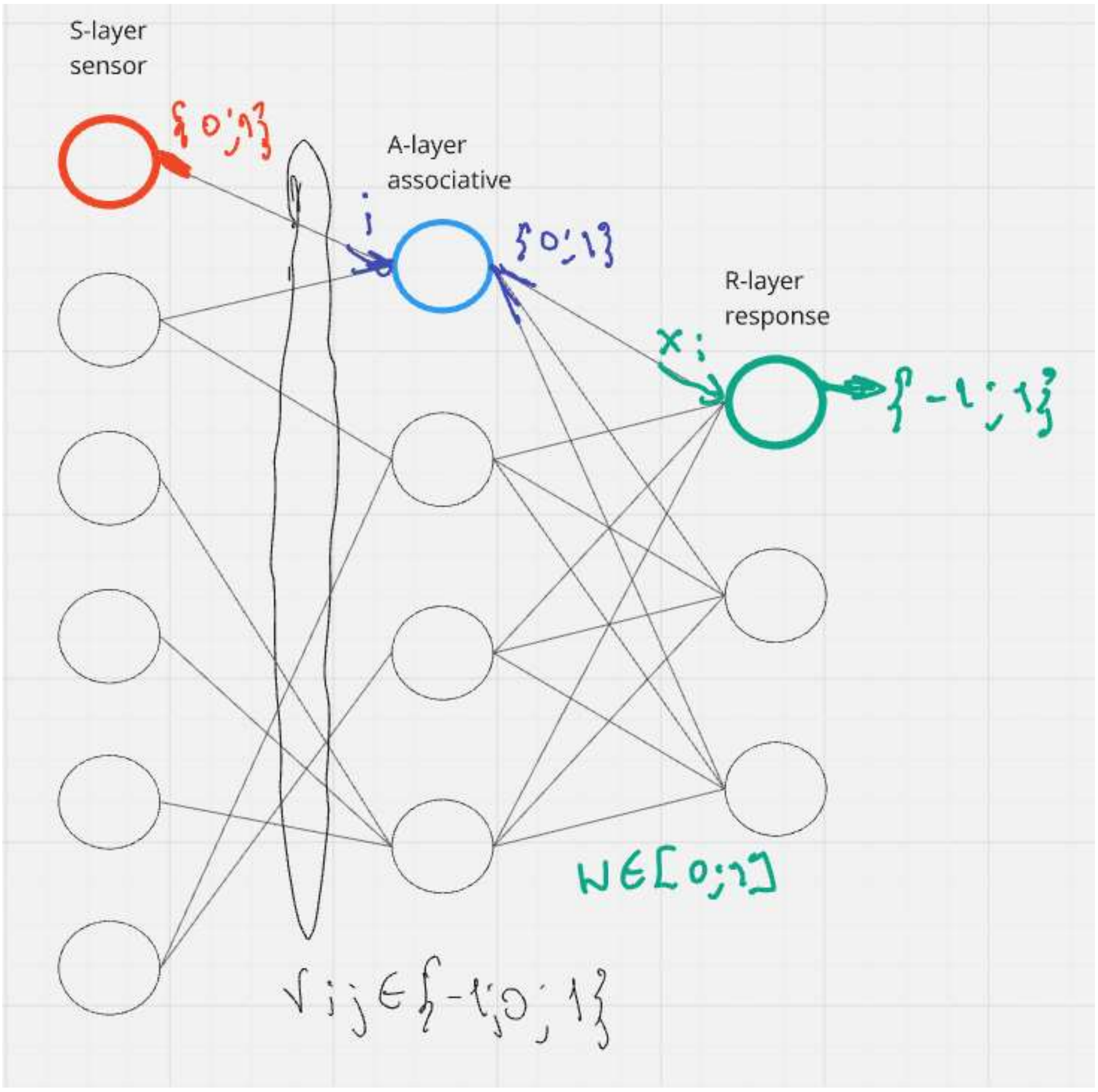
Galitskih Dasha, 5130203/20102

Mathematical formulation of algorithm Perceptron

Model architecture drawing

One of the earliest and most basic machine learning methods used for binary classification is perceptron. Frank Rosenblatt created it in the late 1950s, and it is a key component of more complex neural network topologies.



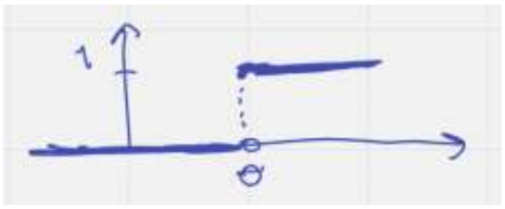
A perceptron is an artificial neural network consisting of three layers: S-layer (sensor), A-layer (associative), R-layer (response). Each circle in the drawing is an adder that works differently in each layer.

There are connections between the layers that are completely random in the model. But between the A-layer and the R-layer, the connections are usually all-to-all.

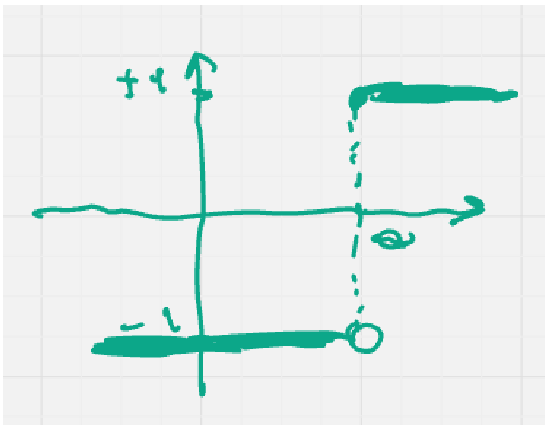
Each element of the S-layer is a sensor, the output of which can be one of two values: 0 or 1.

There are weights of the connection $V_{ij} \in \{-1; 0; 1\}$. If the link is 0, then there is no connection. If the connection is -1, then the neuron is inhibitory. If the connection is equal to 1, then the neuron is excitatory.

The associative neurons of the A-layer are excited and transmit 0 or 1 at the output as well. These neurons have a transfer function in which a certain threshold is allocated $\theta \ (hyperparameter)$. If $\sum i V_{ij} \geq \theta$ (the sum of the input pulses is greater than or equal to this threshold), then the output is 1:



Response neurons are more complicated, they can return -1 or 1. In fact, they return the value of the following function: $f(x) = \text{sgn}(\sum_{i=1}^{n} x_i w_i - \theta)$, где n – number of input connections, $x_i \in \{0; 1\}$, a $w_i \in [0; 1]$ – weight.



Vector representation of data (inputs and outputs)

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

$$X = \begin{bmatrix} x_i \\ \vdots \\ x_n \end{bmatrix} \quad W = \begin{bmatrix} w_i \\ \vdots \\ w_n \end{bmatrix}$$

$$\text{Output: } out = \varphi(sum)$$

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

Activation function

The result of linear combination is passed to the activation function. Some popular activation features are:

1) Threshold function

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

2) ReLU

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

3) Sigmoid function

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

Loss function

Loss functions quantify the difference between predicted and actual outputs. They guide the learning process by penalizing incorrect predictions. Here are the loss functions you described, with clearer explanations and corrected formulas:

1. Mean Squared Error (MSE): Calculates the average of the squared differences between predicted ($\hat{y}_i$) and actual (y_i) values. It heavily penalizes large errors, making it sensitive to outliers.

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

2. Root Mean Squared Error (RMSE): Similar to MSE, but takes the square root of the average squared difference. This allows RMSE to be expressed in the same units as the target variable, making it easier to interpret the magnitude of the error.

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

3. Mean Absolute Error (MAE): Calculates the average of the absolute differences between predicted and actual values. It's less sensitive to outliers than MSE because it doesn't square the differences.

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

4. Cross-Entropy (for classification):Measures the dissimilarity between two probability distributions (predicted and true).

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

There are $y_i - real\ value, \hat{y}_i - predicted\ value.$

How neural networks make prediction

The prediction of a neural network is calculated by sequentially converting input data in the process of direct propagation. This process includes linear transformations, activation functions.

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

Explanation of gradient descendent algorithm

The purpose of neural network training is simple - to minimize the loss function. One of the ways to find the minimum of the function is to change the weights of the connections in the direction opposite to the gradient vector ∇ at each subsequent stage of training - the gradient descent method. To do this, a mathematical formula is used that determines which direction to move in order to get closer to the correct answer. The process is repeated many times until the algorithm can predict the answer as accurately as possible.

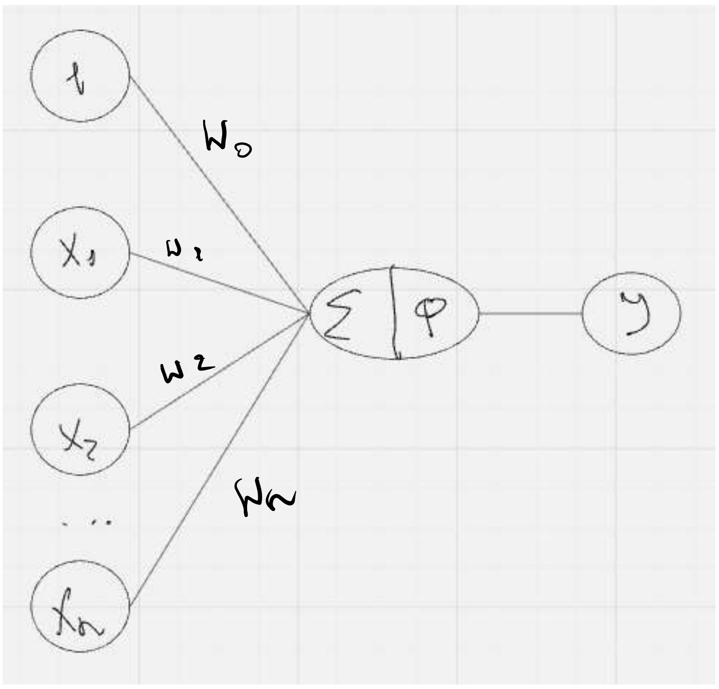
$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_n}\right]^T$$

Formulas of gradients and weights/biases updates

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

Mathematical formulation of algorithm Logistic Regression

Model architecture drawing



Vector representation of data (inputs and outputs)

$$\text{Input: } X = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$\text{Output: } y \in R$$

Linear combination

$$sum = w_0 + w_1x_1 + \ w_2x_2 + \cdots + w_nw_n = [w_0\ w_1\ \dots\ w_n]\begin{bmatrix}1\\x_1\\\vdots\\x_n\end{bmatrix} = \ W^TX = z$$

Activation function

$$\Phi(z) = \frac{1}{1+e^{-z}}$$ - sigmoid function

Loss function

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i\,log(\hat{y}_i) + (1-y_i)log(1-\hat{y}_i)]$$ – cross entropy loss

How neural networks make prediction

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

Explanation of gradient descendent algorithm

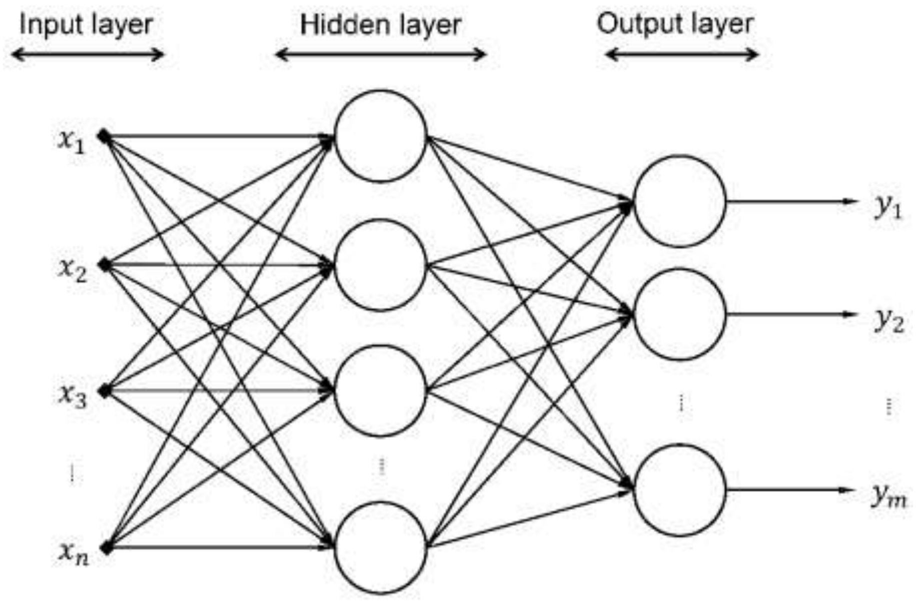
Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters that minimize a cost function as far as possible.

Formulas of gradients and weights/biases updates

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

Mathematical formulation of algorithm Multilayer Perceptron

Model architecture drawing



Vector representation of data (inputs and outputs)

$$\text{Input } X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$\text{Hidden layer } H = \begin{bmatrix} h_1 \\ \vdots \\ h_m \end{bmatrix}$$

$$\text{Weights: } W = \begin{bmatrix} w_{11}^h & \cdots & w_{1n}^h \\ \vdots & w_{ik}^h & \vdots \\ w_{m1}^h & \cdots & w_{mn}^h \end{bmatrix}$$

$$\text{Output: } Y = \begin{bmatrix} y_1 \\ \vdots \\ y_b \end{bmatrix},$$

Linear combination: $net_i^h = \sum_{k=1}^{n} (x_k \cdot w_{ik}^h) + \theta_j^h$

Activation function: $\Phi(net_i^h) = \frac{\mathbf{1}}{\mathbf{1} + \mathrm{e}^{-net_i^h}}$

Loss function: $min\left(L = \frac{1}{2} \sum_{i=1}^{b} (y_i - \hat{y}_i)^2\right)$

How neural networks make prediction: $\delta_i^h = f'(net_i^h) \sum_{j=1}^{C} \delta_j^0 W_{ji}^0$