Reinforcement Learning - Homework 2

Andrii Skliar, 11636785
Gabriele Bani, 11636758

deadline: December 3, 2018

Problem 1 (Instructions)**:**
(empty for correct numeration)
Problem 2 (Gradient Descent Methods)**:**

1. Why is the Monte Carlo target, G_t , an unbiased estimate of $v_\pi(S_t)$?

Solution: The fact that G_t is an unbiased estimate of $v_\pi(S_t)$ follows directly from the definition of $v_\pi(S_t) = \mathbb{E}_\pi[G_t|S_t = s]$. So, if we take an expectation of the Monte-Carlo target, we get the definition of $v_\pi(S_t)$. Therefore, Monte Carlo target, G_t, is an unbiased estimate of $v_\pi(S_t)$.

2. Why does using the Dynamic Programming target

$$\sum_{a,s',r} \pi(a|S_t)p(s',r|S_t,a)[r + \gamma\hat{v}(s',\mathbf{w}_t)] \tag{1}$$

result in a semi-gradient method?

Solution: The main reason is that r will depend on $\hat{v}(s', \mathbf{w}_t), s' \in S$, where S is a set of possible states. This implies, that bootstrapping target depends on vector $\mathbf{w}_t$, which is used for approximating state value and might not correspond to the true state value.

Therefore, if we update our $\mathbf{w}_t$, the target value will also change, which also means that bootstrapping values will be biased unless we reach true value function, which we don't know and therefore can't check if our function has actually converged to the true value function.

As we are updating our function in the direction of the sum of received reward and estimated target and not real target thus including only part of the gradient, this method is called semi-gradient and not full gradient.

3. Despite not being unbiased, semi-gradient methods that use bootstrapping have certain advantages w.r.t. Monte Carlo approaches. Why, for example, would you prefer bootstrapping in the Mountain Car problem?

Solution: The main benefit of semi-gradient methods that use bootstrapping over Monte Carlo approaches is that we don't need to wait for the end of the episode. Semi-gradient methods that use bootstrapping allow us to update our gradient towards more optimal solution at each timestep without waiting for the episode to end. This might result in faster convergence especially in the cases when we might not be able to reach the end of episode at all when using current policy.

If we look at the Mountain Car problem, initially we have bad policy which will make the car oscillate back and forth at the bottom of the hill without actually being able to solve the problem. This is due to that moving forward seems to yield better rewards because we can reach the top of the hill faster, but in reality it is not possible as we don't have enough force to reach the top. Therefore, with Monte Carlo approach, it might take us extensively long to be able to reach top of the hill thus we might not have even a single episode to learn from in a reasonable time. However, with bootstrapping methods and good initialization, we can see that our model will slowly converge to the optimal policy due to that semi-gradient update happens at each timestep.

Problem 3 (Basis functions)**:**

1. Tabular methods can be seen as a special case of linear function approximation. Show that this is the case and give the corresponding feature vectors.

Solution: We can choose to represent states as one-hot vectors. This means that for an environment with N states, we represent each state with an N dimensional feature vector, which is made of all zeros but a one in the position corresponding of the (index of the) represented state. For example, if we have three states, the representation of the second state will be [0, 1, 0]. We will have then that the general formula for linear function approximators can be written as, for state s_k

$$\hat{v}(s_k, \mathbf{w}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{N} w_i x_i(s) = w_k$$

2. You want to design the feature vectors for a state space with $s = [x, y]$. You expect that x and y interact in some unknown way. How would you design a polynomial feature vector for s?

Solution: First, we assume that the interactions between the two features will be up to order n. These interactions can be fully represented by using a polynomial feature vector in the following way (notice that we use the letter z for features as x is already used for the first dimension of the state):

$$z_{ij}(s) = x^i y^j, i = 0..n, j = 0..n$$

We have defined here for simplicity of notation a matrix $\mathbf{Z}$ of features, which can be converted to the standard row vector representation by using the vectorizization operation $\mathbf{z} = \text{vec}(\mathbf{Z})$, which stacks vertically all the columns of $\mathbf{Z}$.

3. What happens to the size of the polynomial feature vector if the number of variables in your state space increases?

Solution: The size of the polynomial feature vector for a k dimensional state space representation, using polynomials up to order n, is $(n+1)^k$. Thus, we can see that if the number of variables k in the state space representation increases, then the number of features needed to represent it using polynomials of up to order n (which is fixed) increases exponentially.

4. You are working on a problem with a state space consisting of two dimensions. You expect that one of them will have a larger impact on the performance than the other. How might you encode this prior knowledge in your basis function?

Solution: Encoding prior knowledge on a problem can be made in different ways, depending on the type of basis functions we are working with. For example, if we are using polynomials, we can use higher order terms for the dimension with more impact on performance. When using RBF instead, we can use different variances for the two dimensions, so that we attribute more importance to one of the two. In tile coding we can instead use different types of tiles than just vertical and horizontal, and favor tiles that can capture salient changes in the larger impact dimension. In general, depending on the type of features, we can either choose to use more features to represent the most relevant dimension, such as in the case of polynomials, or change the importance of the relevant dimension in the features themselves, such as in the case of RBF.

5. You can view coarse coding as a special case of Radial Basis Functions. Why?

Solution: We can see coarse coding as a simplified version of RBFs. First, notice that in both cases we have a circular area of effect around the coordinate of the center of each circle/RBF. In case of RBF, the effect of each RBF is determined as a continuous function of the distance to the current state. Instead, with coarse coding we have a discrete function which takes only the values one or zero. In particular, we can obtain a circle with center c and width d from a RBF of center c and variance σ^2, by using the function

$$f(s) = \begin{cases} 1 & \text{if } \exp\left(\frac{||s-c||^2}{2\sigma^2}\right) \geq d \\ 0 & \text{if } \exp\left(\frac{||s-c||^2}{2\sigma^2}\right) < d \end{cases}$$

Problem 4 (Geometry of linear value-function approximation)**:** Consider the MDP given. It consists of two states with one action, that transition into one another with reward 0. The features for both states are $\phi = 1$ for state s_0 and $\phi = 2$ for state s_1. We will now predict the value of the states using $v_w = w \cdot \phi$.

1. We can write the Bellman error vector as

$$\bar{\delta}_w = B^\pi v_w - v_w, \tag{2}$$

where B^π is the Bellman operator. What is the Bellman error vector after initialization with $w = 1$ and using $\gamma = 1$?

Solution: First, we need to write down the definition of the Bellman operator, which is $(B^\pi v_w)(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$. In our MDP, due to the deterministic policies and transitions, it reduces to $(B^\pi v_w)(s) = r + \gamma v(s')$. Now, using this definition and the give way of state value prediction $v_w = w \cdot \phi$, we can calculate Bellman error vector (note, that we define vector of states as $\mathbf{s} = (s_0, s_1)^T$ and ϕ_i corresponds to the ϕ of state s_i):

$$\begin{aligned} v_w(\mathbf{s}) &= \begin{pmatrix} w \cdot \phi_0 \\ w \cdot \phi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ (B^\pi v_w)(\mathbf{s}) &= \begin{pmatrix} r + \gamma(w \cdot \phi_1) \\ r + \gamma(w \cdot \phi_0) \end{pmatrix} = \begin{pmatrix} 0 + 1 \cdot (1 \cdot 2) \\ 0 + 1 \cdot (1 \cdot 1) \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \\ \bar{\delta}_w &= \begin{pmatrix} (B^\pi v_w)(s_0) - v_w(s_0) \\ (B^\pi v_w)(s_1) - v_w(s_1) \end{pmatrix} = \begin{pmatrix} 2 - 1 \\ 1 - 2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \end{aligned}$$

2. What is the Mean Squared Bellman Error?

Solution: Assuming that $\mu(s_0) = \mu(s_1) = \frac{1}{2}$, so we are equally interested in accurately estimating value of both states, we can calculate Mean Squared Bellman Error as following:

$$\|\bar{\delta}_w\|_\mu^2 = \sum_{s \in \{s_0, s_1\}} \mu(s) v(s)^2 = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1$$

3. What w results in the value functions that is closest (in least-squares sense) to the target values $B^\pi v_w$?

Solution: To calculate w that results in the value functions that is closest (in least-squares sense) to the target values $B^\pi v_w$, we simply need to find point on v_w onto which $B^\pi v_w$ will be projected. To do that, we should calculate following function: $w = \arg\min_{w \in \mathbb{R}} \|B^\pi v'_w - v_w\|_\mu^2$ (note that v'_w stands for v_w with current $w = 1$).

$$f(w) = \| \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ 2 \end{pmatrix} w \|_\mu^2 = \| \begin{pmatrix} 2 - w \\ 1 - 2w \end{pmatrix} \|_\mu^2$$

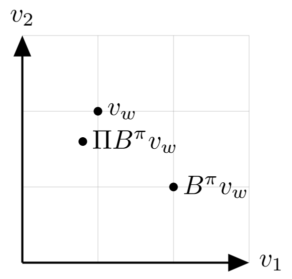
$$
\begin{aligned}
&= \frac{1}{2}(2-w)^2 + \frac{1}{2}(1-2w)^2 \\
&= \frac{1}{2}(4 - 4w + w^2 + 1 - 4w + 4w^2) \\
&= \frac{1}{2}(5w^2 - 8w + 5) \\
\frac{df}{dw} &= 10w - 80 \\
w &= \frac{4}{5}
\end{aligned}
$$

Therefore, $w = \frac{4}{5}$ results in the value functions that is closest (in least-squares sense) to the target values $B^\pi v_w$.

4. Plot v_w, B^π_w and $B^\pi v_w$. Explain what is happening. (hint: Refer to Figure 11.3 in the book).

Solution: As you can see in Figure 4, we have value function for defined $w = 1$, which is located in a subspace of dimension one in two-dimensional space (i.e. line). This allows us to project it into the larger space with B^π_w and then project it down onto the line using projection operator. This projection is denoted as $B^\pi v_w$. It corresponds to the value w, which is the closest to B^π_w in least-squares sense.

So, reaching true value function v_π consists of sequential interchanging steps of something similar to expectation maximization algorithm, where in expectation step we calculate result of applying Bellman operator B^π_w with given w and in the maximization step we find w to get as close to B^π_w using projection operator.



Problem 5 (Neural Networks)**:**

1. Consider the state distribution, $\mu(s)$. How does it depend on the parameters of the value function approximator?

Solution: It depends on the parameters of the value function approximator through the policy $\pi(a|s)$. In fact, changing the parameters of the value functiona approximator also changes the resulting policy, which in turns changes the fraction of time spent in each state, which is represented by $\mu(s)$

2. How does this differ from standard (un-)supervised learning problems?

Solution: This differs from supervised and unsupervised learning asa there the targets and the weightings (when present) are fixed and do not depend on the learnable parameters, while in this case both $\mu_(s)$ and the targets depend by the parametrization.

Also, in supervised/unsupervised learning the data are fixed and chosen in advance, while in reinforcement learning we dynamically explore the environment, often without fully knowing is dynamics, and the data seen can be different based on our current policy, for example.

3. What does this mean for the weighting of the errors (such as in e.g. Eq. 9.1)?

Solution: This means that the errors made in each state are weighted differently based on the values of $\mu(s)$. This makes intuitive sense, as when minimizing the mean squared value error, we want errors made in states which are visited more often to be weighted more.

4. The DQN paper [1] relies, amongst others, on the use of an earlier idea called experience replay [2]. What does this trick do that is important for the algorithm?

Solution: Memory replay addresses the problems that neural networks have when trained online. One main drawback of neural networks is that they suffer from *catastrophic forgetting*, which means that they tend to forget information learned in previous samples, the more samples are seen. So, by having a memory and allowing sampling of already seen states, we can partially address the problem as the network can see the same state multiple times.

Furthermore, typically first order optimizers require the data to be independent in order to converge to good local minima. This is not the case in reinforcement learning, where experience is gained online and thus samples are very related in the same episode. By training only on data sampled on the replay memory, we address the dependence problem as we take random samples from the past.

5. An other important trick is the use of a separate target network that is frozen for periods of time. What does this trick do that is important for the algorithm?

Solution: Using a separate target network is important to stabilize the learning process and prevent drastic changes in the policy. If we would not use a separate target network, then every update to the action-value function would also modify the target of the update, which can lead to oscillations or divergence of the policy. Thus, updating only periodically the target network allows the action-value function to approximate a stable target, and makes oscillations or divergence less likely.

6 Policy Gradient

6.1 Policy Gradient: REINFORCE

In the lecture, we have seen that introducing a constant baseline b does not introduce a bias to our policy gradient.

$$\nabla J = \mathbb{E}\left[(G(\tau) - b)\, \nabla \log p(\tau)\right]$$

We now want to consider the variance when introducing a baseline.

1. Derive the optimal constant baseline that minimizes the variance of the policy gradient. Interpret your result.

Solution:

$$\begin{aligned}
\underset{\tau}{\mathbb{V}}[(G_\tau - b)\nabla \log p(\tau)] &= \underset{\tau}{\mathbb{E}}[(G_\tau - b)^2(\nabla \log p(\tau))^2] - (\underset{\tau}{\mathbb{E}}[(G_\tau - b)\nabla \log p(\tau)])^2 \\
&= \underset{\tau}{\mathbb{E}}[(G_\tau^2 - 2bG_\tau + b^2)(\nabla \log p(\tau))^2] - (\underset{\tau}{\mathbb{E}}[(G_\tau - b)\nabla \log p(\tau)])^2 \\
&= \underset{\tau}{\mathbb{E}}[(G_\tau(\nabla \log p(\tau)))^2] - 2b\,\underset{\tau}{\mathbb{E}}[G_\tau(\nabla \log p(\tau))^2] + b^2\,\underset{\tau}{\mathbb{E}}[(\nabla \log p(\tau))^2] \\
&- (\underset{\tau}{\mathbb{E}}[(G_\tau - b)\nabla \log p(\tau)])^2 \\
&= \mathbb{V}[G_\tau(\nabla \log p(\tau))] - 2b\,\underset{\tau}{\mathbb{E}}[G_\tau(\nabla \log p(\tau))^2] + b^2\,\underset{\tau}{\mathbb{E}}[(\nabla \log p(\tau))^2]
\end{aligned}$$

Now, to minimize variance, we can find extremum point of it, which will also correspond to the minimum point.

$$\begin{aligned}
\frac{\partial\, \mathbb{V}_\tau[(G_\tau - b)\nabla \log p(\tau)]}{\partial b} &= -2\,\underset{\tau}{\mathbb{E}}[G_\tau(\nabla \log p(\tau))^2] + 2b\,\underset{\tau}{\mathbb{E}}[(\nabla \log p(\tau))^2] = 0 \\
2b\,\underset{\tau}{\mathbb{E}}[(\nabla \log p(\tau))^2] &= 2\,\underset{\tau}{\mathbb{E}}[G_\tau(\nabla \log p(\tau))^2] \\
b &= \frac{\mathbb{E}_\tau[G_\tau(\nabla \log p(\tau))^2]}{\mathbb{E}_\tau[(\nabla \log p(\tau))^2]}
\end{aligned}$$

2. Consider the simple example from the lecture in a bandit setting (i.e. no states):

$$\begin{aligned}
r &= a + 2 \\
a &\sim N(\theta, 1) \\
\nabla_\theta \log \pi(a) &= a - \theta
\end{aligned}$$

Can you argue what should be the optimal constant baseline in this case? You can use your result from 1.

Solution:

$$
\begin{aligned}
b &= \frac{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[G_\tau(\nabla_\theta \log p(\tau))^2]}{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[(\nabla_\theta \log p(\tau))^2]} \\
&= \frac{\mathbb{E}_a[G_a(\nabla \log p(a))^2]}{\mathbb{E}_a[(\nabla \log p(a))^2]} \\
&= \frac{\mathbb{E}_a[(a+2)(a-\theta)^2]}{\mathbb{E}_a[(a-\theta)^2]} \\
&= \frac{\mathbb{E}_a[a^3 - 2\theta a^2 + a\theta^2 + 2a^2 - 4a\theta + 2\theta^2]}{\mathbb{E}_a[a^2 - 2a\theta + \theta^2]} \\
&= \frac{\mathbb{E}_a[a^3] + 2(1-\theta)\,\mathbb{E}_a[a^2] + (\theta^2 - 4\theta)\,\mathbb{E}_a[a] + 2\theta^2}{\mathbb{E}_a[a^2] - 2\theta\,\mathbb{E}_a[a] + \theta^2} \\
&= \frac{\theta^3 + 3\theta + 2(1-\theta)(1+\theta^2) + (\theta^2 - 4\theta)\theta + 2\theta^2}{(1 + \theta^2 - 2\theta^2 + \theta^2)} \\
&\qquad\qquad \text{(using known moments of Gaussian distribution)} \\
&= \theta^3 + 3\theta + 2 + 2\theta^2 - 2\theta - 2\theta^3 + \theta^3 - 4\theta^2 + 2\theta^2 \\
&= \theta + 2
\end{aligned}
$$

3. Now, consider a baseline that is not constant, but dependent on the state $b(s_t)$. We want to establish that in this case, the policy gradient remains unbiased. Show that

$$\mathop{\mathbb{E}}_{\tau}[b(s_t)\nabla \log p(\tau)] = 0$$

Hint: you can use the linearity of expectation or the law of iterated expectation to decouple parts of the full trajectory τ.

Solution:

$$
\begin{aligned}
\mathop{\mathbb{E}}_{\tau}[b(s_t)\nabla \log p(\tau)] &= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\tau}\left[b(s_t)\nabla \log \pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{a_t, s_t}\left[b(s_t)\nabla \log \pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[\sum_{a_t} \pi(a_t|s_t) b(s_t) \nabla \log \pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[\sum_{a_t} \pi(a_t|s_t) b(s_t) \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)}\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[\sum_{a_t} b(s_t)\nabla\pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[b(s_t)\sum_{a_t} \nabla\pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[b(s_t)\nabla\sum_{a_t} \pi(a_t|s_t)\right] \\
&= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_t}\left[b(s_t)\nabla 1\right] \\
&= 0
\end{aligned}
$$

6.2 Policy Gradient: Compatible Function Approximation Theorem

Consider the actor-critic method that optimizes the expected return J using the gradient

$$\nabla J = \mathop{\mathbb{E}}_{\tau}\left[\sum_{t=1}^{T} \nabla \log \pi(a_t|s_t) q_\pi(s_t, a_t)\right]$$

where π is our parameterized policy (actor) and q_π is the true state-action value under policy π. In the lecture we have seen that we can approximate q_π with a parameterized value function $\hat{q}_w$ which is unbiased if it fulfills the condition

$$
\begin{aligned}
\nabla_w \hat{q}_w &= \nabla_\theta \log \pi_\theta(s, a) \\
\text{e.g.}\hat{q}_w &= w^T \nabla_\theta \log \pi_\theta(s, a)
\end{aligned}
$$

1. Show that $\mathbb{E}_a\left[\hat{q}_w(s, a)\right] = 0, \forall s \in S$ Briefly interpret this result.

Solution: In the case of linear function approximator, we have

$$
\begin{aligned}
\mathop{\mathbb{E}}_{a}\left[\hat{q}_w(s, a)\right] &= \mathop{\mathbb{E}}_{a}\left[w^T \nabla_\theta \log \pi_\theta(a|s)\right] \\
&= w^T \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \\
&= w^T \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= w^T \sum_a \nabla_\theta \pi_\theta(a|s) \\
&= w^T \nabla_\theta (\sum_a \pi_\theta(a|s)) \\
&= w^T \nabla_\theta 1
\end{aligned}
$$

$$= 0$$

We can see that our estimation $\hat{q}(s,a)$ is not actually an estimator of the true action-value function q_π, as its espected value is 0 and not q_π. However, this is reasonable, as we only care about satisfying the compatible function approximation theorem, which guarantees that the average of the gradients using our estimation $\hat{q}(s,a)$ is the same as the average using the true action-value function, so that the actor receives correct gradients.

2. Evaluate the expectation $\mathbb{E}_a\left[q_\pi(s,a) - v_\pi(s),\right], \forall s \in S$. The term $A(s,a) = q_\pi(s,a) - v_\pi(s)$ is also known as the advantage function.

Solution:

$$\begin{aligned}\mathop{\mathbb{E}}_a\left[q_\pi(s,a) - v_\pi(s),\right] &= \mathop{\mathbb{E}}_a[q_\pi(s,a)] - \mathop{\mathbb{E}}_a[v_\pi(s)] \\ &= v_\pi(s) - v_\pi(s) \\ &= 0\end{aligned}$$

3. What do you conclude from the results in 1 and 2?

Solution: We have that the expectation of our approximation $\hat{q}(s,a)$ is the same as the advantage function. Therefore, optimizing $\hat{q}(s,a)$ can be also seen as optimizing the advantage function.

4. Consider the following policy

$$\pi_\theta(s,a) = \frac{e^{\theta^T \phi_{sa}}}{\sum_b e^{\theta^T \phi_{sb}}}$$

corresponding to a softmax with linear parametrization with respect to the state-action features ϕ_{sa} . Give an expression for the parametrization of $\hat{q}_w$ that satisfies the compatible function approximation theorem.

Solution: To satisfy the compatible function approximation theorem, we can use a linear function approximator $\hat{q}_w = w^T \nabla_\theta \log \pi_\theta(s,a)$. We first calculate the derivative with respect to the log policy

$$\nabla_\theta \log \pi_\theta(s,a) = \nabla_\theta \log \frac{e^{\theta^T \phi_{sa}}}{\sum_b e^{\theta^T \phi_{sb}}}$$

$$
\begin{aligned}
&= \frac{\sum_b e^{\theta^T \phi_{sb}}}{e^{\theta^T \phi_{sa}}} \nabla_\theta \frac{e^{\theta^T \phi_{sa}}}{\sum_b e^{\theta^T \phi_{sb}}} \\
&= \frac{\sum_b e^{\theta^T \phi_{sb}}}{e^{\theta^T \phi_{sa}}} \frac{(\nabla_\theta e^{\theta^T \phi_{sa}}) \sum_b e^{\theta^T \phi_{sb}} - e^{\theta^T \phi_{sa}} \nabla_\theta \sum_b e^{\theta^T \phi_{sb}}}{(\sum_b e^{\theta^T \phi_{sb}})^2} \\
&= \frac{1}{e^{\theta^T \phi_{sa}}} \frac{\phi_{sa} e^{\theta^T \phi_{sa}} \sum_b e^{\theta^T \phi_{sb}} - e^{\theta^T \phi_{sa}} \sum_b \phi_{sb} e^{\theta^T \phi_{sb}}}{\sum_b e^{\theta^T \phi_{sb}}} \\
&= \frac{\phi_{sa} \sum_b e^{\theta^T \phi_{sb}} - \sum_b \phi_{sb} e^{\theta^T \phi_{sb}}}{\sum_b e^{\theta^T \phi_{sb}}} \\
&= \phi_{sa} - \sum_b \phi_{sb} \frac{e^{\theta^T \phi_{sb}}}{\sum_b e^{\theta^T \phi_{sb}}} \\
&= \phi_{sa} - \sum_b \pi(s,b) \phi_{sb}
\end{aligned}
$$

Using this result, we can express $\hat{q}_w$ as

$$
\begin{aligned}
\hat{q}_w &= w^T \nabla_\theta \log \pi_\theta(s,a) \\
&= w^T (\phi_{sa} - \sum_b \pi(s,b) \phi_{sb})
\end{aligned}
$$

6.3 Policy Gradient: Natural Gradient

In this section we parameterize our policy with a univariate Gaussian probability density over real-valued actions. We consider a stateless bandit scenario where you can assume that an episode solely consists of one action and one reward. Mean and variance are learned.

$$
\pi(a|\theta) = \frac{1}{\sigma(\theta_\sigma)\sqrt{2\pi}} \exp\left(- \frac{(a - \mu(\theta_\sigma))^2}{2\sigma(\theta_\sigma)^2} \right)
$$

where we approximate the mean with a single parameter and the standard deviation with the exponential of a parameter

$$
\begin{aligned}
\mu(\theta_\mu) &= \theta_\mu \\
\sigma(\theta_\sigma) &= \exp(\theta_\sigma)
\end{aligned}
$$

1. Calculate $\nabla_\theta \log \pi(a|\theta)$ w.r.t. θ_π and θ_σ

Solution:

$$
\begin{aligned}
\nabla_{\theta_\mu} \log \pi(a|\theta) &= \nabla_{\theta_\mu} \left(- \frac{(a - \theta_{\theta_\mu})^2}{2\sigma(\theta_\sigma)^2} \right) - \nabla_{\theta_\mu} \log(\sigma(\theta_\sigma)\sqrt{2\pi}) \\
&= \nabla_{\theta_\mu} \left(- \frac{(a - \theta_\mu)^2}{2\sigma(\theta_\sigma)^2} \right) \\
&= -\frac{1}{2\sigma(\theta_\sigma)^2} \nabla_{\theta_\mu} (a - \theta_\mu)^2
\end{aligned}
$$

$$
\begin{aligned}
&= -\frac{1}{2\sigma(\theta_\sigma)^2} 2(a-\theta_\mu)(-\theta_\mu) \\
&= \frac{(a-\theta_\mu)}{\sigma(\theta_\sigma)^2} \\
&= \frac{(a-\theta_\mu)}{\exp(\theta_\sigma)^2} \\
&= e^{-2\theta_\sigma}(a-\theta_\mu)
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\sigma \log \pi(a|\theta) &= \nabla_\sigma \left(-\frac{(a-\theta_\mu)^2}{2\sigma(\theta_\sigma)^2} \right) - \nabla_\sigma \log(\sigma(\theta_\sigma)\sqrt{2\pi}) \\
&= \nabla_\sigma \left(-\frac{(a-\theta_\mu)^2}{2} \frac{1}{\sigma(\theta_\sigma)^2} \right) - \frac{1}{\sigma(\theta_\sigma)\sqrt{2\pi}} \sqrt{2\pi} \\
&= \frac{(a-\theta_\mu)^2}{\sigma(\theta_\sigma)^3} - \frac{1}{\sigma(\theta_\sigma)}
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{\theta_\sigma} \log \pi(a|\theta) &= \nabla_\sigma [\log \pi(a|\theta)] \nabla_{\theta_\sigma} \sigma(\theta_\sigma) \\
&= \left(\frac{(a-\theta_\mu)^2}{\exp(\theta_\sigma)^3} - \frac{1}{\exp(\theta_\sigma)} \right) \exp(\theta_\sigma) \\
&= \frac{(a-\theta_\mu)^2}{\exp(\theta_\sigma)^2} - 1 \\
&= e^{-2\theta_\sigma}(a-\theta_\mu)^2 - 1
\end{aligned}
$$

2. Why would we want use the natural policy gradient as opposed to the ”vanilla” policy gradient?

Solution:

The vanilla policy gradient updates are dependent on the learning rate, and based on the magnitude of the gradients this can lead to worse policies after an update. This does not happen when using natural gradients, which guarantee an improvement in the policy. Also, the natural gradient is independent on the parametrization of the policy, and make the improvements directly in the policy space rather than the parameter space. This is done through the fisher information matrix F, which measures the curvature of the policy with respect to the model parameters. Notice also that the natural gradient constraints the actual amount of change in the policy at each step.

These features make the natural more reliable in terms of quality of updates, and typically lead to better policies.

3. By applying the natural policy gradient we want to the solve the contraint optimization problem

$$\theta^* - \theta_0 = \max_{d\theta} J(\theta_0 + d\theta) \text{ s.t. } d\theta^T F_\theta d\theta = c$$

where we limit our gradient change to c in KL-divergence between the old and updated policy. The fisher information matrix F and the natural policy gradient update step is then given by

$$F_\theta = \mathop{\mathbb{E}}_{\tau}\left[\nabla_{d\theta} \log \pi(a|\theta_0 + d\theta) \nabla_{d\theta} \log \pi(a|\theta_0 + d\theta)^T\right]$$
$$\theta^* - \theta_0 \sim F^{-1} \nabla_\theta J(\theta_0)$$

Calculate the Fisher information matrix F_θ for our Gaussian policy.

Solution:

$$\nabla_{d\theta} \log \pi(a|\theta_0 + d\theta) = \begin{bmatrix} \nabla_{\theta_\mu} \log \pi(a|\theta) \\ \nabla_{\theta_\sigma} \log \pi(a|\theta) \end{bmatrix} = \begin{bmatrix} e^{-2\theta_\sigma}(a - \theta_\mu) \\ e^{-2\theta_\sigma}(a - \theta_\mu)^2 - 1 \end{bmatrix}$$

$$\begin{aligned} \nabla_{d\theta} \log \pi(a|\theta_0 + d\theta) \nabla_{d\theta} \log \pi(a|\theta_0 + d\theta)^T &= \begin{bmatrix} \nabla_{\theta_\mu} \log \pi(a|\theta) \\ \nabla_{\theta_\sigma} \log \pi(a|\theta) \end{bmatrix} \left[\nabla_{\theta_\mu} \log \pi(a|\theta), \nabla_{\theta_\sigma} \log \pi(a|\theta)\right] \\ &= \begin{bmatrix} e^{-2\theta_\sigma}(a - \theta_\mu) \\ e^{-2\theta_\sigma}(a - \theta_\mu)^2 - 1 \end{bmatrix} \left[e^{-2\theta_\sigma}(a - \theta_\mu), e^{-2\theta_\sigma}(a - \theta_\mu)^2 - 1\right] \\ &= \begin{bmatrix} e^{-4\theta_\sigma}(a - \theta_\mu)^2 & e^{-4\theta_\sigma}(a - \theta_\mu)^3 - e^{-2\theta_\sigma}(a - \theta_\mu) \\ e^{-4\theta_\sigma}(a - \theta_\mu)^3 - e^{-2\theta_\sigma}(a - \theta_\mu) & e^{-4\theta_\sigma}(a - \theta_\mu)^4 - 2e^{-2\theta_\sigma}(a - \theta_\mu)^2 + 1 \end{bmatrix} \end{aligned}$$

Before calculating the expectation of the matrix, we write down the known results for the central moments of the univariate gaussian distribution

$$\begin{aligned} \mathop{\mathbb{E}}_a[a - \theta_\mu] &= 0 \\ \mathop{\mathbb{E}}_a[(a - \theta_\mu)^2] &= \mathbb{V}(a) = e^{2\theta_\sigma} \\ \mathop{\mathbb{E}}_a[(a - \theta_\mu)^3] &= 0 \\ \mathop{\mathbb{E}}_a[(a - \theta_\mu)^4] &= 3\,\mathbb{V}(a)^2 = 3e^{4\theta_\sigma} \end{aligned}$$

Where μ_3 Using the same approach as in 6.1.3 where we have shown that expectation of the trajectories can be simplified down to expectation over actions, we get following Fisher Information Matrix:

$$\begin{aligned} F_\theta &= \begin{bmatrix} \mathbb{E}_a\left[e^{-4\theta_\sigma}(a - \theta_\mu)^2\right] & \mathbb{E}_a\left[e^{-4\theta_\sigma}(a - \theta_\mu)^3 - e^{-2\theta_\sigma}(a - \theta_\mu)\right] \\ \mathbb{E}_a\left[e^{-4\theta_\sigma}(a - \theta_\mu)^3 - e^{-2\theta_\sigma}(a - \theta_\mu)\right] & \mathbb{E}_a\left[e^{-4\theta_\sigma}(a - \theta_\mu)^4 - 2e^{-2\theta_\sigma}(a - \theta_\mu)^2\right] + 1 \end{bmatrix} \\ &= \begin{bmatrix} e^{-4\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)^2\right] & e^{-4\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)^3\right] - e^{-2\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)\right] \\ e^{-4\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)^3\right] - e^{-2\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)\right] & e^{-4\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)^4\right] - 2e^{-2\theta_\sigma}\,\mathbb{E}_a\left[(a - \theta_\mu)^2\right] + 1 \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
&= \begin{bmatrix} e^{-4\theta_\sigma} e^{2\theta_\sigma} & e^{-4\theta_\sigma} \cdot 0 - e^{-2\theta_\sigma} \cdot 0 \\ e^{-4\theta_\sigma} \cdot 0 - e^{-2\theta_\sigma} \cdot 0 & e^{-4\theta_\sigma} \cdot 3e^{4\theta_\sigma} - 2e^{-2\theta_\sigma} e^{2\theta_\sigma} + 1 \end{bmatrix} \\
&= \begin{bmatrix} e^{-2\theta_\sigma} & 0 \\ 0 & 3 - 2 + 1 \end{bmatrix} \\
&= \begin{bmatrix} e^{-2\theta_\sigma} & 0 \\ 0 & 2 \end{bmatrix}
\end{aligned}
$$

4. Explain the effect of the Fisher information matrix on gradient updates of θ_μ when using the natural gradient.

Solution: To do gradient update with Fisher information matrix, we need to multiply gradient by the inverse of Fisher information matrix. As our information matrix is diagonal, we can see that taking an inverse of it is the same as taking an inverse of each element on the diagonal. Also, in that case we can treat elements on the diagonal of Fisher information matrix as scaling factors for corresponding parameters.

Therefore, when we multiply θ_μ by $e^{2\theta_\sigma}$, it increases the update, which means that this direction is less sensitive to the updates. Fish information matrix thus controls which directions of the gradient should be updated more in order to reach optimal policy and which should be updated less. This also reminds of adaptive learning rate approach used in deep learning.

5. Consider a slightly different parametrization of the standard deviation as

$$\sigma(\theta_\sigma) = \theta_\sigma$$

where we simply learn directly. We will ignore that the variance can technically become negative in this case. Calculate $\nabla \log \pi(a|\theta)$ and F w.r.t. θ_σ for this alternative parametrization.

Solution: Using the result found in the first part of this exercise, we have

$$\nabla_\sigma \log \pi(a|\theta) = \frac{(a - \theta_\mu)^2}{\sigma(\theta_\sigma)^3} - \frac{1}{\sigma(\theta_\sigma)}$$

$$
\begin{aligned}
\nabla_{\theta_\sigma} \log \pi(a|\theta) &= \nabla_\sigma [\log \pi(a|\theta)] \nabla_{\theta_\sigma} \sigma(\theta_\sigma) \\
&= \left(\frac{(a - \theta_\mu)^2}{\theta_\sigma^3} - \frac{1}{\theta_\sigma} \right) 1 \\
&= \frac{(a - \theta_\mu)^2}{\theta_\sigma^3} - \frac{1}{\theta_\sigma}
\end{aligned}
$$

While for θ_μ the result is, reusing the calculations of previous points

$$\nabla_{\theta_\mu} \log \pi(a|\theta) = \frac{(a-\theta_\mu)}{\sigma(\theta_\sigma)^2} = \frac{(a-\theta_\mu)}{\theta_\sigma^2}$$

The Fisher information matrix can be written then as

$$\begin{aligned}
\nabla_{d\theta} \log \pi(a|\theta_0 + d\theta) \nabla_{d\theta} \log \pi(a|\theta_0 + d\theta)^T &= \begin{bmatrix} \nabla_{\theta_\mu} \log \pi(a|\theta) \\ \nabla_{\theta_\sigma} \log \pi(a|\theta) \end{bmatrix} \left[\nabla_{\theta_\mu} \log \pi(a|\theta), \nabla_{\theta_\sigma} \log \pi(a|\theta) \right] \\
&= \begin{bmatrix} \frac{(a-\theta_\mu)}{\theta_\sigma^2} \\ \frac{(a-\theta_\mu)^2}{\theta_\sigma^3} - \frac{1}{\theta_\sigma} \end{bmatrix} \left[\frac{(a-\theta_\mu)}{\theta_\sigma^2}, \frac{(a-\theta_\mu)^2}{\theta_\sigma^3} - \frac{1}{\theta_\sigma} \right] \\
&= \begin{bmatrix} \frac{(a-\theta_\mu)^2}{\theta_\sigma^4} & \frac{(a-\theta_\mu)^3}{\theta_\sigma^5} - \frac{(a-\theta_\mu)}{\theta_\sigma^4} \\ \frac{(a-\theta_\mu)^3}{\theta_\sigma^5} - \frac{(a-\theta_\mu)}{\theta_\sigma^4} & \frac{(a-\theta_\mu)^4}{\theta_\sigma^6} - 2\frac{(a-\theta_\mu)^2}{\theta_\sigma^4} + \frac{1}{\theta_\sigma^2} \end{bmatrix}
\end{aligned}$$

As in 6.3.3, we can write

$$\begin{aligned}
F_\theta &= \begin{bmatrix} \mathbb{E}_a[\frac{(a-\theta_\mu)^2}{\theta_\sigma^4}] & \mathbb{E}_a[\frac{(a-\theta_\mu)^3}{\theta_\sigma^5} - \frac{(a-\theta_\mu)}{\theta_\sigma^4}] \\ \mathbb{E}_a[\frac{(a-\theta_\mu)^3}{\theta_\sigma^5} - \frac{(a-\theta_\mu)}{\theta_\sigma^4}] & \mathbb{E}_a[\frac{(a-\theta_\mu)^4}{\theta_\sigma^6} - 2\frac{(a-\theta_\mu)^2}{\theta_\sigma^4} + \frac{1}{\theta_\sigma^2}] \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\theta_\sigma^2} & 0 \\ 0 & \frac{3\theta_\sigma^4}{\theta_\sigma^6} - 2\frac{\theta_\sigma^2}{\theta_\sigma^4} + \frac{1}{\theta_\sigma^2} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\theta_\sigma^2} & 0 \\ 0 & \frac{3-2+1}{\theta_\sigma^2} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\theta_\sigma^2} & 0 \\ 0 & \frac{2}{\theta_\sigma^2} \end{bmatrix}
\end{aligned}$$

6. Compare how the Fisher information matrix influences the gradient of θ_σ considering both parametrizations of σ. Explain your observations.

Solution: We can easily see that in the first parametrization, standard deviation will be updated less as $2 < 8$. This is due to the fact that in case of parametrization with exponent, changing θ_σ influences the value of σ more than in case of linear parametrization. Therefore, Fisher information matrix controls our updates not to change the value of the parameter too radically, which might potentially result in non-converging performance.

7. Suppose our current parameters are $\mu = 0$ and $\sigma = 4$. We now observe an episode with $a = 8$ and $F(\tau) = 1$. Perform a gradient update on θ_σ with learning rate $\alpha = 0.01$ using the vanilla policy gradient as well as using the natural policy gradient for both parametrizations,

i.e. a total of four independent updates. Report the updated values of σ and discuss your observations.

Solution: The main thing to notice is that in our case with one episode and parametrization on $\boldsymbol{\theta}$, gradient can be calculated as $\nabla_{\boldsymbol{\theta}} J = G(t)\nabla \log \pi(a_t|\boldsymbol{\theta})$.

For the first suggested parametrization, $\sigma = 4, \mu = 0$, therefore $\theta_\sigma = \ln 4, \theta_\mu = 0$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J &= 1 \cdot \begin{bmatrix} e^{-2\theta_\sigma}(a - \theta_\mu) \\ e^{-2\theta_\sigma}(a - \theta_\mu)^2 - 1 \end{bmatrix} \\
&= 1 \cdot \begin{bmatrix} 4^{-2}(8 - 0) \\ 4^{-2}(8 - 0)^2 - 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{2} \\ 3 \end{bmatrix} \\
\boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J && \text{(Vanilla policy gradient)} \\
&= \begin{bmatrix} 0 \\ \ln 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} \frac{1}{2} \\ 3 \end{bmatrix} \\
&\approx \begin{bmatrix} 0.005 \\ 1.416294 \end{bmatrix} \\
\mu &= 0.005 \\
\sigma &= 4.121816 \\
\boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha F_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J && \text{(Natural policy gradient)} \\
&= \begin{bmatrix} 0 \\ \ln 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} e^{-2\theta_\sigma} & 0 \\ 0 & 2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{1}{2} \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ \ln 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} \frac{1}{16} & 0 \\ 0 & 2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{1}{2} \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ \ln 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} 16 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2} \\ 3 \end{bmatrix} && \text{(property of diagonal matrix)} \\
&\approx \begin{bmatrix} 0 \\ \ln 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} 8 \\ 1.5 \end{bmatrix} \\
&\approx \begin{bmatrix} 0.08 \\ 1.401294 \end{bmatrix} \\
\mu &= 0.08 \\
\sigma &= 4.060450
\end{aligned}$$

For the second suggested parametrization, $\sigma = 4, \mu = 0$, therefore $\theta_\sigma = 4, \theta_\mu = 0$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J &= 1 \cdot \begin{bmatrix} \frac{(a-\theta_\mu)}{\theta_\sigma^2} \\ \frac{(a-\theta_\mu)^2}{\theta_\sigma^3} - \frac{1}{\theta_\sigma} \end{bmatrix} \\
&= 1 \cdot \begin{bmatrix} \frac{8}{16} \\ \frac{64}{62} - \frac{1}{4} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J && \text{(Vanilla policy gradient)} \\
&= \begin{bmatrix} 0 \\ 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix} \\
&\approx \begin{bmatrix} 0.005 \\ 4.0075 \end{bmatrix} \\
\mu &= 0.005 \\
\sigma &= 4.0075 \\
\boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha F_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J && \text{(Natural policy gradient)} \\
&= \begin{bmatrix} 0 \\ 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} \frac{1}{\theta_\sigma^2} & 0 \\ 0 & \frac{2}{\theta_\sigma^2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} \frac{1}{16} & 0 \\ 0 & \frac{2}{16} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} 16 & 0 \\ 0 & 8 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix} && \text{(property of diagonal matrix)} \\
&= \begin{bmatrix} 0 \\ 4 \end{bmatrix} + 0.01 \cdot \begin{bmatrix} 8 \\ 6 \end{bmatrix} \\
&\approx \begin{bmatrix} 0.08 \\ 4.06 \end{bmatrix} \\
\mu &= 0.08 \\
\sigma &= 4.06
\end{aligned}$$

The observed result in the case of natural gradient confirms our suggestion in 6.6, as, indeed, in the first parametrization we update θ_σ by 0.015 and in the second parametrization we update it by 0.06. This is due to the fact that changing parameter of the exponent changes the value of the function much more than when updating parameter of linear function. Therefore, Fisher information matrix keeps our updates more careful.

On the other hand, in the case of vanilla gradient, our update increases for the exponential case, which also means that the value of σ changes more rapidly. This might result in that in the case of exponential parametrization, we might never be able to reach the optimal value as we are changing parameter too fast, which is not as bad in the linear case. Natural gradient helps us to avoid this issue.

Both of these observations can also be seen from the fact that for both parametrizations, value of the σ stays very similar when using natural gradient but gets quite different with vanilla gradient.

References