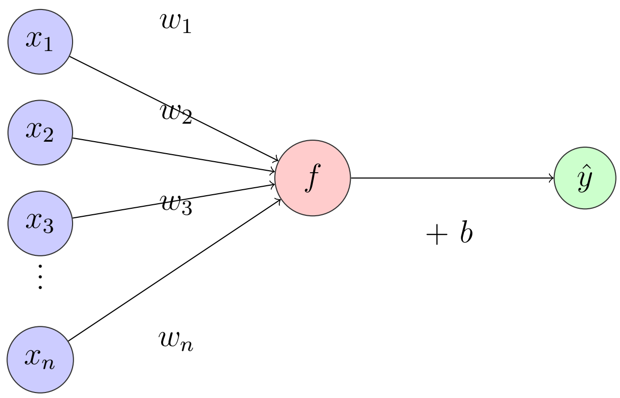
Tsarkova Alina - 5130203/20102

Report about Neural Networks

Perceptron

1. Model Architecture

The Perceptron is one of the simplest types of neural networks, consisting of a single layer with a single neuron. Below is a diagram of the Perceptron model architecture:



2. Vector Representation of Data

Let the input data be represented as a vector:

$\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$

The corresponding weights are represented as:

$\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$

The bias term is denoted as (b). The output (y) is a scalar value.

3. Mathematical Formulation

The Perceptron performs the following operations:

Linear Combination

The linear combination of inputs and weights is computed as:

$z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$

Activation Function

The activation function (f(z)) used in the Perceptron is typically the step function:

$$\text{f(z)} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Loss Function

The Perceptron loss function is defined for misclassified points only:

$L(\mathbf{w}, b) = -y(\mathbf{w}^T\mathbf{x} + b), \quad \text{if } y(\mathbf{w}^T\mathbf{x} + b) \leq 0$

4. Prediction Calculation

To calculate the prediction ($\hat{y}$), the Perceptron performs the following steps:

- Compute the linear combination: ($z = \mathbf{w}^T\mathbf{x} + b$)
- Apply the activation function: ($\hat{y}$ = f(z))

Thus:

$$\hat{y} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

5. Gradient Descent Algorithm

The Perceptron uses an iterative update rule to minimize classification errors by adjusting weights and biases.

Update Rule for Weights and Bias

When a misclassification occurs (($y(\mathbf{w}^T\mathbf{x}+b) \leq 0$)), the weights and bias are updated as follows:

$w_i := w_i + \eta y x_i, \quad \forall i = 1, 2, \ldots, n$

$b := b + \eta y$

where ($\eta > 0$) is the learning rate.

Explanation of Gradient Descent

Gradient descent adjusts the parameters ((w_i) and (b)) to reduce the loss function by moving in the direction of steepest descent in the parameter space.

For the Perceptron loss function:

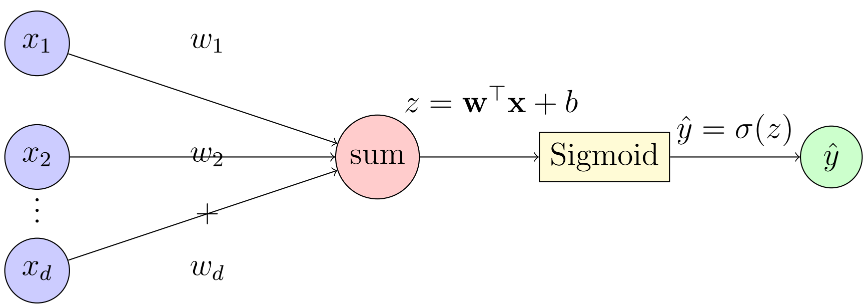
$\nabla_{\mathbf{w}} L = -y\mathbf{x}, \quad \nabla_b L = -y$

The weight and bias updates are derived from these gradients.

Logistic Regression

1. Model Architecture

The logistic regression model can be visualized as follows:



The model consists of input features $\mathbf{x}$, weights $\mathbf{w}$, a bias b, and a sigmoid activation function to produce the output $\hat{y}$.

2. Vector Representation of Data

The input data can be represented as: $\mathbf{x} = \begin{bmatrix} x_1 \; x_2 \; \dots \; x_d \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \; w_2 \; \dots \; w_d \end{bmatrix}, \quad bin\mathbb{R}$. The output $\hat{y}$ is a scalar representing the predicted probability that the input belongs to class 1.

The true label y is also a scalar:
y in 0, 1.

3. Mathematical Formulation

The logistic regression algorithm involves the following steps:

Linear Combination

The model computes a linear combination of the input features:
$z = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{d} w_i x_i + b.$

Activation Function

The sigmoid activation function is applied to z to produce a probability:
$\sigma(z) = \frac{1}{1+e^{-z}}.$
Thus, the predicted probability is:
$\hat{y} = \sigma(z).$

Loss Function

The loss function used in logistic regression is the binary cross-entropy loss:
$L(\hat{y}, y) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y}),$
where y is the true label and $\hat{y}$ is the predicted probability.
For a dataset with n samples, the total loss is:
$J(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}^{(i)}, y^{(i)}).$

4. Prediction Calculation

To calculate predictions $\hat{y}$ for all samples: 1. Compute $z^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$ for each sample i. 2. Apply the sigmoid function:
$\hat{y}^{(i)} = \sigma(z^{(i)}).$

The final predictions are probabilities $\hat{y}^{(i)}$. To classify into binary labels: Predicted label = $\begin{cases} 1 & \text{if } \hat{y}^{(i)} > 0.5, \\ 0 & \text{otherwise.} \end{cases}$

5. Gradient Descent Algorithm

Gradient descent is used to minimize the loss function $J(\mathbf{w}, b)$ by updating the weights $\mathbf{w}$ and bias b. The updates are performed iteratively as follows:

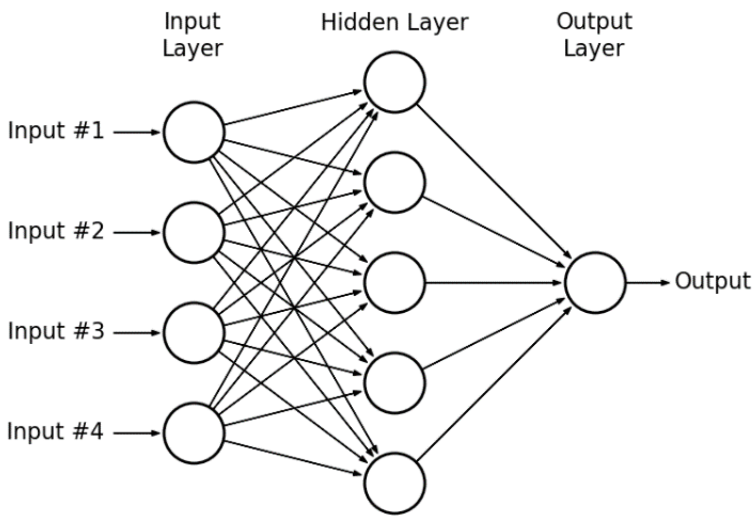
1. Compute gradients of the loss with respect to weights and bias:
$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)},$
$\frac{\partial J}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}).$

2. Update weights and bias using the gradients:
$w_j := w_j - \alpha \frac{\partial J}{\partial w_j}$,
$b := b - \alpha \frac{\partial J}{\partial b}$,
where α is the learning rate.
This process is repeated until convergence.

Multilayer Perceptron

1. Model Architecture

The architecture of a Multilayer Perceptron (MLP) consists of an input layer, one or more hidden layers, and an output layer. The following diagram illustrates the architecture of an MLP with one hidden layer:



2. Vector Representation of Data

Let the input data be represented as a vector:
$\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$,
where x_i represents the i-th feature of the input.
The output is represented as:
$\mathbf{y} = [y_1, y_2, \ldots, y_m]^\top$,
where y_j represents the j-th output.
Weights and biases for each layer are represented as:
$\mathbf{W}^{(l)} = [w_{ij}^{(l)}], \quad \mathbf{b}^{(l)} = [b_j^{(l)}]$,
where l denotes the layer index.

3. Mathematical Formulation

Linear Combination

For each neuron in layer l, the linear combination is computed as:

$z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$,
where:

- $z_j^{(l)}$: Linear combination for neuron j in layer l.
- $w_{ij}^{(l)}$: Weight connecting neuron i from layer $(l-1)$ to neuron j in layer l.
- $a_i^{(l-1)}$: Activation from neuron i in layer $(l-1)$.
- $b_j^{(l)}$: Bias term for neuron j in layer l.

Activation Function

The activation function introduces non-linearity into the model. Common activation functions include:

- Sigmoid: $sigma(z) = \frac{1}{1+e^{-z}}$
- ReLU: $\text{ReLU}(z) = max(0, z)$
- Tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

The activation for neuron j in layer l is given by:
$a_j^{(l)} = f(z_j^{(l)})$,
where $f(\cdot)$ is the activation function.

Loss Function

The loss function quantifies the difference between predicted outputs and true outputs. For regression tasks, the Mean Squared Error (MSE) is commonly used:
$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$,
where $\hat{\mathbf{y}}$ is the predicted output.
For classification tasks, the Cross-Entropy Loss is often used:
$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(\hat{y}_i)$,
where $\hat{\mathbf{y}}$ is the predicted probability distribution.

4. Prediction Calculation

The prediction $\hat{\mathbf{y}}$ is calculated by propagating inputs through the network:
$a_j^{(l)} = f(z_j^{(l)}), \quad z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$,
starting from the input layer and ending at the output layer.
The final prediction is:
$\hat{\mathbf{y}} = [a_1^{(L)}, a_2^{(L)}, \ldots, a_m^{(L)}]^\top$,
where L is the output layer.

5. Gradient Descent Algorithm

Gradient Descent is used to minimize the loss function by updating weights and biases iteratively.

At each iteration: 1. Compute gradients of the loss function with respect to weights and biases. 2. Update weights and biases using:

$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \frac{\partial L}{\partial w_{ij}^{(l)}}$,

$b_j^{(l)} := b_j^{(l)} - \eta \frac{\partial L}{\partial b_j^{(l)}}$,

where η is the learning rate.

6. Gradients and Updates

The gradients are computed using backpropagation:

For output layer:

$\delta_j^{(L)} = a_j^{(L)} - y_j$,

where $delta_j^{(L)}$ is the error term for neuron j in the output layer.

For hidden layers:

$\delta_j^{(l)} = f'(z_j^{(l)}) \sum_{k=1}^{n^{(l+1)}} w_{jk}^{(l+1)} \delta_k^{(l+1)}$,

where $f'(z)$ is the derivative of the activation function.

Gradients for weights and biases are computed as:

$\frac{\partial L}{\partial w_{ij}^{(l)}} = a_i^{(l-1)} \delta_j^{(l)}$,

$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$.

Weights and biases are updated using Gradient Descent as described above.