# Basic Neural Networks

## Introduction

*Neural networks* were originally designed as a way to mimic the operation of neurons in the brain. The idea is that we form a graph with some nodes corresponding to inputs and other nodes corresponding to outputs, and with some extra nodes between them. These extra nodes are organized into layers, and the output of each layer is the input to the next layer. Each node is meant to play the role of a neuron in the brain. The neural network is also known as the *multilayer perceptron*, or *MLP*.

The idea is that activations in one layer determine the activations in the next layer, similarly to the behavior of neurons. In this way, the original data, which is fed into the input layer, activates nodes in the successive layers, ending with the nodes in the output layer. In each layer of a typical neural network, logistic regressions are performed. One could view neural networks as successive logistic regressions taking place at each node and the resulting model comprises multiple layers of logistic regression models.

## Set-ups

Assume that we have categories $\mathcal{C}_k$ for $k = 1, \dots, K$, and suppose that we want to classify an input into one of the categories $\mathcal{C}_k$. Then we can choose our network to have $n$ output nodes, and each output node will produce probability for the input to be in the corresponding category.

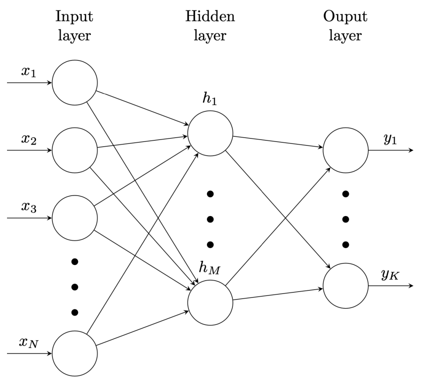

Figure 1: Neural Network

$$\begin{aligned}
\text{input:}\quad & x_1, x_2 \ldots, x_N \\
\text{hidden layer:}\quad & h_i = \sigma(w_{i,1}^{(1)} x_1 + \cdots + w_{i,N}^{(1)} x_N + b_i^{(1)}), \quad i = 1, 2, \ldots, M \\
\text{output:}\quad & y_i = \sigma(w_{i,1}^{(2)} h_1 + \cdots + w_{i,M}^{(2)} h_M + b_i^{(2)}), \quad i = 1, 2, \ldots, K \\
\text{actual class } \mathcal{C}_k\text{:}\quad & \hat{y}_k = 1 \quad \text{and} \quad \hat{y}_i = 0 \quad \text{for } i \neq k
\end{aligned}$$

Here the parameters $w_{i,j}^{(\ell)}$ are called *weights* and $b_i^{(\ell)}$ *biases* for $\ell = 1, 2$. Initially, we might make random choices for the parameters $w_{i,j}^{(\ell)}$ and $b_i^{(\ell)}$ at each node. Using the training data, we want to learn the best values of these parameters.

In a matrix form, write $\boldsymbol{x} = [x_i]^T = [x_1, x_2, \ldots, x_N]^T$, $\boldsymbol{y} = [y_i]^T = [y_1, y_2, \ldots, y_K]^T$, $\boldsymbol{h} = [h_i]^T = [h_1, h_2, \ldots, h_M]^T$. Similarly,

$$\mathbf{b}^{(\ell)} = [b_i^{(\ell)}]^T \quad \text{and} \quad \boldsymbol{w}^{(\ell)} = [w_{i,j}^{(\ell)}].$$

Then we have

$$\boldsymbol{h} = \sigma(\mathbf{w}^{(1)} \boldsymbol{x} + \mathbf{b}^{(1)}) \quad \text{and} \quad \boldsymbol{y} = \sigma(\mathbf{w}^{(2)} \boldsymbol{h} + \mathbf{b}^{(2)}).$$

This process is considered as *forward propagation.*

We can develop more general network mappings by considering more complex network diagrams with many layers. However, these must be restricted to a feed-forward architecture, in other words to one having no closed directed cycles, to ensure that the outputs are deterministic functions of the inputs. In other words, a network must be an *acyclic quiver.*

## Backpropagation

We need to measure how far the results are from the correct classes, and update the weights and biases to improve the classification. We will do this with gradient descent after choosing a cost function. For simplicity in notations, write

$$z_i^{(1)} = \left( \sum_{k=1}^{N} w_{i,k}^{(1)} x_k \right) + b_i^{(1)} \quad \text{and} \quad z_i^{(2)} = \left( \sum_{k=1}^{M} w_{i,k}^{(2)} h_k \right) + b_i^{(2)}.$$

Then we have

$$h_i = \sigma(z_i^{(1)}) \quad \text{and} \quad y_i = \sigma(z_i^{(2)}).$$

The cost function for a particular input $\boldsymbol{x}$ can be taken to be

$$C_{\boldsymbol{x}} = \sum_{i=1}^{K} (y_i - \hat{y}_i)^2,$$

where $\hat{y}_k = 1$ and $\hat{y}_i = 0$, $i \neq k$, for actual class $\mathcal{C}_k$. As in logistic regression, we may also take

$$C_{\boldsymbol{x}} = -\sum_{i=1}^{K} \hat{y}_i \ln y_i.$$

The total cost to be minimized is given by $\sum_{\boldsymbol{x}\in\mathcal{T}} C_{\boldsymbol{x}}$ where $\mathcal{T}$ is a training set. For simplicity, we write $C = C_{\boldsymbol{x}}$ in what follows.

We need to take the partial derivatives of the cost function $C$ with respect to all the weights and biases. This process of taking the partial derivatives is called *backpropagation*. Define

$$\delta_j^{(\ell)} = \frac{\partial C}{\partial z_j^{(\ell)}}$$

This can be interpreted as an error. (See [Bishop, p.243])

From the chain rule,

$$\delta_j^{(2)} = \frac{\partial C}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j^{(2)}} = \frac{\partial C}{\partial y_j} \cdot \sigma'(z_j^{(2)}),$$

where $\frac{\partial C}{\partial y_j}$ can be calculated from the formula of $C$ and $z_j^{(2)}$ has been calculated before backpropagation. Using the chain rule again, we obtain

$$\begin{aligned} \delta_j^{(1)} &= \frac{\partial C}{\partial h_j} \cdot \frac{\partial h_j}{\partial z_j^{(1)}} = \frac{\partial C}{\partial h_j} \cdot \sigma'(z_j^{(1)}) \\ &= \sum_{i=1}^{K} \frac{\partial C}{\partial z_i^{(2)}} \cdot \frac{\partial z_i^{(2)}}{\partial h_j} \cdot \sigma'(z_j^{(1)}) = \sum_{i=1}^{K} \delta_i^{(2)} w_{i,j}^{(2)} \sigma'(z_j^{(1)}). \end{aligned}$$

The formula

$$\boxed{\delta_j^{(1)} = \sum_{i=1}^{K} \delta_i^{(2)} w_{i,j}^{(2)} \sigma'(z_j^{(1)})}$$

is usually referred to as *backpropagation formula*.

Now we have

$$\frac{\partial C}{\partial w_{i,j}^{(1)}} = \frac{\partial C}{\partial z_i^{(1)}} \cdot \frac{\partial z_i^{(1)}}{\partial w_{i,j}^{(1)}} = \delta_i^{(1)} x_j,$$

and similarly,

$$\frac{\partial C}{\partial w_{i,j}^{(2)}} = \delta_i^{(2)} h_j \quad \text{and} \quad \frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \quad \text{for} \quad \ell = 1, 2.$$

Using these partial derivatives, we can apply gradient descent.

## Example–MNIST hand-written digits

The MNIST (Modified National Institute of Standards and Technology) database is a large database of handwritten digits that is commonly used for training various image processing systems. We design a neural network to classify the MNIST handwritten digits.

The input layer will contain 784 nodes, with each node corresponding to a pixel, and the input value would be between 0 (white) and 1 (black). After passing through the network, we want the output layer to tell us what the computer thinks the digit is. We therefore have 10 nodes in the output layer, corresponding to the digits 0 to 9. We would classify an image by selecting the output with the largest probability (this is *softmax* classification).

The standard softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \quad i = 1, \dots, K, \quad \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K.$$

The sigmoid function is a special case of softmax function.

## Miscellanies

### Convolutional Neural Networks (CNN)

An important property of images is that nearby pixels are more strongly correlated than more distant pixels, and recognizing local features can enhance the performance of a classifier. The **convolutional neural network** is a neural network with *local receptive fields* in a hidden layer where the weights are shared across the layer. Consequently, the number of parameters is reduced and the resulting network has the translation invariance property. The reason behind parameter-sharing is that any useful features that are detected in some portion of the input may be valid in other parts.

### Deep Learning

**Deep learning** uses multiple layers to progressively extract higher-level features from the raw input to imitate how the human brain works. Most deep learning models are based on convolutional neural networks.

### TensorFlow and Keras

**TensorFlow** is an open-source software library for machine learning. It has a particular focus on training and inference of deep neural networks.

**Keras** is an open-source library that provides a Python interface for neural networks. It acts as an interface for the TensorFlow library.