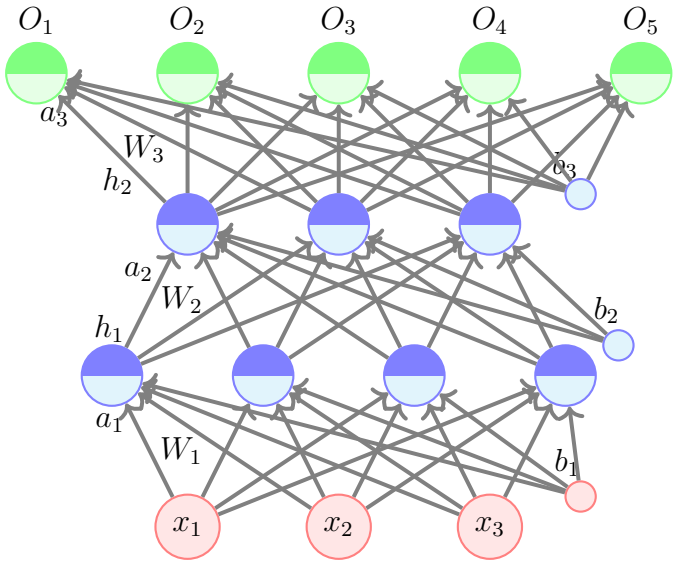
1. Consider the feedforward neural network in the figure shown below which has one input layer, two hidden layers and one output layer. The input x to this network $\in \mathbb{R}^3$ and the number of neurons in the two hidden layers and the output layer is 4,3,5 respectively. Each layer is fully connected to the next layer, *i.e.*, there is a weight connecting every neuron in layer i to every neuron in layer $i+1$. Further, every neuron in the hidden and output layers also has a bias connected to it. What is the total number of parameters in this feedforward neural network?



A. 60

B. 51

C. 39

D. 44

Solution: Let us compute the number of parameters in layer 1 first. Each of the 3 inputs (x_1, x_2, x_3) is connected to each of the 4 hidden units results in 3*4 = 12 parameters. In addition, each of the 4 hidden units has a bias resulting in 4 more parameters. Thus, there are a total of 12+4 = 16 parameters in layer 1. Similarly, layer 2 has 4*3 + 3 = 15 parameters and layer 3 has 3*5 + 5 = 20 parameters. Thus, the total number of parameters in the network is 16 + 15 + 20 = 51.

Option B is the correct answer.

2. Consider the function $f(\theta) = f(x, y, z) = x^2 + y^2 + z^2 - 8$. What is the gradient of this function at at $\theta = \{1, -1, 1\}$. Note that $\theta = [x, y, z]$ is a collection of all the parameters of this function.

   A. [2, -2, 2]

   B. [-2, 2, -2]

   C. [2, -2, -2]

   D. [-2, -2, 2]

Solution:

Option A is the correct answer.

$$\nabla\theta = [\frac{\partial f(\theta)}{\partial x}, \frac{\partial f(\theta)}{\partial y}, \frac{\partial f(\theta)}{\partial z}]$$

Now,

$$\frac{\partial f(\theta)}{\partial x} = \frac{\partial}{\partial x}(x^2 + y^2 + z^2 - 8) = 2x$$
$$\frac{\partial f(\theta)}{\partial y} = \frac{\partial}{\partial y}(x^2 + y^2 + z^2 - 8) = 2y$$
$$\frac{\partial f(\theta)}{\partial z} = \frac{\partial}{\partial z}(x^2 + y^2 + z^2 - 8) = 2z$$

Hence,

$$\nabla\theta = [2x, 2y, 2z]$$

Substituting, $x = 1, y = -1, z = 1$, we get

$$\nabla\theta|_{\theta=\{1,-1,1\}} = [2, -2, 2]$$

3. Consider the function $f(\theta) = f(x, y, z) = x^2 + y^2 + z^2 - 8$. Suppose you start with $\theta_0 = \{1, -1, 1\}$ and run one step of gradient descent with the learning rate $\eta = 1$. What will be the updated value of θ?

A. [1, -1, 1]

B. [1, -1, -1]

C. [-1, -1, 1]

D. [-1, 1, -1]

Solution:

Option D is the correct answer.

Using the update rule of gradient descent, the value of θ after running one iteration starting from θ_0 is given by :

$$\begin{aligned} \theta_1 &= \theta_0 - \eta \nabla \theta_0 \\ &= \theta_0 - \nabla \theta_0 \quad [\because \eta = 1] \end{aligned}$$

In the previous question we computed,

$$\nabla \theta_0 = \nabla \theta|_{\theta=\{1,-1,1\}} = [2, -2, 2]$$

Substituting the value of θ_0 and $\nabla \theta_0$ in the above equation we get,

$$\begin{aligned} \theta_1 &= [1, -1, 1] - [2, -2, 2] \\ &= [-1, 1, -1] \end{aligned}$$

4. Consider the vector $a = [1.2, -2.5, 2.4, 3]$. What will the following lines of code do ?

```
import numpy as np

a = np.asarray([1.2, -2.5, 2.4, 3])
a = np.exp(a)
a = a/np.sum(a)
```

A. It will compute the softmax of a

B. It will compute the element-wise sigmoid of a

C. It will compute the e-th power of each element of a

Solution: Option A is the correct answer. You can just run the code and check it.

5. Consider a box which contains 100 balls of which 30 are red, 50 are green and 20 are blue. Your friend peeps into the box and estimates the number of red, green and blue balls as 50, 25, 25. What is the cross entropy between the true distribution over the colors in the box and the distribution predicted by your friend?

   A. 1.5

   B. 1.0

   C. 1.7

   D. 2.3

Solution: Let X be the random variable denoting the color of a ball. X can thus take on 3 values: red, green and blue. The true probability distribution of X can be computed as $p = [P(X = red), P(X = green), P(X = blue)]$, i.e., $p = [\frac{30}{100}, \frac{50}{100}, \frac{20}{100}]$, *i.e.*, $p = [0.3, 0.5, 0.2]$. Similarly, the probability distribution estimated by your friend can be computed as $q = [0.5, 0.25, 0.25]$. Given these two distributions the cross entropy can be computed using the following formula:

$$\begin{aligned} CE(p,q) &= -\sum_{i \in \{red,blue,green\}} p_i \log_2 q_i \\ &= -(0.3 \log_2 0.5 + 0.5 \log_2 0.25 + 0.2 \log_2 0.25) \\ &= -(0.3 * (-1) + 0.5 * (-2) + 0.2 * (-2)) \\ &= 1.7 \end{aligned}$$

Option C is the correct answer.

6. Consider a vector $a \in \mathbb{R}^n$ and let $b \in \mathbb{R}^n$ be the output of the softmax function applied to this vector. The i-th entry of b is given by:

$$b_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}$$

Now suppose we introduce a parameter k such that

$$b_i = \frac{e^{k*a_i}}{\sum_{j=1}^{n} e^{k*a_j}}$$

Can b still represent a probability distribution ?

A. True

B. False

Solution: Yes, every entry of b will still be ≥ 0 and $\sum_{i=1}^{n} b_i = 1$.
Option A is the correct answer.

7. Continuing the above question where $a \in \mathbb{R}^n$, $b \in \mathbb{R}^n$ and we have a parameter k such that

$$b_i = \frac{e^{k*a_i}}{\sum_{j=1}^{n} e^{k*a_j}}$$

If $k = 1$ then we get the default softmax function. Now, for simplicity, let us assume that all elements of a are positive. Further, let us assume that the j-th element of a is the maximum/largest element of a. It should be obvious that the corresponding j-th element of b will also be the maximum/largest element of b. Now suppose we set $k = 2$, then what will happen to the j-th entry of b.

A. It will remain the same as the case when $k = 1$ because now $k = 2$ appears in the denominator also

B. It will be greater than the case when $k = 1$

C. It will be lesser than the case when $k = 1$

D. Can't say as it will depend on the other values in a

Solution: Option B is the correct answer. As you increase the value of k, the softmax function becomes sharper and sharper. In other words, rich becomes richer. As we increase the value of k, the largest element of b will become even greater than what it was when $k = 1$. To get a better feel for this, you can experiment with different values of k using the following code:

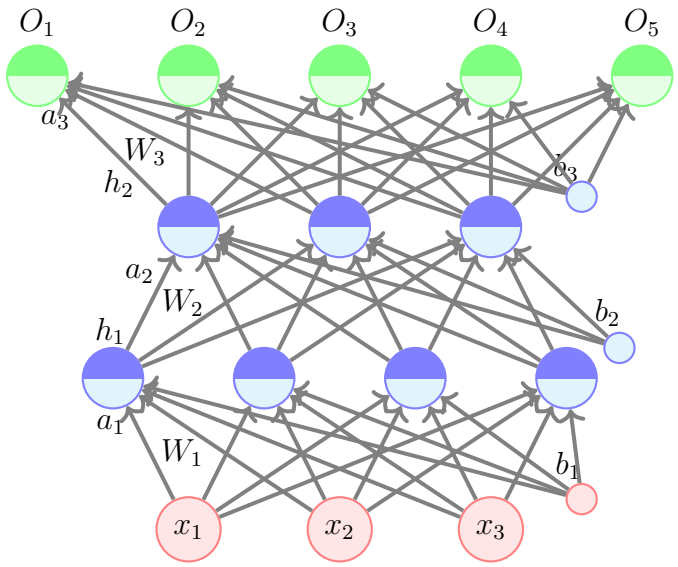
```
import numpy as np

k = 1 # increase/decrease the value of a and see how b changes
a = np.asarray([1.2, 2.5, 2.4, 3])
b = np.exp(k * a)
b = b/np.sum(b)
print (b)
```

8. Consider the feedforward neural network in the figure shown below which has one input layer, two hidden layers and one output layer. The input x to this network $\in \mathbb{R}^3$ and the number of neurons in the two hidden layers and the output layer is 4,3,5 respectively. Each layer is fully connected to the next layer, *i.e.*, there is a weight connecting every neuron in layer i to every neuron in layer $i+1$. Also note that every neuron in the hidden and output layers has a bias connected to it. The activation function used in the two hidden layers is the logistic function as defined in the lecture and the output function is the softmax function. Now suppose that all the weights in layer 1 are 0.05, *i.e.*, each of the 3*4=12 elements of the matrix W_1 has a value 0.05. Similarly, let us assume that all the weights in layer 2 are 0.025, *i.e.*, each of the 4*3=12 elements of the matrix W_2 has a value 0.025. Also, let us assume that all the weights in layer 3 are 1.0, *i.e.*, each of the 3*5=15 elements of the matrix W_3 has a value 1. Finally, the bias vectors for the 3 layers are as follows:

$$b_1 = [0.1, 0.2, 0.3, 0.4]$$
$$b_2 = [5.2, 3.2, 4.3]$$
$$b_3 = [0.2, 0.45, 0.75, 0.55, 0.95]$$

Now, suppose we feed the input $x = [1.5, 2.5, 3]$ to this network, what will be the value of O_3 (i.e., the value output by the third neuron in the output layer).



A. 0.132

B. 0.189

C. 0.229

D. 0.753

Solution:

Option C is the correct answer. You can verify this by running the following code:

```
import numpy as np
import math

#This will take a vector as input and
#compute the sigmoid of each element of the vector
def logistic(x):
  return 1 / (1 + np.exp(-x))

#This will take a vector as input and return the softmax of this vector
def softmax(x):
  a = np.exp(x)
  a = a/np.sum(a)
  return a

#the input to the network
x = [1.5, 2.5, 3]

#the weights in the first layer
W1 = np.full((3, 4), 0.05)

#the weights in the second layer
W2 = np.full((4, 3), 0.025)

#the weights in the second layer
W3 = np.full((3, 5), 1.0)

#the biases in the 3 layers
b1 = np.asarray([0.1, 0.2, 0.3, 0.4])
b2 = np.asarray([5.2, 3.2, 4.3])
b3 = np.asarray([0.2, 0.45, 0.75, 0.55, 0.95])

#Compute the pre-activation and activation for the first hidden layer
a1 = np.matmul(x, W1) + b1  # W1*x + b1
h1 = logistic(a1)      # logistic(W1*x + b1)

#Compute the pre-activation and activation for the second hidden layer
#Note: the input to the second layer is the output of the 1st hidden
    layer
a2 = np.matmul(h1, W2) + b2 # W2*h1 + b2
h2 = logistic(a2)      # logistic(W2*h1 + b2)

#Compute the pre-activation and output for the output layer
#Note: the input to the output layer is the output of the 2nd layer
a3 = np.matmul(h2, W3) + b3 # W3*h2 + b3
o = softmax(a3)        # logistic(W3*h2 + b3)

print (o)
```