Lesson 05

Gavshin Artem 5130203/20101

Report about Neural Networks

Introduction

A **perceptron** is the simplest form of an artificial neuron used for binary classification. It takes several input values, multiplies them by corresponding weights, sums them up, and passes the result through an *activation function* (usually a threshold function). Perceptrons are used for classification tasks where data needs to be divided into two categories. Examples include pattern recognition, text classification, and other tasks where data can be linearly separable. The perceptron was proposed by Frank Rosenblatt in 1957. It is based on the idea that a neuron can be modeled as a simple linear combination of input data followed by activation.

Logistic regression is a statistical method for binary classification that uses the logistic function to model the probability of belonging to one of two classes. It computes a *linear combination* of input data and applies the logistic function to obtain the probability. Logistic regression is widely used in classification tasks such as medical diagnosis, credit scoring, marketing research, and other areas where the probability of an event needs to be predicted. Logistic regression is based on linear regression methods but uses the logistic function to transform the linear combination of input data into a probability. This method was developed in the early 20th century and became popular due to its simplicity and effectiveness.

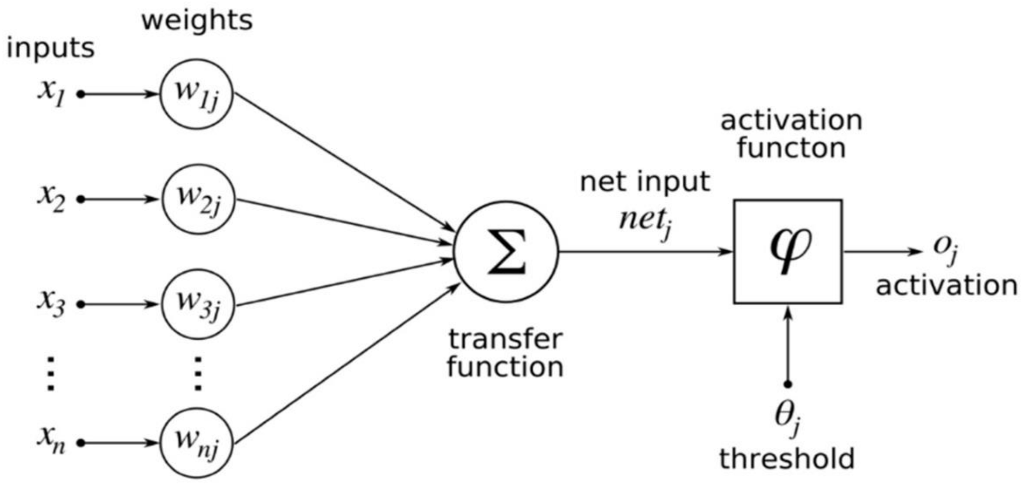
A **multilayer perceptron** (MLP) is a type of artificial neural network that consists of *multiple layers of neurons*: an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to neurons in the next layer, and data passes through the network undergoing nonlinear transformations. MLPs are used to solve complex classification and regression tasks such as pattern recognition, natural language processing, time series forecasting, and many other tasks where data cannot be linearly separated. MLPs are based on the concept of the perceptron but extend it by adding hidden layers and nonlinear *activation functions** (e.g., *ReLU* or sigmoid function, and e.t.c.). This allows the network to model complex dependencies in the data. MLPs became popular with the development of *backpropagation algorithms***, which allow efficient training of deep neural networks.

* **Activation functions** determine the output of a neuron based on its input. Types of activation functions:

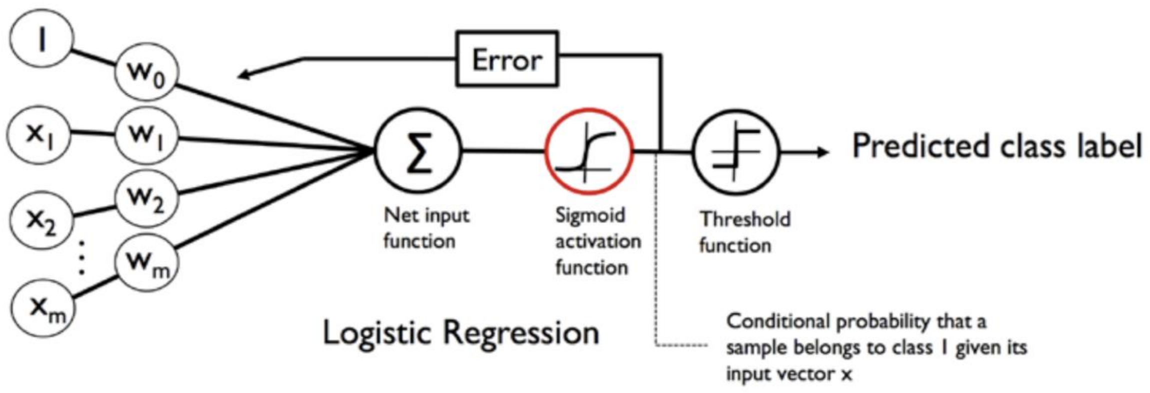
- **Sigmoid:** Transforms the input into a range between 0 and 1. It's often used in logistic regression and some neural networks.
- **Tanh (Hyperbolic Tangent):** Transforms the input into a range between -1 and 1. It's frequently used in hidden layers of neural networks.
- **ReLU (Rectified Linear Unit):** Converts negative values to zero and leaves positive values unchanged. It's widely used in modern neural networks due to its simplicity and effectiveness.
- **Leaky ReLU:** A variation of ReLU that allows small negative values, helping to avoid the problem of "dead" neurons.

** **Backpropagation** is a fundamental algorithm for training neural networks. It works by adjusting the weights to minimize the error between the predicted and actual outputs. After the network makes a prediction, the error is calculated. This error is then propagated backward through the network, layer by layer. During this process, the algorithm computes how much each weight contributed to the error. Using these calculations, the weights are updated to reduce the error. This process is repeated many times, allowing the network to learn and improve its predictions.

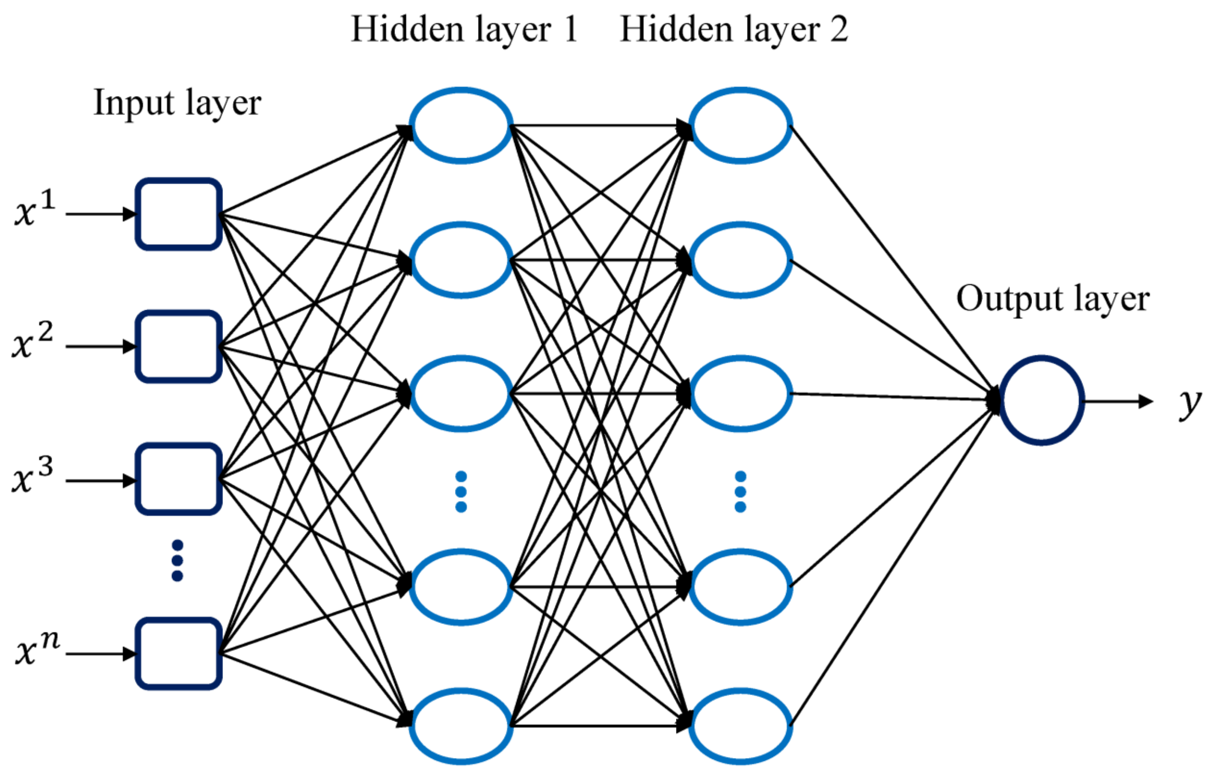
Model architecture



Pic. 1. Single – layer perceptron



Pic 2. Logistic Regression



Pic 3. MLP (Multilayer perseptron)

Vector Representation of Data

Perceptron:

- **Inputs:** In a perceptron, the input data is represented as a vector, where each element of the vector corresponds to one of the features of the input data.
  Input vector: $x = [x_1, x_1 \ldots x_n]^T$
- **Outputs:** The output is the result of the activation function applied to the weighted sum of the input data. It is usually a binary value (e.g., 0 or 1) for classification tasks.
  Output: $y \in \{-1, 1\}$

Logistic Regression:

- **Inputs:** In logistic regression, the input data is also represented as a vector, where each element of the vector corresponds to one of the features of the input data.
  Input vector: $x = [x_1, x_1 \ldots x_n]^T$
- **Outputs:** The output is a probability value that indicates the likelihood of the input data belonging to a particular class. This probability is obtained by applying the logistic function to the weighted sum of the input data.
  Output: $y \in \{0, 1\}$

Multilayer Perceptron (MLP):

- **Inputs:** In an MLP, the input data is represented as a vector, which is fed into the input layer of the network.
  Input vector: $x = [x_1, x_1 \ldots x_n]^T$

- **Outputs:** The output can be either a single value or a vector, depending on the task. For binary classification, it is typically a single probability value. For multi-class classification, the output is a vector of probabilities, each representing the likelihood of the input data belonging to a specific class. The output is obtained after passing through multiple layers of neurons, each applying its own weights and activation functions.

Output vector: y = [y_1,y_1 …y_m] ^{T}

Math formulation of linear combinations, loss func. and activation func.

Perceptron:

- Linear combination:

$$z = \sum_{i=1}^{n} \omega_i x_i + b = \omega^T x + b$$

- Activation function (returns 1 if the result of the linear combination is positive and -1 if it is negative):

$$\Phi(z) = \begin{cases} -1, z \geq 0 \\ 1, z < 0 \end{cases}$$

Or:

$$\hat{y} = sign\{\overline{W} \cdot \overline{X}\} = sign\left\{\sum_{j=1}^{d} w_j x_j\right\}$$

- Loss function:

$$L(x, y, w) = \max(0, -yw^T x + b)$$

Logistic regression:

- Linear combination:

$$z = \sum_{i=1}^{n} \omega_i x_i + b = \omega^T x + b$$

- Activation function (sigmoid):

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

- Loss function:

$$L(y, \hat{y}) = -\tfrac{1}{n} \textstyle\sum_{L=1}^{n} (\hat{y}i \cdot \log(y_i) + (1 - \hat{y}) \cdot \log(1 - y_i))$$

MLP:

- Linear combination (same as perceptron, but for each layer):

$$\mathrm{z} = w * x + \mathrm{b}$$

- Activation function (most popular functions):

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

And:

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

- Loss function:

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log(\hat{y})$$

Math formulation how neural nets calculate the predictions

Formula for **Perceptron**: $\hat{y} = \phi(z)$

Formula for **Logistic Regression** : $\hat{y} = \sigma(z)$

Formula for **Multilayer Perceptron** (same as the Perceptron): $\hat{y} = \phi(z)$

Explanation of gradient descendent algorithm, formulas of it and weight / bias updates

The **Perceptron** algorithm uses gradient descent to minimize classification errors. It starts by initializing the weights and bias with random values. For each training example, it calculates the prediction. If the prediction is incorrect, it updates the weights and bias based on the difference between the predicted and actual values. This process continues until the model achieves a satisfactory level of accuracy.

Formula for updating weights and bias:

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i$$

$$\mathrm{b} \leftarrow \mathrm{b} + \eta(y - \hat{y})x_i$$

In **Logistic Regression**, gradient descent is used to minimize the logistic loss function, also known as cross-entropy. The algorithm begins by initializing the weights and bias randomly. For each training example, it computes the prediction using the sigmoid function. The error is calculated as the difference between the predicted probability and the actual label. The weights and bias are then updated by subtracting the gradient of the loss function with respect to the weights and bias, scaled by the learning rate. This process is repeated until the model converges to a minimum loss.

Formula for updating weights and bias:

$$\frac{\partial L}{\partial w_i} = (y - \hat{y})x_i$$

$$w_i \leftarrow w_i \ + \eta \ \frac{\partial L}{\partial w_i}$$

$$\frac{\partial L}{\partial w_i} = (y - \hat{y})$$

$$\mathrm{b} \leftarrow \mathrm{b} \ + \eta \ \frac{\partial L}{\partial w_i}$$

For **MLP**s, gradient descent is combined with backpropagation to minimize the loss function. The algorithm starts by initializing the weights and biases for all layers randomly. During the forward pass, it computes the predictions by passing the input through the network layers. The error is calculated as the difference between the predicted and actual values. During the backward pass, the algorithm computes the gradients of the loss function with respect to the weights and biases, starting from the output layer and moving backward through the network. The weights and biases are then updated by subtracting the gradients, scaled by the learning rate. This process is repeated until the model achieves a satisfactory level of performance.

Formula for updating weights and bias:

$$\frac{\partial L}{\partial w_i} = \delta * (\alpha)^T$$

$$w_i \leftarrow w_i \ + \eta \ \frac{\partial L}{\partial w_i}$$

$$\mathrm{b} \leftarrow \mathrm{b} \ + \eta \ \frac{\partial L}{\partial w_i}$$