Artificial Neural Networks (Gerstner). Solutions for week 11

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.

Solution:

a.
$$\ln \pi(a_i|x) = \sum_k w_{ik} y_k - \ln \sum_j \exp[\sum_k w_{jk} y_k]. \tag{3}$$

$$\frac{d}{dw_{35}} ln[\pi(a_i|x)] = \delta_{i3} y_5 - \frac{\exp[\sum_k w_{ik} y_k]}{\sum_j \exp[\sum_k w_{jk} y_k]} y_5 = [a_3 - \pi(a_3|x)]y_5. \tag{4}$$

b. Note that w_{35} connects the presynaptic neuron with index 5 in the layer of y-neurons with the action neuron a_3. Hence, the presynaptic facor is y_5. The postsynaptic is $[a_3 - \pi(a_3|x)]$ Hence, if presynaptic and postsynaptic neuron are both active, the eligibility trace is increased by an amount $[1 - \pi(a_3|x)]y_5$. Second, if another action is taken, we have $a_3 = 0$. Hence, the eligibity trace decays by an amount which is proportional to y_5 and $\pi(a_3|x)$. Note that $\pi(a_3|x)$ can be interpreted as the 'drive' or 'membrane potential' of neuron a_3. Yes, the rule would be implementable in biology.

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.

Solution:

a. $\Delta w_{11} = 1 \cdot (1 + 1 \max_{a'} 0 - 0) \cdot 1 = 1$, similarly $\Delta w_{12} = 1$, $\Delta w_{13} = 1 \cdot (1 + 1 \max_{a'} 0 - 0) \cdot 0 = 0$. With these updates we get $Q^{(1)}, 1) = \sum_i w_{1i} s_i^{(1)} = 2$

b. $Q(s^{(2)}, 1)$ was 0 before the update and is now $Q(s^{(2)}, 1) = \sum_i w_{1i} s_i^{(2)} = 1$.

c. Lets call the maximal expected Q-value $Q(s, a^*) = \max_a \langle Q(s, a) \rangle$. If the noise terms where always such that $\arg\max_a Q(s, a) = a^*$, $\langle \max_a Q(s, a) \rangle$ would be equal to $Q(s, a^*) = \max_a \langle Q(s, a) \rangle$, but for all cases where $\arg\max Q(s, a) = \hat{a} \neq a^*$ we have $Q(s, \hat{a}) > Q(s, a^*)$ and thus $\langle \max_a Q(s, a) \rangle > \max_a \langle Q(s, a) \rangle$ and $\langle Q(s, \hat{a}) \rangle < Q(s, a^*)$ and thus $\max_a \langle Q(s, a) \rangle > \langle Q(s, \arg\max_a Q(s, a)) \rangle$.

Exercise 3. Q–learning with function approximation

$$\begin{tikzcd}
s_1 \arrow[r, "{a_1,\, r_{11} = 0}"] \arrow[drr, "{a_2,\, r_{12} = 1}"] & s_2 \arrow[r, "{a_1,\, r_{21} = 1}"] \arrow[dr, "{a_2,\, r_{22} = 2}"] & T \\
 & & T
\end{tikzcd}$$

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.

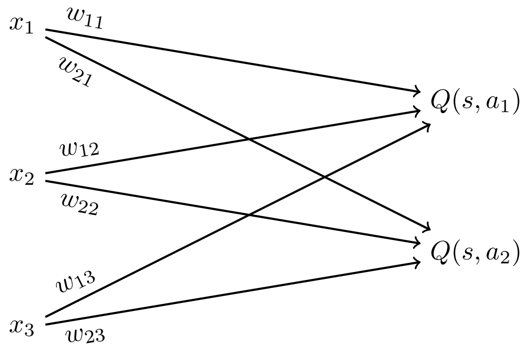
   (ii) After s_2 convergence, what is the policy in s_1? How does this differ from the tabular case after s_2 convergence, and why?

   (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.

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

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



Solution:

a. (i)

$$\begin{bmatrix} Q(s_1,a_1) & Q(s_1,a_2) \\ Q(s_2,a_1) & Q(s_2,a_2) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{(Start)}$$

$$\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} \quad \text{(After } s_2 \text{ convergence)}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad \text{(After } s_1 \text{ convergence)}$$

(ii) With a discount factor of 1, the optimal policy is to take a_1 in s_1 and a_2 in s_2, achieving a reward of 2. After s_2 convergence, the Q–values in s_1 are equal, so they do not represent an optimal policy. The final Q–values do represent the optimal policy.

b. (i) Noting that the updates should result in $w_{12} = w_{13}$ and $w_{22} = w_{23}$, we can expect the weights to converge to

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{(Start)}$$

$$\begin{bmatrix} 0 & 0.5 & 0.5 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{(After } s_2 \text{ convergence)}$$

(ii) Unlike the tabular case, the Q–values for s_1 are now $Q(s_1, a_1) = 0.5$ and $Q(s_1, a_2) = 1$, resulting in a policy that does not give equivalent weight to the two actions (and in this case is suboptimal). This occurs because $x_2 = 1$ for both states, so their updates are correlated.

(iii) One possible solution is to fix $w_{12} = w_{22} = 0$. In this case, the value of x_2 is irrelevant and the state representations become effectively orthogonal. The optimal weights are given by

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

which we can see as equivalent to the tabular solution given above.
In general, we can notice that the weights for $Q(s, a_2)$ can be solved without using the values of any other states, and amount to solving a system of 2 equations with 3 unknowns. Since this solution will not be unique, the optimal weights are not unique either.

(iv) The semi–gradient descent rule for the network above gives the weight update

$$\Delta w_{ji} = \alpha\Big(r_{t+1} + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_t)\Big)\frac{\partial Q(s_t, a_t)}{\partial w_{ji}}$$

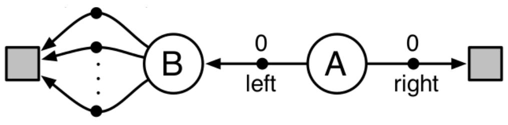
applied only for the observed action $a_t = j$. Note that this is equivalent to tabular Q–learning with $w_{ji} = Q(i, j)$ if

$$\frac{\partial Q(s_t, a_t)}{\partial w_{ji}} = \begin{cases} 1, & \text{if } s_t = i \text{ and } a_t = j \\ 0, & \text{otherwise.} \end{cases}$$

Since $\frac{\partial Q(s_t,a_t)}{\partial w_{ji}} = x_i$, this can be enforced by using a one–hot input representation with N inputs for N states.

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 $Q_\cdot(terminal, \cdot) = 0$

Loop for each episode:
   Initialize $S$
   Loop for each step of episode:
      Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
      Take action $A$, observe $R$, $S'$
      With 0.5 probabilility:
         $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \mathrm{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
      else:
         $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \mathrm{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
      $S \leftarrow S'$
   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 $V_1(B) = Q_1(B, \mathrm{argmax}_{a_i}(Q_2(B, a_i))$ and $V_2(B) = Q_2(B, \mathrm{argmax}_{a_i}(Q_1(B, a_i))$. Do these estimates more accurately reflect the true value of State B?

Solution:

a. Since all actions from State B give the same reward distribution, the policy is irrelevant to the value of the state. The true expected value is therefore simply the mean, -0.1. As a result, $Q(A, \text{left}) = -0.1$. Since $Q(A, \text{right}) = 0$, the opimal policy is to take the right action.

b. A possible function (in Python) is

```python
import numpy as np
def simulate(num_steps):
    num_actions = 20
    qs = np.zeros(num_actions)
    for i in range(num_steps):
        a = np.random.choice(num_actions)
        qs[a] += 0.05*(np.random.normal(-0.1,0.5) - qs[a])
    return qs
print(np.max(simulate(5000)))
```

The result is usually positive, despite the negative bias of the Normal distribution. As a result, $Q(A, \text{left})$ will also usually be positive and the agent will favour the "left" action, although we know from above that "right" is optimal.

To understand why, we note that the mean $\frac{1}{M}\sum_{a_i} Q(B, a_i)$ is usually close to -0.1, where each $Q(B, a_i)$ is computed from a subsample of $\mathcal{N}(-0.1, 0.5)$. However, we take $\max_{a_i} Q(B, a_i)$ to determine the state value. The maximum subsample mean will usually be higher than the true mean, and it is probable that at least one subsample will result in a positive Q–value. This Maximization Bias becomes problematic when the task stochasticity (i.e. Q–value variance) dominates over the difference in expected values of different actions (which in this case is 0).

c. The new function using Double Q–learning should look like

```
import numpy as np
def simulate2(num_steps):
    num_actions = 20
    qs1 = np.zeros(num_actions)
    qs2 = np.zeros(num_actions)
    for i in range(num_steps):
        a = np.random.choice(num_actions)
        if np.random.random() < 0.5:
            qs1[a] += 0.05*(np.random.normal(-0.1,0.5) - qs1[a])
        else:
            qs2[a] += 0.05*(np.random.normal(-0.1,0.5) - qs2[a])
    return qs1, qs2
qs1, qs2 = simulate2(5000)
print(qs1[np.argmax(qs2)], qs2[np.argmax(qs1)])
```

and the resulting values should be close to -0.1.

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 $E[Return(s_{start} \to S_{target})] = \langle r_0 + \gamma r_1 + \gamma^2 r_2 + ...\rangle$.

We proceed in five steps.

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

$$\begin{aligned}
\Delta\theta_j = \quad & [r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_0|s_0,\theta)] \\
+ \quad & [\gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_1|s_1,\theta)] \\
+ \quad & [\gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_2|s_2,\theta)] \\
+ \quad & [\gamma^3 r_3 + \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_3|s_3,\theta)] \\
+ \quad & \gamma^4 r_4] \frac{d}{d\theta_j} ln[\pi(a_4|s_4,\theta)] \qquad (5)
\end{aligned}$$

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 $E[Return(s_t \to S_{target})] = \langle r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...\rangle$.

   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

$$\Delta\theta_j = c\, r_n \left\{ \frac{d}{d\theta_j} ln[\pi(a_n|s_n,\theta)] + \gamma \frac{d}{d\theta_j} ln[\pi(a_{n-1}|s_{n-1},\theta)] + \gamma^2 \frac{d}{d\theta_j} ln[\pi(a_{n-2}|s_{n-2},\theta)] + ... \right. \qquad (6)$$

   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 $\lambda < 1$

$$z_j \longleftarrow \lambda z_j \qquad (7)$$

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

$$z_j \longleftarrow \frac{d}{d\theta_j} ln[\pi(a_t|s_t,\theta)] \qquad (8)$$

   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?

Solution:

a. We will take $G_{s_0,a_0} = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ as a Monte Carlo sample of the total discounted future returns from taking action a_0 in state s_0. Our goal is to maximize $E_\pi[< G_{s_0,a_0} >]$, where $< G_{s_0,a_0} >$ is the expected discounted future returns starting from (s_0, a_0).

   We will start by only optimizing over our policy in the first state, $\pi(a_0|s_0,\theta)$. We then have

$$E_{\pi(a_0|s_0)}[< G_{s_0,a_0} >] = \int_{a_0} < G_{s_0,a_0} > \pi(a_0|s_0) da_0.$$

Taking the derivative and moving it inside the integral gives us

$$\frac{\partial E_{\pi(a_0|s_0)}[< G_{s_0,a_0} >]}{\partial \theta_j} = \int_{a_0} < G_{s_0,a_0} > \frac{\partial}{\partial \theta_j} \pi(a_0|s_0,\theta) da_0$$

The log–likelihood trick tells us that

$$\frac{d}{dx} p(x) = \frac{p(x)}{p(x)} \frac{d}{dx} p(x) = p(x) \frac{d}{dx} \ln p(x).$$

Applying this above gives

$$\frac{\partial E_{\pi(a_0|s_0)}[G_{s_0,a_0}]}{\partial \theta_j} = \int_{a_0} < G_{s_0,a_0} > \pi(a_0|s_0,\theta) \frac{\partial}{\partial \theta_j} \ln \pi(a_0|s_0,\theta) da_0$$

Unfortunately, we don't have direct access to the expected discounted returns $< G_{s_0,a_0} >$ under every action a_0. However, we can approximate it from the batch returns under each action, just as we can approximate $\pi(a_0|s_0,\theta)$ from the proportion of times the agent took action a_0 in state s_0 under the policy. Assume that, over M episodes, a particular state–action pair (s_0, a_0) was experienced N_{s_0,a_0} times. Then,

$$< G_{s_0,a_0} > \pi(a_0|s_0,\theta) \approx \frac{\sum_{k=1}^{N_{s_0,a_0}} G^k_{s_0,a_0}}{N_{s_0,a_0}} \frac{N_{s_0,a_0}}{M} = \frac{\sum_{k=1}^{N_{s_0,a_0}} G^k_{s_0,a_0}}{M}$$

where $G^k_{s_0,a_0}$ represent the returns on a particular episode k. Replacing the integral with a sum over all the batch episodes, we can then approximate the gardient as

$$\Delta\theta^0_j = \frac{1}{M} \sum_{i=1}^{M} G^i_{s_0,a_0} \frac{\partial}{\partial \theta_j} \ln \pi(a_0|s_0,\theta)$$

where $\Delta\theta^0_j$ represents the contribution to the gradient from $\pi(a_0|s_0,\theta)$.

Finally, the full return $< G_{s_0,a_0} >$ depends on all the other actions taken in the episode as well. However, an action a_t only affects the components of $G^i_{s_0,a_0}$ that came after time step t, which is equal to $\gamma^t G^i_{s_t,a_t}$. Therefore,

$$\Delta\theta^t_j = \frac{1}{M} \sum_{i=1}^{M} \gamma^t G^i_{s_t,a_t} \frac{\partial}{\partial \theta_j} \ln \pi(a_t|s_t,\theta)$$

Adding them together gives

$$\Delta\theta_j = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{T_i} \gamma^t G^i_{s_t,a_t} \frac{\partial}{\partial \theta_j} \ln \pi(a_t|s_t,\theta)$$

b. Transforming the batch algorithm into an online algorithm can be done by simply removing the averaging over M, i.e.

$$\Delta\theta_j = \sum_{t=0}^{T} \gamma^t G_{s_t,a_t} \frac{\partial}{\partial \theta_j} \ln \pi(a_t|s_t,\theta)$$

For the given episode, we have

$$\begin{aligned}\Delta\theta_j &= \gamma^0 G_{s_0,a_0} \frac{\partial}{\partial \theta_j} \ln \pi(a_0|s_0,\theta) + \gamma^1 G_{s_1,a_1} \frac{\partial}{\partial \theta_j} \ln \pi(a_1|s_1,\theta) \\ &+ \gamma^2 G_{s_2,a_2} \frac{\partial}{\partial \theta_j} \ln \pi(a_2|s_2,\theta) + \gamma^3 G_{s_3,a_3} \frac{\partial}{\partial \theta_j} \ln \pi(a_3|s_3,\theta) + \gamma^4 G_{s_4,a_4} \frac{\partial}{\partial \theta_j} \ln \pi(a_4|s_4,\theta).\end{aligned}$$

Evaluating $G_{s_t,a_t} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots$ gives the result above.

c. Optimizing for the returns starting from an arbitrary step m on the trajectory gives us

$$\Delta\theta_j = \sum_{t=0}^{T} \sum_{m=-\infty}^{t} \gamma^{t-m} G_{s_t,a_t} \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta)$$

where we recover the original gradient when m runs from 0 to 0. Performing a change of variables with $b = t - m$ yields

$$\Delta\theta_j = \sum_{t=0}^{T} \sum_{b=0}^{\infty} \gamma^{b} G_{s_t,a_t} \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta)$$

and recognizing the infinite geometric series gives us

$$\Delta\theta_j = \frac{1}{1-\gamma} \sum_{t=0}^{T} G_{s_t,a_t} \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta).$$

d. Substituting for the returns in the above yields

$$\Delta\theta_j = \frac{1}{1-\gamma} \sum_{t=0}^{T} \left[\sum_{i=0}^{T-t} \gamma^{i} r_{t+i} \right] \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta)$$

Performing another change of variables with $n = t + k$ and eliminating t gives

$$\begin{aligned}\Delta\theta_j &= \frac{1}{1-\gamma} \sum_{n=0}^{T} \sum_{i=0}^{n} \gamma^{k} r_n \frac{\partial}{\partial\theta_j} \ln \pi(a_{n-i}|s_{n-i},\theta) \\ &= c \sum_{n=0}^{T} r_n \sum_{i=0}^{n} \gamma^{i} \frac{\partial}{\partial\theta_j} \ln \pi(a_{n-i}|s_{n-i},\theta) \qquad (9)\end{aligned}$$

which is equivalent to the expression above, with $c = \frac{1}{1-\gamma}$.

e.

$$\begin{aligned} z_j^t &= \lambda z_j^{t-1} + \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta) \\ &= \lambda(\lambda z_j^{t-2} + \frac{\partial}{\partial\theta_j} \ln \pi(a_{t-1}|s_{t-1},\theta)) + \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta) \\ &= \lambda^2 z_j^{t-2} + \lambda \frac{\partial}{\partial\theta_j} \ln \pi(a_{t-1}|s_{t-1},\theta) + \frac{\partial}{\partial\theta_j} \ln \pi(a_t|s_t,\theta) \\ &= \sum_{i=0}^{t} \lambda^{i} \frac{\partial}{\partial\theta_j} \ln \pi(a_{t-i}|s_{t-i},\theta), \end{aligned}$$

where in the last line we have assumed that $z_j^0 = 0$ (i.e. the shadow variables were all initialized to 0). With $\gamma = \lambda$, we note that this is equivalent to the last sum in Equation 9. In this case, we can express the policy gradient update using our shadow variables as

$$\Delta\theta_j = c \sum_{n=0}^{T} r_n z_j^n \qquad (10)$$

f. In this case, Equation 10 simplifies to

$$\begin{aligned}
\Delta\theta_j &= cr_4 z_j^4 \\
&= cr_4 \sum_{i=0}^{4} \lambda^i \frac{\partial}{\partial\theta_j} \ln\pi(a_{4-i}|s_{4-i},\theta) \\
&= c\lambda^2 r_4 \frac{\partial}{\partial\theta_j} \ln\pi(a_2|s_2,\theta)
\end{aligned}$$

Since $\ln\pi(a_2|s_2,\theta) > 0$, an increase in the value of the parameter θ_j will increase the probability of taking a_2 in s_2 again. In addition, since $r_4 > 0$, all terms are positive and the value of θ_4 will increase.

The magnitude of increase depends on the magnitude of r_4. In other words, θ_j will increase more if it contributed to a larger reward, due to its effect on the policy 2 steps before receiving the reward.

The magnitude of increase also depends on λ^2. If the discount factor λ is small, it suggests that earlier actions contribute little to later rewards; as a result, the gradient will also be small since it relates to the policy several steps before actually receiving the reward.