You got me looking for Attention!

Monit Sharma

November 3, 2024

Contents

1 Introduction to Logistic Regression

Logistic regression is a binary classification model used to predict the probability that a given input belongs to a particular class. Unlike linear regression, logistic regression outputs a probability value between 0 and 1 by applying a sigmoid activation function to the weighted sum of inputs.

1.1 The Sigmoid Function

The sigmoid function, $\sigma(z)$, is used to map any real-valued number to the range $(0,1)$. This enables interpreting the output as a probability. The formula for the sigmoid function is:

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

where $z = w^T x + b$ represents the weighted sum of the input features, and w and b are the parameters (weights and bias) of the model.

For example, suppose we have:

$$z = 0.5$$

Applying the sigmoid function:

$$\sigma(0.5) = \frac{1}{1+e^{-0.5}} \approx 0.622$$

This means that, given the input and model parameters, there is a 62.2% probability that the input belongs to the positive class.

1.2 Loss and Cost Functions

After making a prediction, we compute the error between the prediction and the true label using the **binary cross-entropy loss function**. This function penalizes incorrect predictions, allowing the model to update its parameters to improve future predictions. The loss for a single example is defined as:

$$\text{Loss} = -\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

where y is the actual label and $\hat{y}$ is the predicted probability.

1.2.1 Example Calculation of Loss

Suppose:

- The true label $y = 1$ (positive class)
- The model's predicted probability $\hat{y} = 0.9$

Then,

$$\text{Loss} = -\left(1 \cdot \log(0.9) + (1-1) \cdot \log(1-0.9)\right) = -\log(0.9) \approx 0.105$$

If the model predicted $\hat{y} = 0.1$, the loss would be much higher:

$$\text{Loss} = -\log(0.1) \approx 2.302$$

This higher loss penalizes the model for incorrect predictions.

The **cost function** is the average loss over all samples:

$$\text{Cost} = \frac{1}{m} \sum_{i=1}^{m} \text{Loss}(y_i, \hat{y}_i)$$

where m is the number of samples. The cost function provides an overall measure of the model's error across the dataset.

1.3 Optimization Using Gradient Descent

To minimize the cost function, we use **gradient descent**. Gradient descent iteratively adjusts the weights and bias in a direction that reduces the cost. The update rules for the parameters are:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

where α is the **learning rate**, and $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$ are the gradients of the cost function with respect to the weights and bias, respectively.

1.3.1 Numerical Example of Gradient Descent

Suppose:

- Initial weight $w = 5$
- Initial bias $b = 0$
- Learning rate $\alpha = 0.1$
- Gradient of cost with respect to w is 3
- Gradient of cost with respect to b is 2

The update for w and b is:

$$w := 5 - 0.1 \times 3 = 4.7$$

$$b := 0 - 0.1 \times 2 = -0.2$$

After updating, the parameters w and b are closer to values that reduce the cost.

1.4 Learning Rate

The **learning rate** α controls the size of each update step in gradient descent:

- A small α results in slow convergence, requiring many iterations to reach the minimum cost.
- A large α may cause overshooting, where the model fails to converge.

Choosing an optimal learning rate is essential for effective learning.

2 Artificial Neural Networks (ANN)

An Artificial Neural Network (ANN) extends logistic regression by adding one or more **hidden layers** between the input and output layers. Each hidden layer enables the network to learn more complex patterns, as it extracts multiple levels of representations from the data.

2.1 Structure of a Neural Network

A neural network consists of the following components:

- **Input Layer**: Receives the data features.
- **Hidden Layers**: Learn intermediate representations that help the model capture complex patterns.
- **Output Layer**: Produces the final prediction.

A network with multiple hidden layers is referred to as a **deep neural network**. Each hidden layer increases the model's capacity to learn and represent data.

2.2 Activation Functions

Activation functions introduce non-linearity into the model, allowing it to learn non-linear relationships. Common activation functions include:

- **Sigmoid**: Used for binary classification, mapping values to $(0, 1)$.
- **Tanh**: Often used in hidden layers, as it centers outputs around zero, facilitating faster learning.

The Tanh function is given by:

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

3 Building a 2-Layer Neural Network

A 2-layer neural network contains one hidden layer between the input and output layers. This example walks through the process of building a simple 2-layer network with detailed explanations.

3.1 Size of Layers and Initializing Parameters

For a dataset with 348 samples, the network structure is as follows:

- **Input Layer**: Each sample in the input data is represented as $x(348)$.
- **Hidden Layer**:
$$z^{[1]}(348) = W^{[1]}x(348) + b^{[1]}(348)$$
$$a^{[1]}(348) = \tanh(z^{[1]}(348))$$
- **Output Layer**:
$$z^{[2]}(348) = W^{[2]}a^{[1]}(348) + b^{[2]}(348)$$
$$\hat{y}(348) = a^{[2]}(348) = \sigma(z^{[2]}(348))$$

3.1.1 Parameter Initialization Strategy

- **Random Initialization for Weights**: Random initialization prevents neurons from performing identical computations, which would slow learning. Small random values help avoid vanishing gradients.
- **Bias Initialization**: Biases can be initialized to zero without affecting the learning capacity of the network.

3.2 Forward Propagation

During forward propagation, we compute intermediate values for each layer:

- Hidden layer linear combination: $z^{[1]} = W^{[1]}x + b^{[1]}$
- Hidden layer activation: $a^{[1]} = \tanh(z^{[1]})$
- Output layer linear combination: $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
- Output activation (sigmoid): $\hat{y} = a^{[2]} = \sigma(z^{[2]})$

3.3 Loss and Cost Functions

The loss for each sample measures the error between prediction and actual label:

$$\text{Loss} = -\left(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\right)$$

The cost function, which is the average of all losses, is:

$$\text{Cost} = \frac{1}{m} \sum_{i=1}^{m} \text{Loss}(y_i, \hat{y}_i)$$

3.4 Backward Propagation

Backward propagation calculates the gradients of the cost function with respect to weights and biases. These gradients are then used to adjust the parameters in the direction that minimizes the cost.

3.5 Parameter Update

Using gradient descent, the parameters are updated as follows:

$$W := W - \alpha \frac{\partial J}{\partial W}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

3.6 Prediction with Trained Parameters

After training, the network uses the learned weights and biases to make predictions on new data by performing forward propagation. Predictions are generated by applying a threshold to $\hat{y}$, the output probability.

—

4 Conclusion

In summary, this document covers:

- Logistic regression as a binary classification method with the sigmoid activation function.
- Building a 2-layer neural network with hidden layers and the tanh activation function.
- Forward and backward propagation in neural networks.
- Optimization using gradient descent, with weight and bias updates.

These concepts form the foundation of deeper neural network architectures and more complex learning algorithms.