Mathematical Model of Neural Network

© Hanfeng Zhai

School of Mechanics and Engineering Science, Shanghai Univeristy
Shanghai 200444, PRC

Abstract

Machine learning (ML) has attracted great attentions in recent years due to their satisfying results and robust functions. With the developing technologies, machine learning is widely applied on data mining, facial recognition, and engineering disciplines. Neural network (NN) is among the most important algorithms in ML. Here, we present basic model and equations for neural network to show how we applied NN on real world problems.

Neuron

The neural network is a multi-nonlinear regression model inspired from the information transforming process existed in neurons. The output data is obatined through neurons from the input data. Here, in figure 1, a schematic indicate how the information is conveyed through the neuron(s). From figure 1 we observe that the information is transmitted through an activation function σ acting on the linear model of the input data. The linear model consists of weights and thresholds with multiple inputs acting on a single neuron. Here, we first introduce how the data is transmitted through the linear model.

Linear Regression

For neural network, the basic neural model is fitted by the multiple linear regression model:

$$y_w(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n + b \tag{1}$$

Where w_n is the weight and b is the threshold.

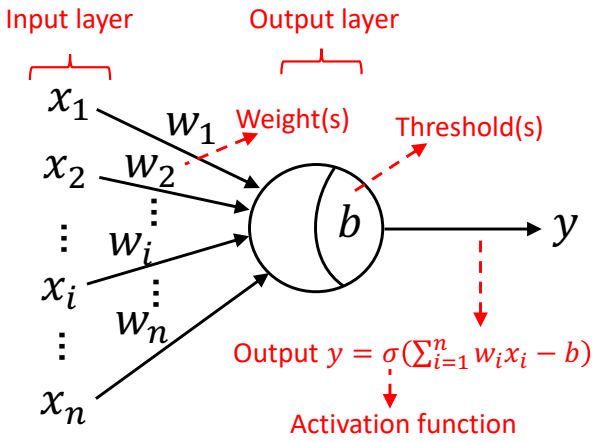

Figure 1: The schematic for a single neurons for neural network.

Based on the regression model, which the connecting neurons is activated by the function to convey the information.

$$\hat{y}(x) = \sigma(w^T \mathbf{x} + b) \tag{2}$$

In which $\hat{y}_i = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_n)$ are the prediction values; $\sigma = \sigma(z)$ is the activation function;

Activation Function

In the information convey process as it is shown in figure 1, we need an activation function to convey the data from input to output. In actual applications, any function values from 0 to 1 in range [-1, 1] can be adopted as an activation function. Here, we introduce three most widely used activation functions, i.e. the Sigmoid function, the ReLU function and the pure linear function.

$$\sigma_{sigmoid}(z) = \frac{1}{1+e^{-z}}; \quad \sigma_{ReLU}(z) = max(0, z); \quad \sigma_{linear}(z) = z \tag{3}$$

In actual applications such as running on Python or MATLAB, the weight(s) are randomly generated. But the accurate prediction values requires a weight that fits the input data. Hence, we introduce the concept of learning rate l_r

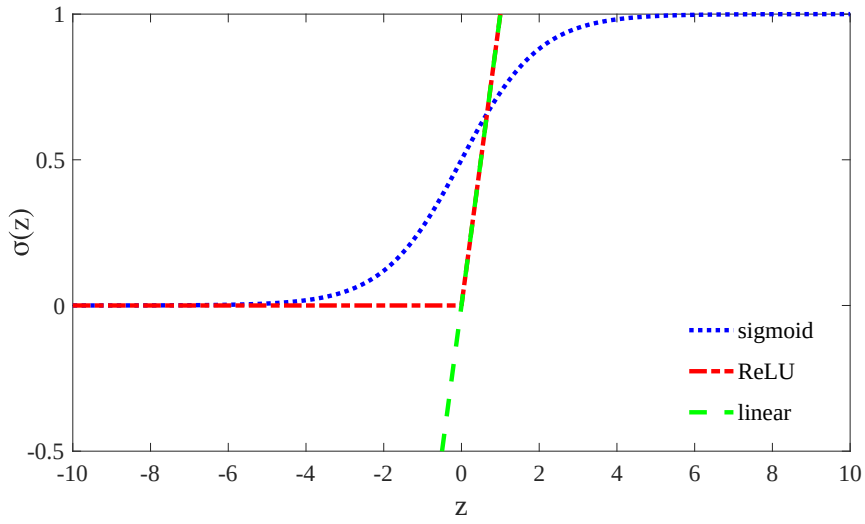

Figure 2: Diagrams for the three most applied activation functions.

to show how to obtain the best weight. the regression modle is adjusted in the way:

$$w_i \Leftarrow w_i + \Delta w_i$$

$$\Delta w_i = l_r(y - \hat{y})x_i \tag{4}$$

Based on equation 4, we know that we need to adopt a specific value of l_r for update the weight to adjust the regression model.

Another question there fore arises, *How do we update the term Δw in the calculation process?* Here, we introduce an algorithm most adopted for NN called the back-propagation algorithm, which means the errors in process are conveyed backwards to the input layer for a loop.

With previously introduced the output values (predictions) are $\hat{y}_i = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_n)$, we can write the predictions in the form:

$$\hat{y_{n_j}} = \sigma(\beta_j - b_j) \tag{5}$$

Where β_j is the input in the i^{th} neurons and b_j is the threshold.

The mean suqare error MSE generated from the transmition of neurons takes the form:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 \tag{6}$$

Now the goal is clear: minimize the term MSE to update the weight w.

To achieve such goal, we need an **optimizer** function. Here, we adopt the most commonly used *Gradient Descent* method as an example. Now, let us assume the weight connecting the h^{th} layer to the j^{th} layers takes the form:

$$\Delta w_{hj} = -l_r \frac{\partial MSE}{\partial w_{hj}} \tag{7}$$

The partial derivatives involved in equation 7 can be written as:

$$\frac{\partial MSE}{\partial w_{hj}} = \frac{\partial MSE}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \beta_j}\frac{\partial \beta_j}{\partial w_{hj}} \tag{8}$$

With the definition of β, we have

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h \tag{9}$$

Based on equations 5 and 6, we define g_j as:

$$\begin{aligned} g_j &= -\frac{\partial MSE}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial \beta_j} \\ &= -(\hat{y}_j - y_j)\dot{\sigma}(\beta_j - b_j) \\ &= \hat{y}_j(1 - \hat{y}_j)(y_j - \hat{y}_j) \end{aligned} \tag{10}$$

Therefore we obtain the update equation for weight w_j through layer h to layer j in the back-propagation algorithm:

$$\Delta w_{hj} = l_r g_j b_h \tag{11}$$

Although we adopt the gradient descent as an example in the back-propagation process, this does not mean that the gradient descent is the best algorithm for optimization.

In the next section we will introduce basic knowledge about the optimizer function.

Optimizer

In the calculation epochs, we introduce the cost function:

$$J(w) = \frac{1}{N}\sum_{i}^{N} Cost(y_w(x_i), y_i) \tag{12}$$

In which $Cost(y_w(x_i), y_i) = \begin{cases} -\log_{10}(y_w(x)), & \text{if } y = 1 \\ -\log_{10}(1 - y_w(x)), & \text{if } y = 0 \end{cases}$.

Based on the cost function $J(w)$ as given in equation 12, we introduce the optimizer, as to achieve our training goal. we first give the gradient descent algorithm, which is a general minimization method which updates parameter values in the "downhill" direction: the direction opposite to the gradient of the objective function [3]. The update parameter $\mathbf{h}$ in step i, that moves the parameters in the direction of steepest descent is given by:

$$\mathbf{h}_{\text{gd}} = l_r J(w) w_i (y_i - \hat{y}_i) \tag{13}$$

In which l_r is the learning rate, which determines the length of the step in the steepest-descent direction.

We simultaneously gives another method for minimizing a sum-of-squares objective function, the Gauss-Newton method [3], in which the resulting normal equations for the Gauss-Newton update are:

$$\left[J(w)^T w_i J(w)\right] \mathbf{h}_{\text{gn}} = J(w)^T w_i (y_i - \hat{y}_i) \tag{14}$$

Based on the equations 13 to 14, we obtain the Levenberg-Marquardt method, which adaptively varies the parameter updates between the gradient descent update and the Gauss-Newton update:

$$\left[J(w)^T w_i J(w) + \mathbf{I}\lambda\right] \mathbf{h}_{\text{lm}} = J(w)^T w_i (y_i - \hat{y}_i) \tag{15}$$

Networks

With the knowledge of how neurons transform the input data into outputs, we can thence move further into how we form a neural network structure consisting of neurons.

Here, in figure 3, we present a classic full connected neural network. The input layer with data $\mathbf{x} = (x_1, x_2, ..., x_i, ..., x_n)$ are the input variables, which

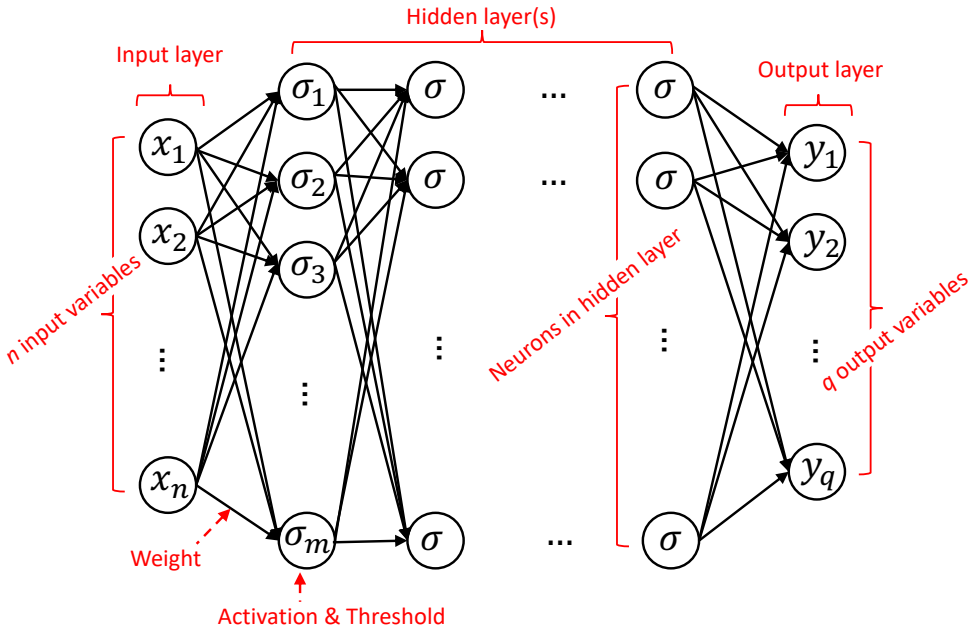

Figure 3: The basic structure for neural network.

are further transmitted into the hidden layer through the neuron model we introduced as shown through the arrays. With multi-hidden layers the initial input datasets are transformed through the activation σ on the linaer model for multiple times. Then the last hidden layer with transmit the data to the output layer with the predictions of output data of q output variables $\hat{y}_i = (y_1, y_2, \ldots, y_q)$.

Example

For better understanding, here, we will provide an example to visualize how neural network works. Provided that a class has many students preparing for a sport constest consisting running, weight lifting, jumping, and push-ups. Here we need to use the neural network to predict how well they will perform on the sport contest. In this problem we will detect their food ingest (pounds per day), sleeping hours, and trainings (work out hours per day).

With the provided informations, we deduce that there are 3 input data

$$\mathbf{x} = (x_{food}[lbs/day],\ x_{sleep}[hrs],\ x_{train}[hrs/day])$$

and 4 output data

$$\mathbf{y} = (y_{run}[m/sec],\, y_{weight}[kg],\, y_{jump}[m],\, y_{push}[times])$$

We deduce that these datasets all has different units, which are troublesome for computations. Thence, we need to nondimensionalize the data preceding to input the data. The nondimensionalized variables x' obeys

$$x' = \frac{x}{x_{max}} \tag{16}$$

The data can be written in the form:

Input layer	Hidden layer(s)	Output layer
x_{food}	x_σ	y_{run}
x_{sleep}	...	y_{weight}
x_{train}	x_σ	y_{jump}
		y_{push}

Table 1: Parameter setting in simulation and experiment.

In actual trainings, for supervised trainings, we set training sets and testing sets. For big datasets we sometimes set validation sets. The training process can be summariezed as train the training sets on the NN and verify the network with the testing sets. For validation, MSE can be one parameter to verify the accuracy. However, we usually adopt R^2 for verification:

$$R^2 = 1 - \frac{(\sum_i^n y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \hat{y}_i)^2} \tag{17}$$

These are the basic information for neural network. More details and other knowledge available at http://hanfengzhai.net/categories/note.

References

[1] Z.-H. Zhou (**2016**). Machine Learning. *THU Press*, ISBN 978-7-302-42328-7.

[2] A.Y. Ng (**2016**). Machine Learning. *Coursera*.

[3] H.P. Gavin (**2019**). The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems. *Matlab Tutorials*, Duke University.