Neural Network

Luu Duc Quy

July 2024

1 Introduction

What is neural network? We begin with a general idea of what neural networks are and why you might be interested in them. Neural networks, also called Artificial Neural Networks (though it seems, in recent years, we've dropped the "artificial" part), are a type of machine learning often conflated with deep learning. The defining characteristic of a deep neural network is having two or more hidden layers — a concept that will be explained shortly, but these hidden layers are ones that the neural network controls. It's reasonably safe to say that most neural networks in use are a form of deep learning.

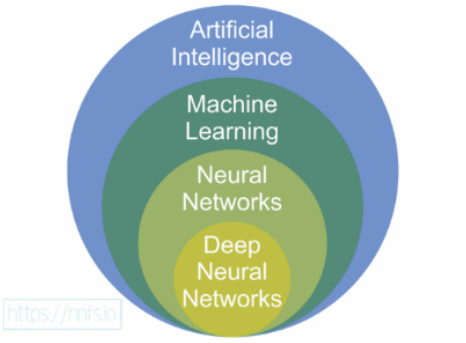

Figure 1: Overall demonstration

2 Input layer

For the neural network to start the computation, the first thing we need to consider about is the in put layer. Consider a picture with 3×3 pixels as an example. (Table 2)

a1	a2	a3
a4	a5	a6
a7	a8	a9

Table 1: 3×3 pixels picture

To become a valid input, we have to reshape it become a 9×1 vector. (Table)

a1
a2
a3
a4
a5
a6
a7
a8
a9

Table 2: Reshape to 9×1

After reshaping, it is ready to be used as an input layer.

3 Activation Function

1. **Neuron**:
   Everything start with a simple neuron. Neuron is the smallest scale in a neural network. A single neuron by itself is relatively useless, but, when combined with hundreds or thousands (or many more) of other neurons, the inter connectivity produces relationships and results that frequently outperform any other machine learning methods.

2. **Weight**:
   Each connection between neurons has a weight associated with it, which is a trainable factor of how much of this input to use, and this weight gets multiplied by the input value. These weights are assigned randomly on the first epoch and then will be optimized later by appropriate methods.

3. **Bias**:
   Once all of the inputs·weights flow into our neuron, they are summed, and a bias, another trainable parameter, is added. The purpose of the bias

is to offset the output positively or negatively, which can further help us map more real-world types of dynamic data.

4. **Formula**:

$$N = \sigma(\sum_{i=1}^{n} a_i w_i + \beta) \tag{1}$$

The formula above is used to calculate the sum of previous to n neurons to find the value of 1 neuron in the next layer. With a is the activation value of each neuron in the previous layer. w is the weight of each neuron in the calculation. β is the bias value assign after we get the sum of weights and activation value. σ is the value to normalize the final result into the form of value between 0 and 1.
One equation can be provided as example is the sigmoid formular:

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

Another popular method is the application of ReLU: The purpose of this method is to make every activation value that less than 0 to become 0. This will make the process become a non-linear trend and prevent negative value for later derivative process.

This process will be repeated for every neurons in the hidden layer and these values will be use as activation value to calculate the next layer.

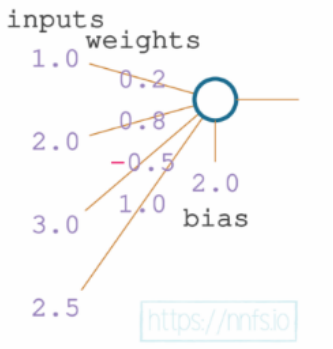

Figure 2: Activation

4 Layers

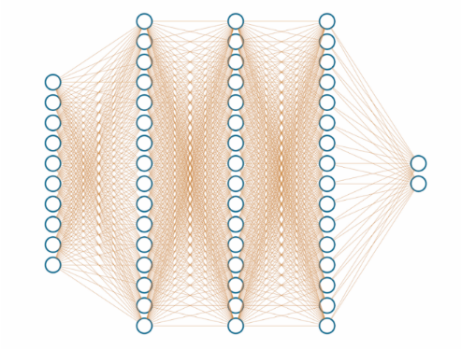

Figure 3: Example of 3 hidden layers with 16 neurons each

Neural networks typically have layers that consist of more than one neuron. Layers are nothing more than groups of neurons. Each neuron in a layer takes exactly the same input — the input given to the layer (which can be either the training data or the output from the previous layer), but contains its own set of weights and its own bias, producing its own unique output. The layer's output is a set of each of these outputs — one per each neuron. (Figure 4)

5 Cost-Loss Function

After we get the value in each neuron of the last layer (output layer), we use the Softmax method to find the percentages of each value. The softmax function for a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is given by:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{for} \quad i = 1, \ldots, n$$

After getting the percentage in each neuron, we can tell the likelihood of each output. The next step is to One-hot Encode the real value so we can compare the differences between each output and the real value. Consider the output to be a binary. After One-hot Encode, it will be labeled as: $\alpha = \begin{vmatrix} 0 \\ 1 \end{vmatrix}$

However, because the weight values were decided randomly, the output of the first run will not have good accuracy and we have to improve it by a method called the Loss Function.

1. Formula
   The Loss function is given by:

$$\mathrm{L} = \sum_{i=1}^{n} (x_i - \alpha_i)^2$$

2. Purposes
   The main purpose of calculating the Loss Function is to minimize the Cost and improve the accuracy of this network. There are several ways to minimize the Cost,

6 Gradient Descent

To manipulate the Loss Function, there are several parameters that can be changed to find the minimum outcome. Those parameters are α for activation value of the previous layer, β for bias, and w for the weight. To find to min value of this function, we need to find the slope of or **Gradient** of the function. One popular method for this is to calculate the derivative.

$$\frac{\Delta y}{\Delta x}$$

The gradient is a vector composed of all of the partial derivatives of a function, calculated with respect to each input variable.

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{\partial}{\partial_x} f(x,y,z) \\ \frac{\partial}{\partial_y} f(x,y,z) \\ \frac{\partial}{\partial_z} f(x,y,z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial_x} \\ \frac{\partial}{\partial_y} \\ \frac{\partial}{\partial_z} \end{bmatrix} f(x,y,z)$$

We'll be using derivatives of single-parameter functions and gradients of multivariate functions to perform gradient descent using the chain rule, or, in other words, to perform the backward pass, which is a part of the model training.

What is Backpropagation? After finding the derivative of the Loss function, the next step is to decide either moving ahead or moving back depends on the current position of the Loss value. A complete cycle start from activation layer until we find the Loss value (forward). Then we find the derivative of that function to optimize the w and β of the previous layer (backward). Finally, we update the Loss value. The complete loop is call an epoch. This process will end when we find the min value of function, or the derivative of Loss function equal to 0.

There are several methods to apply Gradient Descent

1. **Batch**: Optimize bias and weight after the whole process is finished.
2. **Stochastic**: Optimize bias and weight after every activation step
3. **Mini-Batch**: Divide the network to small batch and optimize loss value for each batch.

	Batch	Stochastic	Mini-Batch
Effective	High	Low	Good
Process time	Long	Fast	Good

Table 3: Performance of each method.

7 Learning Rate

Deciding learning rate is a very significant step during the Backpropagation step. Having a appropriate learning rate can increase the accuracy of the model. To explain this phenomenon, if the learning rate is too small, it can not escape the local minimum. On the other hand, having a too huge learning rate may help finding the global minimum but bound out. Observing Loss rate can help deciding which one is the best learning rate.

Figure 4: Learning rate and Loss relationship.