient estimate with temporal structure (takes causality into account) is given by

$$\nabla_\theta J(\theta) \quad = \quad \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log\pi(a_t|s_t)G_{t:\infty}(\tau)\right]$$

where

$$G_{a:b}(\tau) = \sum_{t=a}^{b}\gamma^t r_{t+1}$$

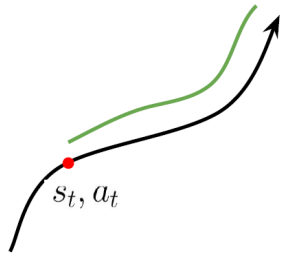
Sample estimate is given by

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}\left[\sum_{t=0}^{\infty}\nabla_\theta \log\pi(a_t^{(i)}|s_t^{(i)})\cdot\left[\sum_{t'=t}^{\infty}\gamma^{t'-t}r_{t'+1}^{(i)}\right]\right]$$

- This gradient estimate is the starting point of actor-critic algorithms

Temporal Structure and Actor-Critic Algorithms

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}^{(i)} \right] \right]$$

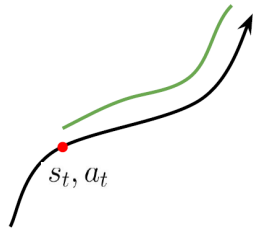


The green curve represents the entity in the inner summation

Figure Source: Jie-Han-Chen:SlideShare

The Critic

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}^{(i)} \right] \right]$$

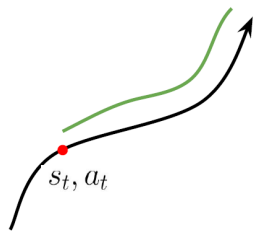


The inner summation is an estimate of $Q^{\pi_\theta}(s_t, a_t)$!!

Figure Source: Jie-Han-Chen:SlideShare

The Critic

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t^{(i)} | s_t^{(i)}) \cdot \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}^{(i)} \right] \right]$$



The inner summation is an estimate of $Q(s_t, a_t)$ and it gives an estimate of how 'good' the action a_t was in state s_t (and hence the name '**critic**')

Figure Source: UCB : Sergey Levine

Policy Gradient Theorem

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t) G_{t:\infty}(\tau)\right\}\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t) \underbrace{\mathbb{E}_{\tau\sim\pi_\theta}\left(G_{t:\infty}(\tau)\middle|\, s_t, a_t\right)}_{??}\right\}\right]
\end{aligned}$$

TCS Research & Innovation

Policy Gradient Theorem

For suitable objective function $J(\theta)$, we have,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \underbrace{\pi_\theta(a_t|s_t)}_{\text{Actor}} \underbrace{Q^{\pi_\theta}(s_t,a_t)}_{\text{Critic}}\right\}\right]$$

- Replacaes the single path reward by $Q^{\pi_\theta}(s_t, a_t)$
- Policy gradient theorem applies to start state objective, average reward objective and average value objective

(More on this later !!)

Policy Gradient with Baseline

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t)(G_{t:\infty} - b(s_t))\right\}\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t)(G_{t:\infty} - \underbrace{\mathbb{E}_{\pi_\theta}(G_{t:\infty}|s_t)}_{??})\right\}\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t)(Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t))\right\}\right]
\end{aligned}
$$

Advantage Function

- Advantage function

$$A^{\pi_\theta}(s,a) \stackrel{\text{def}}{=} Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t,a_t)\right\}\right]$$

How can we estimate the advantage function using samples ?

We already had a way in the lecture on policy gradient !

Algorithm Vanilla Policy Gradient Algorithm

1: Initialize policy network π with parameters θ_1 learning rate α and baseline b
2: **for** $n = 1$ to N **do**
3: Sample K trajectories by executing the policy π_{θ_n}
4: At each time step of each trajectory compute $G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t+1}$ and advantage estimate $A_t = G_t - b(s_t)$
5: Calculate gradient estimate

$$\nabla_{\theta_n} J(\theta_n) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{\infty} \nabla_{\theta_n} \log \pi(a_t^{(i)} | s_t^{(i)}) A_t \right]$$

6: Perform gradient update

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

7: **end for**

Advantage Function Estimate

$$A^{\pi_\theta} = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t+1} - b(s_t)$$

where

$$b(s_t) = \frac{1}{K} \sum_{i=1}^{K} G_{t:\infty}(\tau^{(i)})$$

(time dependent baseline)

- Unbiased estimate but variance is high due to the fact that it is a single sample estimate
- But, we can't roll out trajectories from state s_t as we also need a the algorithm to be online

Estimator for Advantage Function

- Consider the definition of advantage function

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

- Try having function approximator V_{ϕ} for V^{π_θ}
- Consider one-step TD error for V^{π_θ}

$$\delta_t^{\pi_\theta} = r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)$$

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}(\delta^{\pi_\theta}|s_t,a_t) &= \underbrace{E_{\pi_\theta}(r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1})|s_t,a_t)}_{??} - V^{\pi_\theta}(s_t) \\
&= Q^{\pi_\theta}(s_t,a_t) - V^{\pi_\theta}(s_t) = A^{\pi_\theta}(s_t,a_t)
\end{aligned}$$

- The one-step TD error is an unbiased estimate of the advantage function

$$\therefore \quad \nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\left\{\nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t^{\pi_\theta}\right\}\right]$$

Estimator for Advantage Function

- In practice, use the approximate TD error using the function approximator V_ϕ (for V^{π_θ}) as an estimate of the advantage function

$$A^{\pi_\theta}(s_t, a_t) \approx r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

- Note : If we fit V_ϕ using Fitted V iteration, the approximator is biased

Actor Critic Algorithms

Batch Actor Critic Algorithm

Algorithm Batch Actor-Critic Algorithm

1: Initialize critic ϕ, actor θ
2: **for** Repeat over several transitions **do**
3: Sample K transitions (s_i, a_i, r_i, s_i') using π_θ
4: Fit $V_\phi(s_i)$ to sampled reward sums from s_i
5: Evaluate the advantage function (for all K samples) using

$$A^{\pi_\theta}(s_i, a_i) \approx r_i + \gamma V_\phi(s_i') - V_\phi(s_i)$$

6: Update actor $\theta \leftarrow \theta + \alpha \sum_{i=1}^{K} \nabla_\theta \log \pi_\theta(a_i|s_i) A^{\pi_\theta}(s_i, a_i)$
7: **end for**

The V function can be fitted using fitted V iteration

Online Actor Critic Algorithm

Algorithm Online Actor-Critic Algorithm

1: Initialize state s, critic ϕ, actor θ
2: **for** Repeat over several transitions **do**
3: Let a be the action suggested by policy π_θ at state s
4: Take action a, observe reward r and next state s' and get a transition (s, a, r, s')
5: Fit $V_\phi(s)$ using target $r + V_\phi(s')$
6: Evaluate the advantage function using

$$A^{\pi_\theta}(s, a) \approx r + \gamma V_\phi(s') - V_\phi(s)$$

7: Compute $\nabla_\theta J(\theta) \leftarrow \nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s, a)$
8: Update actor $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
9: **end for**

- Fitting V_ϕ has moving target and data correlation problem
- The gradient update of the actor in Step 6 has lot of variance (single sample estimate)

Advantage Actor Critic Algorithms

Steps 5 and 7 works best with a batch (parallel workers)

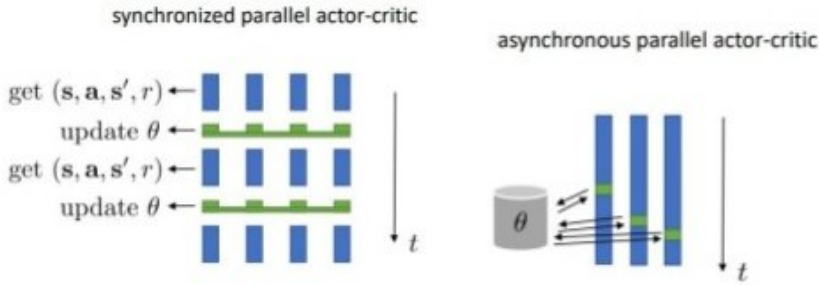


Figure Source: UCB : Sergey Levine

On Applicabilty of A3C Algorithms

- The A3C (with its synchronous version) requires multiple worker threads to simulate samples for gradient computation
- Useful when simulators are available. Instantiate multiple copies of simulator
- In many real applications, this can be an expensive step
  - ★ Navigation of physical robots (require many physical robots)
  - ★ Driving a car (requires samples generated from multiple cars)

Towards n-step returns

- One step TD error based Advantage estimate

$$A_{\mathrm{C}}^{\pi_\theta}(s,a) \approx r + \gamma V_\phi(s') - V_\phi(s)$$

  - ★ Low variance
  - ★ Biased due to the use of function approximators

- Monte Carlo based Advantage estimate

$$A_{\mathrm{MC}}^{\pi_\theta}(s,a) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t+1} - b(s)$$

  - ★ High variance
  - ★ No bias

Towards n-step returns

- We considered the critic who provides one-step TD error

$$\delta_t^{(1)} = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

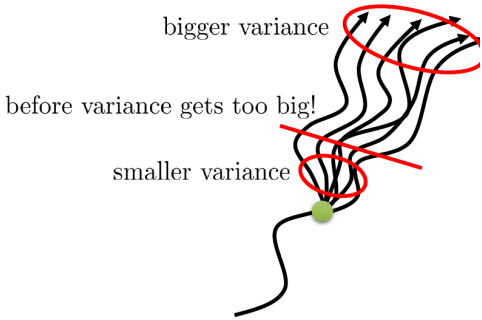
  as feedback to the actor

- We could also consider a critic that provides n-step TD error as feedback to the actor where the n-step TD error

$$\delta_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) - V(s_t)$$

- In theory, $\delta_t^{(n)}$ is also an unbiased estimate of A^{π_θ} if $V = V^{\pi_\theta}$
- Gives rise to a method called Generalized Advantage Estimation (GAE)

Towards n-step returns



$$A_n^{\pi_\theta}(s_t, a_t) \approx \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s'_t, a'_t) + \gamma^n V_\phi(s_{t+n}) - V_\phi(s_t)$$

Figure Source: UCB : Sergey Levine

TD-λ as Targets

- We could also consider the TD(λ) error given by

$$\delta_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\delta_t^{(n)}$$

  for the critic formulation (again unbiased in theory)
- The critic itself can be updated using TD(λ)
- Both TD(λ) (critic and the feedback) updates can be implemented using eligibility traces

A3C

- Asynchronous methods for deep Asynchronous methods for deep reinforcement learning (2016)
- Online actor crtic and paralleized batch
- N-step returns with N = 4 steps
- Single network for actor and critic

Different Policy Gradient Formulations

Gradient of the performance measure is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \Psi_t \right]$$

1. $\Psi_t = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = G_0$, Total reward of the trajectory
2. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} = G_{t:\infty}$,Total reward following action a_t
3. $\Psi_t = \sum_{t'=t}^{\infty} \gamma^{t'} r_{t'+1} - b(s_{t'}) = G_{t:\infty} - b(s_t)$, Baseline version of the previous formula
4. $\Psi_t = \gamma^t Q^{\pi_\theta}(s_t, a_t)$, State action value function
5. $\Psi_t = \gamma^t A^{\pi_\theta}(s_t, a_t) = \gamma^t \left[Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right]$, Advantage function
6. $\Psi_t = \gamma^t \left[r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \right]$, TD residual

Towards Deterministic Policy Gradient Formulations

Stationary Distribution of Markov Chain

- Given a MDP $< \mathcal{M} = \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ and a policy π_θ, we have an induced Markov chain given by $< \mathcal{S}, \mathcal{P}^{\pi_\theta} >$
- Imagine that you can travel along the Markov chain's states forever, and eventually, as the time progresses, the probability of you ending up with at state s from state s_0 (start state) becomes unchanged and is given by

$$d^{\pi_\theta}(s) = \lim_{t \to \infty} \mathbb{P}(s_t = s | s_0, \pi_\theta)$$

- The entity $d^{\pi_\theta}(s)$ is the limiiting (stationary as well) distribution of Markov chain and is assumed to independent of s_0
- Existence of such stationary distribution can be guaranteed under certain some conditions on the Markov chain

Objective Function Formulations

- In episodic environments, we can use the value of the start state as the objective function given by

$$J_1(\theta) = V^{\pi_\theta}(s) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

- In **continuing** environments we have a slightly different formulation for the objective function given by,

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$$

where

$$d^{\pi_\theta}(s) = \lim_{t \to \infty} \mathbb{P}(s_t = s | s_0, \pi_\theta)$$

- **Idea** : Average of $V^{\pi_\theta}(s)$ computed using $d^{\pi_\theta}(s)$ as weights (for all $s \in \mathcal{S}$).
- Average is computed from the tail of episodic sequence starting at state s_0
- Second equality uses the relationship between V^{π_θ} and Q^{π_θ}

Stochastic Policy Gradient Theorem

For any differentiable policy π_θ, for any of the policy objective functions $J(\theta) = J_1(\theta)$, $\frac{1}{1-\gamma} J_{avV}(\theta)$, the gradient estimate of the objective function with respect to the parameter θ, under some conditions, is given by,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \left\{ \nabla_\theta \log \underbrace{\pi_\theta(a_t|s_t)}_{\text{Actor}} \underbrace{Q^{\pi_\theta}(s_t, a_t)}_{\text{Critic}} \right\} \right]$$

Deterministic Policy Gradient Algorithm : Key Ideas

- Thus far, considered the policy function $\pi(\cdot|s)$ as a probability distribution over actions space and thus considered stochastic policies
- **Deterministic policy gradient algorithms (DPG)** instead models the policy as a deterministic decision : $a = \pi(s)$
- Specifically DDPG, an off-policy actor-critic algoirthm, can be thought of as DQN for continuous action space setting
- Interleaves between learning optimal action-value function $Q^*(s, a)$ and learning optimal policy $\pi^*(s)$
- Uses Bellman equation to learn $Q^*(s, a)$ and policy gradients to learn $\pi^*(s)$

Deterministic Policy Gradient Algorithm : Key Ideas

- Bellman equation is the starting point for learning optimal action-value function $Q^*(s, a)$.
- Optimal action-value function int the DQN setting is learnt using the following MSBE function

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

- However, in the DDPG setting, we calculate the max over actions using the policy netwoek as follows,

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

Deterministic Policy Gradient Algorithm : Key Ideas

- Policy is learnt by recoginizing that we are looking for a deterministic policy $\pi_\theta(s)$ that gives an action that maximizes $Q_\phi(s, a)$. Achieved by, performing gradient ascent on the following objective function

$$\max_\theta \mathbb{E}_{s \in \mathcal{D}} Q_\phi(s, \pi_\theta(s))$$

- Because the policy that is being learnt is deterministic, to make DDPG policies explore better, we add noise to their actions at training time.
  - ★ OU noise
  - ★ zero-mean Gaussian noise
- Target networks are updated using Polyak averaging
- The idea of deterministic policy gradient has connections to the stochastic policy gradient setting (in the limiting case)

Deep Deterministic Policy Gradient (DDPG)

Algorithm Deep Deterministic Policy Gradient

Initialize state s, critic ϕ, actor θ and replay buffer
Initialize target critic $\phi' \leftarrow \phi$, target actor $\theta' \leftarrow \theta$
for Repeat over several episodes **do**
    Initialize a random process N for exploration (eg. Ornstein-Uhlenbeck process), and observe initial state s
    for Repeat over transitions **do**
        Apply action $a = \pi_\theta(s) + N_t$, observe reward r and next state s', and store the transition (s, a, r, s') in the replay buffer
        Sample a random minibatch of transitions (s_i, a_i, r_i, s'_i) from the buffer
        Compute SARSA target values $y_i = r_i + Q_{\phi'}(s'_i, \pi_{\theta'}(s'_i))$
        Update critic by minimizing MSE loss $\frac{1}{n}\sum_i (y_i - Q_\phi(s_i, a_i))^2$
        Update actor using sampled deterministic policy gradient $\frac{1}{n}\sum_i \nabla_a Q_\phi(s_i, \pi_\theta(s_i)) \nabla_\theta \pi_\theta(s_i)$
        Perform soft updates on target networks
            $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
            $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
    end for
end for

Advanced Policy Gradients

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

Novemer 02, 2024

Overview of this Lecture

TCS Research & Innovation

Introduction

Policy Gradients : The Story So Far

Policy gradient algorithms try to solve the optimization problem

$$\max_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$$

by taking stochastic gradient ascent on the policy parameters θ, using the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \Psi_t \right]$$

1. $\Psi_t = \gamma^t Q^{\pi_\theta}(s_t, a_t)$, State action value function
2. $\Psi_t = \gamma^t A^{\pi_\theta}(s_t, a_t) = \gamma^t \left[Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right]$, Advantage function
3. $\Psi_t = \gamma^t \left[r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \right]$, TD residual

Limitation 1 : Sample Inefficiency

Gradient of the performance measure is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\Psi_t\right]$$

- Policy Gradient seen thus far is **on-policy**
- Gradient update is performed using samples collected from current policy
- Above formulation does not allow to recycle old data
- If we want to use samples from other policies, then the above gradient term needs correction using **importance sampling** weights

Importance Sampling in PG : Possible Remedy ?

What if we do not samples from π_θ instead samples are from π_η(i.e. $\tau \sim \pi_\eta$) ?

We can rewrite the gradient estimate as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\eta} \left[\frac{P(\tau|\pi_\theta)}{P(\tau|\pi_\eta)} \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\ \Psi_t \right]$$

$$\frac{P(\tau|\pi_\theta)}{P(\tau|\pi_\eta)} = \frac{\mu(s_0) \prod_{t=0}^{\infty} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)}{\mu(s_0) \prod_{t=0}^{\infty} P(s_{t+1}|s_t, a_t)\pi_\eta(a_t|s_t)} = \prod_{t=0}^{\infty} \frac{\pi_\theta(a_t|s_t)}{\pi_\eta(a_t|s_t)}$$

Even for policies only slightly different from each other, many small differences multiply to become a big difference and IS weights can **explode** or **vanish**

Problem : How can we efficiently use samples from old policies while avoiding the challenges posed by importance sampling ?

Limitation 2 : Updates in Parameter Space

- Policy gradient takes step in parameter space

$$\theta_{n+1} = \theta_n + \alpha \nabla_{\theta_n} J(\theta_n)$$

- **Distance** in parameter space $\neq$ Distance in policy space
- Hard to get step size right as a result
- Example of a policy space : For finite state and action case, the policy space is given by

$$\Pi = \left\{ \pi : \pi \in \mathbb{R}^{|S| \times |A|}, \sum_a \pi_{sa} = 1, 0 \leq \pi_{sa} \leq 1 \right\}$$

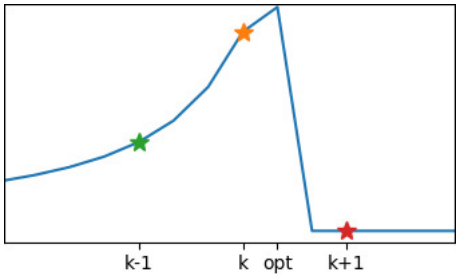
The Step Size Issue

Policy Gradient algorithms perform stochastic gradient ascent

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$$

with step $\Delta_k = \alpha_k \hat{g}_k$

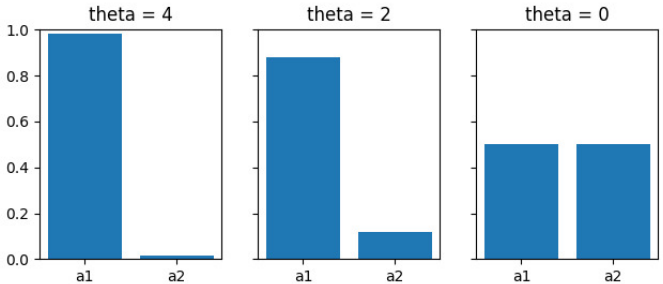
- Step size is too large, performance collapse
  - ★ Consequences in RL setting is more severe than for supervised learning
- Step size is too low, slow progress



Difference in Policy Space : Discrete Case

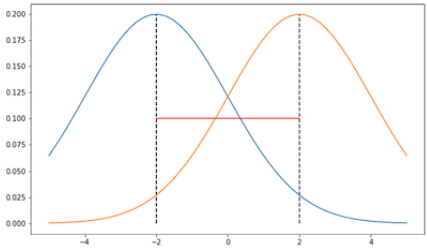
Consider a family of policies with the following parametrization

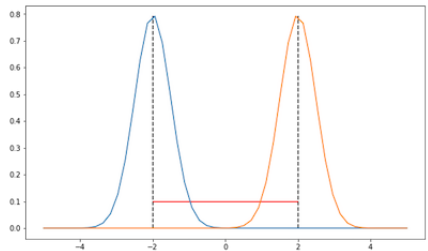
$$\pi_\theta(a) = \begin{cases} \sigma(\theta), & \text{if } a = 1 \\ 1 - \sigma(\theta), & \text{if } a = 2 \end{cases}$$



Small changes in the policy parameters can unexpectedly lead to big changes in the policy

Difference in Policy Space : Continuous Case





Desiderata

- How to make use of data from old policy while avoiding challenges that arise from importance sampling ?
  - ★ At least use roll-outs from **most recent policy** as effectively as possible
- How to design an update rule that doesn't change the policy more than we intend to ?
  - ★ Take steps that respect notion of **distance in policy space** rather than in parameter space

Policy Performance Bounds

Relative Performance of Two Policies

Recall that the performance of a policy π is given by

$$J(\pi) = V^{\pi}(s_0)$$

where $s_0 \in \mu(s)$ is the start state of the trajectory π

Relative Performance Identity of Two Policies

For any two policies π' and π_0 we have,

$$J(\pi') - J(\pi_0) = \mathop{\mathbb{E}}_{\tau \sim \pi'} \left[\gamma^t A^{\pi_0}(s_t, a_t) \right]$$

Proof of Relative Performance Identity of Two Policies

$$
\begin{aligned}
J(\pi') - J(\pi_0)) &\stackrel{?}{=} \mathop{\mathbb{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_0}(s_t,a_t)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t\left[r_{t+1}+\gamma V^{\pi_0}(s_{t+1})-V^{\pi_0}(s_t)\right]\right] \\
&= J(\pi') + \mathop{\mathbb{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^{t+1}V^{\pi_0}(s_{t+1}) - \sum_{t=0}^{\infty}\gamma^t V^{\pi_0}(s_t)\right] \\
&= J(\pi') + \mathop{\mathbb{E}}_{\tau\sim\pi'}\left[\sum_{t=1}^{\infty}\gamma^t V^{\pi_0}(s_t) - \sum_{t=0}^{\infty}\gamma^t V^{\pi_0}(s_t)\right] \\
&= J(\pi') - \mathop{\mathbb{E}}_{\tau\sim\pi'}\left[V^{\pi_0}(s_0)\right] \\
&= J(\pi') - J(\pi_0)
\end{aligned}
$$

Policy Improvement Idea

Can we use the identity,

$$J(\pi') - J(\pi_0) = \mathop{\mathbb{E}}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_0}(s_t, a_t) \right]$$

for policy improvement to go from **old policy** π_0 to **new policy** π' ?

$$\arg\max_{\pi'} J(\pi') = \arg\max_{\pi'} \left[J(\pi') - J(\pi_0) \right] = \arg\max_{\pi'} \left[\mathop{\mathbb{E}}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_0}(s_t, a_t) \right] \right]$$

- Define performance of π' in terms of advantages from π_0
- Problem : Still requires trajectories from π' !!

Discounted State Distribution

Define discounted future state distribution d^π for a state s as

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}(s_t = s|\pi)$$

Consider the expectation

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t,a_t)\right]$$

The above expectation can be rewritten using the discounted future state distribution d^π as

$$\frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d^\pi\\ a\sim\pi}}\left[A^\pi(s,a)\right]$$

That is,

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t,a_t)\right] = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d^\pi\\ a\sim\pi}}\left[A^\pi(s,a)\right]$$

Performance Identity and Discounted State Distribution

$$
\begin{aligned}
J(\pi') - J(\pi_0) &= \mathop{\mathbb{E}}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_0}(s_t, a_t) \right] \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d^{\pi'} \\ a \sim \pi'}} \left[A^{\pi_0}(s, a) \right] \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d^{\pi'} \\ a \sim \pi_0}} \left[\frac{\pi'(a|s)}{\pi_0(a|s)} A^{\pi_0}(s, a) \right]
\end{aligned}
$$

Only problem is $s \sim d^{\pi'}$!

Approximation of Discounted State Distribution

Under what conditions is it OK to assume

$$\frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi_0}} \left[\frac{\pi'(a|s)}{\pi_0(a|s)} A^{\pi_0}(s,a) \right] \stackrel{?}{\approx} \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_0} \\ a\sim\pi_0}} \left[\frac{\pi'(a|s)}{\pi_0(a|s)} A^{\pi_0}(s,a) \right]$$

In other words, under what conditions can we have $d^{\pi'} \approx d^{\pi_0}$?

We can have $d^{\pi'} \approx d^{\pi_0}$, if π' and π_0, are close !!

- Closeness is measured in terms of **KL divergence**
- If the policies are close in **KL divergence**, then the above approximation is good

KL Divergence Between Policies

For any two probability distributions P and Q $D_{KL}(P||Q)$ is defined as

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

KL divergence has the following properties

- $D_{KL}(P||P) = 0$
- $D_{KL}(P||Q) \geq 0$
- $D_{KL}(P||Q) \neq D_{KL}(Q||P)$

KL divergence between policies π' and π_0 is given by

$$D_{KL}(\pi'||\pi_0)[s] = \sum_{a \in \mathcal{A}} \pi'(a|s) \log \frac{\pi'(a|s)}{\pi_0(a|s)}$$

Surrogate Loss Function

If policies π' and π_0 are **'close'** in terms of their KL divergence, then,

$$\begin{aligned} J(\pi') - J(\pi_0) &= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi_0}} \left[\frac{\pi'(a|s)}{\pi_0(a|s)} A^{\pi_0}(s,a) \right] \\ &\approx \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_0} \\ a\sim\pi_0}} \left[\frac{\pi'(a|s)}{\pi_0(a|s)} A^{\pi_0}(s,a) \right] \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_0} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi_0(a_t|s_t)} A^{\pi_0}(s_t,a_t) \right] \end{aligned}$$

Define the quantity $\mathcal{L}_{\pi_0}(\pi')$ as the **surrogate loss function**, as,

$$\mathcal{L}_{\pi_0}(\pi') = \mathop{\mathbb{E}}_{\tau\sim\pi_0} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi_0(a_t|s_t)} A^{\pi_0}(s_t,a_t) \right]$$

- $J(\pi') - J(\pi_0) \approx \mathcal{L}_{\pi_0}(\pi')$

Validity of Approximation

If policies π' and π_0 are 'close' in terms of their KL divergence, then,

$$J(\pi') - J(\pi_0) \approx \mathcal{L}_{\pi_0}(\pi')$$

We can have a **relative policy performance bound** using KL divergence as

$$\Big[J(\pi') - \big(J(\pi_0) + \mathcal{L}_{\pi_0}(\pi')\big)\Big] \leq C\sqrt{\mathop{\mathbb{E}}_{s\sim d^{\pi_0}}\left[D_{KL}(\pi'||\pi_0)[s]\right]}$$

where $C = \frac{4\varepsilon\gamma}{1-\gamma^2}\alpha^2$ with $\alpha = \max_{s\sim d^{\pi_0}}\left[D_{KL}(\pi'||\pi_0)[s]\right]$ and $\varepsilon = \max_{s,a} A^{\pi_0}(s,a)$

Surrogate Loss Function : Properties

$$\mathcal{L}_{\pi_0}(\pi') = \mathop{\mathbb{E}}_{\tau\sim\pi_0}\left[\sum_{t=0}^{\infty}\gamma^t\frac{\pi'(a_t|s_t)}{\pi_0(a_t|s_t)}A^{\pi_0}(s_t,a_t)\right]$$

- $\mathcal{L}_{\pi_0}(\pi')$ is something that we can evaluate using samples from old policy π_0
- Importance sampling is used; but weights depends only on current time-step (not preceding history); hence importance sampling weights don't vanish or explode
- Let π_{θ_k} and π_θ be two parametrized policies. Then, $\nabla_\theta\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)|_{\theta=\theta_k}$, is equal to the policy gradient

$$\begin{aligned}\nabla_\theta\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)|_{\theta=\theta_k} &= \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_k}}\left[\sum_{t=0}^{\infty}\frac{\nabla_\theta\pi_\theta(a_t|s_t)|_{\theta=\theta_k}}{\pi_{\theta_k}(a_t|s_t)}\ \gamma^t A^{\pi_{\theta_k}}(s_t,a_t)\right]\\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_k}}\left[\sum_{t=0}^{\infty}\nabla_\theta\log(\pi_{\theta_k}(a_t|s_t)|_{\theta=\theta_k}\ \gamma^t A^{\pi_{\theta_k}}(s_t,a_t)\right]\end{aligned}$$

Policy Improvement Guarantee

We can rewrite the relative policy performance bound equation as

$$\big[J(\pi') - J(\pi_0)\big] \geq \mathcal{L}_{\pi_0}(\pi') - C\sqrt{\mathop{\mathbb{E}}_{s\sim d^{\pi_0}}\big[D_{KL}(\pi'||\pi_0)[s]\big]}$$

Suppose π_{k+1} and π_k are related by

$$\pi_{k+1} = \arg\max_{\pi'}\left[\mathcal{L}_{\pi_k}(\pi') - C\sqrt{\mathop{\mathbb{E}}_{s\sim d^{\pi_k}}\big[D_{KL}(\pi'||\pi_k)[s]\big]}\right]$$

- Note π_k is a feasible point and the objective at π_k is equal to 0
  - ★ $\mathcal{L}_{\pi_k}(\pi_k) \propto \mathbb{E}[A^{\pi_k}(s_t, a_t)] = 0$
  - ★ $D_{KL}(\pi_k||\pi_k) = 0$
- $\implies$ Optimal value ≥ 0
- By the performance bound inequality, we have $J(\pi_{k+1}) - J(\pi_k) \geq 0$

A First-Cut Algorithm

1: Initialize π_0
2: **for** $k = 0, 1, 2, \cdots$ until convergence **do**
3: Sample a trajectory τ from policy π_k
4: Compute advantage function $A^{\pi_{\theta_k}}(a_t, s_t)$ for all (s_t, a_t) pairs in the trajectory τ
5: Solve the optimization problem

$$\pi_{k+1} = \arg\max_{\pi'} L_{\pi_k}(\pi') - C\sqrt{\mathop{\mathbb{E}}_{s\sim d^{\pi_k}}\left[D_{KL}(\pi'||\pi_k)[s]\right]}$$

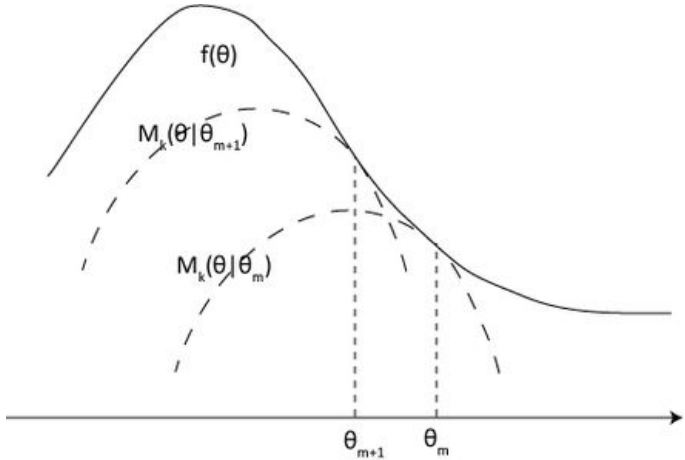
6: **end for**

Issues are :

- C is quite high when γ is close to 1 $\left(C = \frac{4\varepsilon\gamma}{1-\gamma^2}\alpha^2\right)$
- Consequently, step size becomes too small

Majorize Maximize Framework

Majorize-Maximize framework is used to solve the optimization step



Approximate Monotone Improvement

- Instead of KL penalty, use KL constraint
- Can control worst case error through constraint upper limit

$$\pi_{k+1} = \arg\max_{\pi'} \left[L_{\pi_k}(\pi') \right]$$
$$\text{such that} \quad \mathop{\mathbb{E}}_{s \sim d^{\pi_k}} D_{KL}(\pi' || \pi_k)[s] \leq \delta$$

- From the constraint, **steps respect a notion of distance in policy space**
- Above constrained optimization is basis of many algorithms, Natural Policy Gradient (NPG), truncated NPG, TRPO and PPO
- The objective and the constraint can be estimated from the roll-out of old policies – **sample efficient**
- Update is parametrization invariant

Natural Policy Gradient

Trust Region Formulation

We have the following optimization problem

$$\pi_{k+1} = \arg\max_{\pi'} \left[\mathcal{L}_{\pi_k}(\pi')\right]$$
$$\text{such that } \bar{D}_{KL}(\pi' || \pi_k) \leq \delta$$

The constraint on the optimization problem is the trust region with size δ and some guarantees on performance improvement are there within the trust region

For parametrized policies the optimization can be written as

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \left[\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)\right]$$
$$\text{such that } \bar{D}_{KL}(\pi_\theta || \pi_{\theta_k}) \leq \delta$$

How do we solve it ?

- Linear approximation for the objective
- Quadratic approximation for the constraint

Approximation of Objective Function

Taylor series expansion for function $f(x)$ around point a is given by

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \cdots$$

- Using Taylor series expansion on objective function $\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)$ around θ_k (upto first order term) gives us

$$\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) \approx \cancelto{0}{\mathcal{L}_{\pi_{\theta_k}}(\pi_{\theta_k})} + g^T(\theta - \theta_k) \qquad \text{where } g \doteq \nabla_\theta \mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)\,|_{\theta=\theta_k}$$

- Recall that g is exactly the policy gradient

$$\nabla_\theta \mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)|_{\theta=\theta_k} = \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_k}}\left[\sum_{t=0}^{\infty} \nabla_\theta \log(\pi_{\theta_k}(a_t|s_t)|_{\theta=\theta_k}\ \gamma^t A^{\pi_{\theta_k}}(s_t, a_t)\right]$$

- Objective function is simplified to

$$\theta_{k+1} = \arg\max_\theta g^T(\theta - \theta_k)$$

Approximation of Trust Region Constraint

Using Taylor series expansion on the constraint (around θ_k; upto second order) gives us

$$\bar{D}_{KL}(\pi_\theta||\pi_{\theta_k}) \approx \cancelto{0}{\bar{D}_{KL}(\pi_{\theta_k}||\pi_{\theta_k})} + \cancelto{0}{\nabla_\theta \bar{D}_{KL}(\pi_\theta||\pi_{\theta_k})|_{\theta=\theta_k}} + \nabla^2_\theta \bar{D}_{KL}(\pi_\theta||\pi_{\theta_k}) \ |_{\theta=\theta_k}$$

The first order term $\nabla_\theta \bar{D}_{KL}(\pi_\theta||\pi_{\theta_k})$ evaluates to zero since the expectation of the score function is zero

$$\nabla_\theta \bar{D}_{KL}(\pi_\theta \,\|\, \pi_{\theta_k}) = \nabla_\theta \mathop{\mathbb{E}}_{\pi_\theta}[\log \pi_\theta] - \nabla_\theta \mathop{\mathbb{E}}_{\pi_\theta}[\log \pi_{\theta_k}] = \mathop{\mathbb{E}}_{\pi_\theta}[\nabla_\theta \log \pi_\theta] = 0$$

Therefore, we are left only with the second order term

$$\bar{D}_{KL}(\pi_\theta||\pi_{\theta_k}) \approx \frac{1}{2}(\theta - \theta_k)^T H \ (\theta - \theta_k) \quad \text{where } H \doteq \nabla^2_\theta \bar{D}_{KL}(\pi_\theta||\pi_{\theta_k}) \ |_{\theta=\theta_k}$$

Natural Policy Gradient

The optimization problem is now simplified as

$$\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k)$$

$$\text{such that } \frac{1}{2}(\theta - \theta_k)^T H\ (\theta - \theta_k) \leq \delta$$

Linear objective with quadratic constraint

Solution to the approximate problem obtained using Lagrange multiplier method

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

The term $H^{-1}g$ is called the Natural gradient

Algorithm : Natural Policy Gradient

Algorithm Natural Policy Gradient

1: Initialize π_0
2: **for** $k = 0, 1, 2, \cdots$ **do**
3: Collect trajectories D_k on policy $\pi_k = \pi_{\theta_k}$
4: Estimate all advantages $A^{\pi_{\theta_k}}(s_t, a_t)$
5: Form sample estimates for policy gradients $\hat{g}_k$ (using advantage estimates)
6: Form sample estimates for the Hessian of KL divergence
7: Compute the Natural Policy Gradient update

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g_k^T H_k^{-1} g_k}} H_k^{-1} g_k$$

8: **end for**

Fisher Information Matrix and KL Divergence

- Let $p(x|\theta)$ be a probability distribution parameterized by θ.
- Score function of a parameterized probability distribution is given by

$$s(\theta) = \nabla_\theta \log p(x|\theta)\,,$$

- For a parameter vector θ, Fisher Information Matrix is given by,

$$\mathrm{F} = \mathop{\mathbb{E}}_{p(x|\theta)} \left[\nabla_\theta \log p(x|\theta)\, \nabla_\theta \log p(x|\theta)^{\mathrm{T}} \right] .$$

- The sample estimate of the above expectation is given by,

$$\mathrm{F} = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(x_i|\theta)\, \nabla_\theta \log p(x_i|\theta)^{\mathrm{T}} . \tag{1}$$

- **Claim** : Fisher Information Matrix F is the Hessian of KL-divergence between two probability distributions $p(x|\theta')$ and $p(x|\theta)$ evaluated at $\theta' = \theta$

$$\mathrm{H}_{\mathrm{KL}[p(x|\theta) \,\|\, p(x|\theta')]} = \mathrm{F}\,.$$

Properties of Natural Policy Gradient

- Natural policy gradient algorithm gives an update-rule in which updates are pre-multiplied by H^{-1}
- The Hessian of the KL-divergence is the Fischer Information Matrix given by

$$\mathrm{F} = \mathop{\mathbb{E}}_{\pi_\theta}\left[\nabla \log \pi_\theta(\cdot|s)\, \nabla \log \pi_\theta(\cdot|s)^{\mathrm{T}}\right]$$

- The NPG direction $H^{-1}g$ is **co-variant**; i.e. it points in same direction irrespective of the parametrization that is used to compute it

Relationship of Natural Gradient to Policy Gradient

Policy Gradient Formulation

Consider the following optimization problem

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \left[\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) \right]$$
$$\text{such that } \left\| \theta - \theta_k) \right\|^2 \leq \delta$$

After **linearising** the objective, the optimization problem is now,

$$\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k) \text{ such that } (\theta - \theta_k)^2 \leq \delta$$

This is the original policy gradient problem !!

We move a small distance in parameter space in the direction of the gradient

Natural Gradient Formulation

Natural policy gradient problem is given by,

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \left[\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) \right]$$
$$\text{such that } \bar{D}_{KL}(\pi_\theta || \pi_{\theta_k}) \leq \delta$$

After **linearising** the objective and **quadratifying** the constraint, fhe optimization problem is then given by,

$$\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k) \text{ such that } \frac{1}{2}(\theta - \theta_k)^T F \ (\theta - \theta_k) \leq \delta$$

Relationship between Formulations

- Vanilla policy gradient has the right objective but *"incorrect"* constraint (Euclidean penalty instead of KL penalty)
- Recall that, policy iteration (from MDP lectures) obtain policy improvement with no constraint

Other Algorithms

Truncated Natural Policy Gradient

- **Problem** : For neural networks, the dimensionality of parameter θ are high. High computational cost in inverting the matrix H
- **Solution** : Use the **conjugate gradient algorithm** to compute $H^{-1}g$ without inverting H
- Resultant algorithm : Truncated Natural Policy Gradient
- ACTKR algorithm uses KFAC technique to solve the inverse Hessian computation problem

Problems with Natural Policy Gradient Update

- Another problem with NPG update is that - might not be robust to trust region size δ
  - ★ δ may be too large in some iterations and can degrade the performance
- Because of quadratic approximation, the KL-divergence constraint may be violated
- Monotonic improvement may not occur in all iterations

TRPO : Line Search Algorithm

- Enforce improvement in surrogate (i.e. $\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) \geq 0$)
- Enforce KL constraint
- How ? Backtracking line search with exponential decay

Algorithm Line Search for TRPO

Compute the proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{g_k^T H_k^{-1} g_k}} H_k^{-1} g_k$
for $j = 0, 1, 2, \cdots N$ **do**
  Compute proposed update $\theta = \theta_k + \alpha_j \Delta_k$
  If $\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) \geq 0$ and $\bar{D}_{KL}(\theta || \theta_k) \leq \delta$
    Accept the update $\theta = \theta_k + \alpha_j \Delta_k$
  Else
    Find another α_j (Reduce α_j)
end for

Algorithm : Trust Region Policy Optimization

Algorithm Trust Region Policy Optimization

1: Initialize π_0
2: **for** $k = 0, 1, 2, \cdots$ **do**
3: Collect trajectories D_k on policy $\pi_k = \pi_{\theta_k}$
4: Estimate all advantages $A^{\pi_{\theta_k}}(s_t, a_t)$
5: Form sample estimates for policy gradients $\hat{g}_k$ (using advantage estimates)
6: Form sample estimates for the Hessian of KL divergence / FIM
7: Use conjugate gradient to obtain FIM estimate H^{-1}
8: Estimate step size α using backtracking line search to enforce KL constraint and monotonic improvement
9: Compute the Natural Policy Gradient update

$$\theta_{k+1} = \theta_k + \alpha \, \Delta_k$$

10: **end for**

Proximal Policy Optimization

Proximal Policy Optimization is a family of methods that approximately enforce without actually computing the natural gradient

- **Adaptive KL Penalty**

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \left[\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) - \beta \bar{D}_{KL}(\pi_\theta || \pi_{\theta_k}) \right]$$

Penalty co-efficient β is changed between iterations to approximately enforce KL constraint

- **Clipped Objective** (Simpler to implement, no need to check KL constraint; works well)

$$\mathcal{L}^{CLIP}_{\pi_{\theta_k}}(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{T} \min \left(r_t(\theta) A_t^{\pi_{\theta_k}}, clip(1-\varepsilon, 1+\varepsilon) A_t^{\pi_{\theta_k}} \right) \right]$$

where $r_t(\theta)$ is the importance sampling ratio between target policy π_θ and behaviour policy π_{θ_k} and policy update takes place as

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \mathcal{L}^{CLIP}_{\pi_{\theta_k}}(\pi_\theta)$$

PPO : KL Constraint

Algorithm Proximal Policy Optimization : KL Constraint

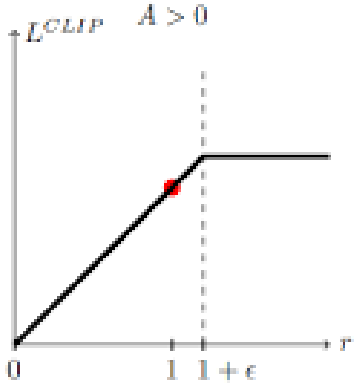
Initialize π_0 initial KL penalty β_0 and target KL divergence δ
for $k = 0, 1, 2, \cdots$ **do**
  Collect set of trajectories on policy $\pi_k = \pi_{\theta_k}$
  Estimate advantages $A^{\pi_{\theta_k}}(s_t, a_t)$
  Compute policy update by solving (using SGD)

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \left[\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta) - \beta_k \bar{D}_{KL}(\pi_\theta || \pi_{\theta_k}) \right]$$

  If $\bar{D}_{KL}(\pi_{\theta_{k+1}} || \pi_{\theta_k}) \geq 1.5\delta$ then
    $\beta_{k+1} = 2\beta_k$
  Else if $\bar{D}_{KL}(\pi_{\theta_{k+1}} || \pi_{\theta_k}) \leq \delta / 1.5$
    $\beta_{k+1} = \beta_k / 2$
end for

- Initial KL penalty not that important; it adapts quickly
- Some iterations may violate KL constraint, but most don't

Clipped Objective : Rationale



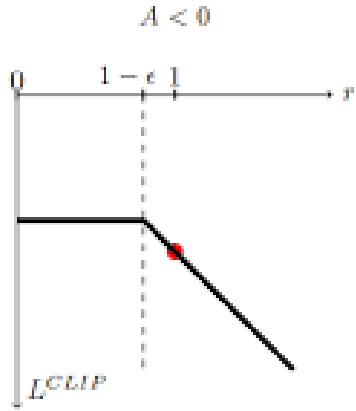


Figure Source: https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe

PPO : Clipped Objective

Algorithm Proximal Policy Optimization : Clipped Objective

1: Initialize π_0
2: **for** $k = 0, 1, 2, \cdots$ **do**
3: Collect set of trajectories on policy $\pi_k = \pi_{\theta_k}$
4: Estimate advantages $A^{\pi_{\theta_k}}(s_t, a_t)$
5: Compute policy update by solving (using SGD)

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} \mathcal{L}^{CLIP}_{\pi_{\theta_k}}(\pi_\theta)$$

where

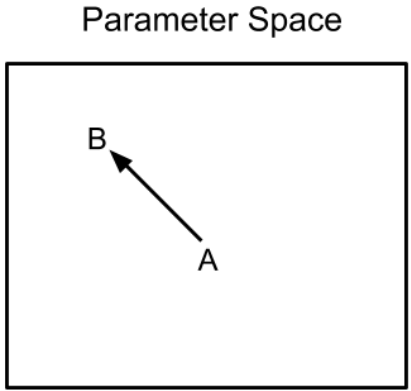
$$\mathcal{L}^{CLIP}_{\pi_{\theta_k}}(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{T} \min \left(r_t(\theta) A_t^{\pi_{\theta_k}}, clip(1-\varepsilon, 1+\varepsilon) A_t^{\pi_{\theta_k}} \right) \right]$$

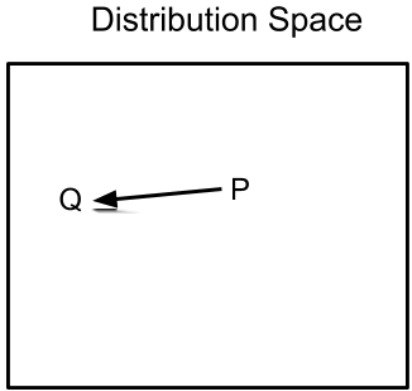
6: **end for**

- Clipping prevents the new policy $\pi_{\theta_{k+1}}$ to go far away from π_{θ_k}
- Clipping is simpler to implement and works well in practice

Insights into Natural Gradients

Parameter and Distribution Space





Natural Gradients

- Natural gradient is not the gradient with respect to model parameters; rather, it is the gradient on the manifold of probabilistic models
- Traditional gradients considers how perturbations of the model parameters affect the loss function; Natural gradients considers how the loss function changes when the probabilistic model moves a little bit on the manifold of model family
- Because the model family does not change with respect to parameterizations, the natural gradient will also remain the same.
- Natural gradients are invariant to parameterization. That is, Natural gradients under two different parameterizations correspond to the same "gradient entity" on the manifold

Consider the following model that captures the joint distribution between two random variables x and y

$$p_\theta(x, y) = p(x)\mathcal{N}(y \mid \theta x + b, \sigma^2), \theta \in \mathbb{R},$$

where θ is a parameter and b and σ are constants.

Let $\theta = 2\mu$. We can rewrite the above model in terms of new parameter μ as

$$p_\mu(x, y) = p(x)\mathcal{N}(y \mid 2\mu x + b, \sigma^2), \mu \in \mathbb{R}.$$

Note that even if $p_\theta(x, y)$ and $p_\mu(x, y)$ have different analytical forms, they represent the same (family) distribution. Specifically,

$$p_{\theta=a}(x, y) \equiv p_{\mu=a/2}(x, y), \text{ for any } a \in \mathbb{R}$$

Gradient Descent and Parameterization

Construct a negative log-likelihood as loss function for the distribution $p(x, y)$ as

$$\begin{aligned}
\mathcal{L}(p(x,y)) &= -[\log p(x,y)] \\
&= \left[\frac{1}{2\sigma^2}(\theta x + b - y)^2 + \log\sqrt{2\pi}\sigma\right] \quad (\text{for } p_\theta(x,y)) \\
&= \left[\frac{1}{2\sigma^2}(2\mu x + b - y)^2 + \log\sqrt{2\pi}\sigma\right] \quad (\text{for } p_\mu(x,y))
\end{aligned}$$

Gradient Descent and Parameterization

- For $p_\theta(x, y)$ the update rule is given by $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}(p_{\theta_t}(x, y))$ where α is the step size with

$$\nabla_{\theta_t} \mathcal{L}(p_{\theta_t}(x, y)) = \left[\frac{x}{\sigma^2} (\theta_t x + b - y) \right]$$

- For $p_\mu(x, y)$ the update rule is given by $\mu_{t+1} = \mu_t - \alpha \nabla_{\mu_t} \mathcal{L}(p_{\mu_t}(x, y))$ where α is the step size with

$$\nabla_{\mu_t} \mathcal{L}(p_{\mu_t}(x, y)) = \left[\frac{2x}{\sigma^2} (2\mu_t x + b - y) \right]$$

Gradient Descent and Parameterization

- Let $\theta_t = 2\mu_t = a$ for some $a \in \mathbb{R}$
- That is, at the t-th step of the gradient descent, the parametric probabilistic models $p_{\theta=a}(x, y)$ and $p_{\mu=a/2}(x, y)$ represent the same distribution
- Because

$$\nabla_{\theta_t} \mathcal{L}(p_{\theta_t}(x, y))|_{\theta_t=a} = \frac{1}{2} \nabla_{\mu_t} \mathcal{L}(p_{\mu_t}(x, y))|_{\mu_t=a/2}$$

  we will have,

$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}(p_{\theta_t}(x, y)) \\ &= 2\mu_t - \frac{\alpha}{2} \nabla_{\mu_t} \mathcal{L}(p_{\mu_t}(x, y)) \\ &\neq 2\mu_t - 2\alpha \nabla_{\mu_t} \mathcal{L}(p_{\mu_t}(x, y)) \\ &= 2\mu_{t+1} \end{aligned}$$

- Hence the $t+1$-th optimization step will result in different probabilistic models $p_{\theta_{t+1}}(x, y) \not\equiv p_{\mu_{t+1}}(x, y)$

Gradient Descent and Parameterization

- One step of gradient descent will result in different models depending on which parameterization is used
- Hence, requires carefully choosing the parameterization to avoid hindering optimization
- Not all optimization procedures are parameterization dependent. For example, the Newton-Raphson method is invariant to affine transformations of model parameters
- Natural gradient methods are invariant to arbitrary differentiable transformations of model parameters when the learning rate is small enough

Sanity Check : Natural Gradients

Recall that the natural gradient of a loss function is given by,

$$\widetilde{\nabla}_\theta \mathcal{L}(p_\theta(x,y)) := \mathbf{F}_\theta^{-1} \nabla_\theta \mathcal{L}(p_\theta(x,y)),$$

where

$$[\mathbf{F}_\theta]_{i,j} := \mathbb{E}_{p_\theta(x,y)} \left[\left(\frac{\partial}{\partial \theta_i} \log p_\theta(x,y) \right) \left(\frac{\partial}{\partial \theta_j} \log p_\theta(x,y) \right) \right],$$

For different parameterizations of Gaussian conditional distributions, the derivatives of the log densities are respectively

$$\begin{aligned} \frac{\partial}{\partial \theta} \log p_\theta(x,y) &= \frac{(y-\theta x-b)x}{\sigma^2} \\ \frac{\partial}{\partial \mu} \log p_\mu(x,y) &= \frac{2(y-2\mu x-b)x}{\sigma^2} \end{aligned}$$

With $\theta_t = 2\mu_t$, we can conclude that $\mathbf{F}_{\mu=\mu_t} = 4\mathbf{F}_{\theta=\theta_t}$. Using

$$\widetilde{\nabla}_\mu \mathcal{L}(p_\mu(x,y))|_{\mu=\mu_t} = \frac{1}{2} \widetilde{\nabla}_\theta \mathcal{L}(p_\theta(x,y))|_{\theta=\theta_t}$$

we can conclude that, $\theta_{t+1} = 2\mu_{t+1}$

Metric Tensor (Distance Function)

In a Reimannian space, the distance between points v and $v + \delta v$ can be expressed as

$$dist^2(v + \delta v, v) = \delta v^T \mathcal{G}(v) \delta v$$

The $\mathcal{G}(v)$ is the **metric tensor** (or the distance function) in the space and it depends on the co-ordinate system that represents vector v

Example

The Euclidean space $\mathbb{R}^2$ can be represented in Cartesian coordinates or polar coordinates. If $v = (x, y)$ in Cartesian coordinates (metric tensor is identity matrix)

$$dist^2(v + \delta v, v) = \delta x^2 + \delta y^2 = \delta v^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \delta v$$

In polar coordinates,

$$dist^2(v + \delta v, v) = (\delta r, \delta \theta)^T \underbrace{diag(1, r^2)}(\delta r, \delta \theta)$$

The Fischer Information matrix F is the metric tensor in policy space.

Covariance of Natural Policy Gradient

Consider the same vector and vector difference in two coordinate systems

- In system 1 (v), we write $(v; \delta v)$, and the metric tensor is written as $\mathcal{G}_v$
- In system 2 (w), we write $(w; \delta w)$, and the metric tensor is written as $\mathcal{G}_w$
- Note that $v = w$ (same vector in different parametrizations).

Because the deltas are also equal. they are related by,

$$\delta v_i = \sum_j \frac{\partial v_i}{\partial w_j} \partial w_j \implies \delta v = A^T \delta w, \text{ where } A_{ji} = \frac{\partial v_i}{\partial w_j}$$

The distances must be the same in both, so metrics are related as follows:

$$\begin{aligned} dist^2(v, v+\delta v) &= dist^2(w, w+\delta w) \\ dist^2(v, v+\delta v) &= \delta v^T \mathcal{G}_v \delta v = \delta w^T A \mathcal{G}_v A^T \delta w \\ dist^2(w, w+\delta w) &= \delta w^T \mathcal{G}_w \delta w \implies \mathcal{G}_w = A \mathcal{G}_v A^T \end{aligned}$$

Gradients are related by chain rule:

$$[g_w]_j = \frac{\partial f}{\partial w_j} = \sum_i \frac{\partial v_i}{\partial w_j} \frac{\partial f}{\partial v_i} \implies g_w = A g_v$$

Slide Credit: Joshua Aicham : OpenAI

Covariance of Natural Policy Gradient

Now we ask the question if $\Delta v = \mathcal{G}_v^{-1} g_v$ and $\Delta w = \mathcal{G}_w^{-1} g_w$ are same vectors

$$\begin{aligned} \Delta w &= \mathcal{G}_w^{-1} g_w \\ &= (A\mathcal{G}_v A^T)^{-1} A g_v \\ &= (A^T)^{-1} \mathcal{G}_v^{-1} A^{-1} A g_v \\ &= (A^T)^{-1} \Delta v \end{aligned}$$

$$\implies A^T \Delta w = \Delta v$$

They are the same vectors and the natural gradient $H^{-1} g$ is invariant to parameterization

Recommended Reading

- "Why Natural Gradient?" S. Amari and S. C. Douglas, 1998
- "Natural Policy Gradient," Sham Kakade, 2001
- "Reinforcement Learning of Motor Skills with Policy Gradients," Jan Peters and Stefan Schaal, 2008
- "Trust Region Policy Optimization," Schulman et al. 2015
- "Benchmarking Deep Reinforcement Learning for Continuous control," Duan et al. 2016
- "Proximal Policy Optimization Algorithms," Schulman et al. 2017