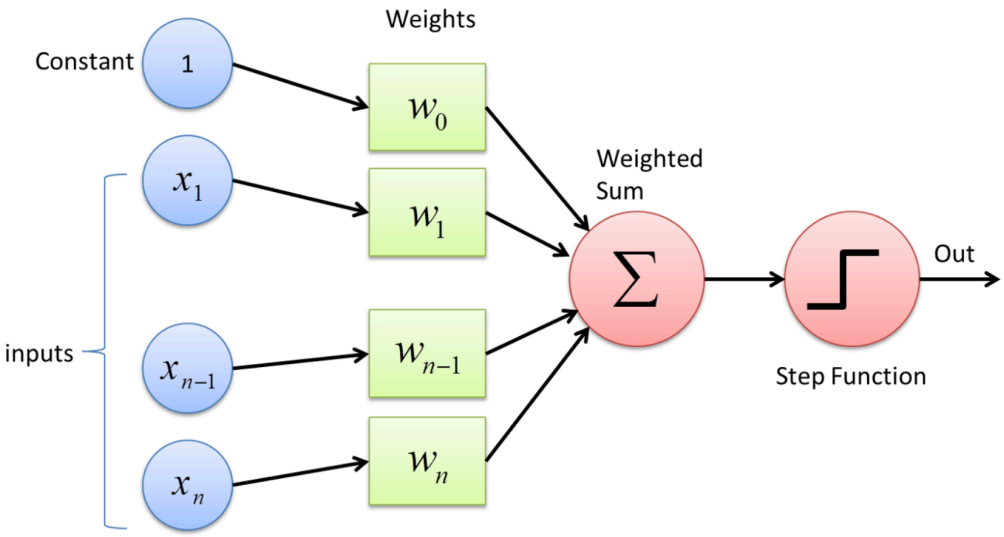
Malysheva Anna 5130203/20102

Report about Neural Networks

Perceptron

Model architecture drawing



The first step is to multiply the inputs to the weights. Once that is done, add the products together to get the weighted sum. Apply the activation function to the sum to classify the data.

Perceptrons need weights and bias. Why? The weight shows the strength of each node. A bias value, meanwhile, lets you shift the activation function curve up or down. But why do you need an activation function? Because a perceptron uses binary, you need to separate values into two parts only. The activation function does just that. It maps the inputs into the required values like (0, 1) or (-1, 1).

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: $out = \sqcap(sum)$.

Linear combination

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

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

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 (fi) and actual (yi) values.

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

2. Root Mean Squared Error (RMSE): Similar to MSE, but takes the square root of the average squared difference.

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

3. Mean Absolute Error (MAE): Calculates the average of the absolute differences between predicted and actual values.

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

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

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

A mathematical formulation of how neural networks make predictions

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

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

Explanation of gradient descendent algorithm

Neural network training aims to minimize the loss function. Gradient descent iteratively adjusts connection weights in the direction opposite the gradient vector to find this minimum. A mathematical formula dictates the weight updates, moving the network closer to optimal predictions with each iteration.

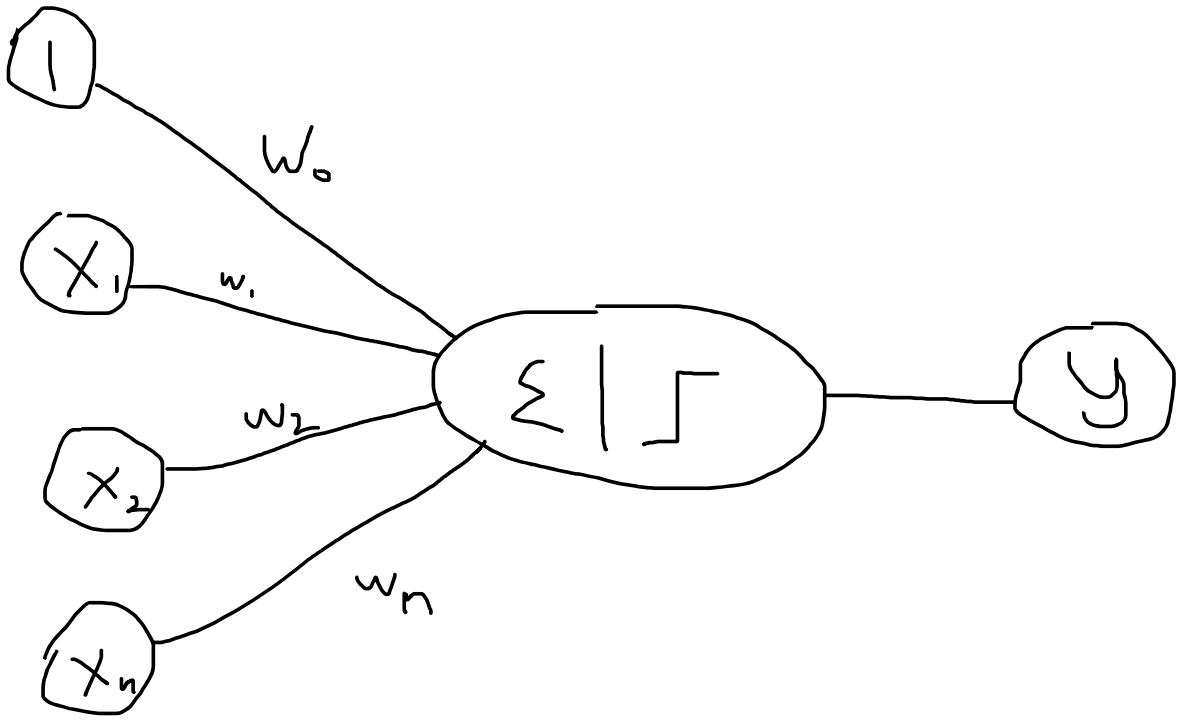
$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T$$

Formulas of gradients and weights/biases updates

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

Logistic Regression

Model architecture drawing



Vector representation of data (inputs and outputs)

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

Output: $y \in R$

Linear combination

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

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

A mathematical formulation of how neural networks make predictions

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

Explanation of gradient descendent algorithm

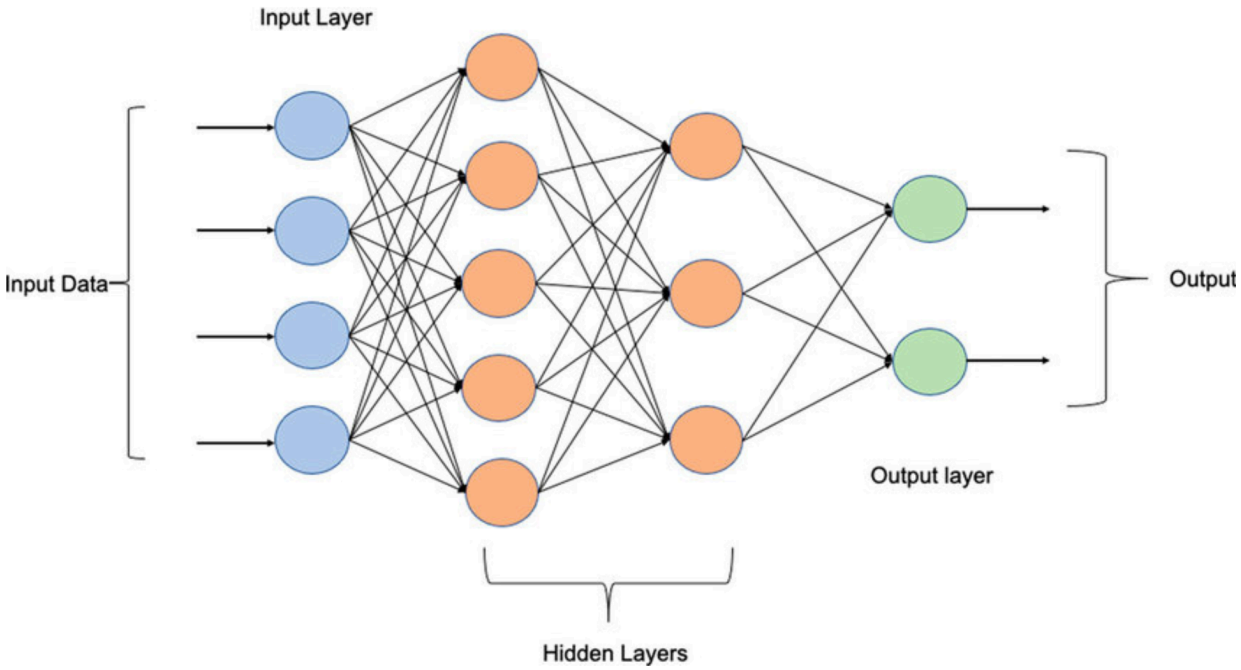
Gradient descent is an optimization algorithm used to find the local minimum of a differentiable function, minimizing a cost function in machine learning by adjusting parameters.

Formulas of gradients and weights/biases updates

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

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

$$\Phi(net_i^h) = \frac{1}{1+e^{-net_i^h}}$$

Loss function

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

A mathematical formulation of how neural networks make predictions

$$\delta_i^h = f'(net_i^h)\sum_{j=1}^{C}\delta_j^0 W_{ji}^0$$