Deep Learning Notes

In this PDF, I am going to list some very basic common concepts and equations about neural networks and deep learning mentioned in the course in Coursera.

What is Structured & Unstructured Data?

Structured Data means basically databases of data. Unstructured data are usually audio, images or text. Historically, it is usually hard to extract features from unstructured data than structured data which is also one of the reasons that AI rises.

What is the downside of Sigmoid Activation Function?

The derivative of Sigmoid when x gets close to -∞ or ∞ is almost zero which makes the learning process extremely difficult when you use SGD.

How to express logistic regression?

First let's set up some notations. n_x stands for the dimension of a single input, $(x^{(i)}, y^{(i)})$ is the i_{th} sample, m stands for the total number of samples. Here we combine every column vector to form a matrix.

$$\mathrm{X} = \begin{pmatrix} x^{(1)} & \dots & x^{(i)} \end{pmatrix}$$

$$\mathrm{Y} = \begin{pmatrix} y^{(1)} & \dots & y^{(i)} \end{pmatrix}$$

Logistic Regression is actually calculating $\hat{y} = \mathrm{P}(\mathrm{y} = 1|\mathrm{x})$. If we use w to represent the weight vector and b to represent the bias item then we have

$$\hat{y} = \sigma(w^T x + b)$$

$$\sigma(\mathrm{z}) = \frac{1}{1 + e^{-z}}$$

The sigmoid function makes sure the output is between 0 and 1.

Our goal is that given $\{(x^{(1)}, y^{(2)}), \dots, ((x^{(m)}, y^{(m)})\}$, we want $\hat{y}^{(i)} \approx y^{(i)}$.

One possible loss function is

$$\mathrm{L}(\hat{y}, \mathrm{y}) = \frac{1}{2}(\hat{y} - \mathrm{y})^2$$

This function is a non-convex function which means we may face local-minimum during our optimization process and we can't find the global minimum.

So, what do we use as our loss function?

$$\mathrm{L}(\hat{y}, \mathrm{y}) = -(y log \hat{y} + (1 + y) \log(1 - \hat{y}))$$

And we define our cost function as

$$\mathrm{J}(\mathrm{w}, \mathrm{b}) = \frac{1}{m} \sum_{i}^{m} \mathrm{L}(\hat{y}^{(i)}, y^{(i)})$$

We can know that

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1 - y}{1 - a} \; where \; a = \sigma(z)$$

$$\frac{dL}{dZ} = A - Y$$

$$\mathrm{A} = [a^{(1)} \quad \dots \quad a^{(n)}]$$

$$\mathrm{Y} = [y^{(1)} \quad \dots \quad y^{(n)}]$$

$$\mathrm{db} = \frac{1}{m} np.sum(dZ)$$

$$\mathrm{dW} = \frac{1}{m}\begin{bmatrix} X^{(1)} & \dots & X^{(m)} \end{bmatrix}\begin{bmatrix} dZ^{(1)} \\ \vdots \\ dZ^{(m)} \end{bmatrix} = \frac{1}{m}\left[X^{(1)}dZ^{(1)} + \cdots + X^{(m)}dZ^{(m)}\right]$$
$$= \frac{1}{m}XdZ^T$$

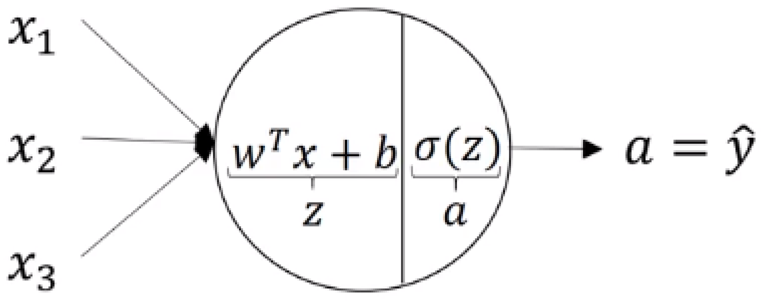
Notice that all the equations above are based on Vectorization which means we deal with matrix instead of loop over each item.

What are the broadcast rules of numpy?

There are more detailed documents on the scipy website. I summarize those rules as:

1. If the length of dimension is not the same, dimension 1 will be added to the shorter one until their length are the same. For example, if we have A = numpy.random.randn(3, 3) and B = numpy.random.randn(3) and we try to do A + B, then the dimension 1 will be added to array B and the shape of B will become (3, 1).
2. If one dimension is not the same, then the 1 dimension will be padded to a specific number. If we continue use the example above, you will see that the shape of B is still not the same as A, then the (3, 1) dimension of B will be padded to (3, 3).

How to know the shape of Weight Matrix?



This picture shows a typical neuron and w inside this neuron is of shape (3, 1) because the input is of shape (3, 1) and we all know that each cell is stacked together vertically, so the weight matrix should be of shape ($nums_{this\ layer\ neurons}, nums_{last\ layer\ neurons}$) and the weight matrix should looks like

$$W^{[i]} = \begin{bmatrix} \_w_1^{[i]T}\_ \\ \vdots \\ \_w_n^{[i]T}\_ \end{bmatrix}$$

<u>Notice that all the parameters inside the NN is stacked vertically and different samples are stacked horizontally.</u>

Tanh Function

$$\tanh(\mathrm{z}) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

The output of this function is between -1 and 1 so the performance is basically always better than sigmoid because the mean of tanh is 0 which means the output has been centralized.

This function also suffers from zero gradient when z is significantly small or large like sigmoid.

Derivative of a standard NN

$$\mathrm{g(z)} = \frac{1}{1+e^{-z}}$$

$$\frac{dg(z)}{dz} = g(z)(1-g(z))$$

or

$$\mathrm{g(z)} = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

$$\frac{dg(z)}{dz} = (1-(g(z))^2)$$

Forward Propagation

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

$$A^{[1]} = g^{[1]}\left(Z^{[1]}\right)$$

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

$$A^{[2]} = g^{[2]}\left(Z^{[2]}\right)$$

$$\text{where } g^{[2]}\text{should be sigmoid if we are doing binary classification}$$

Backpropagation

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]T}$$

$$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^{T}$$

$$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$