# OReL Assignment 4

bkx591

March 2024

## 1 Direct Policy Search

### 1.1 Multi-variate normal distribution

1. Consider $a \in \mathbb{R}$ and $\boldsymbol{a} \in \mathbb{R}^n$ if $\boldsymbol{a}$ is not the zero vector, it has a non-zero rank of 1.

   (a) The outer product of $\boldsymbol{a}\boldsymbol{a}^T$ generates the matrix $C$, whose rows are the vector $\boldsymbol{a}$ scaled by the values of $\boldsymbol{a}$ s.t.

   $$\begin{aligned} C_1 &= \boldsymbol{a}_1[\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n] \\ C_2 &= \boldsymbol{a}_2[\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n] \\ &\vdots \\ C_n &= \boldsymbol{a}_n[\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n] \end{aligned}$$

   and $\boldsymbol{a}$ is the basis - all rows of $C$ are linear combinations of $\boldsymbol{a}$. Therefore the rank of $C$ is 1, as it has exactly one linearly independent row.

   (b) To prove that $\boldsymbol{a}$ is an eigenvector of matrix $C = \boldsymbol{a}\boldsymbol{a}^T$, it must satisfy the equation $Cv_{\text{eig}} = \lambda v_{\text{eig}}$, where $v_{\text{eig}}$ is the eigenvector and $\lambda$ is the eigenvalue. Let us substitute $C = \boldsymbol{a}\boldsymbol{a}^T$ and $v_{\text{eig}} = \boldsymbol{a}$ and get

   $$\boldsymbol{a}\boldsymbol{a}^T\boldsymbol{a} = \lambda\boldsymbol{a}$$

   Since $\boldsymbol{a}^T\boldsymbol{a}$ is a scalar, let's denote it by $\alpha$, we can rewrite the equation as

   $$\alpha\boldsymbol{a} = \lambda\boldsymbol{a}$$

   which shows that $\alpha$ must be the eigenvalue and $\boldsymbol{a}$ the eigenvector of $C$.

   (c) The one-dimensional normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean having the highest likelihood to generate $a \in \mathbb{R}$ has variance $\sigma^2 =$

$a^2$. To prove this, we start by recalling the likelihood function, the probability density function, of a gaussian/normal distribution:

$$f(x; \sigma^2, \mu) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

or

$$f(a; \sigma^2, 0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{a^2}{2\sigma^2}}$$

To find the variance that maximizes this likelihood, we take the derivative of the log-likelihood with respect to $\sigma^2$ and set it to zero. Let's first write out the log-likelihood:

$$\begin{aligned} L(a; \sigma^2) &= f(x; \sigma^2, 0) \\ \ln L(a; \sigma^2) &= \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{a^2}{2\sigma^2} \\ &= -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{a^2}{2\sigma^2} \end{aligned}$$

differentiate with respect to $\sigma^2$

$$\frac{\partial}{\partial\sigma^2} \ln L(a; \sigma^2) = -\frac{1}{2\sigma^2} + \frac{a^2}{2(\sigma^2)^2}$$

and finally, to find the maximum likelihood estimate of $\sigma^2$, we set the derivative to zero and solve for $\sigma^2$

$$0 = -\frac{1}{2\sigma^2} + \frac{a^2}{2(\sigma^2)^2}$$

$$\frac{1}{2\sigma^2} = \frac{a^2}{2(\sigma^2)^2}$$

Examining this equation, considering $\sigma^2 = a^2$, we get

$$\begin{aligned} \frac{1}{2\sigma^2} &= \frac{a^2}{2(a^2)^2} \\ \frac{1}{2\sigma^2} &= \frac{1}{2a^2} \end{aligned}$$

proving that the variance of the normal distribution that maximizes the likelihood of observing a single datapoint $a$ is the datapoint $a$ itself.

Using the log-likelihood for solving for $\sigma$ gives a much cleaner derivation, though it is also possible to solve just using the likelihood function.

2. Consider $m$ random vectors $x_1, ..., x_m \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

(a) The random vector

$$z = \sum_{i=1}^{m} x_i$$

is distributed according to the $x$s that makes it up. Assuming each $x_i$ is independently and identically distributed to a multivariate normal distribution with mean vector 0 and covariance matrice $\boldsymbol{I}$, then, by the properties of the normal distribution, their sum is also normally distributed. Therefore, $z$ is distributed as $z \sim \mathcal{N}(0, C)$. The mean of a sum of random variables is the sum of their means. Since each $x_i$ is normal distributed with a mean of 0, the mean $\mu_z$ of $z$ is also 0:

$$\mu_z = \sum_{i=1}^{m} \mu_{x_i} = \sum_{i=1}^{m} 0 = 0$$

This can also be shown with linearity of expectations. Let $\mathbb{E}[x_i] = \mu$ denote the expected values of a random normal distributed vector $x_i$, then

$$\mathbb{E}[z] = \mathbb{E}[\sum_{i=1}^{m} x_i] = \sum_{i=1}^{m} \mathbb{E}[x_i] = \sum_{i=1}^{m} \mu$$

In this case, $\mu = 0$.

As for the covariance we'll do a bit longer formal proof [1]. Consider the set of independent and standard normally distributed random variables

$$x_i \sim \mathcal{N}(0, 1), i = 1, ..., n$$

Then, these variables together form a multivariate normally distributed random vector $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Because the covariance is zero for independent random variables, we have

$$Cov(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0, \forall i \neq j$$

and for $i = j$ we have

$$Cov(\boldsymbol{x}_i, \boldsymbol{x}_j) = Var(\boldsymbol{x}_i) = 1$$

s.t.

$$Cov(\boldsymbol{x}) = \boldsymbol{I}$$

(In the above, and sorry for this, $\boldsymbol{x}_i$ represents the ith index, or the ith random variable in the n-dimension vector $\boldsymbol{x}$ and not the ith random vector). From this, we can derive the covariance of $z$ simply as the sum of identity matrices - additivity of covariances (like linearity of expectations) also holds here - s.t.

$$Cov(z) = \sum_{i=1}^{m} \boldsymbol{I} = m\boldsymbol{I}$$

[1] See https://statproofbook.github.io/P/mvn-cov.html for my inspiration.

Therefore, the random vector $z$ is distributed according to $\mathcal{N}(\mathbf{0}, m\boldsymbol{I})$. This proof could be further elongated by derivation from the definition of the covariance matrix, setting the mean to 0 and simplify, but I hope this'll do.

(b) Now considering additional $m$ positive real-valued weights $w_1, ..., w_m \in \mathbb{R}^+$, the random vector

$$z_w = \sum_{1=1}^{m} w_i \boldsymbol{x_i}$$

is in some ways distributed similarly to $z$. It's safe to say, that the weighted sum of independent normally distributed random vectors is also normally distributed. The weights $w_i$ scales the random vectors up or down (positive real-valued), so we have that $z_w$ is a linear combination of $x_i$ with mean 0. The covariance is more complex though, as $w_i$ is a real values, random variable, so we almost certainly have that $w_i \neq w_j$. In the simplest form, we can write out the covariance as the weighted sum of $m$ square diagonal matrices with value $w_i$:

$$\sum_{i=1}^{m} w_i \boldsymbol{I}$$

which makes $z_w$ distributed according to $z_w \sim \mathcal{N}(\mathbf{0}, \sum_{i=1}^{m} w_i \boldsymbol{I})$

(c) If $m < n$, the rank of $C$ is at most $m$, because we have fewer vectors than the dimensions in our $n$-dimensional space requires to span all possible dimensions. If $m \geq n$, the rank of $C$ is at most $n$, because even with more vectors than dimensions, the rank is bound by the number of dimensions ($n$). Therefore, with probability 1, by combining the two inequalities, we have the $\mathbb{P}[rank(C) = \min(m, n)] = 1$.

## 1.2 Neuroevolution

1.
```
class CartPolePolicyNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, use_bias=True):
        super(CartPolePolicyNet, self).__init__()
        # Define the hidden layer with tanh activation
        self.hidden = nn.Linear(input_size, hidden_size, bias=use_bias)
        self.tanh = nn.Tanh()

        # Define the output layer with linear activation
        self.output = nn.Linear(hidden_size, output_size, bias=use_bias)

    def forward(self, x):
        x = self.tanh(self.hidden(x))
        x = self.output(x)
        return x
```

```
input_size = 4  # Cart Position, Cart Velocity, Pole Angle, Pole Angular Velocity
hidden_size = 5  # Number of neurons in the hidden layer
output_size = 1  # Single output for the action spacve {0, 1} / {left, right}

# Create the policy network without bias
policy_net = CartPolePolicyNet(input_size, hidden_size, output_size, use_bias=False)
```

2. The training and balancing metrics can be seen in Table 1. 200 models were trained in total, of which 100 were with bias and 100 were without. Their performance was tested in randomly initialized environments (-0.05, 0.05). Analyzing the numbers gives us an indication as to the degree of complexity fit for the task of balancing a single pole. The model without bias trains for drastically fewer steps before being able to balance for 500 steps than the one with bias. Let's dig a bit more into it.

| Model Configuration | Balanced For (steps) | Trained For (steps) |
|---|---|---|
| With Bias | 89.59 | 1960.48 |
| Without Bias | 119.67 | 24.42 |

Table 1: Comparison of NN with and without bias in the CartPole-v1 environment (low: -0.05, high: 0.05)

**Training Time**

- **With Bias**: The significantly longer training time suggests that the network is utilizing the additional degrees of freedom provided by the bias terms to adjust its weights and biases to closely fit the training data. This process can be more computationally intensive and time-consuming due to the larger parameter space, which is seen empirically in Table 1.
- **Without Bias**: The shorter training time indicates a simpler model with fewer parameters to adjust. This simplicity has quite likely lead to the faster convergence but might reduce model flexibility.

**Performance (Balanced for)**

- **With Bias**: Surprisingly, the models with bias performed worse in terms of the time they were able to balance the pole. This might suggest that while these models are theoretically capable of capturing more complex patterns, they might also be more prone to overfitting or not generalizing well to the dynamics of the environment due to their complexity.
- **Without Bias**: The better performance of the bias-free models suggests that, for this particular task, the complexity added by bias terms might not be necessary. It could also indicate that the models

without bias are better at generalizing from their training, possibly because they are forced to focus on the most significant patterns present in the input data.

I did further experiments with a wider range of random initialization of the environment (-0.2, 0.2) which gave more indications as to the impact of the randomness. It seems both models have reached a performance bottleneck that might be defined more by their capacity for generalization than by the specific advantages conferred by bias terms. Both models perform equally bad - it takes ¡60 steps for the pole to tip on a passive balancing cart.

| Model Configuration | Balanced For (steps) | Trained For (steps) |
|---|---|---|
| With Bias | 61.6 | 1969.87 |
| Without Bias | 61.23 | 24.65 |

Table 2: Comparison of NN with and without bias in the CartPole-v1 environment (low: -0.2, high: 0.2)

## 2 Policy Gradient Methods

### 2.1 Analytical Score Function

I realized pretty late, that the assignment asks for the differentiation of the log policy with a different definition. I initially did the differentiation of definition from slide 17, which is still attached below. It is much more thorough and contains more in depth steps of the derivation. Therefore, if you want more details as to which tools and steps to take, look at that derivation first. Alas, here comes the differentiation of the log policy from the last slide in a somewhat more compressed format.

**Differentiation of pi from final slide** For differentiating the softmax policy, we have:

$$\pi(s,a) = \frac{e^{\theta_a^\top s}}{\sum_{a'} e^{\theta_{a'}^\top s}}$$

The logarithm of the softmax policy $\pi$ for state $s$ and action $a$ is:

$$\log \pi(s,a) = \theta_a^\top s - \log\left(\sum_{a'} e^{\theta_{a'}^\top s}\right)$$

To find the gradient of $\log \pi(s,a)$ with respect to the parameters $\theta_i$ for action $i$, differentiate both terms separately: For the first term:

$$\nabla_{\theta_i}(\theta_a^\top s) = \begin{cases} s & \text{if } i = a \\ 0 & \text{otherwise} \end{cases}$$

For the second term, differentiate the log of the sum (using the chain rule and the fact that the derivative of $e^x$ is $e^x$):

$$\nabla_{\theta_i} \log\left(\sum_{a'} e^{\theta_{a'}^\top s}\right) = \frac{1}{\sum_{a'} e^{\theta_{a'}^\top s}} \sum_{a'} \nabla_{\theta_i} e^{\theta_{a'}^\top s}$$

Since the derivative of $e^{\theta_{a'}^\top s}$ with respect to $\theta_i$ is $e^{\theta_{a'}^\top s} s$ if $a' = i$ and 0 otherwise, the expression simplifies to:

$$\nabla_{\theta_i} \log\left(\sum_{a'} e^{\theta_{a'}^\top s}\right) = \frac{e^{\theta_i^\top s} s}{\sum_{a'} e^{\theta_{a'}^\top s}}$$

$$= \pi(s, i) \cdot s$$

Combining both terms, the gradient of $\log \pi(s, a)$ with respect to $\theta_i$ is:

$$\nabla_{\theta_i} \log \pi(s, a) = \begin{cases} s - \pi(s, i) \cdot s & \text{if } i = a \\ -\pi(s, i) \cdot s & \text{otherwise} \end{cases}$$

$$= (s - \pi(s, i) \cdot s) \cdot 1(a = i) - \pi(s, i) \cdot s \cdot 1(a \neq i)$$

$$= s \cdot (1(a = i) - \pi(s, i))$$

**Differentiation of pi from side 17** To differentiate the log of the softmax policy $\pi(s, a|\theta)$, we apply the chain rule and properties of the logarithm function. The softmax policy is given by:

$$\pi(s, a|\theta) = \frac{e^{\theta^T f(s,a)}}{\sum_{a'} e^{\theta^T f(s,a')}}$$

Taking the logarithm of the policy, we get:

$$\ln \pi(s, a|\theta) = \theta^T f(s, a) - \ln\left(\sum_{a'} e^{\theta^T f(s,a')}\right)$$

Differentiating the log-policy with respect to the parameters $\theta$, we obtain:

$$\nabla_\theta \ln \pi(s, a|\theta) = \nabla_\theta(\theta^T f(s, a)) - \nabla_\theta \ln\left(\sum_{a'} e^{\theta^T f(s,a')}\right)$$

The gradient of the first term, $\theta^T f(s, a)$, is straightforward:

$$\nabla_\theta(\theta^T f(s, a)) = f(s, a)$$

For the second term, we apply the gradient to the log-sum-exp function, which yields:

$$\nabla_\theta \ln \left( \sum_{a'} e^{\theta^T f(s,a')} \right) = \frac{\sum_{a'} e^{\theta^T f(s,a')} \nabla_\theta (\theta^T f(s,a'))}{\sum_{a'} e^{\theta^T f(s,a')}}$$

Since $\nabla_\theta(\theta^T f(s,a')) = f(s,a')$, this simplifies to:

$$\nabla_\theta \ln \left( \sum_{a'} e^{\theta^T f(s,a')} \right) = \frac{\sum_{a'} e^{\theta^T f(s,a')} f(s,a')}{\sum_{a'} e^{\theta^T f(s,a')}}$$

Now, by recognizing that this is the expectation of the feature vector $f(s,a')$ under the policy $\pi$, we get:

$$\nabla_\theta \ln \left( \sum_{a'} e^{\theta^T f(s,a')} \right) = \sum_{a'} \pi(s,a'|\theta) f(s,a')$$

Putting it all together, the gradient of the log-policy becomes:

$$\nabla_\theta \ln \pi(s,a|\theta) = f(s,a) - \sum_{a'} \pi(s,a'|\theta) f(s,a')$$

This expression represents the difference between the feature vector for the action taken and the expected feature vector under the policy distribution, and we will use it to update the policy parameters $\theta$ in the REINFORCE algorithm given in the notebook.

## 2.2 Implementation of Analytical Score Function

The implementation is simply a few lines of code, utilizing the existing functions in *Softmax_policy*, implementing the following functionality:

$$\nabla_\theta \ln \pi(s,a|\theta) = f(s,a) - \sum_{a'} \pi(s,a'|\theta) f(s,a')$$

The solution is verified by comparing it to the numerical approximation and checking for some epsilon tolerance of $\epsilon = 10^-4$. The analytical solution is verified if for each element of the gradient, it is within a fraction of the corresponding numerical approximation by said tolerance - and this solution is verified.

```
def gradient_log_pi(self, s, a):
    """
    Computes the gradient of the logarithm of the policy
    :param s: state feature vector
    :param a: action
    :return: gradient of the logarithm of the policy
    """
    # Compute action probabilities for all actions
    pi_s = self.pi(s)
    # Initialize gradient matrix
    d = np.zeros(self.theta.shape)

    # Compute gradient for all actions and features
    for action in range(self.no_actions):
        d[action, :] = s * (int(action == a) - pi_s[action])

    return d
```

## 2.3 Expectation of Control Variable

Let us first differentiate the policy $\pi$ with respect to $\theta$. The softmax policy $\pi(a|s;\theta)$ is given by

$$\pi(a|s;\theta) = \frac{e^{\theta^T f(s,a)}}{\sum_{a'} e^{\theta^T f(s,a')}}$$

and we want to compute the gradient of $\pi(a|s;\theta)$ with respect to the parameters $\theta$:

$$\nabla_\theta \pi(a|s;\theta) = \nabla_\theta \left( \frac{e^{\theta^T f(s,a)}}{\sum_{a'} e^{\theta^T f(s,a')}} \right)$$

First step involves the chain rule for differentiating the exponential function of $\theta$ in the numerator, and the summation of exponentials in the denominator. For the numerator we have that $\nabla_\theta e^{\theta^T f(s,a)} = f(s,a)e^{\theta^T f(s,a)}$. For the denominator, the chain rule is applied to the sum of exponentials, which involves differentiating each term of the sum, resulting in a sum of terms of the form $f(s,a')e^{\theta^T f(s,a')}$.

Now applying the quotient rule for differentiation, which states that for two functions $u(\theta))$ and $v(\theta)$, the derivative of $\frac{u(\theta)}{v(\theta)}$ is:

$$\nabla_\theta \left( \frac{u(\theta)}{v(\theta)} \right) = \frac{v(\theta)\nabla_\theta u(\theta) - u(\theta)\nabla_\theta v(\theta)}{v(\theta)^2}$$

For

$$u(\theta) = e^{\theta^T f(s,a)} \tag{1}$$

$$v(\theta) = \sum_{a'} e^{\theta^T f(s,a')} \tag{2}$$

$$u'(\theta) = f(s,a)e^{\theta^T f(s,a)}) \tag{3}$$

we get:

$$\nabla_\theta \pi(a|s;\theta) = \frac{\sum_{a'} e^{\theta^T f(s,a')} f(s,a) e^{\theta^T f(s,a)} - e^{\theta^T f(s,a)} \sum_{a'} f(s,a') e^{\theta^T f(s,a')}}{\left(\sum_{a'} e^{\theta^T f(s,a')}\right)^2} \tag{4}$$

$$= \frac{\sum_{a'} e^{\theta^T f(s,a')} f(s,a) e^{\theta^T f(s,a)}}{\left(\sum_{a'} e^{\theta^T f(s,a')}\right)^2} - \frac{e^{\theta^T f(s,a)} \sum_{a'} f(s,a') e^{\theta^T f(s,a')}}{\left(\sum_{a'} e^{\theta^T f(s,a')}\right)^2} \tag{5}$$

$$= \frac{e^{\theta^T f(s,a)} f(s,a)}{\sum_{a'} e^{\theta^T f(s,a')}} - \frac{e^{\theta^T f(s,a)}}{\sum_{a'} e^{\theta^T f(s,a')}} \sum_{a'} \frac{e^{\theta^T f(s,a')} f(s,a')}{\sum_b e^{\theta^T f(s,b)}} \tag{6}$$

By recognizing that $e^{\theta^T f(s,a)}$ in the numerator and $\sum_{a'} e^{\theta^T f(s,a')}$ in the denominator represent $\pi(a|s;\theta)$ and $v(\theta)$ respectively, we can simplify the gradient as follows:

$$\begin{aligned}
\nabla_\theta \pi(a|s;\theta) &= \pi(a|s;\theta) f(s,a) - \pi(a|s;\theta) \sum_{a'} \frac{e^{\theta^T f(s,a')}}{\sum_b e^{\theta^T f(s,b)}} f(s,a') \\
&= \pi(a|s;\theta) f(s,a) - \pi(a|s;\theta) \sum_{a'} \pi(a'|s;\theta) f(s,a') \\
&= \pi(a|s;\theta) \left( f(s,a) - \sum_{a'} \pi(a'|s;\theta) f(s,a') \right) \\
&= \pi(a|s;\theta) \left( \nabla_\theta \ln \pi(a|s;\theta) \right)
\end{aligned}$$

Now let's use this to prove that $\mathbb{E}[\sum_a \nabla_\theta \pi(s,a) b(s)] = 0$:

$$\mathbb{E}\left[\sum_a \nabla_\theta \pi(a|s) b(s)\right] = \mathbb{E}\left[\sum_a \pi(a|s) \left( f(s,a) - \sum_{a'} \pi(a'|s) f(s,a') \right) b(s)\right] \tag{7}$$

$$= \mathbb{E}\left[\sum_a \pi(a|s) f(s,a) b(s) - \sum_a \pi(a|s) b(s) \sum_{a'} \pi(a'|s) f(s,a')\right] \tag{8}$$

Using the fact that $\sum_a \pi(a|s) = 1$ and the linearity of expectation, this (the second term) simplifies to:

$$\mathbb{E}\left[\sum_a \pi(a|s) f(s,a) b(s)\right] - \mathbb{E}\left[b(s) \sum_{a'} \pi(a'|s) f(s,a') \sum_a \pi(a|s)\right]$$

$b(s)$ doesn't change in the sum, so we can move it outside as a constant

$$b(s)\mathbb{E}\left[\sum_a \pi(a|s) f(s,a)\right] - b(s)\mathbb{E}\left[\sum_{a'} \pi(a'|s) f(s,a')\right]$$

Now it is obvious, that these two terms are equivalent, because they both represent the expected value of $f(s, a)$ weighted by the policy $\pi$ and the baseline $b(s)$. They thus cancel each other out, and we have that

$$\mathbb{E}[\sum_a \nabla_\theta \pi(s, a) b(s)] = 0$$