Minor in AI

Neural Networks

Activation functions & Propagation

December 30, 2024

1 Introduction

Neural networks are computational models inspired by the human brain. This document introduces key concepts in neural networks, with case studies and minimal mathematics. Each math expression is preceded by a clear explanation.

2 Neuron: Basic Building Block

A neuron performs two main operations:

1. **Net Operation**: Computes the weighted sum of inputs.
2. **Out Operation**: Applies an activation function to produce the output.

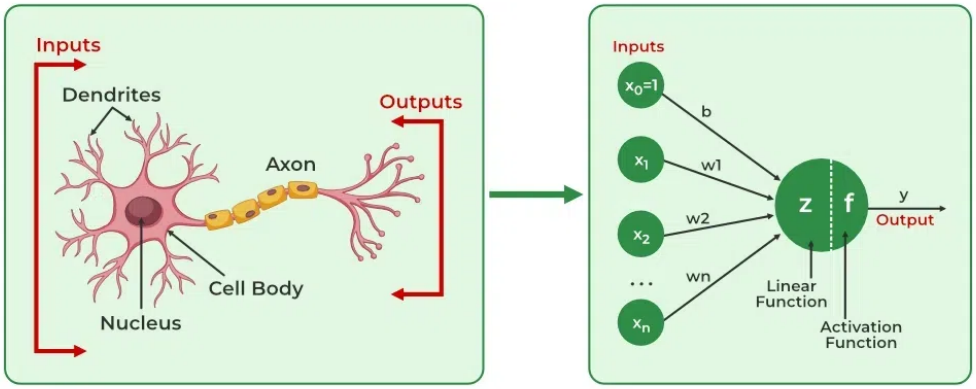

Figure 1: Neuron

2.1 Components of a Neuron

The core elements of a neuron are:

- **Inputs** ($x_1, x_2, \ldots, x_n$): Values that represent features or outputs from the previous layer.
- **Weights** ($w_1, w_2, \ldots, w_n$): Coefficients that signify the importance of each input.
- **Bias** (b): An additional value that shifts the weighted sum to enhance flexibility.
- **Activation Function** (f): A function that introduces non-linearity to the model, enabling it to learn complex patterns.
- **Output** (y): The final result computed by applying the activation function to the weighted sum of inputs and bias.

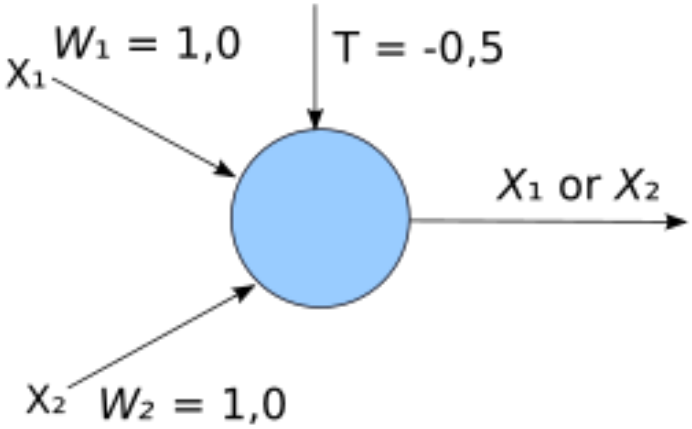

Figure 2: Components of a Neuron

2.2 Mathematical Representation

To compute the output of a neuron, first calculate the weighted sum of inputs and bias:

$$\text{Weighted Sum (Net Output)} = \sum_{i=1}^{n} w_i x_i + b$$

This value is then passed through the activation function:

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

In words: The neuron's output is the result of applying the activation function f to the sum of the weighted inputs ($w_i \times x_i$) and the bias (b).

2.3 Example

Consider a neuron with:

- Inputs: $x_1 = 2$, $x_2 = 3$,
- Weights: $w_1 = 0.5$, $w_2 = 0.8$,
- Bias: $b = 1$,
- Activation Function: Sigmoid ($f(x) = \frac{1}{1+e^{-x}}$).

First, calculate the weighted sum:

$$\text{Net Output} = (0.5 \times 2) + (0.8 \times 3) + 1 = 4.9$$

Then apply the activation function:

$$y = f(4.9) = \frac{1}{1 + e^{-4.9}} \approx 0.9927$$

Thus, the neuron's output is approximately 0.9927.

3 Structure of a Neural Network

A neural network consists of three main layers:

- **Input Layer**: Accepts raw data and passes it to the next layer.
- **Hidden Layer(s)**: Processes inputs using weights and biases, and learns patterns.
- **Output Layer**: Produces the final predictions.

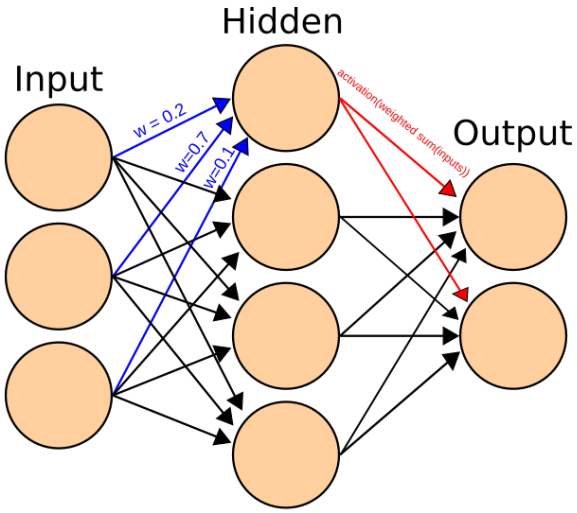

Figure 3: Structure of an Artificial Neural Network

3.1 Example: Single Hidden Layer Network

Consider a network with:

- 2 input neurons (x_1, x_2),
- 2 hidden neurons (h_1, h_2),
- 2 output neurons (o_1, o_2).

Each neuron performs two steps:

1. Compute the weighted sum of inputs (net operation).
2. Apply the activation function to get the output (out operation).

4 Forward and Backward Propagation

4.1 Key Equations and Explanations

The weight update process in backpropagation is governed by the following core formula:

$$w_{\text{new}} = w - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w}$$

This equation describes how weights are updated to reduce the total error. Here: - w_{new} is the updated weight. - w is the current weight. - η is the learning rate, a factor that determines the step size for weight updates. - $\frac{\partial E_{\text{total}}}{\partial w}$ is the gradient of the total error with respect to the weight w.

The gradient $\frac{\partial E_{\text{total}}}{\partial w}$ is calculated using the chain rule of differentiation:

$$\frac{\partial E_{\text{total}}}{\partial w} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{\text{node}}} \cdot \frac{\partial \text{out}_{\text{node}}}{\partial \text{net}_{\text{node}}} \cdot \frac{\partial \text{net}_{\text{node}}}{\partial w}$$

Here: - $\frac{\partial E_{\text{total}}}{\partial \text{out}_{\text{node}}}$: Represents how the total error changes with respect to the output of the node. - $\frac{\partial \text{out}_{\text{node}}}{\partial \text{net}_{\text{node}}}$: Captures the sensitivity of the node's output to its input (typically the derivative of the activation function). - $\frac{\partial \text{net}_{\text{node}}}{\partial w}$: Represents how the weighted sum of inputs changes with respect to the weight being updated.

4.2 Overview

Forward propagation involves passing inputs through the network to compute the outputs. Backward propagation adjusts the weights in the network to minimize the error between the predicted and target outputs. Backpropagation is the fundamental algorithm for training artificial neural networks. It utilizes gradient descent to iteratively minimize the error function by adjusting weights. This iterative process involves two primary steps:

1. **Forward Pass**: Compute the outputs based on the current weights.
2. **Backward Pass**: Calculate gradients (the rate of change of the error) and propagate errors backward using the chain rule of differentiation.

The ultimate goal of backpropagation is to adjust the weights in such a way that the error between the predicted output and the actual target is minimized.

4.3 Forward Pass

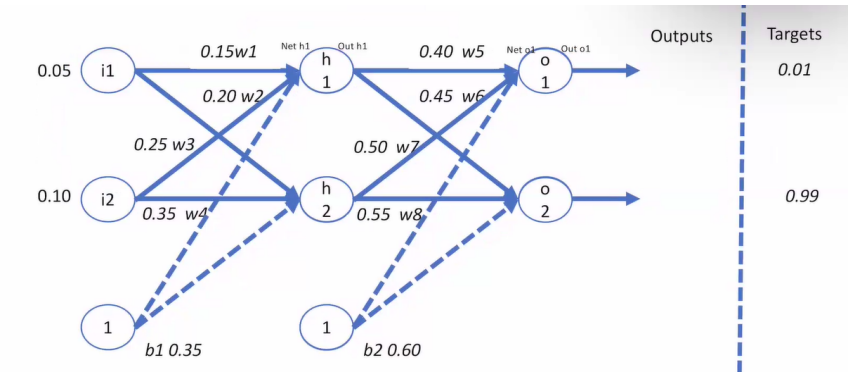

Figure 4: Example of Forward Propagation in a Neural Network

The forward pass is the process of computing the outputs of a neural network given the inputs and the current weights. Here's how it's done:

1. **Weighted Sum of Inputs**: For a neuron, the net input is calculated as the weighted sum of the inputs to the neuron, plus a bias term:

$$\text{net}_j = \sum_i w_{ij} x_i + b_j$$

where:

- w_{ij} are the weights associated with the input x_i,
- b_j is the bias term for neuron j,
- x_i are the input features.

2. **Activation Function**: The output of the neuron is then obtained by applying an activation function to the net input. Common activation functions include:

- **Sigmoid**:

$$f(\text{net}_j) = \frac{1}{1 + e^{-\text{net}_j}}$$

The sigmoid function squashes the output to a range between 0 and 1.

For example, the output of a neuron can be expressed as:

$$y_j = f(\text{net}_j)$$

where y_j is the output of neuron j.

4.4 Backward Pass

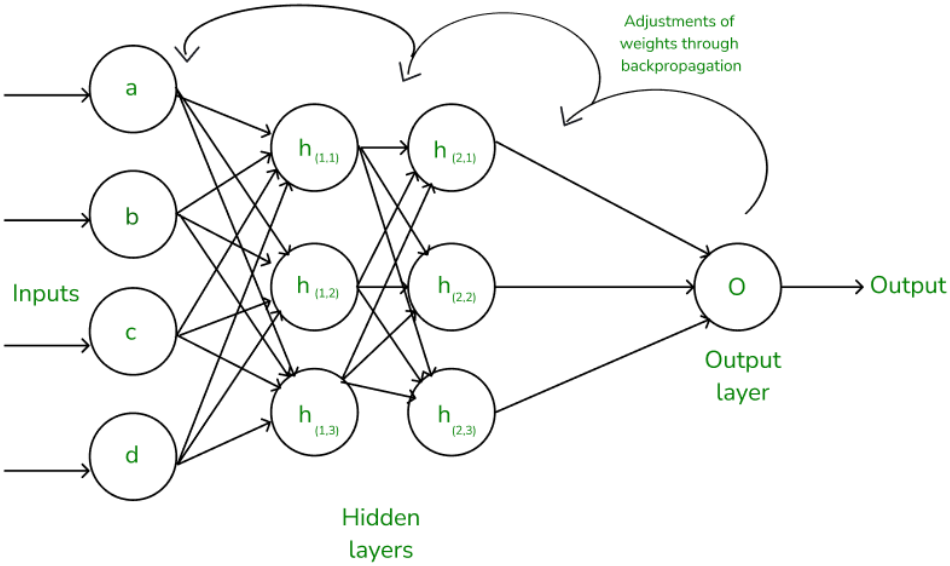

Figure 5: Backward Propagation in an Artificial Neural Network

In the backward pass, we calculate how much each weight in the network contributed to the overall error, and adjust the weights accordingly.

1. **Error Calculation**: The error for each output neuron is computed using the Mean Squared Error (MSE) formula:

$$E_j = \frac{1}{2}(t_j - y_j)^2$$

where:

- t_j is the target output,
- y_j is the predicted output from the forward pass.

This error measures how far the predicted output y_j is from the desired target output t_j.

2. **Total Error**: The total error of the network across all output neurons is the sum of the individual errors:

$$E_{\text{total}} = \sum_j E_j$$

3. **Gradient Calculation**: To adjust the weights, we need to compute the gradient of the total error with respect to each weight. Using the chain rule of differentiation, the gradient of the total error with respect to a weight w_{ij} is:

$$\frac{\partial E_{\text{total}}}{\partial w_{ij}} = \frac{\partial E_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}}$$

Each term is computed as follows:

- $\frac{\partial E_j}{\partial y_j} = -(t_j - y_j)$,
- $\frac{\partial y_j}{\partial \text{net}_j} = f'(\text{net}_j)$, where $f'(\text{net}_j)$ is the derivative of the activation function (e.g., for the sigmoid, $f'(\text{net}_j) = y_j(1 - y_j)$),
- $\frac{\partial \text{net}_j}{\partial w_{ij}} = x_i$, the input to the neuron.

4. **Weight Update**: The weights are then updated using the gradient descent algorithm:

$$w_{ij}^{\text{new}} = w_{ij} - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_{ij}}$$

where η is the learning rate, a hyperparameter that controls the step size of the weight update.

4.5 Worked Example

Let's consider a network with a single weight w_5. Assume:

$$w_5 = 0.40, \quad \eta = 0.5, \quad \frac{\partial E_{\text{total}}}{\partial w_5} = 0.082167$$

Using the weight update formula:

$$w_5^{\text{new}} = w_5 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_5}$$

Substituting the values:

$$w_5^{\text{new}} = 0.40 - 0.5 \cdot 0.082167 = 0.358916$$

Thus, the new weight becomes $w_5^{\text{new}} = 0.358916$.

4.6 Detailed Gradient Derivation

To compute the gradient $\frac{\partial E_{\text{total}}}{\partial w_5}$, we sum the contributions from all output nodes connected to w_5:

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_1}{\partial w_5} + \frac{\partial E_2}{\partial w_5}$$

For a single error term E_j, the chain rule gives:

$$\frac{\partial E_j}{\partial w_{ij}} = \frac{\partial E_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}}$$

4.6.1 Component Breakdown

1. **Error to Output**:
$$\frac{\partial E_j}{\partial y_j} = -(t_j - y_j)$$
This represents how much the error changes with respect to the output of the neuron.

2. **Output to Net Input**:
$$\frac{\partial y_j}{\partial \text{net}_j} = f'(\text{net}_j)$$
For the sigmoid activation function:
$$f'(\text{net}_j) = y_j(1 - y_j)$$
This represents the rate of change of the output with respect to the net input.

3. **Net Input to Weight**:
$$\frac{\partial \text{net}_j}{\partial w_{ij}} = x_i$$
This represents how much the net input changes with respect to the weight.

4.7 Forward Propagation Example

Let's consider a simple neural network with inputs, weights, biases, and target outputs. The network consists of an input layer, a hidden layer, and an output layer.

4.7.1 Step 1: Compute Hidden Layer Outputs

To calculate the outputs of the neurons in the hidden layer, we first compute the weighted sum of inputs for each neuron in the hidden layer. For the first hidden layer neuron h_1, the weighted sum is calculated as:

$$\text{net}_{h_1} = w_1 \cdot i_1 + w_2 \cdot i_2 + b_1$$

where w_1, w_2 are the weights, i_1, i_2 are the input values, and b_1 is the bias term. The output of the hidden neuron is then computed using the sigmoid activation function:

$$\text{Out}_{h_1} = \frac{1}{1 + e^{-\text{net}_{h_1}}}$$

Similarly, for the second hidden layer neuron h_2, the output is calculated in the same manner:

$$\text{Out}_{h_2} = \frac{1}{1 + e^{-\text{net}_{h_2}}}$$

4.7.2 Step 2: Compute Output Layer Outputs

After computing the hidden layer outputs, we move on to the output layer. The output layer neurons compute a weighted sum of the outputs from the hidden layer neurons. For output neuron o_1, the weighted sum is:

$$\text{net}_{o_1} = w_5 \cdot \text{Out}_{h_1} + w_6 \cdot \text{Out}_{h_2} + b_2$$

The output is then computed using the sigmoid function:

$$\text{Out}_{o_1} = \frac{1}{1 + e^{-\text{net}_{o_1}}}$$

Similarly, for the second output neuron o_2, the output is computed as:

$$\text{Out}_{o_2} = \frac{1}{1 + e^{-\text{net}_{o_2}}}$$

4.8 Error Calculation

The total error is calculated using the Mean Squared Error (MSE) function:

$$E_{\text{total}} = \frac{1}{2} \sum_j (t_j - y_j)^2$$

where t_j is the target output and y_j is the predicted output. This error guides the weight update in the backward pass.

4.9 Summary

- Backpropagation computes gradients using the chain rule and updates weights iteratively.
- The forward pass calculates the outputs; the backward pass calculates gradients and propagates errors.
- The weight update formula combines gradients with a learning rate for iterative improvement.
- Careful tuning of the learning rate ensures optimal convergence.

5 Worked Example: Forward Pass and Backward Pass

This section provides a detailed worked example of the forward pass and backward pass in a simple neural network.

5.1 Problem Setup

Given the following parameters:

- Input value: $x = 0.5$

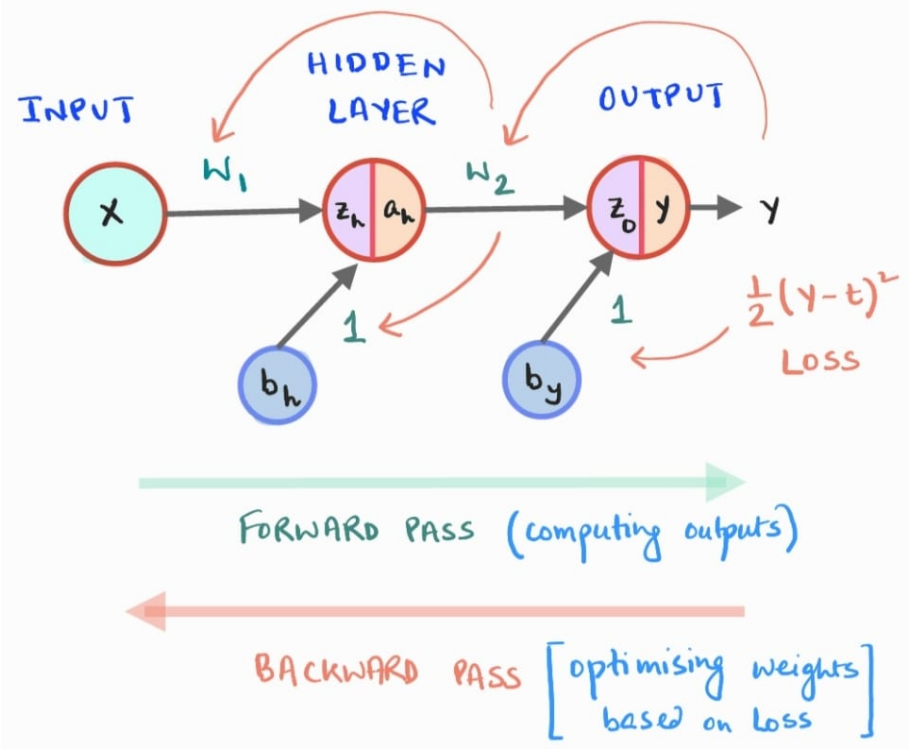

Figure 6: Illustration of FP and BP

- Weight from input to hidden layer: $w_1 = 0.4$
- Bias for hidden layer: $b_h = 0.1$
- Weight from hidden layer to output layer: $w_2 = 0.7$
- Bias for output layer: $b_y = -0.2$
- Target output: $t = 0.6$

5.2 Forward Pass

The forward pass consists of the following steps:

5.2.1 Step 1: Compute the Hidden Layer Output

The net input to the hidden layer (net_h) is calculated as:

$$z_h = w_1 \cdot x + b_h$$

Substituting the values:

$$z_h = 0.4 \cdot 0.5 + 0.1 = 0.2 + 0.1 = 0.3$$

The output of the hidden layer (a_h) is obtained using the sigmoid activation function:

$$a_h = \frac{1}{1 + e^{-\text{net}_h}}$$

Substituting $\text{net}_h = 0.3$:

$$a_h = \frac{1}{1 + e^{-0.3}} \approx 0.5744$$

5.2.2 Step 2: Compute the Output Layer Value

The net input to the output layer (z_o) is calculated as:

$$z_o = w_2 \cdot a_h + b_y$$

Substituting the values:

$$z_o = 0.7 \cdot 0.5744 - 0.2 \approx 0.4021 - 0.2 = 0.2021$$

The output of the network (y) is obtained using the sigmoid activation function:

$$y = \frac{1}{1 + e^{-z_o}}$$

Substituting $z_o = 0.2021$:

$$y = \frac{1}{1 + e^{-0.2021}} \approx 0.5504$$

5.3 Backward Pass

The backward pass involves calculating gradients and updating weights.

5.3.1 Step 1: Compute the Output Error

The error (E) for the output layer is given by:

$$E = \frac{1}{2}(y - t)^2$$

Substituting $t = 0.6$ and $y = 0.5504$:

$$E = \frac{1}{2}(0.6 - 0.5504)^2 = \frac{1}{2}(0.0496)^2 \approx 0.00123$$

5.3.2 Step 2: Compute Gradients for Output Layer

The gradient of the error with respect to the output ($\frac{\partial E}{\partial y}$) is:

$$\frac{\partial E}{\partial y} = -(t - y)$$

Substituting $t = 0.6$ and $y = 0.5504$:

$$\frac{\partial E}{\partial y} = -(0.6 - 0.5504) = -0.0496$$

The gradient of the output with respect to the net input to the output layer ($\frac{\partial y}{\partial z_o}$) is:

$$\frac{\partial y}{\partial z_o} = y \cdot (1 - y)$$

Substituting $y = 0.5504$:

$$\frac{\partial y}{\partial z_o} = 0.5504 \cdot (1 - 0.5504) \approx 0.2475$$

The gradient of the error with respect to the net input to the output layer is:

$$\frac{\partial E}{\partial z_o} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z_o}$$

Substituting values:

$$\frac{\partial E}{\partial z_o} = -0.0496 \cdot 0.2475 \approx -0.01228$$

5.3.3 Step 3: Update the Weight w_2

The gradient of the error with respect to w_2 is:

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \mathrm{z}_o} \cdot \frac{\partial \mathrm{z}_o}{\partial w_2}$$

Substituting $\frac{\partial \mathrm{z}_o}{\partial w_2} = a_h \approx 0.5744$:

$$\frac{\partial E}{\partial w_2} = -0.01228 \cdot 0.5744 \approx -0.00705$$

The updated weight w_2 is:

$$w_2^{\text{new}} = w_2 - \eta \cdot \frac{\partial E}{\partial w_2}$$

Using a learning rate $\eta = 0.1$:

$$w_2^{\text{new}} = 0.7 - 0.1 \cdot (-0.00705) \approx 0.7007$$

Similarly, all the parameters such as w_1, b_1 and b_2 will also get updated based on their gradient as a function of Error E.

5.4 Summary of Results

The final results after performing the forward pass and backward pass are as follows:

- Hidden layer output (a_h): 0.5744
- Network output (y): 0.5504
- Error (E): 0.00123
- Gradient for weight w_2: -0.00705
- Updated weight w_2^{new}: 0.7007

6 Drawbacks of Neural Networks

6.1 Lack of Invariance to Shifting and Other Forms of Distortion

One major limitation of neural networks, especially in traditional architectures, is their inability to be invariant to certain transformations of the input data, such as shifting or other forms of distortion. This means that small changes or shifts in the input data may cause significant changes in the output.

For instance, if a neural network is trained to recognize an object in an image, a slight shift in the position of the object could result in a misclassification. While some models, such as Convolutional Neural Networks (CNNs), partially address this issue, basic neural networks do not inherently possess this capability.

Example: If a neural network is trained to recognize a cat in an image and the cat is moved slightly to the left or right in the input image, the network may not recognize the cat as the same object anymore, leading to an incorrect prediction.

6.2 Large Number of Trainable Parameters

Another drawback is that as the network's architecture becomes more complex (e.g., more layers and neurons), the number of trainable parameters (weights and biases) increases rapidly. This can make the training process computationally expensive, slow, and prone to overfitting, especially when there is insufficient training data.

For example, if a neural network has 1000 neurons in the input layer, 100 neurons in the hidden layer, and 10 neurons in the output layer, the number of parameters will be:

$$\text{Number of parameters} = (1000 \times 100) + (100 \times 10) + 100 + 10 = 101,110 \text{ parameters}$$

Training such a network requires a lot of memory and computational resources, which can make it difficult to scale to larger datasets or more complex tasks.

7 Epochs and Iterations in Neural Network Training

7.1 What is an Epoch?

An epoch refers to one complete pass through the entire training dataset. In one epoch, every training sample has been used once to update the weights. The purpose of multiple epochs is to allow the network to learn and adjust its parameters iteratively to minimize the error over time.

Example: Suppose we have a dataset of 1000 images, and during one epoch, the network processes all 1000 images once to adjust the weights. If the training process involves 10 epochs, then the network will process the 1000 images 10 times.

7.2 What is an Iteration?

An iteration refers to one update of the model's weights during training. This update occurs after processing a batch of data. If the dataset is divided into smaller subsets (batches), each time a batch is processed and the weights are updated, it counts as one iteration.

Example: Let's say we have a dataset of 1000 images and we use a batch size of 100. In this case, there will be 10 iterations per epoch (1000 images $\div$ 100 images per batch = 10 iterations).

7.3 Difference Between Epoch and Iteration

The key difference is that an epoch refers to a complete pass through the entire dataset, while an iteration refers to a single update step based on a subset (batch) of the data.

- **Epoch**: One complete pass through the entire training dataset.
- **Iteration**: One update step based on a batch of data.

Example for Epoch and Iteration: If we have 1000 training examples and a batch size of 100:

- **1 epoch**: Process all 1000 images.

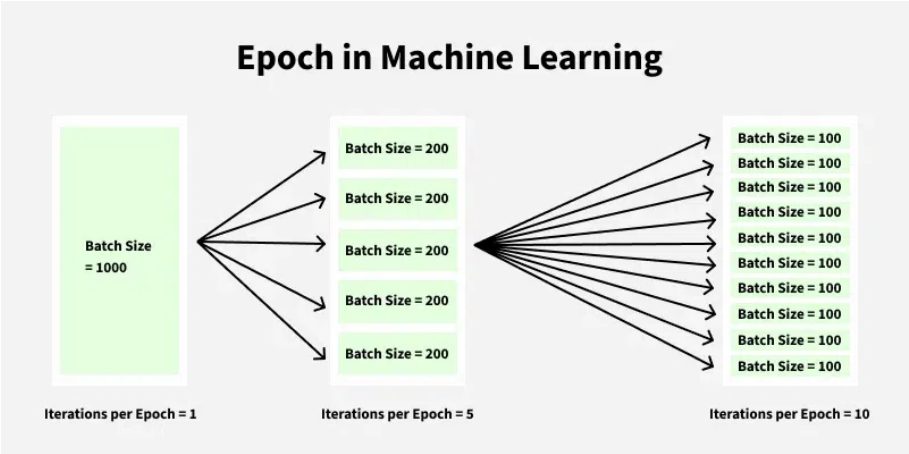

Figure 7: Epoch vs Iteration

- **10 iterations**: Process the images in 10 batches (100 images each) and update the weights after each batch.

Thus, for 1 epoch, there will be 10 iterations, and after completing all 10 iterations, the epoch is complete. This process repeats for several epochs to improve the model's accuracy.

Additional Resources

[1] Fei-Fei Li, *Lecture 4: Backpropagation and Neural Networks*, Stanford University

[2] Michael Nielsen, *Neural Networks and Deep Learning, Chapter 2*, Lambda

[3] Department of CSE, *Activation Functions and Their Derivatives*, IIT Guwahati

[4] Fei-Fei Li, *Lecture 06: Training Neural Networks, Part I*, Stanford University