CSCI S-89C Deep Reinforcement Learning

Harvard Summer School

Dmitry Kurochkin

Summer 2020
Lecture 9

Contents

Contents

Quiz 8

Question 1

4 / 4 pts

TD(λ) with $\lambda = 0$ is equivalent to

(A) 1-step TD

(B) Monte Carlo Control

(C) Monte Carlo Prediction

(D) None of (A), (B), (C)

Please select:

- B
- C
- Correct! A
- D

Quiz 8

Question 2

4 / 4 pts

SARSA(λ) with $\lambda = 0$ is equivalent to

(A) SARSA

(B) Monte Carlo Control

(C) Monte Carlo Prediction

(D) None of (A), (B), (C)

Please select:

Correct!

- A
- C
- B
- B

Quiz 8

Question 3

4 / 4 pts

Q-learning is equivalent to SARSA(λ) for some choice of parameter $\lambda \in [0, 1]$.

- ○ True
- ◉ False — Correct!

Quiz 8

Question 4

4 / 4 pts

The environment has three states: s_A, s_B, and s_C. In each state there are two actions, a_1 and a_2, available.

Suppose we want to obtain optimal policy using Q-learning; and the initial values of the action-value function are

$Q(s_A, a_1) = 6, Q(s_A, a_2) = 5,$

$Q(s_B, a_1) = 4, Q(s_B, a_2) = 3,$

$Q(s_C, a_1) = 2, Q(s_C, a_2) = 1.$

We generate the sequence using ε-greedy ($\varepsilon = 0.05$) policy with respect to current $Q(s, a)$ values and observe:

$s_A, a_1, R_1 = 5, s_B, a_2, \ldots$

If $\alpha = 0.1, \gamma = 0.9$, what is $Q(s_A, a_1)$ after its first update according to the Q-learning algorithm?

Hint: use the following off-policy Q-learning:
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right].$

Correct!

6.26

Correct Answers

6.26 (with margin: 0.01)

Quiz 8

Question 5

4 / 4 pts

Suppose that the environment has three states {stage k} with k=1, 2, 3 (k is a *feature* of the state):

s_A ={stage 1}, s_B ={stage 2}, s_C ={stage 3}.

In each state there are two actions, 0 and 1, available.

The action-value function is being approximated by the following linear function (linear in weights):

$$q_\pi(s,a) \approx \hat{q}(s,a,\mathbf{w}) \doteq w_1 + w_2\,a + (w_3 + w_4\,a)\,k\ ,$$

where $\mathbf{w} = (w_1, w_2, w_3, w_4)^T$ are *weights* and k *corresponds to the state* s ={stage k}.

For example, $q_\pi(s_C, 0) \approx \hat{q}(s_C, 0, \mathbf{w}) \doteq w_1 + w_2 \cdot 0 + (w_3 + w_4 \cdot 0) \cdot 3$.

Here, π is some current policy, usually ε-greedy with respect to the most recent $\hat{q}$.

Suppose we run Q-learning with approximation:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla\hat{q}(S_t, A_t, \mathbf{w}_t),$$

where the initial values (i.e. values at time $t=0$) for weights are set to zero, $\mathbf{w}_0 = (0,0,0,0)^T$, and observe:

$s_B, A_0 = 1, R_1 = 5, s_C, A_1 = 0, \ldots$

If $\alpha = 0.1$, $\gamma = 0.9$, what is $\mathbf{w}_1$ after the first update (i.e. weights at time $t=1$)?

Please select:

- (0.5, 0.5, 1.0, 0)
- Correct! (0.5, 0.5, 1.0, 1.0)
- (0.5, 0, 1.0, 0)
- (5, 0, 10, 0)

Contents

Mini-batch Gradient Descent

Given policy π, assume that for some weights $\mathbf{w} = (w_1, w_2, \ldots, w_d)^T$ (usually $d \ll |\mathcal{S}|$) we can approximate:

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

Mini-batch Gradient Descent

Given policy π, assume that for some weights $\mathbf{w} = (w_1, w_2, \ldots, w_d)^T$ (usually $d \ll |\mathcal{S}|$) we can approximate:

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

The Mini-batch gradient descent (SGD) method that minimizes the mean-squared error

$$J(\mathbf{w}) \doteq E_\pi \left[\left(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}) \right)^2 \right]$$

is then

$$\mathbf{w}_{k+1} \doteq \mathbf{w}_k - \frac{1}{2}\alpha \nabla \sum_{t=mk}^{m(k+1)-1} \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_k) \right]^2$$

$$= \mathbf{w}_k + \alpha \sum_{t=mk}^{m(k+1)-1} \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_k) \right] \nabla \hat{v}(S_t, \mathbf{w}_k),$$

where m is the size of *mini-batches*.

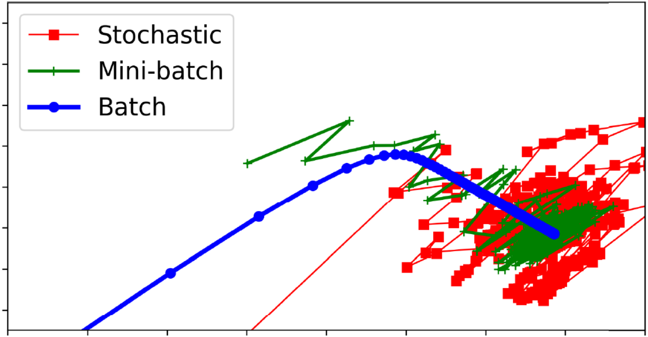
Mini-batch Gradient Descent

Since we do not know $v_\pi(S_t)$, we use an approximation U_t of the state value function (for example G_t in case of MC). The weights then can be obtained as follows:

$$\mathbf{w}_{k+1} \doteq \mathbf{w}_k + \alpha \sum_{t=mk}^{m(k+1)-1} \left[\underbrace{U_t}_{\approx v_\pi(S_t)} - \hat{v}(S_t, \mathbf{w}_k) \right] \nabla \hat{v}(S_t, \mathbf{w}_k)$$

Mini-batch Gradient Descent

Example: Path in (w_1, w_2) plane:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

Supervised Learning for Estimating $v_\pi(s)$

“training data”:

$$\langle S_0, v_\pi(S_0)\rangle, \langle S_1, v_\pi(S_1)\rangle, \langle S_2, v_\pi(S_2)\rangle \ldots$$

We can approximate $v_\pi(s)$ with

- G_t (MC)
- $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)$ (1-step TD)
- $G_{t:(t+n)}$ (n-step TD)
- etc.

-think of these as noisy “measurements” of $v_\pi(s)$.

Supervised Learning for Estimating $q_\pi(s)$

"training data":

$$\langle (S_0, A_0), q_\pi(S_0, A_0)\rangle, \langle (S_1, A_1), q_\pi(S_1, A_1)\rangle, \langle (S_2, A_2), q_\pi(S_2, A_2)\rangle, \ldots$$

We can approximate $q_\pi(s, a)$ with

- G_t (MC)
- $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$ (1-step SARSA)
- $G_{t:(t+n)}$ (n-step SARSA)
- etc.

-think of these as noisy "measurements" of $q_\pi(s, a)$.

Contents

Supervised Learning for Estimating $q_*(s)$ and $\pi_*(a|s)$

"training data":

$$\langle(S_0, A_0), q_*(S_0, A_0)\rangle, \langle(S_1, A_1), q_*(S_1, A_1)\rangle, \langle(S_2, A_2), q_*(S_2, A_2)\rangle, \ldots$$

We can approximate $q_*(s, a)$ with

- G_t (MC) and continuously adjust policy
- $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$ (1-step SARSA) and adjust policy
- $G_{t:(t+n)}$ (n-step SARSA) and adjust policy
- $R_{t+1} + \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$ (Q-learning) - no need to adjust policy!
- etc.

-think of these as noisy (and likely very biased in the beginning of the training!) "measurements" of $q_*(s)$.

Contents

Experience Replay

While updating weights $\mathbf{w}$ using "training data"

$$\langle S_0, v_\pi(S_0)\rangle, \langle S_1, v_\pi(S_1)\rangle, \langle S_2, v_\pi(S_2)\rangle \dots$$

or

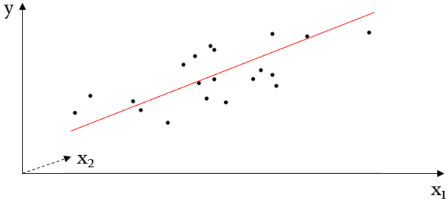
$$\langle (S_0, A_0), q_\pi(S_0, A_0)\rangle, \langle (S_1, A_1), q_\pi(S_1, A_1)\rangle, \langle (S_2, A_2), q_\pi(S_2, A_2)\rangle, \dots,$$

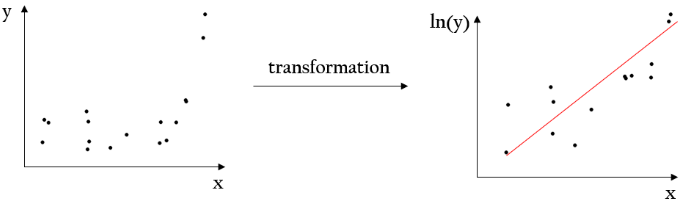
the "samples" do not need to be used in the order of time flow. Moreover, one can re-use the "samples"

- this is called *experience replay*.

Contents

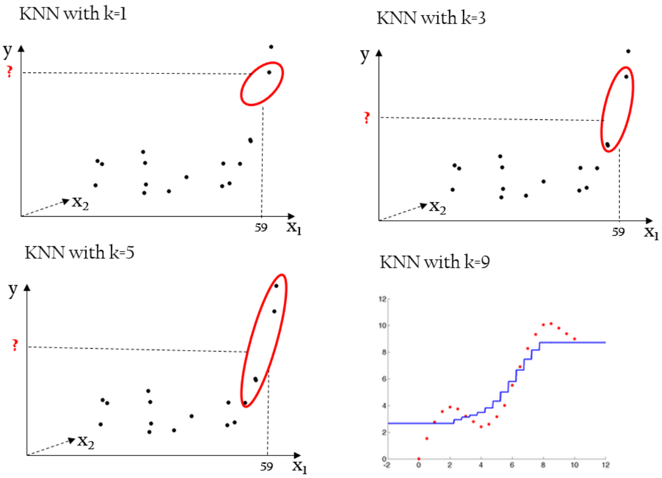
Linear Regression





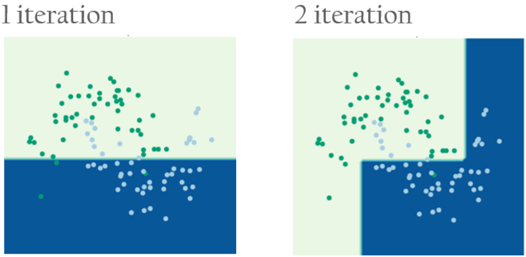
Contents

K Nearest Neighbors (KNN)

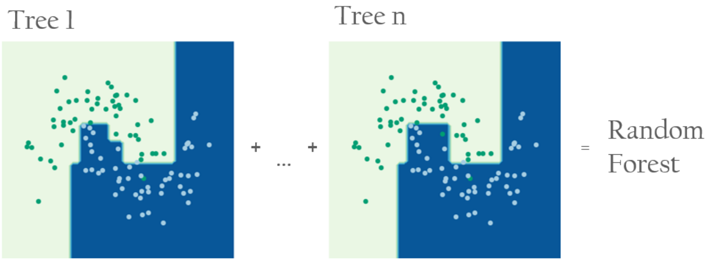


Contents

Random Tree / Forest

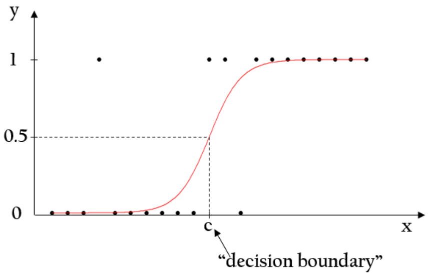


Random Tree / Forest



Contents

Logistic Regression



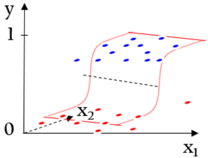
logistic function

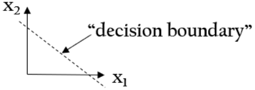
$$\sigma(t) = \frac{e^t}{e^t + 1}$$

Decision rule:
if $\boldsymbol{x} \le \boldsymbol{c}$ then y = 0 and 1 otherwise

(c is called “decision boundary”)

What if we have 2 inputs: x_1 and x_2?

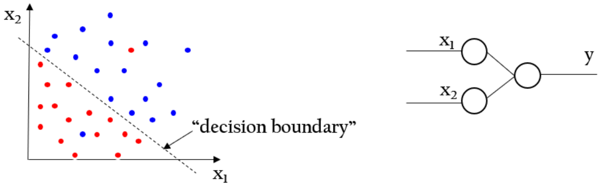


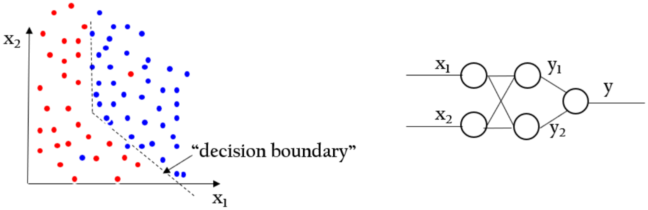


Decision rule:
if point (x_1, x_2) is “below” the decision boundary
then y = 0 and 1 otherwise

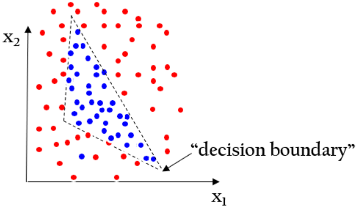
Contents

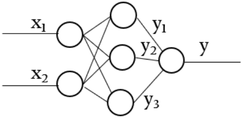
Artificial Neural Networks (NN)

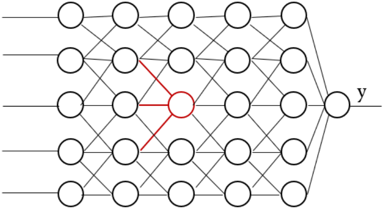




Artificial Neural Networks (NN)







Convolutional NN (CNN)