

Reinforcement Learning

Function approximation and policy gradient methods

Davide Abati

December 11, 2017

University of Modena and Reggio Emilia

All this material is a free re-arrangement of **David Silver's UCL Course on RL**.
You are also encouraged to take a look to his **Youtube lectures**.

Introduction

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

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

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

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

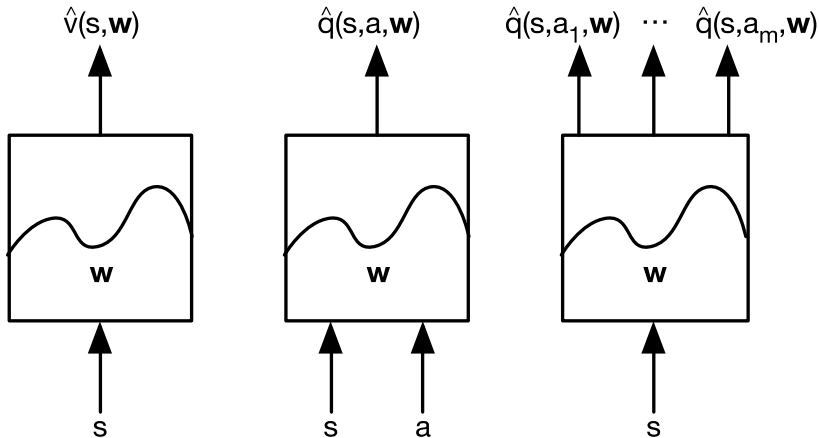
How can we scale up the model-free methods for prediction and control from the last lecture?

- 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 state spaces:
 - 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 state spaces:
 - Estimate value function with function approximation

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

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

- Generalise from seen states to unseen states
- Update parameter w using MC or TD learning



There are many function approximators, e.g.

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

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

Incremental methods

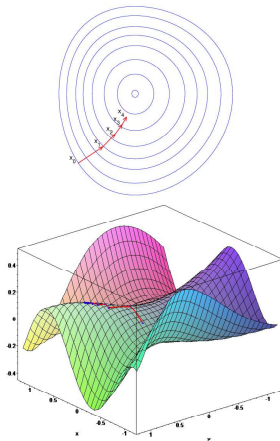
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector \mathbf{w}
- Define the gradient of $J(\mathbf{w})$ to be

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

- 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



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

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

- Gradient descent finds a local minimum

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

- 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

- 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

- Represent value function by a linear combination of features

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

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

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

- 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 = *stepsize* \times *predictionerror* \times *featurevalue*

- 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

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

- 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

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

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

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

- 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

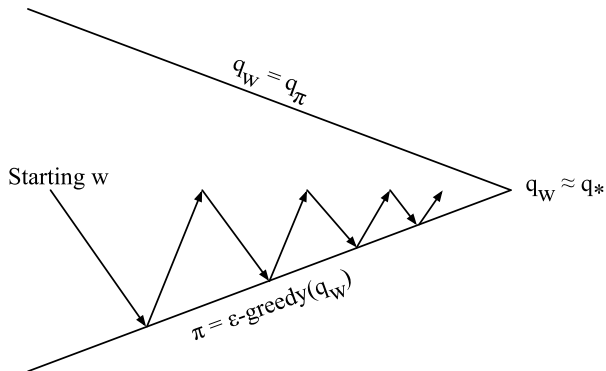
- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to “training data”:

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

- Linear TD(0) update is

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

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



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

Policy improvement ϵ -greedy policy improvement

- Approximate the action-value function

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

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

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

- Use stochastic gradient descent to find a local minimum

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

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

- 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 function by linear combination of features

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

- Stochastic gradient descent update

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

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

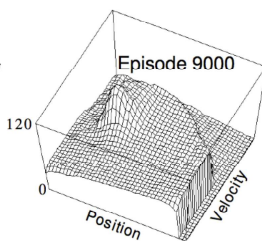
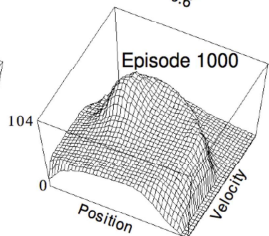
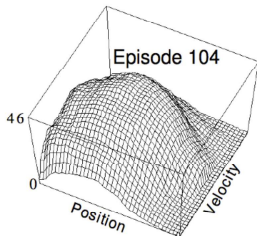
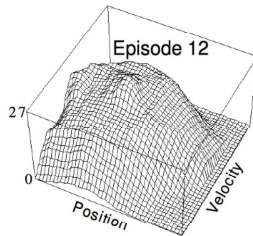
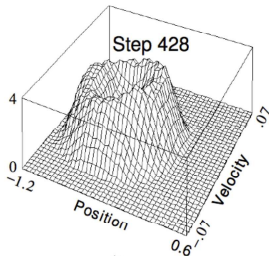
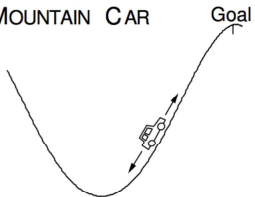
- Like prediction, we must substitute a target for $q_\pi(S, A)$
 - For MC, the target is the return G_t

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

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

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

MOUNTAIN CAR



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

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗

(✓) = chatters around near-optimal value function

Batch methods

- 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")

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_{\pi}(s)$
- And *experience* \mathcal{D} 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 function $\hat{v}(s, \mathbf{w})$?
- **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} ,

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

Given experience consisting of $\langle state, value \rangle$ 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

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

Given experience consisting of $\langle state, value \rangle$ 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

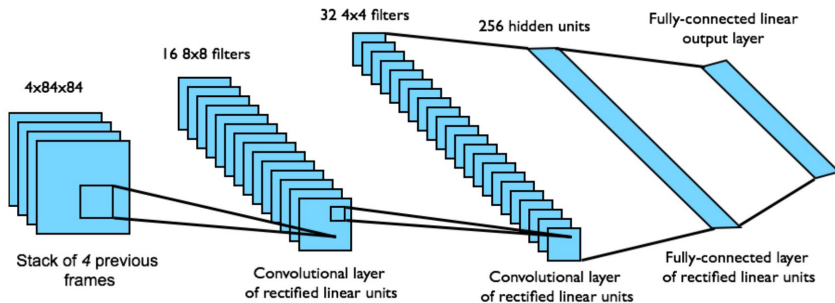
$$\mathbf{w}^\pi = \arg \min_{\mathbf{w}} LS(\mathbf{w})$$

- 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

- 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



DQN results in Atari

