Artificial Neural Networks (Gerstner). Exercises for week 10

Applications of Reinforcement Learning

Exercise 1. Biological learning rules

In this exercise you will show that the softmax output for action selection in combination with a linear read-out function leads to a biologically plausible learning rule.

Consider a network with three output neurons corresponding to actions a_1, a_2 and a_3 with 1-hot coding. If $a_k = 1$, action a_k is taken.

The probability of taking action a_k is given by the softmax function

$$\pi(a_i|x) = \frac{\exp[\sum_k w_{ik} y_k]}{\sum_j \exp[\sum_k w_{jk} y_k]} \tag{1}$$

where $y_k = f(x - x_k)$.

a. Show that

$$\frac{d}{dw_{35}} ln[\pi(a_i|x)] = [a_3 - \pi(a_3|x)] y_5 \,. \tag{2}$$

Hint: simply insert the softmax and then take the derivative.

b. Interpret your result in terms of a 'presynaptic factor' and a 'postsynaptic factor'. Can the rule be implemented in biology?

Hint: Consider the two cases: action a_3 is (or is not) chosen at time t.

Exercise 2. Why target networks help

We look at semi-gradient Q-learning with linear function approximation, i.e. $Q(s^{(j)}, a) = \sum_i w_{ai} s_i^{(j)}$. We start with $w_{ai} = 0$ for all a and i.

Assume we observe state $s^{(1)} = (1, 1, 0)$, take action $a = 1$, receive reward $r = 1$ and observe the next state $s^{(2)} = (0, 1, 1)$.

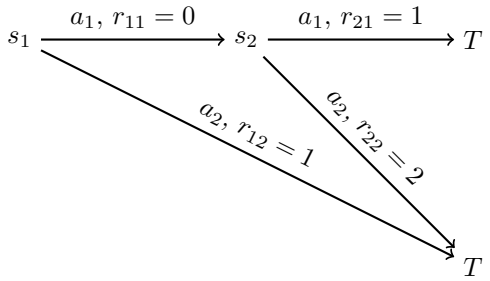
a. Compute $Q(s^{(1)}, 1)$ with the semi-gradient learning rule $\Delta w_{ai} = \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) s_i$ with $\eta = \gamma = 1$.

b. Show that $Q(s^{(2)}, 1)$ has also changed.

c. Assume $Q(s, a) = \sum_i w_{ai} s_i + \epsilon$, where ϵ is a Gaussian noise term with mean 0 and variance σ^2. Show that $\langle \max_a Q(s, a) \rangle > \max_a \langle Q(s, a) \rangle > \langle Q(s, \arg\max_a Q(s, a)) \rangle$.

In the following two exercises you will get a better understanding of the basic intuition gained here.

Exercise 3. Q–learning with function approximation



Consider the MDP shown above, with two states, two actions and deterministic rewards (where T represents the terminal state). We want to learn the Q–values associated with the states using Q–learning, with discount factor $\gamma = 1$.

a. (**Tabular Case**) The agent starts with all Q–values equal to 0. As in Dyna–Q, we assume that the agent can store observed transitions in memory. The agent observes all 4 possible transitions, then updates

the Q–values for s_2 by alternating between observations of (s_2, a_1, r_{21}) and (s_2, a_2, r_{22}) until learning converges. The agent then similarly alternates between observations of (s_1, a_1, r_{11}) and (s_1, a_2, r_{12}) until learning converges.

(i) What are the Q–values after convergence in s_2, and finally after convergence in s_1?

(ii) Do the Q–values after each stage result in the optimal policy?

b. (**Function Approximation**) Now assume that the states are given to us with the vector–based observations shown below. We will learn the Q–values using the linear network shown on the right.

As before, assume a Dyna–Q–style learning where the agent learns the weights after observing all transitions. Start with $w_{11} = w_{12} = w_{13} = w_{21} = w_{22} = w_{23} = 0$.

(i) What will the converged weights be after alternating between the two possible s_2 observations? Hint: Note that certain weights will always be updated in exactly the same way, and should therefore converge to the same value.

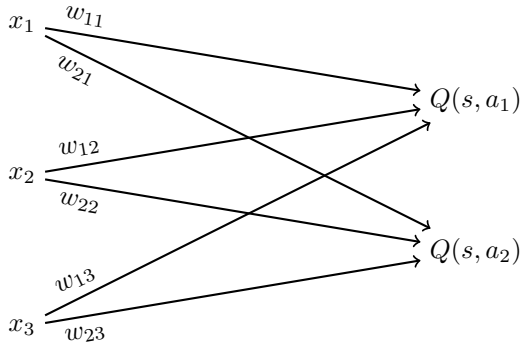
(ii) After s_2 convergence, what is the policy in s_1? How does this differ from the tabular case after s_2 convergence, and why?

(iii) What weights would result in the correct Q–value predictions for all (s, a) pairs? Are they unique?

(iv) How can an arbitrary tabular Q–learning problem be represented using a simple linear neural network like the one shown on the right? Hint: consider how the input space could be represented such that semi–gradient descent results in each weight converging exactly to $Q(s, a)$ for some (s, a) pair.

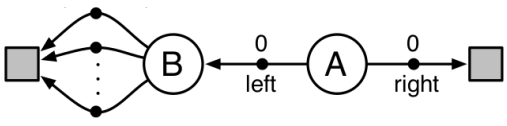
$$s_1 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$s_2 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$



Exercise 4. The maximization bias and Double–Q learning

Consider the MDP given below, with two non–terminal states A and B and terminal states represented by grey boxes.



From State A, the action "right" transitions directly to a terminal state with reward 0, while the action "left" transitions to State B with reward 0. From State B, the agent can take any one of M actions, each of which has a reward distributed on every step according to $\mathcal{N}(-0.1, 0.5)$ (i.e. a Normal distribution with mean -0.1 and variance 0.5). We assume a discount factor $\gamma = 1$.

a. What is the true (expected) value of State B? As a result, what are the optimal Q–values $Q(A, \text{left})$ and $Q(A, \text{right})$, and therefore the optimal policy from State A?

b. Assume $M = 20$. Write a short function on your computer (5-8 lines) to simulate Q–learning starting from State B and taking a random policy. Start with all Q–values equal to 0 and $\alpha = 0.05$, and run it several times for 5000 trials. Is the resulting value of State B according to $V(B) = \max_{a_i} Q(B, a_i)$ usually positive or negative? Why? What effect will this have on the learned policy for State A?

c. This effect is referred to as the Maximization Bias, and can be addressed using the Double Q–learning algorithm below.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q.(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \text{argmax}_a\, Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \text{argmax}_a\, Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Re–write your function above to learn two different sets of Q–values and update one of them at random with each new observation. Check the value of State B according to both $V_1(B) = Q_1(B, \text{argmax}_{a_i}(Q_2(B, a_i))$ and $V_2(B) = Q_2(B, \text{argmax}_{a_i}(Q_1(B, a_i))$. Do these estimates more accurately reflect the true value of State B?

Exercise 5. From Policy Gradient to eligibility traces

In this exercise you will show that eligibility traces appear naturally in any policy gradient algorithm. Eligibility traces are nice because they lead to a transparent and easy–to–interpret algorithm. Moreover, eligibility traces enable a direct online implementation of the algorithm in distributed hardware (or biology).

Consider a discrete multistep reinforcement learning problem with the usual graph, the usual notations and transitions: an action a_t leads you (stochastically) from state s_t to s_{t+1} and on this transition you collect the reward r_t. Suppose that you always start in state $s_{t=0} = s_{start}$. We assume that there is a simple terminal state s_{target}. When you reach this state you get a particularly strong positive reward.

Your policy $\pi(a_t|s_t, \theta)$ depends on parameters θ. For the moment your aim is to optimize the parameters of the policy such that you maximize the expected discounted reward $E[Return(s_{start} \rightarrow S_{target})] = \langle r_0 + \gamma r_1 + \gamma^2 r_2 + ...\rangle$.

We proceed in five steps.

a. Derive a batch version of the policy gradient algorithm over multiple time steps by optimizing $E[Return(s_{start} \rightarrow S_{target}] = \langle r_0 + \gamma r_1 + \gamma^2 r_2 + ...\rangle$ through gradient descent.

   Hint: Use the log-likelihood trick seen in class. Start as for blackboard 3 (slide 35 of Lecture 10) and take the derivative with respect to parameter θ_j.

b. A batch algorithm means averaging over many episodes. Transform the batch algorithm into an online algorithm where you consider one episode at a time. Assume that in one episode you traverse the state-action sequence: $s_0, a_0, r_0; s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; s_4, a_4, r_4; s_5 = s_{target}$.

   Show that the parameter updates can be written as

$$\begin{aligned}
\Delta\theta_j = \quad & [r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_0|s_0, \theta)] \\
+ \quad & [\gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_1|s_1, \theta)] \\
+ \quad & [\gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_2|s_2, \theta)] \\
+ \quad & [\gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_3|s_3, \theta)] \\
+ \quad & \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_4|s_4, \theta)] \qquad (3)
\end{aligned}$$

   Hint: redo the calculation (blackboard 3) on page 35 and compare your result with the result on page 36 (Lecture 10).

c. So far we were only interested in maximizing the discounted future reward from the INITIAL state, with the discount factor computed relative to that state ($t = 0$). However, while you move along the trajectory you pass by other states s_1, s_2, s_3, s_4. For each of these states s_t, you should now also optimize the future expected discounted reward starting from s_t; that is you want to maximize $E[Return(s_t \rightarrow S_{target})] = \langle r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \rangle$.

More generally, you should optimize the future discounted returns from every step t, assuming that the discounting started at the current step or at any possible step m in the past (i.e. $m \leq t$). Assume that m runs from $-\infty$ to t.

Redo the calculation in (b), but calculate the parameter update resulting from returns starting in arbitrary states with arbitrary initial discount factors.

Hint: Copy, but time-shift the results from (b).

d. Sum all the updates from (b) and (c) and reorder all terms from (b) and (c) such that updates that are multiplied with the same reward are grouped together.

Show that this results in updates of the form

$$\Delta\theta_j = c\, r_n \Big\{ \frac{d}{d\theta_j} ln[\pi(a_n|s_n,\theta)] + \gamma \frac{d}{d\theta_j} ln[\pi(a_{n-1}|s_{n-1},\theta)] + \gamma^2 \frac{d}{d\theta_j} ln[\pi(a_{n-2}|s_{n-2},\theta)] + ... \tag{4}$$

with some constant c. What is this constant?

e. Now we introduce eligibility traces by defining for each parameter θ_j a 'shadow variable' z_j which, in each time step t, decreases by a factor $\lambda < 1$

$$z_j \longleftarrow \lambda z_j \tag{5}$$

and then (in the same time step) increase by an amount

$$z_j \longleftarrow \frac{d}{d\theta_j} ln[\pi(a_t|s_t,\theta)] \tag{6}$$

where a_t is the action taken in time step t.

What is the relation of λ and γ? What is the final weight update?

f. Suppose that all rewards are zero, except the reward in the final time step $r_4 > 0$. Furthermore suppose that parameter θ is only sensitive to a_2, s_2. To be specific, say $\frac{d}{d\theta_j} ln[\pi(a_2|s_2,\theta)] > 0$ and $\frac{d}{d\theta_j} ln[\pi(a_t|s_t,\theta)] = 0$ for $t \neq 2$.

How can you interpret the resulting algorithm? How much will the parameter θ_j change?