CSE422: Artificial Intelligence
Artificial Neural Networks

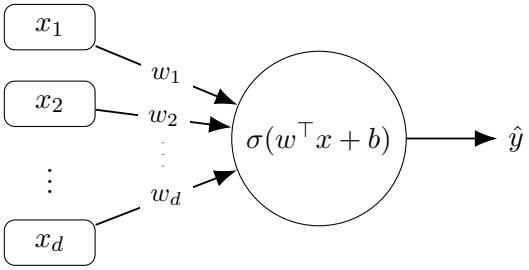
Rafiad Sadat Shahir

An Artificial Neural Network (ANN) is a machine learning model, inspired by biological neural networks, that uses interconnected nodes (artificial neurons) to process information and learn from data.

1 Neural Network with One Neuron

1.1 Logistic Regression as a Connectionist System

Consider the following connectionist system that represents the logistic regression:



Here, the hypothesis is defined as $h_w : x \to y$ and is expressed as follows:

$$h_w(x) = \sigma(w^\top x + b)$$

In the context of neural networks, a one-dimensional non-linear function that maps $\mathbb{R} \to \mathbb{R}$, such as the sigmoid function, is often referred to as an activation function. The model $h_\theta(x)$ is said to have a single neuron partly because it has a single non-linear activation function. Moreover, the term b is often referred to as bias, and the vector w is referred to as the weight vector. Such a neural network has 2 layers, i.e., input layer and output layer.

1.2 Perceptrons

Perceptrons, introduced by Frank Rosenblatt in the late 1950s, are the first trainable artificial neural networks. In contrast to the logistic regression, the perceptrons uses a threshold function as the activation function.

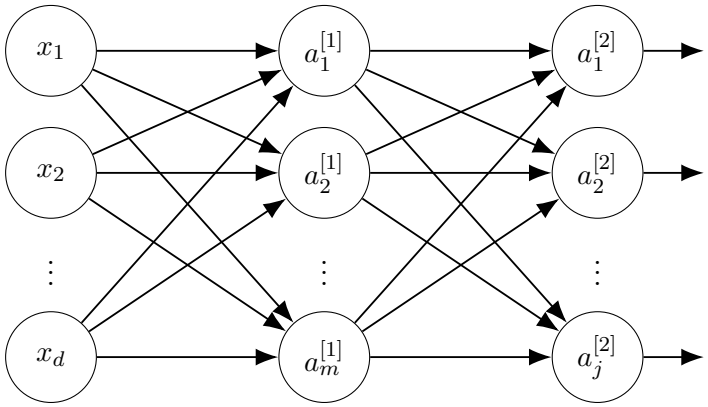
1.3 Limitations of Perceptron

One of the most fundamental limitations of perceptrons is their inability to solve problems that are not linearly separable, i.e. if the data points from different classes cannot be

separated by a hyperplane, the perceptron fails to classify them correctly. The XOR problem, where the output is '1' only when exactly one of the inputs is '1', is a classic case that a perceptron cannot solve. In contrast, the AND and OR problems can be handled by a perceptron.

2 Multilayer Perceptron

In a more complex neural network, the single neurons may be stacked together so that the output of one neuron is passed as the input to the next neuron. Such a network is called a multilayer perceptron(MLP). The diagram of an MLP is as follows:



Formally, the input to a neural network is a set of input features $\{x_1, x_2, \ldots, x_d\}$. The intermediate variables denoted by $a_1^{[1]}, a_2^{[1]}, \ldots, a_m^{[1]}$ are often referred to as hidden neurons. The output is represented by a vector containing the values $a_1^{[2]}, a_2^{[2]}, , \ldots, a_j^{[2]}$. The network is called a fully-connected neural network as all hidden neurons a_i depend on all inputs x_i.

2.1 Forward Propagation

In general, the expressions for neural networks are simplified with tensor notation. An important motivation for that is to leverage parallelism in GPUs.
For the first layer, a weight matrix $W^{[1]} \in \mathbb{R}^{m \times d}$ is defined as the concatenation of all vectors $w_j^{[1]}$:

$$W^{[1]} = \begin{bmatrix} —w_1^{[1]\top}— \\ —w_2^{[1]\top}— \\ \vdots \\ —w_m^{[1]\top}— \end{bmatrix}$$

We can write $z \in \mathbb{R}^m$ as:

$$z^{[1]} = \underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_m^{[1]} \end{bmatrix}}_{z \in \mathbb{R}^{m \times 1}} = \underbrace{\begin{bmatrix} — {w_1^{[1]}}^\top — \\ — {w_2^{[1]}}^\top — \\ \vdots \\ — {w_m^{[1]}}^\top — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m \times 1}}$$

In essence,

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Computing the activations $a^{[1]} \in \mathbb{R}^m$ from $z^{[1]} \in \mathbb{R}^m$ involves an element-wise non-linear application of the activation function. Thus, we have,

$$a^{[1]} = f^{[1]}(z^{[1]})$$

where, f denotes the activation function. The process can be generalized for the l^{th} layer as follows:

$$a^{[l]} = f^{[l]}(W^{[l]}a^{[l-1]} + b^{[l]})$$

The process of inference in neural networks is generally known as forward propagation.

2.2 Activation Functions

The activation function is generally a non-linear function $f(z)$ that maps $\mathbb{R} \to \mathbb{R}$ such as:

$$f(z) = \frac{1}{1 + e^{-z}} \quad \text{(sigmoid)}$$

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \text{(tanh)}$$

$$f(z) = \max(z, 0) \quad \text{(ReLU)}$$

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad \text{(Softmax)}$$

Usually, the identity function for is not used as the activation function. For simplicity,

assume that $b^{[l]} = 0$. Consider the following for a network with L layers:

$$\begin{aligned} h_W(x) &= W^{[L]} a^{[L-1]} \\ &= W^{[L]} W^{[L-1]} a^{[L-2]} \\ &= W^{[L]} W^{[L-1]} \dots W^{[1]} x \\ &= \tilde{W} x \end{aligned}$$

As applying a linear function to another linear function will result in a linear function over the original input, without non-linear activation functions, the neural network will simply perform linear regression.

2.3 Backpropagation

One way of training the network is to use gradient descent. For that, we need to calculate the derivative of the loss function with respect to the weight matrix. Using the chain rule, the gradient $\frac{\partial J}{\partial W^{[L]}}$ for the last layer L^{th} is calculated as follows:

$$\frac{\partial J}{\partial W^{[L]}} = \frac{\partial J}{\partial a^{[L]}} \cdot \frac{\partial a^{[L]}}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial W^{[L]}}$$

The value of $\frac{\partial J}{\partial a^{[L]}}$ and $\frac{\partial a^{[L]}}{\partial z^{[L]}}$ depends on the choice of the loss function and the activation function. For square loss and sigmoid function, $\frac{\partial J}{\partial W^{[L]}}$ is calculated as follows:

$$\frac{\partial J}{\partial W^{[L]}} = (\hat{y} - y) \odot \sigma(1 - \sigma) \cdot a^{[L-1]\top}$$

Now, the gradient $\frac{\partial J}{\partial W^{[L-1]}}$ for the $(L-1)^{th}$ layer is calculated as follows:

$$\frac{\partial J}{\partial W^{[L-1]}} = \frac{\partial J}{\partial a^{[L]}} \cdot \frac{\partial a^{[L]}}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \cdot \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \cdot \frac{\partial z^{[L-1]}}{\partial W^{[L-1]}}$$

We can reuse the calculated value of $\frac{\partial J}{\partial z^{[L]}}$. The gradient $\frac{\partial J}{\partial a^{[L-1]}}$ is often called the upstream gradient and is denoted by $d_u^{[L-1]}$ for the layer $L-1$. By generalization, $\frac{\partial J}{\partial W^{[l]}}$ is expressed as follows:

$$\begin{aligned} \frac{\partial J}{\partial W^{[l]}} &= d_u^{[l]} \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial W^{[l]}} \\ &= d_u^{[l]} \odot f'(z^{[l]}) \cdot a^{[l-1]} \\ \frac{\partial J}{\partial b^{[l]}} &= d_u^{[l]} \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}} \\ &= d_u^{[l]} \odot f'(z^{[l]}) \end{aligned}$$

The backpropagation process for a layer is summarized as follows:

Algorithm 1 Backpropagation algorithm

Calculating the Gradients:

$$\frac{\partial J}{\partial W^{[l]}} = d_u^{[l]} \odot f'(z^{[l]}) \cdot a^{[l-1]}$$

$$\frac{\partial J}{\partial b^{[l]}} = d_u^{[l]} \odot f'(z^{[l]})$$

Updating Weights and Biases:

$$W^{[l]} \to W^{[l]} - \eta \frac{\partial J}{\partial W^{[l]}}$$

$$b^{[l]} \to b^{[l]} - \eta \frac{\partial J}{\partial b^{[l]}}$$

Calculating Downstream Gradient:

$$d_u^{[l-1]} = d^{[l]} \cdot {W^{[l]}}^{\mathsf{T}}$$

2.4 Vectorization Over Training Set

For a training set with n instances $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, The first-layer activations for each instances are as follows:

$$\begin{aligned} z^{[1](1)} &= W^{[1]} x^{(1)} + b^{[1]} \\ z^{[1](2)} &= W^{[1]} x^{(2)} + b^{[1]} \\ &\vdots \\ z^{[1](n)} &= W^{[1]} x^{(n)} + b^{[1]} \end{aligned}$$

Instead of using a loop for the calculations, we can use a matrix representation of the data that is defined as follows:

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{d \times 3} \tag{1}$$

We can then combine this into a single unified formulation:

$$Z^{[1]} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = W^{[1]} X + B^{[1]} \tag{2}$$

where,

$$B^{[1]} = \begin{bmatrix} | & | & | \\ b^{[1]} & b^{[1]} & b^{[1]} \\ | & | & | \end{bmatrix} \tag{3}$$

2.5 Hyperparameters

In the context of neural networks, hyperparameters are the configuration variables set by the user before training that control the learning process and the model architecture. Unlike weights, hyperparameters are not learned during training. The following are some common hyperparameters in a neural network: Number of hidden layers, number of neurons per layer, activation functions, loss function, learning rate, optimizer, batch size, number of epochs, etc.