Value Function Approximation

- RL is usually used for large problems eg.
- Backgammon - 10^{20} states
- Go - 10^{170} states
- Helicopter - continuous state space.
- We cannot (in almost all cases) create tables for states.
- So, we need some way to extend the model-free control methods for prediction and control.

Until now:

- Value function was represented as a lookup table i.e every state s has an entry $V(s)$ or for every (s,a) pair, there was an entry $Q(s,a)$.

Problem with Large MDP's:

1) Too many states / action pairs to store in memory.
2) Too slow to learn the value of each state individually

Solution for Large MDP's:

- Estimate value function with function approximation.

$$\hat{v}(s, w) \approx V_\pi(s) \quad \text{or} \quad \hat{q}(s, a, w) \approx q_\pi(s, a).$$

1) Generalise from seen states to unseen states.
2) Update param w using MC or TD learning.

Types of Value-Function Approximation:

→ We either approximate V or Q.

For V:

$$s \to \boxed{w} \to \hat{v}(s,w).$$

For Q:

① $s, a \to \boxed{w} \to \hat{q}(s,a,w)$

② $s \to \boxed{w} \to \hat{q}(s,a_1,w), \ldots, \hat{q}(s,a_n,w).$

→ Which Function Approximator?

examples:

1) Linear combination of features
2) Neural Network.
3) DT's
4) KNN's. etc.

For RL, we need <u>differentiable</u> function approximators, so they are Linear combination and Neural Networks.

2 Methods for Function Approximation:

1) Incremental Methods
2) Batch Methods.

Gradient Descent

→ Let $J(w)$ be a differentiable function of parameter vector w.

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{pmatrix}_w .$$

→ Goal: to find a local minima of J.

So, we adjust w in the direction of -ve gradient.

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w). \quad (\alpha \text{ is step-size})$$

Value Function Approx by Stochastic Gradient Descent:

Goal: Find param w which minimises MSE between $\hat{v}(s,w)$ and true value f^n $v_\pi(s)$.

$$J(w) = E_\pi\left[(v_\pi(s) - \hat{v}(s,w))^2\right].$$

Assume for now, that v_π is present (or given to you)

→ GD finds a local minimum:

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w).$$

$$= \alpha E_\pi\left[(v_\pi(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w)\right]$$

→ SGD samples the Gradient.

$$\Delta w = \alpha \left(v_\pi(S) - \hat{v}(S,w)\right) \nabla_w \hat{v}(S,w).$$

Feature Vector

Represent state by a feature vector.

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix}$$

ex: 1) Distance of a robot from landmarks

2) Piece configurations in Chess.

Linear Value Function Approximation:

→ Represent Value f^n by a linear combi of features

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$$

→ Objective function is quadratic in w, i.e

$$J(w) = E_\pi\left[\left(v_\pi(S) - x(S)^T w\right)^2\right]$$

→ SGD will converge on global optimum.

$$\nabla_w \hat{v}(S,w) = x(S).$$

$$\Delta w = \alpha \left(v_\pi(S) - \hat{v}(S,w)\right) x(S).$$

So, update = step-size × prediction error × feature value.

Table Lookup Features:

→ Table lookup is a special case of linear value function approximation

→ Using table lookup features

$$x^{table}(S) = \begin{pmatrix} 1(S=S_1) \\ \vdots \\ 1(S=S_n) \end{pmatrix}$$

→ parameter $\underline{w}$ gives value of each individual state

$$\hat{v}(S,w) = \begin{pmatrix} 1(S=S_1) \\ \vdots \\ 1(S=S_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}.$$

Incremental Prediction Algos:

→ we assumed true value f^n $v_\pi(S)$ given by a supervisor.

→ But obviously this is cheating, we only have rewards.

→ In practice, we substitute a <u>target</u> for $v_\pi(S)$.

→ <u>For MC</u>, the target is G_t

$$\Delta w = \alpha\left(G_t - \hat{v}(S_t, w)\right) \nabla_w \hat{v}(S_t, w)$$

→ For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$

$$\Delta w = \alpha\left(R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\right) \nabla_w \hat{v}(S_t, w).$$

→ For TD(λ), the target is the λ return G_t^λ

$$\Delta w = \alpha\left(G_t^\lambda - \hat{v}(S_t, w)\right) \nabla_w \hat{v}(S_t, w)$$

1) Monte Carlo.

→ G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$.

→ We can therefore treat

$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_T, G_T \rangle$ as training data in

Supervised Learning.

→ So, if our loss was MSE, the update would be

$$\Delta w = \alpha\left(G_t - \hat{v}(S_t, w)\right) \nabla_w \hat{v}(S_t, w)$$

~~sed~~

2) TD learning:

→ The TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$ is a biased sample of true value $v_\pi(S_t)$.

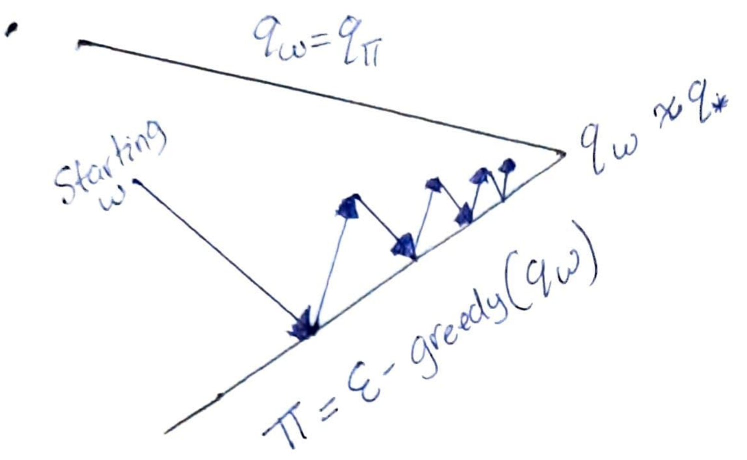
→ we can still apply supervised learning to "training data":

$$\left(S_1, R_2 + \gamma \hat{v}(S_2, w)\right), \ldots, \left(S_{t-1}, R_t\right)$$

ex: For MSE, $\Delta w = \alpha\left(R + \gamma \hat{v}(S', w) - \hat{v}(S, w)\right) \nabla_w \hat{v}(S, w)$

Control with Value Function Approximation:

- **Policy Evaluation**: Approximate policy evaluation, $\hat{q}(\cdot,\cdot,\omega) \approx q_\pi$
- **Policy Improvement** ε-greedy policy improvement.



Action Value Function Approximation:

- Approximate the action-value function
$$\hat{q}(S, A, \omega) \approx q_\pi(S, A)$$
- Minimise MSE between approximate action value f^n $\hat{q}(S, A, \omega)$ and true action-value f^n $q_\pi(S, A)$
$$J(\omega) = \mathbb{E}_\pi\left[\left(q_\pi(s,a) - \hat{q}(s,a,\omega)\right)^2\right]$$
- Use SGD to find a local minimum.

→ Represent state and action by a Feature vector:

$$X(S,A) = \begin{pmatrix} x_1(s,a) \\ \vdots \\ x_n(s,a) \end{pmatrix}$$

Simplest Case: Linear Approximator

$$\hat{q}(s,a,w) = x(s,a)^T w = \sum_{j=1}^{n} x_j(s,a) w_j$$

→ Do the same as with Value f^n approximation to approximate $q(s,a)$.

Policy Based RL

- Until now, we were approximating the value function or the Q-value function.
  The policy was generated directly from the value function
- Now, we will directly parametrise / approximate the policy.

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s, \theta]$$

Value Based

- Learnt Value function
- Implicit Policy

Policy Based

- No value function
- Learnt policy

Actor Critic

- learn both value f^n and policy

Advantages of Policy-Based

- Better convergence properties
- Effective in high-dimensional or continuous action-spaces
  Most important reason
- Can learn stochastic policies.

Disadvantages.

→ Typically converge to a local rather than global optimum

→ Evaluating a policy is typically inefficient and high variance.

Q) Why would we need stochastic policies?

ex: Rock-Paper-Scissors.

→ A deterministic policy is easily exploited.

Policy Objective Functions:

Goal: Given policy $\pi_\theta(s,a)$ with parameters θ, find θ.

→ But how do we measure the quality of a policy π_θ?

1) In episodic environments we can use the start value.

$$J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}[v_1]$$

2) In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s)\, V^{\pi_\theta}(s).$$

or the average reward per time step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a.$$

→ where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain

for π_θ.

Policy Optimisation

→ Policy Based RL is an optimisation problem.

→ Find θ that maximises $J(\theta)$.

Policy Gradient

→ Let $J(\theta)$ be any policy objective function.

→ Policy gradient algos search for a local maxima in $J(\theta)$ by ascending the gradient of the policy w.r.t params θ

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

where $\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{pmatrix}(\theta)$, α is step-size.

Computing Gradients by Finite Differences:

→ To evaluate policy gradient of $\pi_\theta(s,a)$

For each dim $k \in [1, n]$,

→ Estimate the k^{th} p.d of J w.r.t θ as follows,

$$\frac{\partial J}{\partial \theta_k} \approx \frac{J(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon},$$

where u_k is $(0, 0, \ldots, 1, \ldots 0)$ — k^{th} position