Neural_Networks

February 9, 2023

1 Neural Networks: Maths

This notebook is for the mathematical details of the neural network. Here, mathematical details about forward and backward propagation will be given.

The notations used are given in the file

1.1 Forward Propagation

Suppose $X \in \mathbb{R}^{n_x \times m}$ is the input matrix, there are L layers in the network, and $W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$ and $b^{[l]} \in \mathbb{R}^{n^{[l]} \times 1}$ are the weight and bias matrices for the l^{th} layer. Then, the output of the l^{th} layer is given by

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} A^{[l]} = g^{[l]}(Z^{[l]})$$

Where $g^{[l]}$ is the activation function for the l^{th} layer. The shapes of the Z and A's are - $Z^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$ and $A^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$. This is the forward propagation.

Note that $A^{[0]} = X$.

1.1.1 An Example of Forward Propagation: Shapes

We'll take the following parameters as an example:

- $n_x = 10$
- $n_y = 3$
- $m = 100$
- $L = 3$
- $n^{[0]} = 10, n^{[1]} = 8, n^{[2]} = 5, n^{[3]} = 3$ (Note that $n^{[0]} = n_x$ and $n^{[L]} = n_y$)

Then, the shapes of the matrices should be:

$$W^{[1]} \in \mathbb{R}^{8 \times 10}$$
$$b^{[1]} \in \mathbb{R}^{8 \times 1}$$
$$W^{[2]} \in \mathbb{R}^{5 \times 8}$$
$$b^{[2]} \in \mathbb{R}^{5 \times 1}$$
$$W^{[3]} \in \mathbb{R}^{3 \times 5}$$
$$b^{[3]} \in \mathbb{R}^{3 \times 1}$$

While the outputs should have the shape:

$$Z^{[1]} \in \mathbb{R}^{8\times 100}$$
$$A^{[1]} \in \mathbb{R}^{8\times 100}$$
$$Z^{[2]} \in \mathbb{R}^{5\times 100}$$
$$A^{[2]} \in \mathbb{R}^{5\times 100}$$
$$Z^{[3]} \in \mathbb{R}^{3\times 100}$$
$$A^{[3]} \in \mathbb{R}^{3\times 100}$$

Let's calculate the shape of $Z^{[1]}$, $Z^{[2]}$ and $Z^{[3]}$ from the given equation to see this is indeed the case.

$Z^{[1]}$ gives

$$\begin{aligned} Z^{[1]} &= W^{[1]}A^{[0]} + b^{[1]} \\ &= (8\times 10)(10\times 100) + (8\times 1) \\ &= (8\times 100) + (8\times 1) \\ &= (8\times 100) \end{aligned}$$

Note that numpy broadcasting is used here, that is, in numpy the bias with shape (8×1) is added to the matrix with shape (8×100) by braodcasting the bias to the shape (8×100). See the numpy broadcasting documentation for more details.

We see that $Z^{[1]}$ has the shape (8×100) as expected. Let's do the same for $Z^{[2]}$ and $Z^{[3]}$.

$$\begin{aligned} Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ &= (5\times 8)(8\times 100) + (5\times 1) \\ &= (5\times 100) + (5\times 1) \\ &= (5\times 100) \end{aligned}$$

We see that $Z^{[2]}$ has the shape (5×100) as expected. Let's do the same for $Z^{[3]}$.

$$\begin{aligned} Z^{[3]} &= W^{[3]}A^{[2]} + b^{[3]} \\ &= (3\times 5)(5\times 100) + (3\times 1) \\ &= (3\times 100) + (3\times 1) \\ &= (3\times 100) \end{aligned}$$

The output $A^{[3]}$ is sigmoid of $Z^{[3]}$, so it has the same shape as $Z^{[3]}$. We see that $A^{[3]}$ has the shape (3×100) as expected, since Y has the shape (3×100).

1.1.2 How Numpy Broadcasting Takes Care of the Bias

Let's see how numpy broadcasting takes care of the bias. We'll take the following example:

- $n^{[1]} = 2$
- $n^{[2]} = 3$
- $m = 4$

- $l = 2$

The output of the 2^{nd} layer then will be

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

The matrices involved in the euqation will have shape:

$$W^{[2]} \in \mathbb{R}^{3\times 2}$$
$$A^{[1]} \in \mathbb{R}^{2\times 4}$$
$$b^{[2]} \in \mathbb{R}^{3\times 1}$$

Let's suppose that $A^{[1]}$ has the following matrix form

$$A^{[1]} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$$

And, the output $W^{[2]}$ is

$$W^{[2]} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

And, the bias $b^{[2]}$ is

$$b^{[2]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Let's multiply the matrices $W^{[2]}$ and $A^{[1]}$.

$$\begin{aligned} W^{[2]}A^{[1]} &= \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \\ &= \begin{bmatrix} w_{11}a_{11} + w_{12}a_{21} & w_{11}a_{12} + w_{12}a_{22} & w_{11}a_{13} + w_{12}a_{23} & w_{11}a_{14} + w_{12}a_{24} \\ w_{21}a_{11} + w_{22}a_{21} & w_{21}a_{12} + w_{22}a_{22} & w_{21}a_{13} + w_{22}a_{23} & w_{21}a_{14} + w_{22}a_{24} \\ w_{31}a_{11} + w_{32}a_{21} & w_{31}a_{12} + w_{32}a_{22} & w_{31}a_{13} + w_{32}a_{23} & w_{31}a_{14} + w_{32}a_{24} \end{bmatrix} \end{aligned}$$

Numpy broadcasts the bais

$$b^{[2]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

as

$$\begin{bmatrix} b_1 & b_1 & b_1 & b_1 \\ b_2 & b_2 & b_2 & b_2 \\ b_3 & b_3 & b_3 & b_3 \end{bmatrix}$$

Adding the bias to the matrix multiplication gives

$$W^{[2]}A^{[1]} + b^{[2]} = \begin{bmatrix} w_{11}a_{11} + w_{12}a_{21} & w_{11}a_{12} + w_{12}a_{22} & w_{11}a_{13} + w_{12}a_{23} & w_{11}a_{14} + w_{12}a_{24} \\ w_{21}a_{11} + w_{22}a_{21} & w_{21}a_{12} + w_{22}a_{22} & w_{21}a_{13} + w_{22}a_{23} & w_{21}a_{14} + w_{22}a_{24} \\ w_{31}a_{11} + w_{32}a_{21} & w_{31}a_{12} + w_{32}a_{22} & w_{31}a_{13} + w_{32}a_{23} & w_{31}a_{14} + w_{32}a_{24} \end{bmatrix} + \begin{bmatrix} b_1 & b_1 & b_1 & b_1 \\ b_2 & b_2 & b_2 & b_2 \\ b_3 & b_3 & b_3 & b_3 \end{bmatrix}$$

$$= \begin{bmatrix} w_{11}a_{11} + w_{12}a_{21} + b_1 & w_{11}a_{12} + w_{12}a_{22} + b_1 & w_{11}a_{13} + w_{12}a_{23} + b_1 & w_{11}a_{14} + w_{12}a_{24} + b_1 \\ w_{21}a_{11} + w_{22}a_{21} + b_2 & w_{21}a_{12} + w_{22}a_{22} + b_2 & w_{21}a_{13} + w_{22}a_{23} + b_2 & w_{21}a_{14} + w_{22}a_{24} + b_2 \\ w_{31}a_{11} + w_{32}a_{21} + b_3 & w_{31}a_{12} + w_{32}a_{22} + b_3 & w_{31}a_{13} + w_{32}a_{23} + b_3 & w_{31}a_{14} + w_{32}a_{24} + b_3 \end{bmatrix}$$

To see if the output is correct, we calulate $\mathbf{z}_3^{[2]}$ which is defined as the output (before activation) of the 3^{nd} nueuron in the 2^{nd} layer. Phyically, this should be a row vector of length 4. And this should the 3^{rd} row of the matrix $Z^{[2]}$.

We can calculate it by hand as this is nothing but the output of each samples by the third neuron in the second layer. Now, output of the first sample by the third neuron in the second layer is

$$z_3^{[2](1)} = w_{31}a_{11} + w_{32}a_{21} + b_3$$

Similarly, the output of the second sample by the third neuron in the second layer is

$$z_3^{[2](2)} = w_{31}a_{12} + w_{32}a_{22} + b_3$$

Similarly, the output of the third sample by the third neuron in the second layer is

$$z_3^{[2](3)} = w_{31}a_{13} + w_{32}a_{23} + b_3$$

Finally, the output of the fourth sample by the third neuron in the second layer is

$$z_3^{[2](4)} = w_{31}a_{14} + w_{32}a_{24} + b_3$$

Putting these in a row vector, we get

$$\begin{aligned} \mathbf{z}_3^{[2]} &= \begin{bmatrix} z_3^{[2](1)} & z_3^{[2](2)} & z_3^{[2](3)} & z_3^{[2](4)} \end{bmatrix} \\ &= \begin{bmatrix} w_{31}a_{11} + w_{32}a_{21} + b_3 & w_{31}a_{12} + w_{32}a_{22} + b_3 & w_{31}a_{13} + w_{32}a_{23} + b_3 & w_{31}a_{14} + w_{32}a_{24} + b_3 \end{bmatrix} \end{aligned}$$

This is the same as the 3^{rd} row of the matrix $Z^{[2]}$.

As another test, we can calculate $Z^{[2](3)}$ which is the output of the third sample by the second layer. This should be the 3^{rd} column of the matrix $Z^{[2]}$.

Calculating this by hand, we get

$$
\begin{aligned}
Z^{[2](3)} &= \begin{bmatrix} z_1^{[2](3)} \\ z_2^{[2](3)} \\ z_3^{[2](3)} \end{bmatrix} \\
&= \begin{bmatrix} w_{11}a_{13} + w_{12}a_{23} + b_1 \\ w_{21}a_{13} + w_{22}a_{23} + b_2 \\ w_{31}a_{13} + w_{32}a_{23} + b_3 \end{bmatrix}
\end{aligned}
$$

This is the same as the 3^{rd} column of the matrix $Z^{[2]}$.

This completes the forward propagation.

1.2 Backward Propagation

General equation of backpropagation is

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W^{[l]}} &= \delta^{[l]} \odot \mathbf{A}^{[l-1]T} \\
\frac{\partial \mathcal{L}}{\partial b^{[l]}} &= \sum_{i=1}^{m} \delta^{[l](i)}
\end{aligned}
$$

where

$$
\delta^{[l]} = (W^{[l+1]T} \odot \delta^{[l+1]}) g^{[l]'}(Z^{[l]})
$$

where $g^{[l]'}(Z^{[l]})$ is the derivative of the activation function of the l^{th} layer and $\odot$ is the dot product.

The value of $\delta^{[l]}$ is calculated using the value of $\delta^{[l+1]}$ which is calculated using the value of $\delta^{[l+2]}$ and so on. This is called the backpropagation. The value of $\delta^{[L]}$ can be calculated using the following equation

$$
\delta^{[L]} = \frac{1}{m} \frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[L]}} g^{[L]'}(Z^{[L]})
$$

where $\frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[L]}}$ is the derivative of the cost function with respect to the output of the L^{th} layer.

1.2.1 Deriving the Backpropagation Equations

The equations can be derived by using chain rule. Let's define a neural network with 3 layers. The forward propagation is defined as

$$
\begin{aligned}
A^{[0]} &= X \\
Z^{[1]} &= W^{[1]} A^{[0]} + \mathbf{b}^{[\mathbf{1}]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]} A^{[1]} + \mathbf{b}^{[\mathbf{2}]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
Z^{[3]} &= W^{[3]} A^{[2]} + \mathbf{b}^{[\mathbf{3}]} \\
A^{[3]} &= g^{[3]}(Z^{[3]})
\end{aligned}
$$

where $g^{[1]}$, $g^{[2]}$ and $g^{[3]}$ are the activation functions of the first, second and third layers respectively. We will use mean squared error as the cost function. This gives

$$\mathcal{J} = \frac{1}{2m} \sum \left(A^{[3]} - Y \right)^2$$

Now, differentaiation at the last layer is:

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = \frac{\partial \mathcal{J}}{\partial A^{[3]}} \frac{\partial A^{[3]}}{\partial Z^{[3]}} \frac{\partial Z^{[3]}}{\partial W^{[3]}}$$

For the last layer, we can solve this:

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial A^{[3]}} &= \frac{\partial}{\partial A^{[3]}} \frac{1}{2m} \sum \left(A^{[3]} - Y \right)^2 \\ &= \frac{1}{m} \left(A^{[3]} - Y \right) \end{aligned}$$

$$\begin{aligned} \frac{\partial A^{[3]}}{\partial Z^{[3]}} &= \frac{\partial}{\partial Z^{[3]}} g^{[3]}(Z^{[3]}) \\ &= g^{[3]'}(Z^{[3]}) \end{aligned}$$

and

$$\begin{aligned} \frac{\partial Z^{[3]}}{\partial W^{[3]}} &= \frac{\partial}{\partial W^{[3]}} \left(W^{[3]} A^{[2]} + \mathbf{b^{[3]}} \right) \\ &= A^{[2]} \end{aligned}$$

Putting all these together, we get

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = \frac{1}{m} \left(A^{[3]} - Y \right) g^{[3]'}(Z^{[3]}) A^{[2]T}$$

The first two terms are the same as the equation for $\delta^{[3]}$. Next, we will calculate the derivative of the cost function with respect to the bias of the third layer. This is the same as above except that we replace $W^{[3]}$ with $\mathbf{b^{[3]}}$. The final equation is

$$\frac{\partial \mathcal{J}}{\partial \mathbf{b^{[3]}}} = \frac{1}{m} \sum \left(A^{[3]} - Y \right) g^{[3]'}(Z^{[3]})$$

Next, let's proceed to the second layer. The equation is

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = \frac{\partial \mathcal{J}}{\partial A^{[3]}} \frac{\partial A^{[3]}}{\partial Z^{[3]}} \frac{\partial Z^{[3]}}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial W^{[2]}}$$

The first two terms are the same as the equation for $\delta^{[3]}$. Using this and rewriting gives

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = W^{[3]T} \odot \delta^{[3]} g^{[2]'}(Z^{[2]}) \odot A^{[1]T}$$

Writing in termes of the notation of $\delta^{[l]}$ gives

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = \delta^{[2]} \odot A^{[1]T}$$

with

$$\delta^{[2]} = W^{[3]T} \odot \delta^{[3]} g^{[2]'}(Z^{[2]})$$

We see that we are following the general formula given. Let's calculate the derivative of the first layer. The equation is

$$\frac{\partial \mathcal{J}}{\partial W^{[1]}} = \frac{\partial \mathcal{J}}{\partial A^{[3]}} \frac{\partial A^{[3]}}{\partial Z^{[3]}} \frac{\partial Z^{[3]}}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial W^{[1]}}$$

This can be rewritten as

$$\frac{\partial \mathcal{J}}{\partial W^{[1]}} = W^{[2]T} \odot \delta^{[2]} g^{[1]'}(Z^{[1]}) \odot A^{[0]T}$$

This is indeed what we needed.

1.2.2 Backward Propagation: Shapes

We'll take the same example as above, namely:

- $n_x = 10$
- $n_y = 3$
- $m = 100$
- $L = 3$
- $n^{[0]} = 10, n^{[1]} = 8, n^{[2]} = 5, n^{[3]} = 3$ (Note that $n^{[0]} = n_x$ and $n^{[L]} = n_y$)

Then, the shapes of the matrices should be:

$$\begin{aligned} W^{[1]} &\in \mathbb{R}^{8\times 10} \\ b^{[1]} &\in \mathbb{R}^{8\times 1} \\ W^{[2]} &\in \mathbb{R}^{5\times 8} \\ b^{[2]} &\in \mathbb{R}^{5\times 1} \\ W^{[3]} &\in \mathbb{R}^{3\times 5} \\ b^{[3]} &\in \mathbb{R}^{3\times 1} \end{aligned}$$

While the outputs should have the shape:

$$Z^{[1]} \in \mathbb{R}^{8\times 100}$$
$$A^{[1]} \in \mathbb{R}^{8\times 100}$$
$$Z^{[2]} \in \mathbb{R}^{5\times 100}$$
$$A^{[2]} \in \mathbb{R}^{5\times 100}$$
$$Z^{[3]} \in \mathbb{R}^{3\times 100}$$
$$A^{[3]} \in \mathbb{R}^{3\times 100}$$

Meanwhile, the derivatives should have the shape as the parameters:

$$\frac{\partial \mathcal{J}}{\partial W^{[1]}} \in \mathbb{R}^{8\times 10}$$
$$\frac{\partial \mathcal{J}}{\partial b^{[1]}} \in \mathbb{R}^{8\times 1}$$
$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} \in \mathbb{R}^{5\times 8}$$
$$\frac{\partial \mathcal{J}}{\partial b^{[2]}} \in \mathbb{R}^{5\times 1}$$
$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} \in \mathbb{R}^{3\times 5}$$
$$\frac{\partial \mathcal{J}}{\partial b^{[3]}} \in \mathbb{R}^{3\times 1}$$

Let's start from the last layer. We have

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = \frac{1}{m}\left(A^{[3]} - Y\right) g^{[3]'}(Z^{[3]}) A^{[2]T}$$

Writing the shapes:

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = (3, 100).(3, 100) \odot (100, 5) = (3, 5)$$

While

$$\frac{\partial \mathcal{J}}{\partial \mathbf{b^{[3]}}} = \sum \left(A^{[3]} - Y\right) g^{[3]'}(Z^{[3]}) \frac{\partial \mathcal{J}}{\partial \mathbf{b^{[3]}}} = \sum (3, 100) = (3, 1)$$

Now, for the second layer:

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = W^{[3]T} \odot \delta^{[3]} g^{[2]'}(Z^{[2]}) \odot A^{[1]T} \frac{\partial \mathcal{J}}{\partial W^{[2]}} = (5, 3) \odot (3, 100) \cdot (5, 100) \odot (100, 8) = (5, 8)$$

Similarly, we can show that

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}}$$

follows the shape given by the above table.

Hence, the final equation of back-propagation is

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \delta^{[l]} \mathbf{A}^{[l-1]T}$$
$$\frac{\partial \mathcal{L}}{\partial b^{[l]}} = \sum_{i=1}^{m} \delta^{[l](i)}$$
$$\delta^{[l]} = (W^{[l+1]T} \odot \delta^{[l+1]}) g^{[l]'}(Z^{[l]})$$
$$\delta^{[L]} = \frac{1}{m} \frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[L]}} g^{[L]'}(Z^{[L]})$$

2 Neural Network: Implementation

2.1 Implementation of Forward Propagation

[ ]:
```
import numpy as np
```

[ ]:
```
num_layers = 5
nx = 10
ny = 3
m = 10
nuerons = [20, 10, 10, 5, 3]
```

[ ]:
```
def layer_num_to_str(layer_num):
    return f'layer_{layer_num}'
```

[ ]:
```
def create_weights(nuerons, nx):
    weights = {}
    for i in range(len(nuerons)):
        if i == 0:
            weights[layer_num_to_str(i)] = np.random.randn(nuerons[i], nx)
        else:
            weights[layer_num_to_str(i)] = np.random.randn(nuerons[i],␣
 ↪nuerons[i-1])
    return weights

def create_biases(nuerons):
    biases = {}
    for i in range(len(nuerons)):
        biases[layer_num_to_str(i)] = np.zeros((nuerons[i], 1))
    return biases
```

[ ]:
```
weights = create_weights(nuerons, nx)
biases = create_biases(nuerons)
```

[ ]:
```
weights.keys(), biases.keys()
```

[ ]:
```
(dict_keys(['layer_0', 'layer_1', 'layer_2', 'layer_3', 'layer_4']),
 dict_keys(['layer_0', 'layer_1', 'layer_2', 'layer_3', 'layer_4']))
```

[ ]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[ ]:
```
def linear_forward(A, W, b):
    Z = np.dot(W, A) + b
    return Z
```

[ ]:
```
def linear_activation_forward(A_prev, W, b):
    Z = linear_forward(A_prev, W, b)
    A = sigmoid(Z)
    return A
```

[ ]:
```
X = np.random.randn(nx, m)

def forward_propagation(X, weights, biases):
    A = X
    for i in range(num_layers):
        A_prev = A
        A = linear_activation_forward(A_prev, weights[layer_num_to_str(i)],␣
 ↪biases[layer_num_to_str(i)])
    return A
```

`forward_propagation` is the main function that implements the forward propagation. It takes the input data `X` and the parameters `parameters` as input and returns the output `A`

[ ]:
```
output = forward_propagation(X, weights, biases)
```

[ ]:
```
assert output.shape == (ny, m)
```

[ ]:
```
output
```

[ ]:
```
array([[0.39331398, 0.38826564, 0.38970083, 0.38939174, 0.39196332,
        0.38967909, 0.3873361 , 0.38904605, 0.3931707 , 0.38847667],
       [0.51676666, 0.51432102, 0.51292546, 0.51546524, 0.52068796,
        0.51693211, 0.51944096, 0.51348509, 0.51354485, 0.51644627],
       [0.06149044, 0.05909642, 0.05895807, 0.05820592, 0.06068014,
        0.05798942, 0.06320216, 0.05879738, 0.06084828, 0.06036028]])
```