# Neural Network Introduction

Xingzhuo "Gesa" Chen

June 2020

## 1 Introduction

In this note, I will try to introduce a little bit about neural networks, and the mathematics behind it. Before I really start to focus on the neural networks, I shall firstly state the difference between machine learning and deep learning. Machine learning is a broad term including many aspects, everything using some data and let a machine learn to finish some task without human's experience can be included.

Machine learning can be divided into two subgroups, supervised learning and unsupervised learning. Supervised learning use data with labels, like a bunch of picture which are tagged with "cat" or "dog", or a table of house condition and house sell prices. Then, the task for our machine is to classify the cats from dogs (classification problem) or use the house condition to estimate the house sell prices (regression problem). As for the unsupervised learning, the data are not labeled, so the problem itself is more abstract and algorithms are more complicated. To begin with, you can search t-SNE (T-distribution Stochastic Neighbor Embedding) on google and find some interesting web pages (like this Interactive t-SNE). This is a handy unsupervised learning algorithm used for clustering, I guess you will learn more about it in your graduate statistics class. In the future, we may face more complicated algorithms, such as GAN (Generative Adversarial Network) or Auto-encoder, both of which are based on neural networks.

When we are talking about deep learning, we usually refer to the machine learning which use a multi-layer (deep) neural network and not really limited to supervised learning problems.

Okay, here we go.

## 2 Loss Function

When we have a classifier, we always want to introduce a number which tells how accurate that classifier is in doing classifications, so we put a bunch of data into it, wait until the classifier return results, finally compare the results with the truths. Therefore, we use a loss function to connect the predictions and the truths and calculate such a number, smaller is better.

It seems that there will be soooooooo many functions can serve as loss functions, such as precision (maybe its inverse), mean squared error, etc. However, a loss function for a neural network should also satisfy two other conditions: smooth and with derivations, the reason will be shown in Section 4, thus precision cannot be used as loss function.

For a two label classification problem, we usually use a cross-entropy loss function:

$$L = \Sigma y_{true} ln(y_{pred}) \tag{1}$$

Here, $y_{true}$ means the truth label and can only be 0 or 1, $y_{pred}$ is the predicted label also the output of neural network, it can be any number between 0-1, indicating how possible the classifier thinks such an item can be in the "1" group, $\Sigma$ is the sum over all data. Obviously, when neural network predicts everything correctly, with 0 or 1, the cross-entropy reaches its minimum zero.

# 3 To Construct A Neural Network

## 3.1 Layers

There are many pictures about neural networks online, showing many layers of neurons connected with lines. Generally speaking, there are three kinds of layers: input layer, hidden layer and output layer. Input layer is where information input into the network, output layer return the predictions which will be used in loss function, hidden layer is in between and not necessary to be one layer. Usually, we believe a neural network is smarter with more layers.

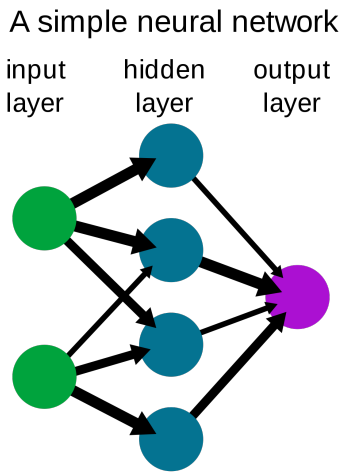


Figure 1: An example of neural network

For a typical fully connected neural network, the calculation between layers is a matrix production. Using the Figure 1 as an example, the input layer propagates a 2-dimension vector, then multiply this vector by a $2 \times 4$ matrix and plus a 4-dimension bias, we get the input for the hidden layer. The formula can be written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{2}$$

Here, the w are called "weights", b are called "bias", both are included in trainable parameters.

### 3.2 Neurons

In each neurons, an activation function will be used. There are many candidates for activation functions, such as tanh functions, elu functions, they are discussed in keras's online document. Usually, we use a rectified linear unit (ReLU) for the hidden layers, which is:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3}$$

To notice, without such an activation function in a multi-layer fully connected neural network, there will be matrix products between the input and the output layers, and these matrix products can be simplified as one matrix product, obviously we don't want such things to happen. For the input layer, we can use a linear activation.

When doing two-labeled classification with a cross-entropy loss function, the output layer activation function shall be a sigmoid function, which shows as:

$$f(x) = \frac{e^x}{e^x + 1} \tag{4}$$

Mathematically, the sigmoid function limit the output between 0 and 1, which conforms to our two-label classification problem. I once remember there is a mathematical proof which justifies that cross-entropy should be used together with sigmoid function, but I cannot find it anywhere.

## 4 Training

When training the neural networks, we try to modify the weights and bias (a.k.a trainable parameters), in order to make the loss function smaller. To notice, based on what we have discussed about neural network structure, all the calculations are analytical, which allow us to write the prediction as a function of input:

$$y_{pred} = F(x) \tag{5}$$

Where x is the input information, F is the neural network with matrix product - activation - matrix product ... So, we can try to write the derivation of the loss function:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y_{pred}} \frac{\partial y_{pred}}{\partial w} \tag{6}$$

The process of calculating the gradient of each trainable parameters layer by layer from the output to the input layer is called back propagation, the process of calculating the loss function begin from the input layer is called forward propagation. Then, when we try to update the trainable parameters with this formula:

$$w_{new} = w_{old} - \frac{\partial L}{\partial w} \times \alpha \tag{7}$$

Where $\alpha > 0$, we are very likely to see the loss function becomes smaller. The $\alpha$ here is called learning rate, we usually use a relatively small value for it, like 0.01 or 0.0001. After several iterations, the loss function will reach a minimum, and the trainable parameters won't change much, then we can stop the training and use the network for prediction. The as-described algorithm is called stochastic gradient descent. There are many other optimization algorithms available in keras, like rmsprop and adam, and all these optimizers are based on the gradient. They may also consider what was the direction of last step, how large was the last step and other information to determine the next step, and their relating papers say they can somehow let the loss function jump out of a local minimum and find a better minimum.