Neural Networks: Forward Propagation, Backpropagation, and Gradient Descent

Neural Networks (NNs), also known as Artificial Neural Networks (ANNs), are a powerful class of machine learning models inspired by the structure and function of the human brain. They are designed to recognize patterns, learn from data, and make predictions, making them fundamental to deep learning. This document will detail the core components of neural networks and their fundamental training algorithms: Forward Propagation, Loss Functions, Gradient Descent, and Backpropagation.

Core Concepts of Neural Networks:

- **Neuron (Perceptron):** The fundamental building block. It receives inputs, computes a weighted sum of these inputs, adds a bias, and then applies an activation function to produce an output.
- **Inputs (x):** Features fed into the network. For a batch of m samples and n_0 input features, $\mathbf{X} \in \mathbb{R}^{n_0 \times m}$.
- **Weights (W):** Parameters that determine the strength of the connection between neurons. Each connection has an associated weight. For a layer l with n_l neurons receiving inputs from n_{l-1} neurons, $\mathbf{W}^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$.
- **Bias (b):** A parameter added to the weighted sum of inputs, allowing the activation function to be shifted. It helps the neuron fire even when all inputs are zero. For a layer l with n_l neurons, $\mathbf{b}^{[l]} \in \mathbb{R}^{n_l \times 1}$.
- **Activation Function (f):** A non-linear function applied to the weighted sum of inputs plus bias (z). It introduces non-linearity, enabling the network to learn complex patterns and make non-linear decisions. Common activation functions include:
  - **Sigmoid:** $f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$. Outputs values between 0 and 1. Used for binary classification in the output layer.
  - **Rectified Linear Unit (ReLU):** $f(z) = \max(0, z)$. Simple, computationally efficient, and widely used in hidden layers.
  - **Hyperbolic Tangent (Tanh):** $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. Outputs values between -1 and 1.
- **Layers:**
  - **Input Layer ($l = 0$):** Receives the raw data $\mathbf{A}^{[0]} = \mathbf{X}$.
  - **Hidden Layers ($l = 1, \ldots, L-1$):** Layers between the input and output layers where the bulk of computation and pattern recognition occurs. Deep neural networks have multiple hidden layers.
  - **Output Layer ($l = L$):** Produces the network's prediction or classification $\hat{\mathbf{Y}} = \mathbf{A}^{[L]}$. The number of neurons and activation function depend on the task (e.g., 1 neuron with sigmoid for binary classification, multiple neurons with softmax for multi-class classification).

Forward Propagation: Mathematical Details

Forward propagation is the process of computing the output of the network given an input. For each layer l (from input to output), we compute two quantities: the pre-activation Z and the activation A.

For a single data sample $\mathbf{x}$ and a single neuron j in layer l:

$$z_j^{[l]} = \left(\sum_k W_{jk}^{[l]} a_k^{[l-1]} \right) + b_j^{[l]}$$
$$a_j^{[l]} = f(z_j^{[l]})$$

Where:

- $a_k^{[l-1]}$ is the activation (output) of neuron k in the previous layer $(l-1)$. For the input layer ($l = 0$), $\mathbf{A}^{[0]} = \mathbf{X}$.

- $W_{jk}^{[l]}$ is the weight connecting neuron k in layer $l-1$ to neuron j in layer l.
- $b_j^{[l]}$ is the bias for neuron j in layer l.
- $z_j^{[l]}$ is the weighted sum of inputs plus bias for neuron j in layer l (pre-activation).
- $a_j^{[l]}$ is the activation (output) of neuron j in layer l.
- $f(\cdot)$ is the activation function.

For an entire layer l with a mini-batch of m samples (vectorized form):

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]}\mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} \tag{1}$$

$$\mathbf{A}^{[l]} = f(\mathbf{Z}^{[l]}) \tag{2}$$

Here, $\mathbf{Z}^{[l]} \in \mathbb{R}^{n_l \times m}$, $\mathbf{W}^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{A}^{[l-1]} \in \mathbb{R}^{n_{l-1} \times m}$, and $\mathbf{b}^{[l]} \in \mathbb{R}^{n_l \times 1}$ (the bias vector is broadcast across all m samples). The final output of the network is $\hat{\mathbf{Y}} = \mathbf{A}^{[L]}$.

Example: Simple Forward Pass (Revisited)

Consider a network with:

- Input Layer: 2 neurons (x_1, x_2)
- Hidden Layer: 1 neuron (h_1)
- Output Layer: 1 neuron (o_1)

Given inputs $x_1 = 0.5, x_2 = 1.0$.

- Weights: $W_{h1,x1} = 0.1, W_{h1,x2} = 0.3$, $W_{o1,h1} = 0.2$.
- Biases: $b_{h1} = 0.05, b_{o1} = 0.01$.
- Activation functions: Sigmoid for hidden layer, Sigmoid for output layer.

1. Calculate Hidden Layer (h_1):

$$\begin{aligned} z_{h1} &= (W_{h1,x1} \cdot x_1) + (W_{h1,x2} \cdot x_2) + b_{h1} \\ &= (0.1 \cdot 0.5) + (0.3 \cdot 1.0) + 0.05 \\ &= 0.05 + 0.3 + 0.05 = 0.4 \\ a_{h1} &= \sigma(z_{h1}) = \frac{1}{1 + e^{-0.4}} \approx 0.5987 \end{aligned}$$

2. Calculate Output Layer (o_1):

$$\begin{aligned} z_{o1} &= (W_{o1,h1} \cdot a_{h1}) + b_{o1} \\ &= (0.2 \cdot 0.5987) + 0.01 \\ &= 0.11974 + 0.01 = 0.12974 \\ \hat{y} = a_{o1} &= \sigma(z_{o1}) = \frac{1}{1 + e^{-0.12974}} \approx 0.5324 \end{aligned}$$

So, the network's prediction $(\hat{y})$ for input $(0.5, 1.0)$ is approximately 0.5324.

Loss Function (Cost Function)

After forward propagation, the network's prediction $(\hat{y})$ is compared to the actual target value (y) using a **Loss Function** (or Cost Function). This function quantifies the error or discrepancy between the predicted and true values. The goal of training is to minimize this loss.

For a single training example, common loss functions include:

- **Mean Squared Error (MSE):** Commonly used for regression tasks.

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

  The factor of $\frac{1}{2}$ simplifies the derivative for gradient descent.

- **Binary Cross-Entropy Loss (BCE):** Commonly used for binary classification, particularly when the output is a probability.

$$L(\hat{y}, y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

- **Categorical Cross-Entropy Loss:** For multi-class classification.

For a mini-batch of m examples, the total loss is typically the average loss over all examples in the batch: $L = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i)$.

Gradient Descent

Gradient Descent is an iterative optimization algorithm used to minimize the loss function. It works by adjusting the network's parameters (weights and biases) in the direction opposite to the gradient of the loss function with respect to those parameters. The gradient indicates the direction of the steepest ascent, so moving in the opposite direction ensures steepest descent towards a minimum.

The update rule for a parameter θ (which can be any weight W or bias b) is:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \cdot \frac{\partial L}{\partial \theta}$$

Where:

- α (alpha) is the **learning rate**, a hyperparameter that determines the size of the steps taken during each iteration. A small learning rate leads to slow convergence, while a large learning rate can cause overshooting and divergence.
- $\frac{\partial L}{\partial \theta}$ is the partial derivative of the loss function L with respect to the parameter θ, representing the slope of the loss function at the current parameter value.

Types of Gradient Descent:

- **Batch Gradient Descent:** Computes the gradient of the loss function for the entire training dataset. It's stable but slow for large datasets.
- **Stochastic Gradient Descent (SGD):** Computes the gradient and updates parameters for each individual training example. Faster updates but more noisy convergence.
- **Mini-batch Gradient Descent:** Computes the gradient and updates parameters for a small, randomly sampled subset (mini-batch) of the training data. This is the most common approach, balancing efficiency and stability.

*The Essence of Backpropagation: Backpropagation computes how much each parameter contributes to the total loss. It essentially answers: "If I change this specific weight or bias by a tiny amount, how much will the total loss change?"

Steps of Backpropagation (Conceptual):

BP Step 1. Forward Pass: Compute the output $\hat{y}$ for a given input $\mathbf{x}$ and current network parameters (W, b). Store all intermediate z and a values for each layer.

BP Step 2. Compute Output Layer Error (Derivative of Loss): Calculate the gradient of the loss function with respect to the output layer's pre-activation $z^{[L]}$ (where L is the output layer).

$$\delta^{[L]} = \frac{\partial L}{\partial \mathbf{A}^{[L]}} \odot f'(\mathbf{Z}^{[L]})$$

Here, $\odot$ denotes element-wise multiplication (Hadamard product), and f' is the derivative of the activation function. This $\delta^{[L]}$ (often called the "error signal" or "delta") represents how much the error changes with respect to the output of the last neuron before activation.

BP Step 3. Backpropagate the Error (Calculate δ for Hidden Layers): For each layer l from $L-1$ down to 1, calculate the error signal $\delta^{[l]}$:

$$\delta^{[l]} = \left((\mathbf{W}^{[l+1]})^\top \delta^{[l+1]}\right) \odot f'(\mathbf{Z}^{[l]})$$

This step effectively propagates the error backward through the network. The error from the next layer is weighted by the transpose of the weights connecting them, and then multiplied by the derivative of the current layer's activation function.

BP Step 4. Compute Gradients for Weights and Biases: Once all δ values are calculated, the gradients for weights and biases for each layer l can be computed:

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \delta^{[l]}(\mathbf{A}^{[l-1]})^\top$$
$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$$

These gradients tell us how to adjust $W^{[l]}$ and $b^{[l]}$ to reduce the loss.

Chain Rule Illustration (Simplified for an output neuron and a weight connecting to it): Let's consider a simple scenario where the loss L depends on the output $\hat{y}$ of an output neuron, which in turn depends on its pre-activation z, and z depends on a weight W_{ij} and an input a_{i-1}.

$$L \xrightarrow{\text{depends on}} \hat{y} \xrightarrow{\text{depends on}} z \xrightarrow{\text{depends on}} W_{ij}$$

To find $\frac{\partial L}{\partial W_{ij}}$, using the chain rule:

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial W_{ij}}$$

- $\frac{\partial L}{\partial \hat{y}}$: How sensitive is the loss to the final output (e.g., $(y - \hat{y})$ for MSE).
- $\frac{\partial \hat{y}}{\partial z}$: The derivative of the activation function ($f'(z)$).
- $\frac{\partial z}{\partial W_{ij}}$: The input to the neuron that W_{ij} is connected to (a_{i-1}).

Backpropagation efficiently generalizes this chain rule application across all layers and parameters.

Backpropagation Algorithm: Mathematical Derivations

Backpropagation efficiently calculates the gradients of the loss function with respect to all the weights and biases in the network by applying the **chain rule** of calculus. It propagates the error backward from the output layer to the input layer.

Derivatives of Common Activation Functions: These are crucial for backpropagation, as they appear in the chain rule. Let $a = f(z)$.

- **Sigmoid:** $f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$
$$f'(z) = \sigma(z)(1 - \sigma(z)) = a(1 - a)$$
- **ReLU:** $f(z) = \max(0, z)$
$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$
- **Tanh:** $f(z) = \tanh(z)$
$$f'(z) = 1 - (\tanh(z))^2 = 1 - a^2$$

Backpropagation Steps (with Mathematical Detail):

BP Step 1: Forward Pass (as described in Section 2). Compute $\mathbf{Z}^{[l]}$ and $\mathbf{A}^{[l]}$ for all layers $l = 1, \ldots, L$. Store these values for use in the backward pass.

BP Step 2: Compute Output Layer Error ($\delta^{[L]}$) This term, $\delta^{[L]}$, represents how much the loss changes with respect to the pre-activation value of the output layer neurons. For a mini-batch of m samples, $\delta^{[L]} \in \mathbb{R}^{n_L \times m}$.

$$\delta^{[L]} = \frac{\partial L}{\partial \mathbf{Z}^{[L]}} = \frac{\partial L}{\partial \mathbf{A}^{[L]}} \odot f'(\mathbf{Z}^{[L]})$$

Where $\odot$ denotes element-wise (Hadamard) product. Let's look at specific cases for L:

- **Case 1: MSE Loss with Sigmoid Output Activation** Consider a single output neuron $\hat{y}$ and true label y. $L = \frac{1}{2}(\hat{y} - y)^2$. Output activation is $\hat{y} = \sigma(z^{[L]})$. We need $\frac{\partial L}{\partial z^{[L]}}$. By chain rule:

$$\frac{\partial L}{\partial z^{[L]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[L]}}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left(\frac{1}{2}(\hat{y} - y)^2 \right) = (\hat{y} - y)$$

$$\frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\partial}{\partial \hat{y}} \left(\sigma(z^{[L]}) \right) = \sigma(z^{[L]})(1 - \sigma(z^{[L]})) = \hat{y}(1 - \hat{y})$$

  Therefore, for a single sample:

$$\delta^{[L]} = (\hat{y} - y) \cdot \hat{y}(1 - \hat{y})$$

  For a mini-batch (vectorized):

$$\delta^{[L]} = (\mathbf{A}^{[L]} - \mathbf{Y}) \odot \mathbf{A}^{[L]}(1 - \mathbf{A}^{[L]})$$

- **Case 2: Binary Cross-Entropy Loss with Sigmoid Output Activation** For a single output neuron $\hat{y}$ and true label y. $L = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$. Output activation is $\hat{y} = \sigma(z^{[L]})$.

$$\frac{\partial L}{\partial z^{[L]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[L]}}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left(-[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \right) = -\left(\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}} \right) = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

  As before, $\frac{\partial \hat{y}}{\partial z^{[L]}} = \hat{y}(1 - \hat{y})$. Therefore, for a single sample:

$$\delta^{[L]} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \hat{y}(1 - \hat{y}) = \hat{y} - y$$

  For a mini-batch (vectorized):

$$\delta^{[L]} = \mathbf{A}^{[L]} - \mathbf{Y}$$

  This remarkably simple derivative is a major reason why Binary Cross-Entropy is preferred with a Sigmoid output for binary classification.

BP Step 3: Backpropagate the Error (Calculate $\delta^{[l]}$ for Hidden Layers) For each hidden layer l from $L - 1$ down to 1, the error signal $\delta^{[l]}$ is computed by taking the error from the next layer ($\delta^{[l+1]}$), weighing it by the transposed weights connecting the layers, and then multiplying by the derivative of the current layer's activation function.

$$\delta^{[l]} = \left((\mathbf{W}^{[l+1]})^\top \delta^{[l+1]} \right) \odot f'(\mathbf{Z}^{[l]})$$

Where:

- $\delta^{[l+1]}$ is the error signal from the subsequent layer.
- $(\mathbf{W}^{[l+1]})^\top$ is the transpose of the weight matrix connecting layer l to layer $l + 1$. This distributes the error appropriately to the neurons in the current layer l.
- $f'(\mathbf{Z}^{[l]})$ is the element-wise derivative of the activation function for layer l, evaluated at the pre-activations $\mathbf{Z}^{[l]}$.

BP Step 4: Compute Gradients for Weights and Biases Once all $\delta^{[l]}$ values are calculated for each layer, the gradients of the loss function with respect to the weights and biases of each layer l can be computed. These are the values needed for Gradient Descent. For a mini-batch of size m:

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{1}{m} \delta^{[l]} (\mathbf{A}^{[l-1]})^\top \tag{3}$$

$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \frac{1}{m} \sum_{\text{samples}} \delta^{[l]} = \frac{1}{m} \text{np.sum}(\delta^{[l]}, \text{axis} = 1, \text{keepdims} = True) \tag{4}$$

The division by m is for averaging the gradients over the mini-batch. $\mathbf{A}^{[l-1]}$ is the activation from the previous layer, which acts as the input to the current layer's weights.

Training a Neural Network: An Iterative Process

The training of a neural network combines these concepts in an iterative loop:

Training Step 1. **Initialize Parameters:** Randomly initialize all weights and biases.

Training Step 2. **Loop (Epochs):** Repeat for a specified number of epochs (passes through the entire training dataset) or until convergence:

a) **Mini-batch Loop:** For each mini-batch in the training data:

i) **Forward Propagation:** Feed the mini-batch inputs through the network to get predictions ($\hat{\mathbf{Y}}$), storing intermediate $\mathbf{Z}^{[l]}$ and $\mathbf{A}^{[l]}$ values.

ii) **Compute Loss:** Calculate the loss using the predictions ($\hat{\mathbf{Y}}$) and true labels ($\mathbf{Y}$).

iii) **Backpropagation:** Calculate the gradients $\frac{\partial L}{\partial \mathbf{W}^{[l]}}$ and $\frac{\partial L}{\partial \mathbf{b}^{[l]}}$ for all layers using the steps detailed above.

iv) **Parameter Update:** Adjust the weights and biases using the Gradient Descent update rule:

$$\mathbf{W}^{[l]} \leftarrow \mathbf{W}^{[l]} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[l]}}$$

$$\mathbf{b}^{[l]} \leftarrow \mathbf{b}^{[l]} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[l]}}$$

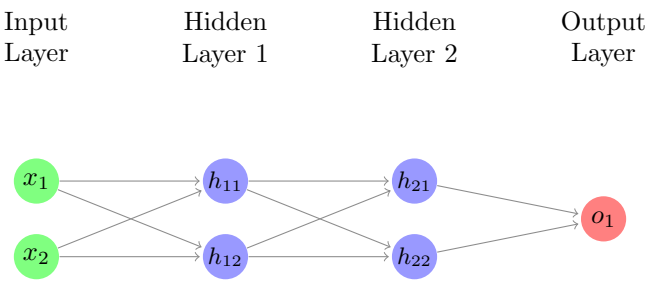
Training Step 3. **Evaluation:** After training, evaluate the network's performance on a separate validation or test set.

Full Mathematical Walkthrough: 2-2-2-1 Neural Network

Architecture and Problem Setup

We will analyze a neural network with the following architecture:

- **Input Layer (Layer 0):** 2 neurons (x_1, x_2)
- **Hidden Layer 1 (Layer 1):** 2 neurons (h_{11}, h_{12})
- **Hidden Layer 2 (Layer 2):** 2 neurons (h_{21}, h_{22})
- **Output Layer (Layer 3):** 1 neuron (o_1)



Activation Functions:

- Hidden Layers (Layer 1, Layer 2): **Rectified Linear Unit (ReLU)**: $f(z) = \max(0, z)$
- Output Layer (Layer 3): **Sigmoid** ($\sigma(z)$): $f(z) = \frac{1}{1+e^{-z}}$

Loss Function: We will use **Binary Cross-Entropy (BCE)** for a single sample:

$$L(\hat{y}, y) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Optimization: We will use **Gradient Descent** with a learning rate $\alpha = 0.1$.
Input Data and Target (Single Sample):

- Input vector: $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix}$
- True target: $y = 1.0$

Initial Parameters (Weights and Biases): These are randomly initialized values.

- **Layer 1 (Input to Hidden 1):** $\mathbf{W}^{[1]} = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \quad \mathbf{b}^{[1]} = \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix}$
- **Layer 2 (Hidden 1 to Hidden 2):** $\mathbf{W}^{[2]} = \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix} \quad \mathbf{b}^{[2]} = \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}$
- **Layer 3 (Hidden 2 to Output):** $\mathbf{W}^{[3]} = \begin{pmatrix} 0.9 & 1.0 \end{pmatrix} \quad \mathbf{b}^{[3]} = \begin{pmatrix} 0.15 \end{pmatrix}$

Step 1: Forward Propagation

The forward pass computes the output of each neuron, layer by layer, until the final prediction $\hat{y}$ is obtained.

Layer 1: Input to Hidden Layer 1

The input activation is $\mathbf{A}^{[0]} = \mathbf{x} = \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix}$.

$$\begin{aligned} \mathbf{Z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{A}^{[0]} + \mathbf{b}^{[1]} \\ &= \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix} + \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} \\ &= \begin{pmatrix} (0.1 \cdot 0.5) + (0.2 \cdot 1.0) \\ (0.3 \cdot 0.5) + (0.4 \cdot 1.0) \end{pmatrix} + \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} \\ &= \begin{pmatrix} 0.05 + 0.2 \\ 0.15 + 0.4 \end{pmatrix} + \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} \\ &= \begin{pmatrix} 0.25 \\ 0.55 \end{pmatrix} + \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix} \end{aligned}$$

Applying ReLU activation: $\mathbf{A}^{[1]} = (\mathbf{Z}^{[1]}) = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix} = \begin{pmatrix} \max(0, 0.30) \\ \max(0, 0.60) \end{pmatrix} = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix}$

Layer 2: Hidden Layer 1 to Hidden Layer 2

The activation from the previous layer is $\mathbf{A}^{[1]} = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix}$.

$$\begin{aligned} \mathbf{Z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{A}^{[1]} + \mathbf{b}^{[2]} \\ &= \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix} \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} \\ &= \begin{pmatrix} (0.5 \cdot 0.30) + (0.6 \cdot 0.60) \\ (0.7 \cdot 0.30) + (0.8 \cdot 0.60) \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} \\ &= \begin{pmatrix} 0.15 + 0.36 \\ 0.21 + 0.48 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} \\ &= \begin{pmatrix} 0.51 \\ 0.69 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix} \end{aligned}$$

Applying ReLU activation: $\mathbf{A}^{[2]} = (\mathbf{Z}^{[2]}) = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix} = \begin{pmatrix} \max(0, 0.61) \\ \max(0, 0.79) \end{pmatrix} = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix}$

Layer 3: Hidden Layer 2 to Output Layer

The activation from the previous layer is $\mathbf{A}^{[2]} = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix}$.

$$\begin{aligned} \mathbf{Z}^{[3]} &= \mathbf{W}^{[3]}\mathbf{A}^{[2]} + \mathbf{b}^{[3]} \\ &= \begin{pmatrix} 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix} + \begin{pmatrix} 0.15 \end{pmatrix} \\ &= ((0.9 \cdot 0.61) + (1.0 \cdot 0.79)) + 0.15 \\ &= (0.549 + 0.79) + 0.15 \\ &= 1.339 + 0.15 = 1.489 \end{aligned}$$

Applying Sigmoid activation: $\hat{y} = \mathbf{A}^{[3]} = \sigma(\mathbf{Z}^{[3]}) = \sigma(1.489) = \frac{1}{1+e^{-1.489}} \approx 0.816$

Loss Calculation

Given true target $y = 1.0$ and prediction $\hat{y} = 0.816$:

$$\begin{aligned} L(\hat{y}, y) &= -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})] \\ &= -[1.0 \cdot \log(0.816) + (1-1.0) \cdot \log(1-0.816)] \\ &= -[1.0 \cdot (-0.203) + 0 \cdot \log(0.184)] \\ &= -(-0.203) = 0.203 \end{aligned}$$

The loss for this single sample is 0.203.

Step 2: Backpropagation

Backpropagation computes the gradients of the loss function with respect to each weight and bias, propagating error backward from the output layer.

Derivatives of Activation Functions

- **Sigmoid:** $f'(z) = a(1-a)$
- **ReLU:** $f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$

Layer 3: Output Layer Gradients

The error signal for the output layer $\delta^{[3]}$: For BCE loss with Sigmoid output, this simplifies greatly to $\hat{y} - y$.

$$\begin{aligned} \delta^{[3]} &= \hat{y} - y \\ &= 0.816 - 1.0 = -0.184 \end{aligned}$$

Now, calculate gradients for $\mathbf{W}^{[3]}$ and $\mathbf{b}^{[3]}$. Recall $\mathbf{A}^{[2]} = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix}$.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}^{[3]}} &= \delta^{[3]} \cdot \mathbf{A}^{[2]} \\ &= (-0.184) \begin{pmatrix} 0.61 & 0.79 \end{pmatrix} \\ &= \begin{pmatrix} -0.184 \cdot 0.61 & -0.184 \cdot 0.79 \end{pmatrix} \\ &= \begin{pmatrix} -0.11224 & -0.14536 \end{pmatrix} \\ \frac{\partial L}{\partial \mathbf{b}^{[3]}} &= \delta^{[3]} \\ &= -0.184 \end{aligned}$$

Layer 2: Hidden Layer 2 Gradients

First, calculate the error signal $\delta^{[2]}$. We need $(\mathbf{W}^{[3]})^\top \delta^{[3]}$:

$$\mathbf{W}^{[3]} = \begin{pmatrix} 0.9 \\ 1.0 \end{pmatrix}$$

$$\mathbf{W}^{[3]}\delta^{[3]} = \begin{pmatrix} 0.9 \\ 1.0 \end{pmatrix}(-0.184) = \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix}$$

Next, we need $f'(\mathbf{Z}^{[2]})$. Recall $\mathbf{Z}^{[2]} = \begin{pmatrix} 0.61 \\ 0.79 \end{pmatrix}$. Since both components are > 0, ReLU derivative is 1 for both.

$$f'(\mathbf{Z}^{[2]}) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Now, calculate $\delta^{[2]}$:

$$\begin{aligned} \delta^{[2]} &= \left(\mathbf{W}^{[3]}\delta^{[3]}\right) \odot f'(\mathbf{Z}^{[2]}) \\ &= \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} \odot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} \end{aligned}$$

Now, calculate gradients for $\mathbf{W}^{[2]}$ and $\mathbf{b}^{[2]}$. Recall $\mathbf{A}^{[1]} = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix}$.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}^{[2]}} &= \delta^{[2]} \cdot \mathbf{A}^{[1]} \\ &= \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} \begin{pmatrix} 0.30 & 0.60 \end{pmatrix} \\ &= \begin{pmatrix} -0.1656 \cdot 0.30 & -0.1656 \cdot 0.60 \\ -0.1840 \cdot 0.30 & -0.1840 \cdot 0.60 \end{pmatrix} \\ &= \begin{pmatrix} -0.04968 & -0.09936 \\ -0.05520 & -0.11040 \end{pmatrix} \\ \frac{\partial L}{\partial \mathbf{b}^{[2]}} &= \delta^{[2]} \\ &= \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} \end{aligned}$$

Layer 1: Hidden Layer 1 Gradients

First, calculate the error signal $\delta^{[1]}$. We need $(\mathbf{W}^{[2]})^\top \delta^{[2]}$:

$$\mathbf{W}^{[2]} = \begin{pmatrix} 0.5 & 0.7 \\ 0.6 & 0.8 \end{pmatrix}$$

$$\mathbf{W}^{[2]}\delta^{[2]} = \begin{pmatrix} 0.5 & 0.7 \\ 0.6 & 0.8 \end{pmatrix} \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} = \begin{pmatrix} (0.5 \cdot -0.1656) + (0.7 \cdot -0.1840) \\ (0.6 \cdot -0.1656) + (0.8 \cdot -0.1840) \end{pmatrix} = \begin{pmatrix} -0.0828 - 0.1288 \\ -0.09936 - 0.1472 \end{pmatrix} = \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix}$$

Next, we need $f'(\mathbf{Z}^{[1]})$. Recall $\mathbf{Z}^{[1]} = \begin{pmatrix} 0.30 \\ 0.60 \end{pmatrix}$. Since both components are > 0, ReLU derivative is 1 for both.

$$f'(\mathbf{Z}^{[1]}) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Now, calculate $\delta^{[1]}$:

$$\begin{aligned} \delta^{[1]} &= \left(\mathbf{W}^{[2]}\delta^{[2]}\right) \odot f'(\mathbf{Z}^{[1]}) \\ &= \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix} \odot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix} \end{aligned}$$

Finally, calculate gradients for $\mathbf{W}^{[1]}$ and $\mathbf{b}^{[1]}$. Recall $\mathbf{A}^{[0]} = \mathbf{x} = \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix}$.

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}^{[1]}} &= \delta^{[1]} \cdot \mathbf{A}^{[0]} \\
&= \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix} \begin{pmatrix} 0.5 & 1.0 \end{pmatrix} \\
&= \begin{pmatrix} -0.2116 \cdot 0.5 & -0.2116 \cdot 1.0 \\ -0.24656 \cdot 0.5 & -0.24656 \cdot 1.0 \end{pmatrix} \\
&= \begin{pmatrix} -0.10580 & -0.21160 \\ -0.12328 & -0.24656 \end{pmatrix} \\
\frac{\partial L}{\partial \mathbf{b}^{[1]}} &= \delta^{[1]} \\
&= \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix}
\end{aligned}$$

Step 3: Parameter Update (First Iteration)

Using the calculated gradients and a learning rate $\alpha = 0.1$, we update the weights and biases using the rule: $\theta_{\text{new}} = \theta_{\text{old}} - \alpha \cdot \frac{\partial L}{\partial \theta}$.

Layer 3 Parameters

$$\begin{aligned}
\mathbf{W}^{[3]}_{\text{new}} &= \mathbf{W}^{[3]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[3]}} \\
&= \begin{pmatrix} 0.9 & 1.0 \end{pmatrix} - 0.1 \begin{pmatrix} -0.11224 & -0.14536 \end{pmatrix} \\
&= \begin{pmatrix} 0.9 & 1.0 \end{pmatrix} - \begin{pmatrix} -0.011224 & -0.014536 \end{pmatrix} \\
&= \begin{pmatrix} 0.9 + 0.011224 & 1.0 + 0.014536 \end{pmatrix} \\
&= \begin{pmatrix} 0.911224 & 1.014536 \end{pmatrix} \\
\mathbf{b}^{[3]}_{\text{new}} &= \mathbf{b}^{[3]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[3]}} \\
&= \begin{pmatrix} 0.15 \end{pmatrix} - 0.1(-0.184) \\
&= \begin{pmatrix} 0.15 \end{pmatrix} - (-0.0184) \\
&= \begin{pmatrix} 0.15 + 0.0184 \end{pmatrix} = \begin{pmatrix} 0.1684 \end{pmatrix}
\end{aligned}$$

Layer 2 Parameters

$$\begin{aligned}
\mathbf{W}^{[2]}_{\text{new}} &= \mathbf{W}^{[2]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[2]}} \\
&= \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix} - 0.1 \begin{pmatrix} -0.04968 & -0.09936 \\ -0.05520 & -0.11040 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix} - \begin{pmatrix} -0.004968 & -0.009936 \\ -0.005520 & -0.011040 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 + 0.004968 & 0.6 + 0.009936 \\ 0.7 + 0.005520 & 0.8 + 0.011040 \end{pmatrix} \\
&= \begin{pmatrix} 0.504968 & 0.609936 \\ 0.705520 & 0.811040 \end{pmatrix} \\
\mathbf{b}^{[2]}_{\text{new}} &= \mathbf{b}^{[2]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[2]}} \\
&= \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} - 0.1 \begin{pmatrix} -0.1656 \\ -0.1840 \end{pmatrix} \\
&= \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} - \begin{pmatrix} -0.01656 \\ -0.01840 \end{pmatrix} \\
&= \begin{pmatrix} 0.1 + 0.01656 \\ 0.1 + 0.01840 \end{pmatrix} = \begin{pmatrix} 0.11656 \\ 0.11840 \end{pmatrix}
\end{aligned}$$

Layer 1 Parameters

$$\begin{aligned}
\mathbf{W}^{[1]}_{\text{new}} &= \mathbf{W}^{[1]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[1]}} \\
&= \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} - 0.1 \begin{pmatrix} -0.10580 & -0.21160 \\ -0.12328 & -0.24656 \end{pmatrix} \\
&= \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} - \begin{pmatrix} -0.010580 & -0.021160 \\ -0.012328 & -0.024656 \end{pmatrix} \\
&= \begin{pmatrix} 0.1 + 0.010580 & 0.2 + 0.021160 \\ 0.3 + 0.012328 & 0.4 + 0.024656 \end{pmatrix} \\
&= \begin{pmatrix} 0.110580 & 0.221160 \\ 0.312328 & 0.424656 \end{pmatrix} \\
\mathbf{b}^{[1]}_{\text{new}} &= \mathbf{b}^{[1]}_{\text{old}} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[1]}} \\
&= \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} - 0.1 \begin{pmatrix} -0.2116 \\ -0.24656 \end{pmatrix} \\
&= \begin{pmatrix} 0.05 \\ 0.05 \end{pmatrix} - \begin{pmatrix} -0.02116 \\ -0.024656 \end{pmatrix} \\
&= \begin{pmatrix} 0.05 + 0.02116 \\ 0.05 + 0.024656 \end{pmatrix} = \begin{pmatrix} 0.07116 \\ 0.074656 \end{pmatrix}
\end{aligned}$$

Second Iteration (Conceptual Overview)

To perform the second step of update, you would repeat the entire process:

1. **Forward Propagation (using NEW weights and biases):** Feed the same input $\mathbf{x} = \begin{pmatrix} 0.5 \\ 1.0 \end{pmatrix}$ through the network, but now using $\mathbf{W}^{[1]}_{\text{new}}, \mathbf{b}^{[1]}_{\text{new}}$, $\mathbf{W}^{[2]}_{\text{new}}, \mathbf{b}^{[2]}_{\text{new}}$, and $\mathbf{W}^{[3]}_{\text{new}}, \mathbf{b}^{[3]}_{\text{new}}$. This will result in new $\mathbf{Z}^{[l]}$ and $\mathbf{A}^{[l]}$ values, and a new prediction $\hat{y}_{\text{new}}$.
2. **Loss Calculation (using new $\hat{y}$):** Calculate the loss $L(\hat{y}_{\text{new}}, y)$ with the new prediction. You should observe that this new loss is (ideally) lower than the loss from the first iteration (0.203).

3. **Backpropagation (using new A and old W for delta propagation):** Calculate all the new error signals ($\delta^{[3]}, \delta^{[2]}, \delta^{[1]}$) and subsequently the new gradients ($\frac{\partial L}{\partial \mathbf{W}^{[l]}}, \frac{\partial L}{\partial \mathbf{b}^{[l]}}$) using the $\mathbf{A}^{[l]}$ values obtained from the *second* forward pass and the *updated* weights $W^{[l+1]}$ for backpropagating the delta.

4. **Parameter Update (second update):** Apply the gradient descent update rule again using the gradients calculated in this second backward pass.

$$\mathbf{W}^{[l]}_{\text{newest}} = \mathbf{W}^{[l]}_{\text{new}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[l]}}_{\text{from second pass}}$$

$$\mathbf{b}^{[l]}_{\text{newest}} = \mathbf{b}^{[l]}_{\text{new}} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[l]}}_{\text{from second pass}}$$

This iterative process continues for many epochs until the network's performance converges or reaches a desired level of accuracy. Each step refines the parameters, gradually minimizing the loss function.

Challenges and Advanced Topics:

- **Vanishing/Exploding Gradients:** During backpropagation, gradients can become extremely small (vanishing) or extremely large (exploding), hindering effective training, especially in deep networks. Solutions include careful initialization (e.g., Xavier/He initialization), ReLU activation functions, batch normalization, and gradient clipping.
- **Overfitting:** When a network learns the training data too well, losing its ability to generalize to unseen data. Mitigated by regularization techniques (L1/L2 regularization, Dropout), early stopping, and data augmentation.
- **Optimizers:** While Gradient Descent is the core, more sophisticated optimizers (e.g., Adam, RMSprop, Adagrad) adapt the learning rate for each parameter, often leading to faster and more stable convergence.
- **Network Architectures:** Beyond simple feedforward networks, various architectures exist for specific tasks (e.g., Convolutional Neural Networks for images, Recurrent Neural Networks for sequences, Transformers for language).

Prepared By:
Md. Atikuzzaman
Lecturer
Department of Computer Science and Engineering
Green University of Bangladesh
Email: atik@cse.green.edu.bd