Value Function Approximation

Introduction to Reinforcement Learning

Bogdan Ivanyuk-Skulskyi, Dmytro Kuzmenko

Department of Mathematics,
National University of Kyiv-Mohyla Academy

March 13, 2023

Table of Contents

A note on Monte Carlo vs TD estimates

- Policy evaluation: $V^{\pi} \leftarrow (1-\alpha)\hat{V}^{\pi} + \alpha V_{target}$
- MC: $V_{target}(s_t) = G_t$ (sum of discounted returns until the episode terminates)
    - Target is unbiased estimate of V^{π}
    - Target can be high variance
- $TD(0)$: $V_{target}(s_t) = r_t + \gamma \hat{V}(s)$
    - Target is a biased estimate of V^{π}
    - Target is lower variance
- n-step TD: $V_{target}(s_t) = r_t + \gamma r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^n \hat{V}(s_{t+n})$

Table of Contents

Monte Carlo Value Function Approximation

- Return G_t is an unbiased but noisy sample of the true expected return $V^\pi(s_t)$
- Therefore can reduce MC VFA to doing supervised learning on a set of (state,return) pairs: $<s_1, G_1>, <s_2, G_2>, \ldots, <s_T, G_T>$
  - Substitute G_t for the true $V^\pi(s_t)$ when fit function approximator
- Concretely when using linear VFA for policy evaluation

$$\Delta w = \alpha(G_t - \hat{V}(s_t; w))\nabla_w \hat{V}(s_t, w)$$

$$= \alpha(G_t - \hat{V}(s_t; w))x(s_t)$$

$$= \alpha(G_t - x(s_t)^T w)x(s_t)$$

- Note: G_t may be a very noisy estimate of true return

MC Linear Value Function Approximation for Policy Evaluation

Init $w = 0$, $k = 1$
Loop

- Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,L_k})$ given π
- for $t = 1, ..., L_k$ do
  - if First visit to (s) in episode k then
    - $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$
    - Update weights: $w = \alpha(G_t(s) - x(s)^T w)x(s)$

$k = k + 1$

Table of Contents

Temporal Difference (TD(0)) Learning with Value Function Approximation

- Uses bootstrapping and sampling to approximate true V^{π}
- Updates estimate V^{π} after each transition (s, a, r, s):

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(r + \gamma V^{\pi}(s') - V^{\pi}(s))$$

- Target is $r + \gamma V^{\pi}(s')$, a biased estimate of the true value $V^{\pi}(s)$
- In value function approximation, target is $r + \gamma V^{\pi}(s'; w)$, a biased and approximated estimate of the true value $V^{\pi}(s)$
- 3 forms of approximation:
  - Sampling
  - Bootstrapping
  - Value function approximation

Temporal Difference (TD(0)) Learning with Value Function Approximation

- In value function approximation, target is $r + \gamma V^\pi(s'; w)$, a biased and approximated estimate of the true value $V^\pi(s)$
- Can reduce doing TD(0) learning with value function approximation to supervised learning on a set of data pairs: $<s_1, r_1 + \gamma V^\pi(s_2; w)>$, $<s_2, r_2 + \gamma V^\pi(s_3; w)>$...
- Find weights to minimize mean squared error

$$J(w) = \mathbb{E}_\pi\left[(r_j + \gamma \hat{V}^\pi(s_{j+1}, w) - \hat{V}^\pi(s_j; w))^2\right]$$

TEMPORAL DIFFERENCE (TD(0)) LEARNING WITH VALUE FUNCTION APPROXIMATION

- In value function approximation, target is $r + \gamma V^{\pi}(s'; w)$, a biased and approximated estimate of the true value $V^{\pi}(s)$
- Can reduce doing TD(0) learning with value function approximation to supervised learning on a set of data pairs: $<s_1, r_1 + \gamma V^{\pi}(s_2; w)>, <s_2, r_2 + \gamma V^{\pi}(s_3; w)> \ldots$
- In linear TD(0)

$$\begin{aligned}\Delta w &= \alpha(r + \gamma \hat{V}^{\pi}(s; w) - \hat{V}^{\pi}(s; w)) \nabla_w \hat{V}^{\pi}(s; w) \\ &= \alpha(r + \gamma \hat{V}^{\pi}(s; w) - \hat{V}^{\pi}(s; w)) x(s) \\ &= \alpha(r + \gamma x(s')^T w - x(s)^T w) x(s)\end{aligned}$$

TD(0) Linear Value Function Approximation for Policy Evaluation

- Initialize $w = 0$,$k = 1$
- Loop
  - Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
  - Update weights:
$$w = w + (r + \gamma x(s')^T w - x(s)^T w)x(s)$$
  - $k+ = 1$

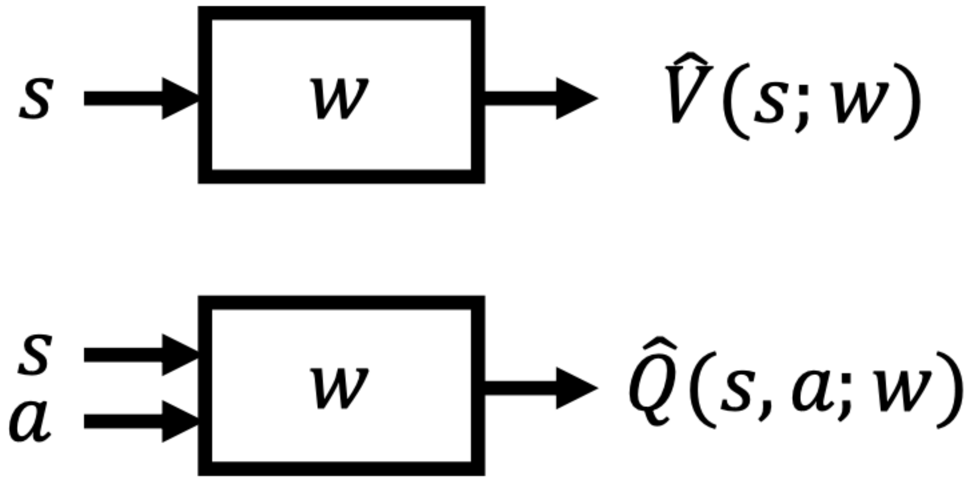
Table of Contents

Control using Value Function Approximation

- Use value function approximation to represent state-action values $\hat{Q}^{\pi}(s, a, w) \approx Q^{\pi}$
- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform $\epsilon - greedy$ policy improvement
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping
  - Off-policy learning

Control with VFA

Represent state-action value function by Q-network with weights w $\hat{Q}(s,a;w) \approx Q(s,a)$



Action-Value Function Approximation with an Oracle

- $Q^{\pi}(s,a;w) \approx Q^{\pi}$
- Minimize the mean-squared error between the true action-value function $Q^{\pi}(s,a)$ and the approximate action-value function:

$$J(w) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(s,a) - \hat{Q}^{\pi}(s,a;w)\right)^2\right]$$

- Use stochastic gradient descent to find a local minimum

$$(w) = -\frac{1}{2}\alpha\nabla_w J(w) = \alpha\mathbb{E}\left[\left(Q^{\pi}(s,a) - \hat{Q}^{\pi}(s,a;w)\right)\nabla_w \hat{Q}^{\pi}(s,a;w)\right]$$

- Stochastic gradient descent (SGD) samples the gradient

Linear State Action Value Function Approximation with an Oracle

- Use features to represent both the state and action

$$x(s,a) = (x_1(s,a), x_2(s,a), ..., x_n(s,a))$$

- Represent state-action value function with a weighted linear combination of features

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

- Stochastic gradient descent update:

$$\nabla_w J(w) = \nabla_w \mathbb{E}_\pi \left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a;w))^2 \right]$$

Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta w = \alpha(G_t - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

- For SARSA instead use a TD target $r + \gamma\hat{Q}(s, a; w)$ which leverages the current function approximation value

$$\Delta w = \alpha(r + \gamma\hat{Q}(s, a; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

- For Q-learning instead use a TD target $r + \gamma max_a \hat{Q}(s, a; w)$ which leverages the max of the current function approximation value

$$\Delta w = \alpha(r + \gamma max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

RL with Function Approximation

- Linear value function approximators assume value function is a weighted combination of a set of features, where each feature a function of the state
- Linear VFA often work well given the right set of features
- But can require carefully hand designing that feature set
- An alternative is to use a much richer function approximation class that is able to directly go from states without requiring an explicit specification of features
- Local representations including Kernel based approaches have some appealing properties (including convergence results under certain cases) but can't typically scale well to enormous spaces and datasets

The Benefit of Deep Neural Network Approximators

- Uses distributed representations instead of local representations
- Universal function approximator
- Can potentially need exponentially less nodes/parameters (compared to a shallow net) to represent the same function
- Can learn the parameters using stochastic gradient descent

Model-Free Control with General Function Approximators

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- Similar to linear value function approximation, but gradient with respect to complex function
- Monte Carlo: use return G_t as target:

$$\Delta w = \alpha(G_t - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

- SARSA: use a TD target $r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; w)$, with current function approximation value

$$\Delta w = \alpha(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; w) - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

- For Q-learning

$$\Delta w = \alpha(r + \gamma max_a \hat{Q}(s_{t+1}, a; w) - \hat{Q}(s_t, a_t; w))\nabla_w \hat{Q}(s_t, a_t; w)$$

Q-Learning with Value Function Approximation

- Q-learning converges to the optimal $Q^*(s, a)$ using table lookup representation
- In value function approximation Q-learning we can minimize MSE loss by stochastic gradient descent using a target Q estimate instead of true Q (as we saw with linear VFA)
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses these challenges by
  - Experience replay
  - Fixed Q-targets

DQNs: Experience Replay

- To help remove correlations, store dataset D from prior experience

$$s_1, a_1, r_2, s_2$$

$$s_2, a_2, r_3, s_3$$

$$s_3, a_3, r_4, s_4$$

$$\dots$$

$$s_t, a_t, r_{t+1}, s_{t+1}$$

- To perform experience replay, repeat the following:
  - $(s, a, r, s) \sim D$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma max_{a'} \hat{Q}(s', a'; w)$
  - Use stochastic gradient descent to update the network weights

$$\Delta w = \alpha(r + \gamma max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w)$$

- Uses target as a scalar, but function weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters w^- be the set of weights used in the target, and w be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim D$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma max_{a'} \hat{Q}(s', a'; w)$
  - Use stochastic gradient descent to update the network weights

$$\Delta w = \alpha(r + \gamma max_{a'} \hat{Q}(s', a'; w^-) - \hat{Q}(s, a; w)) \nabla_w \hat{Q}(s, a; w)$$

DQNs Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old, fixed parameters w
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

DEEP RL

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)