

Lecture 6: Value Function Approximation

David Silver

Outline

1 Introduction

2 Incremental Methods

3 Batch Methods

Outline

1 Introduction

2 Incremental Methods

3 Batch Methods

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up the model-free methods for *prediction* and *control* from the last two lectures?

Value Function Approximation

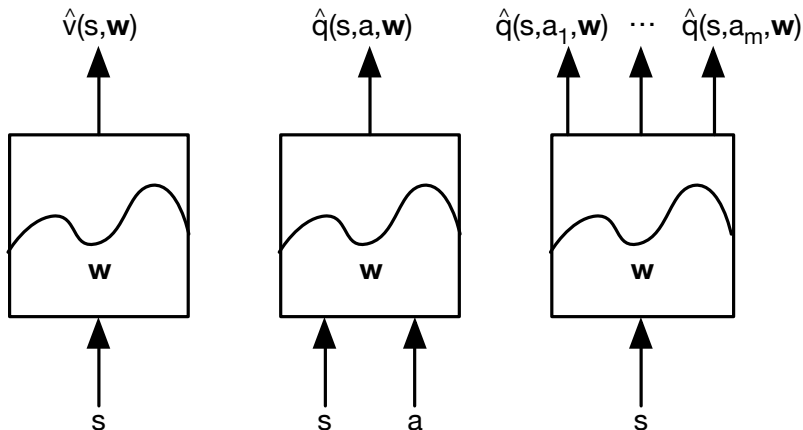
- So far we have represented value function by a *lookup table*
 - Every state s has an entry $V(s)$
 - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
 - There are too many states and/or actions to store in memory
 - It is **too slow** to learn the value of each state individually
- Solution for large MDPs:
 - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_{\pi}(s)$$

or $\hat{q}(s, a, \mathbf{w}) \approx q_{\pi}(s, a)$

- Generalise from **seen states to unseen states**
- Update parameter \mathbf{w} using MC or TD learning

Types of Value Function Approximation



Which Function Approximator?

There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Which Function Approximator?

We consider **differentiable** function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Furthermore, we require a training method that is suitable for **non-stationary**, **non-iid** data



Outline

1 Introduction

2 Incremental Methods

3 Batch Methods

Gradient Descent

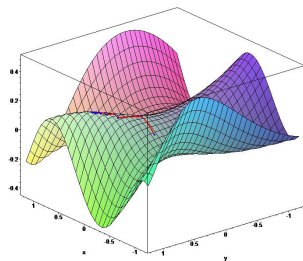
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector \mathbf{w}
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust \mathbf{w} in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter



Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector \mathbf{w} minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
 - Distance of robot from landmarks
 - Trends in the stock market
 - Piece and pawn configurations in chess



Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^n \mathbf{x}_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters \mathbf{w}

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

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

Update = *step-size* \times *prediction error* \times *feature value*

Table Lookup Features

- Table lookup is a **special** case of linear value function approximation
- Using *table lookup features*

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

- Parameter vector \mathbf{w} gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
 - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(\mathbf{G}_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha(\mathbf{R}_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha(\mathbf{G}_t^\lambda - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Monte-Carlo with Value Function Approximation

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to “training data”:

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \dots, \langle S_T, G_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(\textcolor{red}{G}_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

TD Learning with Value Function Approximation

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to “training data”:

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, \dots, \langle S_{T-1}, R_T \rangle$$

- For example, using *linear* $TD(0)$

$$\begin{aligned} \Delta \mathbf{w} &= \alpha (\mathbf{R} + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S) \end{aligned}$$

- Linear $TD(0)$ converges (close) to global optimum



TD(λ) with Value Function Approximation

- The λ -return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to “training data”:

$$\langle S_1, G_1^\lambda \rangle, \langle S_2, G_2^\lambda \rangle, \dots, \langle S_{T-1}, G_{T-1}^\lambda \rangle$$

- Forward view linear TD(λ)

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(\mathbf{G}_t^\lambda - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(\mathbf{G}_t^\lambda - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)\end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

TD(λ) with Value Function Approximation

- The λ -return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to “training data”:

$$\langle S_1, G_1^\lambda \rangle, \langle S_2, G_2^\lambda \rangle, \dots, \langle S_{T-1}, G_{T-1}^\lambda \rangle$$

- Forward view linear TD(λ)

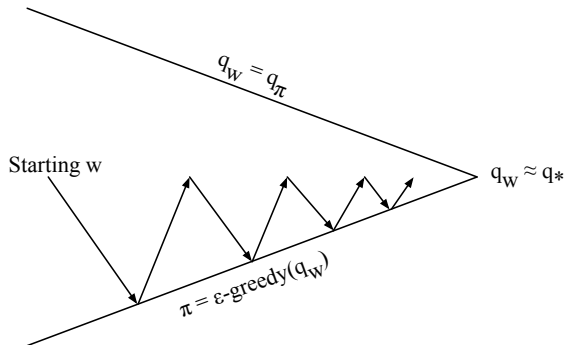
$$\begin{aligned}\Delta \mathbf{w} &= \alpha(\mathbf{G}_t^\lambda - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(\mathbf{G}_t^\lambda - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)\end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma \lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t\end{aligned}$$

Forward view and backward view linear TD(λ) are equivalent

Control with Value Function Approximation



Policy evaluation **Approximate** policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_{\pi}(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_{\pi}(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_{\pi} [(q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w}))^2]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^n \mathbf{x}_j(S, A) \mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \mathbf{x}(S, A)$$

Incremental Control Algorithms

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
 - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(\mathbf{G}_t - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(\mathbf{R}_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For forward-view TD(λ), target is the action-value λ -return

$$\Delta \mathbf{w} = \alpha(\mathbf{q}_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For backward-view TD(λ), equivalent update is

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$

$$E_t = \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \delta_t E_t$$

Convergence of Prediction Algorithms

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---------------|-----------------|--------------|--------|------------|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✓ | ✗ |
| | TD(λ) | ✓ | ✓ | ✗ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✗ | ✗ |
| | TD(λ) | ✓ | ✗ | ✗ |

Outline

1 Introduction

2 Incremental Methods

3 Batch Methods

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is *not* sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience ("training data")

Least Squares Prediction

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* \mathcal{D} consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$\mathcal{D} = \{ \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle \}$$

- Which parameters \mathbf{w} give the *best fitting* value fn $\hat{v}(s, \mathbf{w})$?
- **Least squares** algorithms find parameter vector \mathbf{w} minimising sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and target values v_t^π ,

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^T (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_{\mathcal{D}} [(v^\pi - \hat{v}(s, \mathbf{w}))^2] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$\mathcal{D} = \{ \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle \}$$

Repeat:

- 1 Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$



- 2 Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha (v^\pi - \hat{v}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$\mathcal{D} = \{ \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle \}$$

Repeat:

- 1 Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

- 2 Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha (v^\pi - \hat{v}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}} LS(\mathbf{w})$$

Experience Replay in Deep Q-Networks (DQN)



DQN uses **experience replay** and **fixed Q-targets**

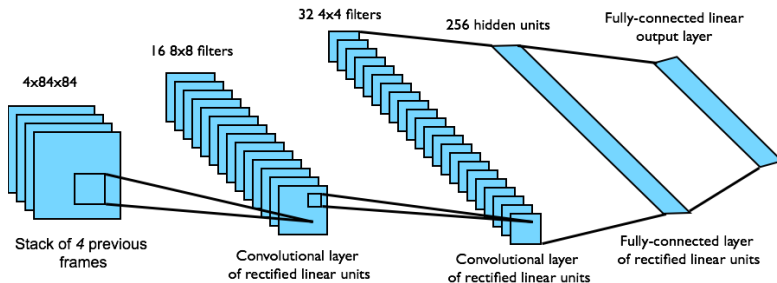
- Take action a_t according to ϵ -greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory \mathcal{D}
- Sample **random** mini-batch of transitions (s, a, r, s') from \mathcal{D}
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent

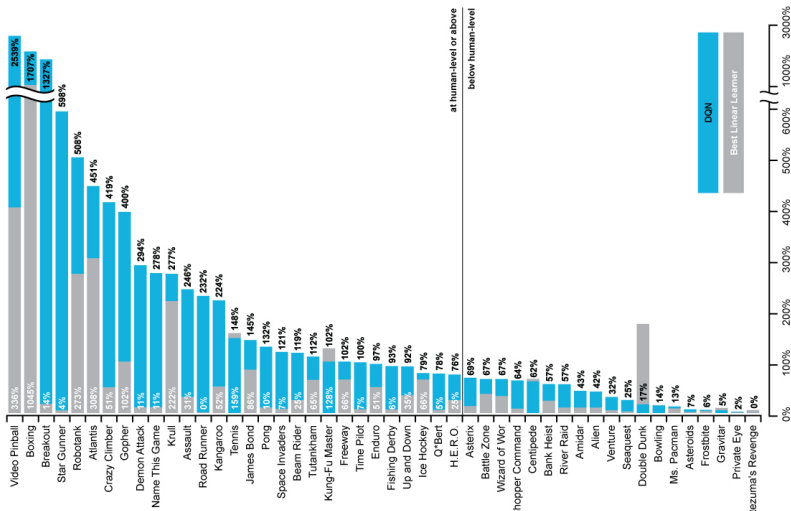
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from **last 4 frames**
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

DQN Results in Atari



How much does DQN help?



| | Replay Fixed-Q | Replay Q-learning | No replay Fixed-Q | No replay Q-learning |
|----------------|-------------------|----------------------|----------------------|-------------------------|
| Breakout | 316.81 | 240.73 | 10.16 | 3.17 |
| Enduro | 1006.3 | 831.25 | 141.89 | 29.1 |
| River Raid | 7446.62 | 4102.81 | 2867.66 | 1453.02 |
| Seaquest | 2894.4 | 822.55 | 1003 | 275.81 |
| Space Invaders | 1088.94 | 826.33 | 373.22 | 301.99 |