Peter the Great St. Petersburg Polytechnic University
Institute of Computer Science and Cybersecurity
Higher School of Artificial Intelligence Technologies

Neural Networks Report

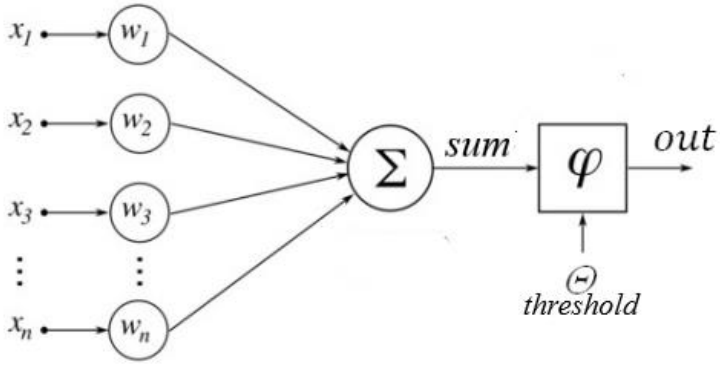
Completed by
student of group 5130203/20102 Butyrevskaya G.V.
Accepted by Espinola Rivera Holger Elias

Санкт-Петербург, 2024

Content

Perceptron

Model architecture drawing



Pic. 1. Perceptron model

$\sum - the\ sum\ of\ the\ input\ signals\ taking\ in\ to\ account\ their\ weights$
$\varphi\ -\ activation\ function\ to\ the\ total\ sum\ of\ the\ effects\ of\ input\ signals$
$\Theta\ - threshold$

Vector representation of data (inputs and outputs)

Inputs: *sum* = $X^T\ W\ +\ B = \sum_{i=1}^{n} x_i w_i + b$, $b -$ displacement by the value b (bias). This is not indicated on the drawing.

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

Outputs: $out\ = \varphi(sum)$.

Mathematical formulation of linear combination, activation function and loss function

Linear combination (sum of weighted inputs)

The perceptron takes as input n values $x_1, x_2, \ldots, x_n$, each of which is multiplied by the corresponding weight $w_1, w_2, \ldots, w_n$. The result is a weighted sum:

$$sum\ = X^T\ W\ +\ B = \sum_{i=1}^{n} x_i w_i + b$$

Activation function

The result of linear combination *sum* is passed to the activation function, which determines the output of the neuron $out = \varphi(sum)$.

Some popular activation features are:

1. Threshold function

$$\varphi(sum) = \begin{cases} 1, sum \geq 0 \\ 0, sum < 0 \end{cases}$$

2. Sigmoid function

$$\varphi(sum) = \frac{1}{1 + e^{-sum}}$$

3. ReLU

$$\varphi(sum) = max(0, sum)$$

Loss function

The loss function is a metric that reflects the discrepancy between expected and actual output data.

1. Mean Squared Error, MSE. This loss function is particularly sensitive to outliers in the training set, since it uses the square of the difference between the actual and expected values (an outlier is a value that is very far from other values in the data set).

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

2. Root MSE. Same as root mean square error, but can reflect a real physical unit of measurement.

$$L = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

3. Mean Absolute Error, MAE (not so sensitive to emissions).

$$L = \frac{1}{n}\sum_{i=1}^{n}|(y_i - \hat{y}_i)|$$

4. Cross entropy (for classification problems)

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

In this case $\hat{y}_i$ – predicted value, y_i – real value.

A mathematical formulation of how neural networks make predictions

Neural network forecast is calculated by successively transforming the input data in a forward propagation process. This process involves linear transformations, activation functions.

$$\hat{y} = \varphi\left(\sum(X, W)\right)$$

Explanation of the Gradient Descent Algorithm

The goal of training a neural network is simple - it is to minimize the loss function. One way to find the minimum of a function is to modify the connection weights in the direction opposite to the gradient vector ∇ at each successive learning step - the gradient descent method. To do this, a mathematical formula is used to determine which direction to move in to get closer to the correct answer. The process is repeated many times until the algorithm can predict the answer as well as possible.

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T$$

Pic. 2. The gradient vector ∇

Formulas for updating gradients and weights/biases

Gradient descent method mathematically looks like this:

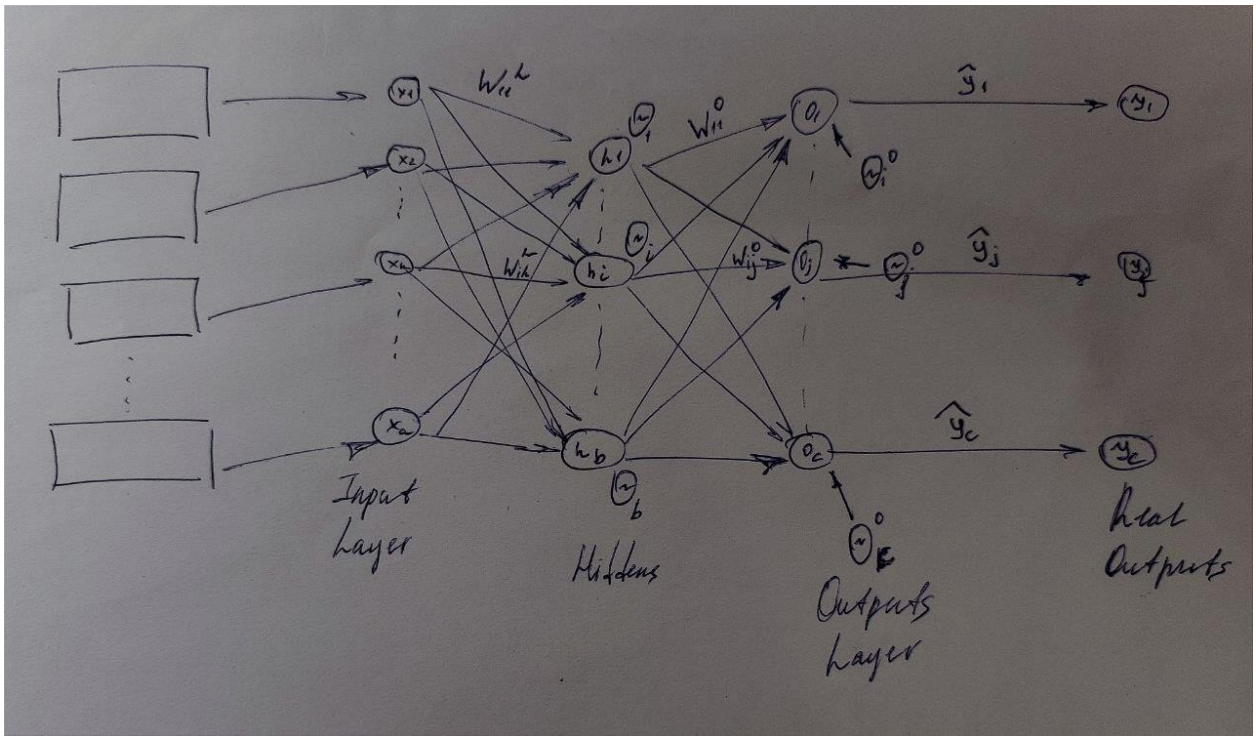
$$\vec{w}^{(k+1)} = \vec{w}^k - \mu\nabla L(\vec{w}^k),$$

where $k -$ neural network training iteration, μ –learning rate, $\nabla L -$ gradient of the loss function. To find the gradient, we use partial derivatives with respect to configurable arguments $w_1, w_2, \ldots, w_n$:

$$\nabla L(\vec{w}) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \end{bmatrix}$$

Multilayer perceptron

Model architecture drawing



Pic.3. Multilayer perceptron

Vector representation of data (inputs and outputs)

Input layer

$$X = [x_1, \ldots, x_a]$$

Hidden layer

$$H = [h_1, \ldots, h_i, \ldots, h_b]$$

$$W^h = \begin{bmatrix} w_{11}^h & \cdots & w_{1k}^h & \cdots & w_{1a}^h \\ \vdots & & \vdots & & \vdots \\ w_{b1}^h & \cdots & w_{bk}^h & \cdots & w_{ba}^h \end{bmatrix}$$

$$\theta^h = \begin{bmatrix} \theta_1^h \\ \vdots \\ \theta_b^h \end{bmatrix}$$

Output layer

$$O = [O_1 \quad \ldots \quad O_j \quad \ldots \quad O_c]$$

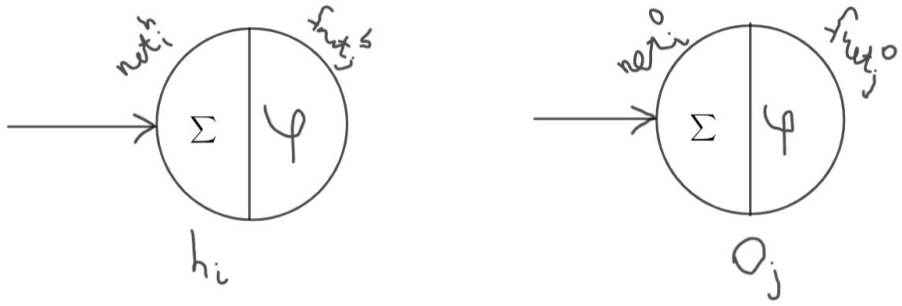
$$W^O = \begin{bmatrix} w_{11}^O & \cdots & w_{1i}^O & \cdots & w_{1b}^O \\ \vdots & & \vdots & & \vdots \\ w_{c1}^O & \cdots & w_{ci}^O & \cdots & w_{cb}^O \end{bmatrix}$$

$$\theta^O = \begin{bmatrix} \theta_1^O \\ \vdots \\ \theta_c^O \end{bmatrix}$$

Mathematical formulation of linear combination, activation function and loss function

Two algorithms to do this:

1. $\overrightarrow{Forward}$



Pic. 4. Forward algorithm

- hidden

$$net_i^h = \sum_{k=1}^{a} x_k w_{ik}^h + \theta_i^h$$

$$fnet_i^h = \varphi(net_i^h)$$

- outputs

$$net_j^O = \sum_{i=1}^{b} fnet_i^h w_{ji}^O + \theta_j^O$$

$$fnet_j^O = \varphi(net_j^O)$$

2. $\overleftarrow{backpropagation}$

- loss function

$$L = \frac{1}{2}\sum_{j=1}^{c}(y_j - \hat{y}_j)^2 \quad MSE$$

- optimiser (algorithm SBG)

$for\ each\ epochs$:

$for\ each\ (x_i, y_i) \in (X, Y)$

$$\xrightarrow{D.G.R} W^{(t+1)} = W^t - \eta\frac{\partial L}{\partial W} - this\ is\ updating$$

- outputs

$$W_{ji}^{O(t+1)} = W_{ji}^{O(t)} - \eta\frac{\partial L}{\partial W_{ji}^O}$$

$$\frac{\partial L}{\partial W_{ji}^{O}} = \delta_j^O(fnet_i^h)$$

$$\theta_j^{O(t+1)} = \theta_j^{O(t)} - \eta\frac{\partial L}{\partial \theta_j^O}$$

$$\eta\frac{\partial L}{\partial \theta_j^O} = \delta_j^O(1)$$

- hidden

$$W_{ik}^{h(t+1)} = W_{ik}^{h(t)} - \eta\frac{\partial L}{\partial W_{ik}^h}$$

$$\eta\frac{\partial L}{\partial W_{ik}^h} = \delta_i^h(X_k), X_k - input$$

$$\theta_i^{h(t+1)} = \theta_i^{h(t)} - \eta\frac{\partial L}{\partial \theta_i^h}$$

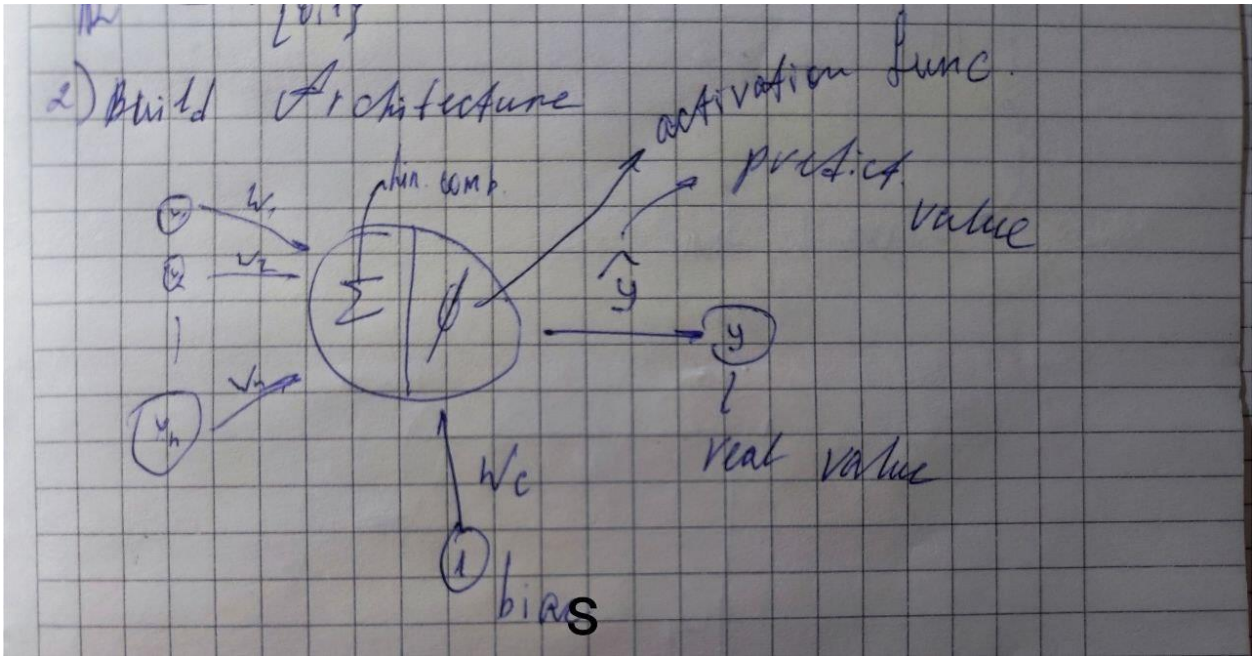
$$\eta\frac{\partial L}{\partial \theta_i^h} = \delta_i^h(1), 1 - inputs$$

How we can calculate $\delta_{\mathrm{i}}^{\mathrm{h}}(\mathrm{X_k}), \delta_{\mathrm{i}}^{\mathrm{h}}(1)$?

$$\delta_i^h = f'(net_i^h)\sum_{j=1}^{c}\delta_j^O\, w_{ji}^O$$

Logistic regression

Model architecture drawing



Pic. 1. Logistic regression model

Vector representation of data (inputs and outputs)

$$z = w_1x_1 + \cdots + w_nx_n + w_0$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

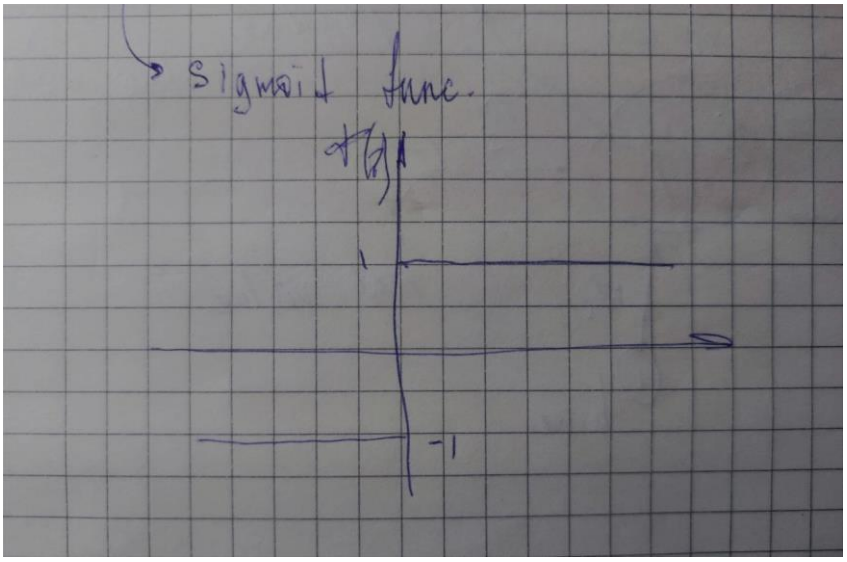
Mathematical formulation of linear combination, activation function and loss function

Linear combination (sum of weighted inputs)

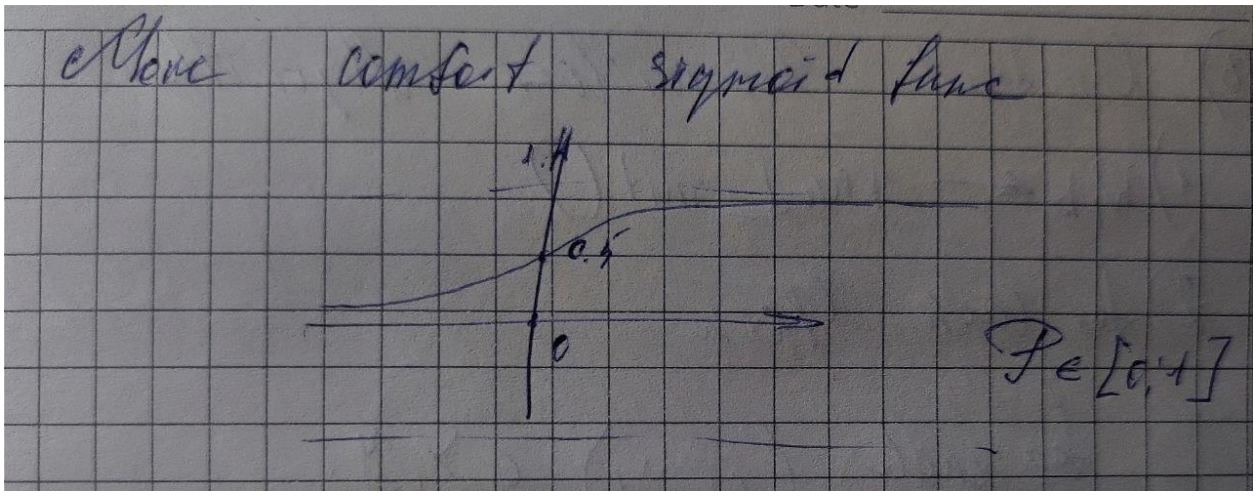
$$z = W^T X$$

Activation function

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



Pic. 2. Sigmoid function



Pic. 3. More comfort sigmoid function

Loss function

$$L(y, \hat{y}) = -\frac{1}{N}\sum [y\ log(\hat{y}) + (1 - y)\ log(1 - \hat{y})]$$

A mathematical formulation of how neural networks make predictions

$$\hat{y} = \varphi\left(\sum (X, W)\right), \sum (X, W)\ =\ z$$

Explanation of the Gradient Descent Algorithm

1. $w_i b \leftarrow rand\_init()$
2. $for\ each\ epoch$:
   $for\ each (x_i, y_i) \in X, Y$:
   $W^{t+1} = W^t - \eta \frac{\partial L}{\partial W}$
   $\eta - learning\ rate$

Formulas for updating gradients and weights/biases

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial L}{\partial \hat{y}} = -\left[y \frac{1}{\hat{y}} + (1 - \hat{y}) \frac{-1}{1 - \hat{y}}\right] = -\frac{(y - \hat{y})}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = 1 - \frac{1}{1 + e^{-z}}$$

$$\frac{\partial z}{\partial w_i} = x_i, i \neq 0$$

$$i = 0: \frac{\partial z}{\partial w_0} = 1$$

$${W_0}^{(t+1)} = {W_0}^t - \eta[-y(y - \hat{y})]$$