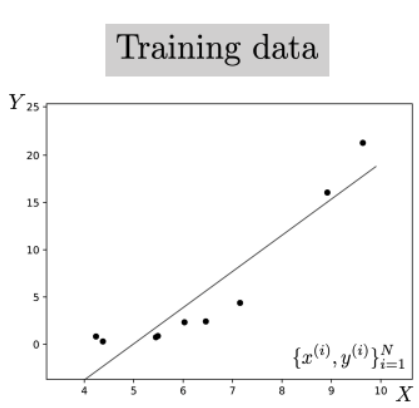
CMSC 491/691: Computer Vision **Fall 2023**

Lecture 7: 2024-02-28 ML-for-CV II (Neural Networks)

Lecturer: Tejas Gokhale *Scribe: Jedan Davis*

7.1 Linear Regression

In linear regression, we aim to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. However, in some cases, a simple linear model may not capture the underlying relationships adequately.



$$f_\theta(x) = \theta_0 + \theta_1 x$$

Figure 7.1: This figure depicts a linear regression model fitted to scattered data points. The straight line represents the best-fitting equation that minimizes the squared errors between the data points and the predicted values.

7.1.1 Underfitting

In contrast, underfitting occurs when a regression model is too simplistic and fails to capture the underlying relationships between features and the target variable. This leads to poor performance on both the training and test datasets.

7.1.1.1 Manifestations of Underfitting in Regression

1. High bias: An underfitted model exhibits high bias, making oversimplified assumptions about the relationships between variables.

2. Inaccurate predictions: These oversimplified assumptions lead to inaccurate predictions on both the training and test data.

3. Flattened trendlines: Underfitted models often produce trendlines that appear as flat lines or straight lines, failing to capture the non-linear patterns present in the data.

4. Poor performance metrics: Metrics like R-squared and mean squared error (MSE) reflect the poor fit of the model, indicating high errors in predictions.

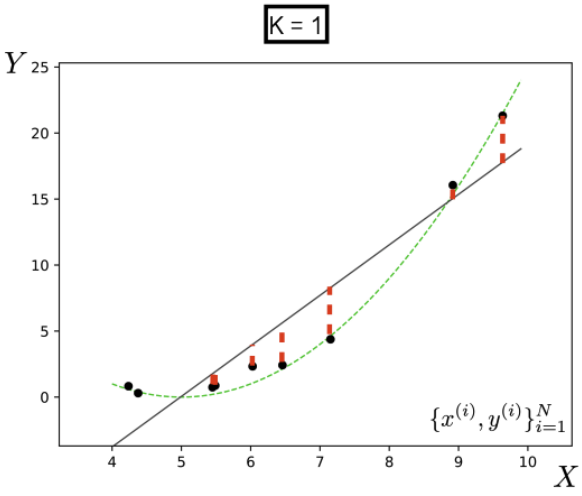
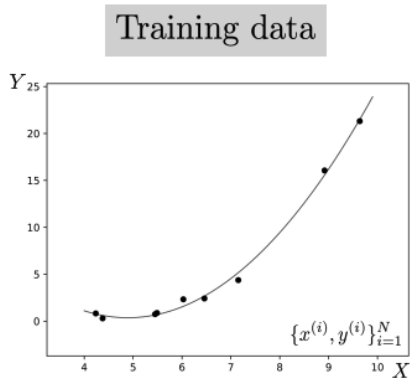


Figure 7.2: When the model does not have the capacity to capture the true function, we call this underfitting.

7.2 Polynomial Regression

Polynomial regression, which fits a polynomial equation to the data, often provides a better fit by allowing for more flexibility in the regression line. By increasing the degree of the polynomial (adding more basis functions), the regression line becomes more flexible and can better capture the nuances in the data.



$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k$$

K-th degree polynomial regression

Figure 7.3: This figure shows a polynomial regression model fitted to scattered data points. The curved line represents the best-fitting polynomial function that captures the non-linear relationship between the independent variable (x-axis) and the dependent variable (y-axis).

7.2.1 Overfitting

Overfitting occurs when a regression model becomes too complex and starts capturing noise in the training data instead of the true underlying relationships between variables. This results in a model that performs exceptionally well on the training data but fails to generalize to new, unseen data.

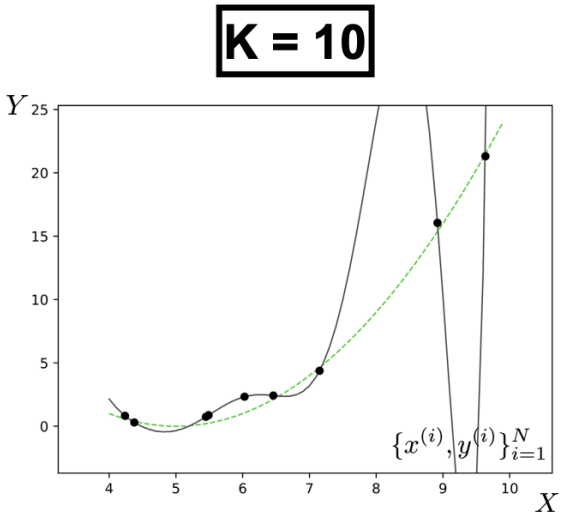


Figure 7.4: It occurs when we have too high capacity a model, e.g., too many free parameters, too few data points to pin these parameters down.

7.2.2 Performance Calculation

To evaluate the performance of a regression model, we typically calculate the error between the estimated and actual values for all data points and find the average error. This error metric helps us understand how

well the model predicts the target variable across the entire dataset.

7.2.3 IID Assumption

IID (Independent and Identically Distributed) refers to the assumption that the samples in both the training and test datasets are drawn from the same probability distribution and are mutually independent. This assumption is crucial for ensuring that machine learning models generalize well to unseen data. Violations of the IID assumption can lead to poor model performance in real-world applications.

7.3 Parametric Approach and Linear Classifiers

In a parametric approach, linear classifiers are used to classify data points into different categories based on a linear decision boundary. The bias parameter (often denoted as "b") helps adjust the position of this decision boundary.

7.3.1 Limitations of Linear Classifiers

One significant limitation of linear classifiers is their inability to classify data points belonging to the XOR function correctly. The XOR function represents a non-linear relationship between input features and target labels, which linear classifiers fail to capture effectively.

7.4 The Perceptron: A Building Block for Neural Networks

The perceptron, introduced by Frank Rosenblatt in 1957, is a fundamental unit in the field of artificial neural networks. It serves as the simplest model of an artificial neuron and lays the groundwork for more complex neural network architectures.

A perceptron takes a weighted sum of its inputs, applies an activation function to the sum, and produces a single output. Mathematically, the output (y) of a perceptron can be expressed as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{7.1}$$

where:

- x_i represents the i-th input value.
- w_i represents the weight associated with the i-th input.
- b represents the bias term.
- f is the activation function, which introduces non-linearity into the model. Common activation functions include the step function, sigmoid function, and ReLU (Rectified Linear Unit).

The perceptron can be trained using an algorithm called the perceptron learning rule. This algorithm iteratively adjusts the weights of the perceptron to minimize the error between the desired output and the actual output for a given set of training data.

Despite its limitations (e.g., inability to learn non-linearly separable patterns), the perceptron played a crucial role in the early development of artificial neural networks. It paved the way for more sophisticated architectures that have revolutionized various fields, including machine learning, computer vision, and natural language processing.

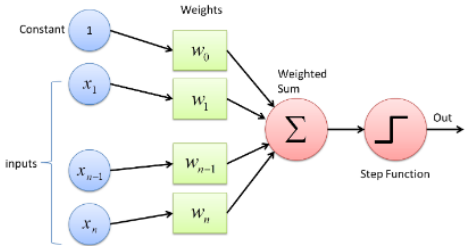


Figure 7.5: Perceptron: A basic artificial neuron receiving weighted inputs, applying an activation function, and generating an output.

Source: [The Definitive Perceptron Guide](https://towardsdatascience.com/the-definitive-perceptron-guide-fd384eb93382)

7.5 Linear Layer

We start by considering a 1D input vector x that we want to transform into a new feature space. A linear layer performs a weighted sum of the input elements, followed by adding a bias term. Here's the mathematical notation:

$$y_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \tag{7.2}$$

where:

- y_j is the j-th element of the output vector.
- w_{ij} is the weight connecting the i-th input element to the j-th output element.
- x_i is the i-th element of the input vector.
- b_j is the bias term for the j-th output element.

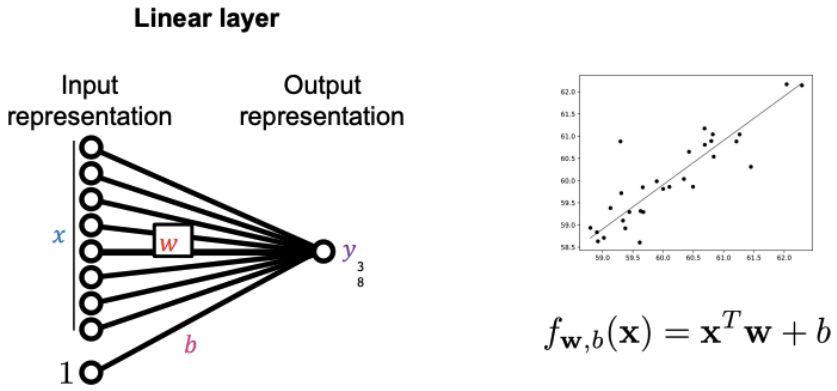


Figure 7.6: A visual representation of a linear layer, showing the weights and bias term.

7.6 Full Layer

A full layer, also known as a densely connected layer, takes an input vector x and transforms it into an output vector y. Each element in the output vector is calculated by performing a linear combination of the input elements, followed by adding a bias term. Mathematically, this can be expressed as:

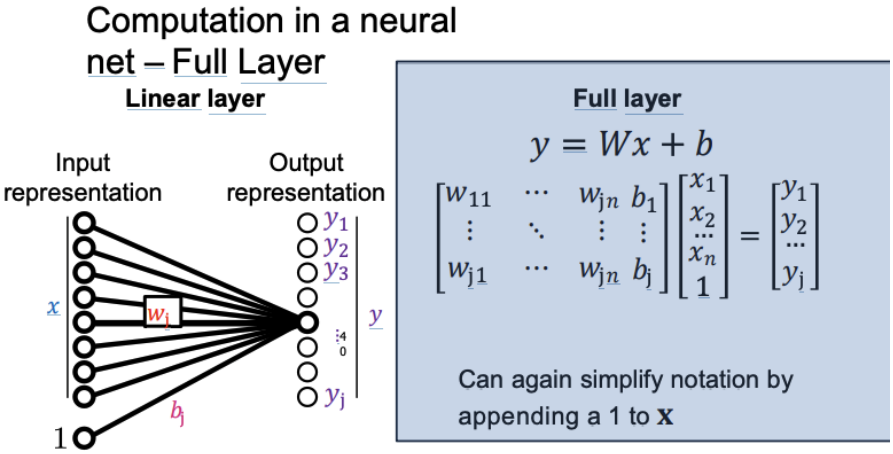
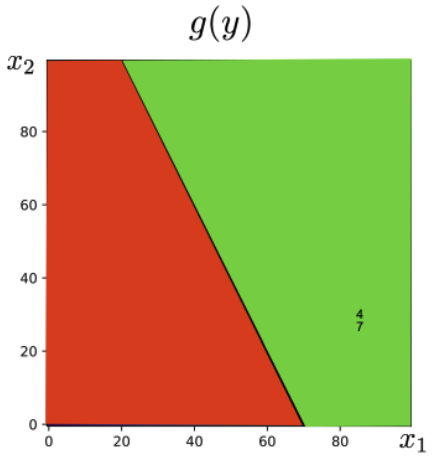


Figure 7.7: Computation in a neural net – Full Layer

7.7 Non-linearity

While linear layers are powerful tools, they have a significant limitation: they can only represent linear relationships between the input and output. This limitation becomes apparent when considering problems like classifying data that cannot be separated by a straight line.

To overcome this limitation, we introduce non-linear activation functions. These functions transform the linear output of a layer before passing it to the next layer. This allows the network to learn complex, non-linear relationships between the input and output.



$$y = \mathbf{x}^T\mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Figure 7.8: linear classification with a perceptron

7.8 Non-linear Activation Functions: ReLU and Leaky ReLU

These functions transform the output of a layer before passing it to the next layer, allowing the network to learn complex, non-linear relationships.

The ReLU (Rectified Linear Unit) is a popular non-linear activation function.

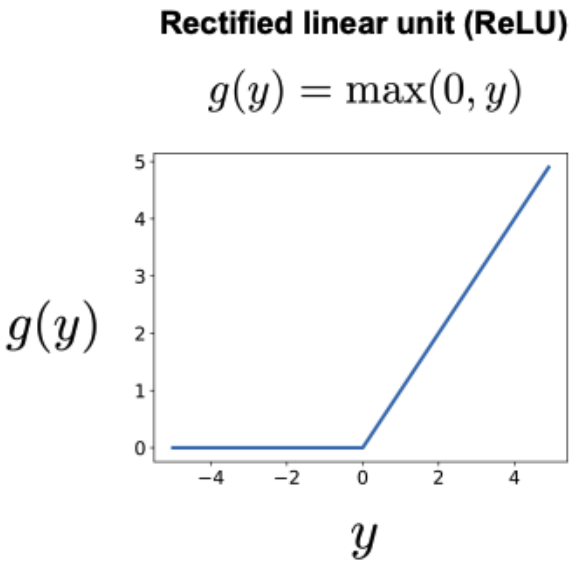
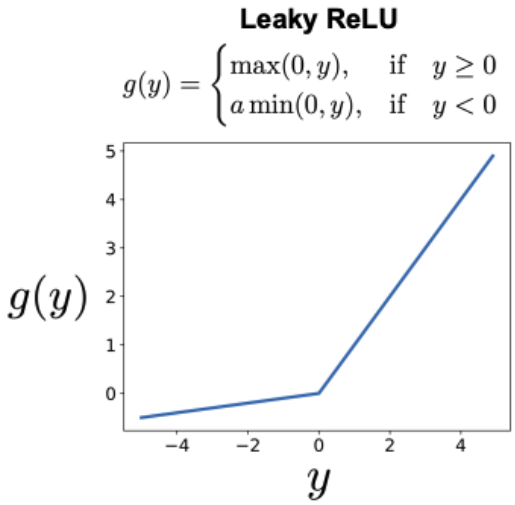


Figure 7.9: ReLU and Leaky ReLU Activation Functions

However, ReLU also has a drawback: Dead neurons. Leaky ReLU addresses this issue by introducing a small positive slope for negative inputs.



Both ReLU and Leaky ReLU are popular non-linear activation functions used in neural networks. The choice between these functions often depends on the specific task and network architecture.

7.9 Stacking Layers

By stacking multiple layers with appropriate hidden units, non-linear activation functions, and connectivity patterns, neural networks can progressively extract higher-level features from the input data. The first layers learn basic features, while subsequent layers learn increasingly complex combinations of these features, ultimately leading to the desired output representation.