

Machine Learning

Video 71:

Logistic Regression Part 3 | Sigmoid Function

What was the issue with Perceptron trick?

The Perceptron trick in machine learning faced limitations because it could only solve linearly separable problems. It failed with complex, non-linear datasets, as it couldn't adapt to patterns that required more sophisticated models, like multi-layer networks. This led to the development of more advanced architectures, such as neural networks.

What was the approach in perceptron trick?

The Perceptron trick involves iteratively adjusting weights based on classification errors. The algorithm starts with random weights and updates them whenever it misclassifies a data point. The update rule involves adding or subtracting the input features, scaled by a learning rate, to minimize the classification error. This process continues until all data points are correctly classified or a set number of iterations is reached. The approach was effective for linearly separable data but struggled with non-linear patterns.

What can be the solution for the issue that exists with perceptron trick?

The solution to the issue with the Perceptron trick—its inability to handle non-linearly separable data—was the development of more advanced models like the **Multilayer Perceptron (MLP)**, which uses multiple layers of neurons (i.e., a neural network). The key advancements that addressed the limitations include:

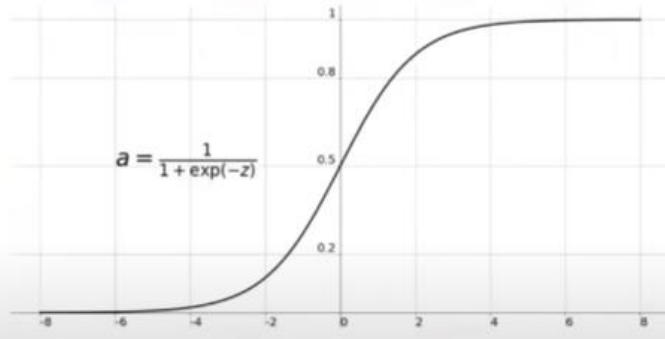
1. **Non-linear activation functions:** Introducing non-linear activation functions (like sigmoid, ReLU) allows the network to learn complex, non-linear patterns in the data.
2. **Backpropagation:** This algorithm enables efficient training of multi-layer networks by updating weights through gradient descent, allowing for deep learning and learning non-linear relationships.
3. **Support Vector Machines (SVM):** For some cases, SVMs were introduced as an alternative, using the kernel trick to map data into higher dimensions where it becomes linearly separable.

These approaches enabled more powerful models capable of handling a wider variety of real-world, non-linearly separable problems.

The Sigmoid Function

Thursday, June 17, 2021 12:19 PM

Sigmoid Function



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

Adding the **sigmoid function** to the Perceptron (or in a neural network) significantly enhances its ability to handle complex, non-linear relationships in data. Here's the impact:

1. **Non-linearity:** The sigmoid function introduces non-linearity to the model. While the Perceptron itself is a linear classifier, the sigmoid function allows the model to learn non-linear decision boundaries, making it capable of solving more complex problems.
2. **Smooth Gradients:** The sigmoid function provides smooth gradients, which are essential for efficient training with gradient-based optimization methods like backpropagation. This makes it possible to adjust weights more effectively during training, improving the model's accuracy and performance.
3. **Squashing Outputs:** The sigmoid function outputs values between 0 and 1, which can be interpreted as probabilities. This makes it suitable for binary classification tasks, where the output represents the likelihood of a class belonging to a certain category.
4. **Gradient Descent Compatibility:** With a sigmoid function, the network can propagate errors back through multiple layers during training, which is necessary for training deep networks. This allows multi-layer networks (like MLPs) to learn complex patterns by adjusting weights through backpropagation.

In short, adding a sigmoid function enables the Perceptron to go beyond linear separability and solve more complex, real-world problems, making it a key building block for neural networks.

Example:

Code link:

<https://colab.research.google.com/drive/1XnxCPAeWrZqOIgKWDMipjlujNXQ2r3Su?usp=sharing>

Video 72:

Logistic Regression Part 4 | Loss Function | Maximum Likelihood | Binary Cross Entropy

Even with the sigmoid function, faults in graphs can occur due to issues like vanishing gradients. Sigmoid squashes outputs between 0 and 1, causing gradients to become very small during backpropagation, leading to slow or ineffective learning, especially in deep networks. This hampers model performance and convergence.

In **logistic regression**, we use **binary cross-entropy** as the loss function because it is designed for binary classification tasks, where the output is either 0 or 1.

Maximum Likelihood:

In logistic regression, we treat the parameters (weights) as unknowns and use the **Maximum Likelihood Estimation (MLE)** method to find the optimal parameters that maximize the likelihood of the observed data.

The likelihood for logistic regression is the probability of getting the actual labels based on the predicted probabilities, which is modeled using the logistic function.

Binary Cross-Entropy:

Binary Cross-Entropy is derived from the log-likelihood function for binary classification.

The logistic regression model outputs probabilities between 0 and 1, and **binary cross-entropy** computes the negative log-likelihood of the predicted probabilities compared to the true labels.

The formula for binary cross-entropy for a single prediction is: $L = -(y \log(p) + (1-y) \log(1-p))$ where:

y is the true label (0 or 1),

p is the predicted probability for class 1.

Why Binary Cross-Entropy in Logistic Regression?

- **Binary Cross-Entropy** is used because it measures how well the predicted probabilities match the actual labels.
- **Logistic regression** predicts probabilities, not hard labels, and binary cross-entropy is perfect for comparing probabilistic predictions with binary true labels.
- Minimizing this loss function during training helps the model improve its parameters (weights) to give more accurate predictions.

In summary, logistic regression uses **maximum likelihood estimation** to find the parameters that maximize the likelihood of the data, and **binary cross-entropy** is the loss function that corresponds to this likelihood maximization in the context of binary classification.

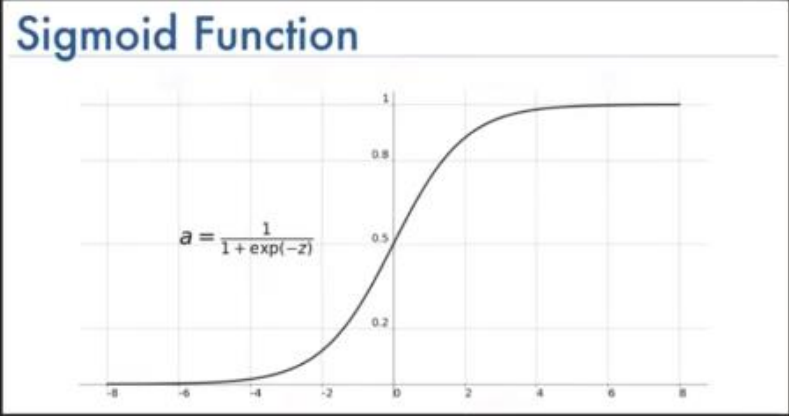
Video 73:

Derivative of Sigmoid Function

Derivative of Sigmoid

Thursday, June 17, 2021 12:20 PM

Sigmoid Function



The graph shows the Sigmoid Function, which is an S-shaped curve. The x-axis ranges from -8 to 8, and the y-axis ranges from 0 to 1. The curve passes through the point (0, 0.5). A label on the graph indicates the formula $a = \frac{1}{1 + \exp(-z)}$.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
$$\sigma'(x) = \frac{d}{dx} \left(\frac{1}{1 + e^{-x}} \right)$$
$$\frac{d}{dx} \left(\frac{1}{x} \right) = \frac{d}{dx} (x)^{-1}$$
$$= -x^{-2} = -\frac{1}{x^2}$$

$$\begin{aligned} \frac{d}{dx} \left[\frac{1}{1 + e^{-x}} \right] &= \frac{d}{dx} \left[(1 + e^{-x})^{-1} \right] = -\frac{1}{(1 + e^{-x})^2} \frac{d}{dx} (1 + e^{-x}) \\ &= -\frac{1}{(1 + e^{-x})^2} \frac{d}{dx} (e^{-x}) = -\frac{e^{-x}}{(1 + e^{-x})^2} \frac{d}{dx} (-x) = \frac{e^{-x}}{(1 + e^{-x})^2} \end{aligned}$$
$$\frac{1 \cdot e^{-x}}{(1 + e^{-x})(1 + e^{-x})} = \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = \sigma(x) \left[\frac{e^{-x}}{1 + e^{-x}} \right]$$
$$= \sigma(x) \left[\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right] = \sigma(x) \left[\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right]$$

$$= \sigma(x) \left[\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right] = \sigma(x) \left[\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right]$$
$$\sigma(x) [1 - \sigma(x)] \Rightarrow \sigma'(x) = \sigma(x) [1 - \sigma(x)]$$

Video 74:

Logistic Regression Part 5 | Gradient Descent & Code from Scratch

Example:

Code link:

<https://colab.research.google.com/drive/16QJz1KH2vazcGwllfTVA52Z67ea5ltuY?usp=sharing>

Video 75:

Accuracy and Confusion Matrix | Type 1 and Type 2 Errors | Classification Metrics Part 1

Accuracy in machine learning is a metric that measures the proportion of correctly predicted instances out of the total instances. It is calculated as the ratio of correct predictions to the total number of predictions, providing a general indication of model performance.

Confusion Matrix is a table used to evaluate the performance of classification models. It shows the number of correct and incorrect predictions broken down by each class, including True Positives, True Negatives, False Positives, and False Negatives, helping to assess the model's classification accuracy.

A "good" accuracy depends on the context of the problem and the specific dataset. In some cases, an accuracy of 70-80% can be considered good, especially with complex or noisy data. However, for imbalanced datasets or in critical applications (e.g., healthcare, finance), higher accuracy (above 90%) might be needed. It's important to also consider other metrics like precision, recall, and F1 score, especially if the classes are imbalanced. Thus, it depends on the context of the problem and data.

The main issue with accuracy is that it can be misleading, especially in cases of **class imbalance**. For example, in a dataset where 95% of instances belong to one class and only 5% to another, a model that always predicts the majority class will still achieve 95% accuracy, despite failing to detect the minority class. This makes accuracy an unreliable metric in such situations, and other metrics like **precision**, **recall**, and the **F1 score** should be considered for a more accurate assessment of model performance.

Confusion Matrix

Tuesday, June 22, 2021 1:12 PM



		Prediction	
		1	0
Actual	1	True Positive (TP)	False Negative (FN)
	0	False Positive (FP)	True Negative (TN)

Type 1 Error (False Positive): This occurs when a model incorrectly predicts a positive outcome when the true outcome is negative. In other words, the model wrongly classifies a negative instance as positive.

Type 2 Error (False Negative): This occurs when a model incorrectly predicts a negative outcome when the true outcome is positive. In other words, the model misses identifying a positive instance and classifies it as negative.

In simpler terms:

- **Type 1** = "False alarm" (predicts positive when it's actually negative).
- **Type 2** = "Miss" (predicts negative when it's actually positive).

Accuracy can be misleading in the following situations:

1. **Class Imbalance:** When one class dominates the dataset, a model can achieve high accuracy by simply predicting the majority class. For instance, in a dataset where 95% of the samples belong to one class, a model predicting only that class will still reach 95% accuracy, but it won't correctly classify the minority class.
2. **Irrelevant Metrics:** When the problem requires more nuanced predictions (e.g., in healthcare or fraud detection), high accuracy might not reflect the model's ability to identify critical patterns, even if overall accuracy is high.
3. **Skewed Cost of Errors:** In some applications (like spam detection or medical diagnoses), false negatives (Type 2 errors) or false positives (Type 1 errors) may have very different consequences, so accuracy alone doesn't reflect the model's performance effectively.

In these cases, metrics like **precision**, **recall**, **F1 score**, and the **confusion matrix** offer a better understanding of how well the model performs across all classes.