CS 412 — Introduction to Machine Learning (UIC) April 15, 2025

Lecture 23

Instructor: Aadirupa Saha *Scribe(s): Charles Pipim / Rithish Reddy Chichili*

Overview

In the last lecture we covered the following topics:

- Spectral clustering (SC)
- Neural Net Intro

This lecture mainly focuses on the transition from Logistic Regression (LR) to deeper Neural Networks, and the mathematical foundation behind training such models. The topics covered include:

- Feed-Forward Neural Network (FFNN)
- Back-Propagation (BP)
- Application of Chain Rule for differentiation

1 Logistic Regression Model for Classification

We consider a supervised learning task, where the goal is to learn a mapping:

$$f : \mathcal{X} \to \{0,1\}, \quad \text{where} \quad x \in \mathcal{X} \subseteq \mathbb{R}^d$$

Dataset

We are given a labeled dataset:

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}, \quad x_i \in \mathbb{R}^d,\ y_i \in \{0,1\}$$

Architecture Diagram

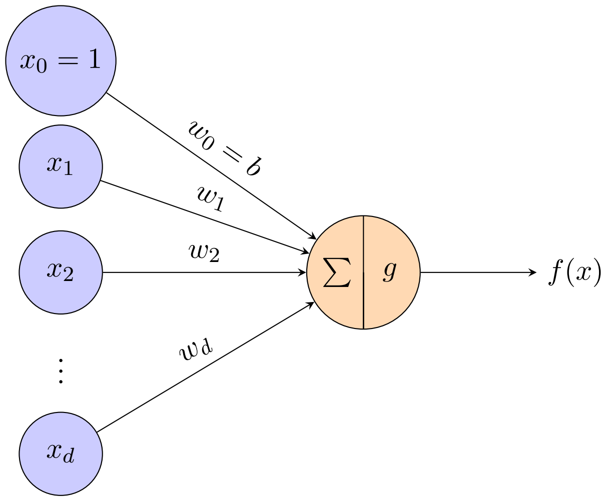

Figure 1: Architecture of a Single Layer Feedforward Neural Network (FFNN)

Explanation of the Architecture Diagram

This diagram represents the computation performed by a single neuron in a logistic regression model or a 1-layer Feedforward Neural Network (FFNN):

- **Input Nodes:**
  - $x_0 = 1$ is a bias input node, which is always fixed to 1.
  - $x_1, x_2, \ldots, x_d$ are the actual input features, where $x \in \mathbb{R}^d$.
- **Weights:**
  - Each input x_i is associated with a corresponding weight w_i.
  - The weight w_0 corresponds to the bias term b, and is connected to the bias input x_0.

- **Neuron Computation:**
  - The left half of the neuron ($\sum$) computes the linear combination:

$$z = \sum_{i=1}^{d} w_i x_i + b$$

  - The right half (g) applies a non-linear activation function $g(\cdot)$ to the result:

$$f(x) = g(z)$$

- **Output:**
  - The final output $f(x)$ is typically our predicted output, i.e., $\hat{y}$.

Logistic Regression Prediction

Here we define the prediction function using the sigmoid (logistic) function as activation function:

$$f(x) = g(z) = \sigma(\sum_{i=1}^{d} w_i x_i + b) = \frac{1}{1 + e^{-(\sum_{i=1}^{d} w_i x_i + b)}}$$

Activation Function

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

Other activation functions $g : \mathbb{R} \to \mathbb{R}$ include:

- Tanh: $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
- ReLU: $\max(0, z)$

Logistic Regression: Parameter Optimization

Let the model parameters be:

$$\theta = (w, b) \in \mathbb{R}^d \times \mathbb{R}, \quad \tilde{w} \in \mathbb{R}^d$$

Given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, with $y_i \in \{0, 1\}$, the logistic regression model makes predictions as:

$$\hat{y} = f_{LR}(x; w, b) = \sigma(w^\top x + b) = \frac{1}{1 + e^{-(w^\top x + b)}}$$

Loss Function

We define the cross-entropy loss (log loss) function:

$$\ell(y, \hat{y}) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

Log-Likelihood:

$$\log L(\mathcal{D}) = \sum_{i=1}^{n} \left[y_i \log\left(\frac{1}{1+e^{-w_i x_i + b}}\right) + (1-y_i)\log\left(\frac{e^{-w_i x_i + b}}{1+e^{-w_i x_i + b}}\right)\right]$$

Loss Minimization View:

Minimize the negative log-likelihood:

$$argmin_{w,b\in\mathbb{R}^{d+1}} - \log L(\mathcal{D})$$

The optimization objective becomes:

$$argmin_{w,b\in\mathbb{R}^{d+1}} \sum_{i=1}^{n} \left[y_i \log\left(1+e^{w_i x_i + b}\right) + (1-y_i)\log\left(1+e^{w_i x_i + b}\right)\right]$$

2 Multi Layer Feedforward Neural Network (FFNN)

When we go beyond a single-layer logistic regression model, we introduce one or more **hidden layers** consisting of multiple neurons. Each neuron performs a weighted sum of its inputs followed by a non-linear activation function g. This architecture is called a **Multi Layer Feedforward Neural Network (FFNN)**. We consider a FFNN with:

- Input dimension d
- One hidden layer with k neurons
- Output is a scalar (for binary classification)

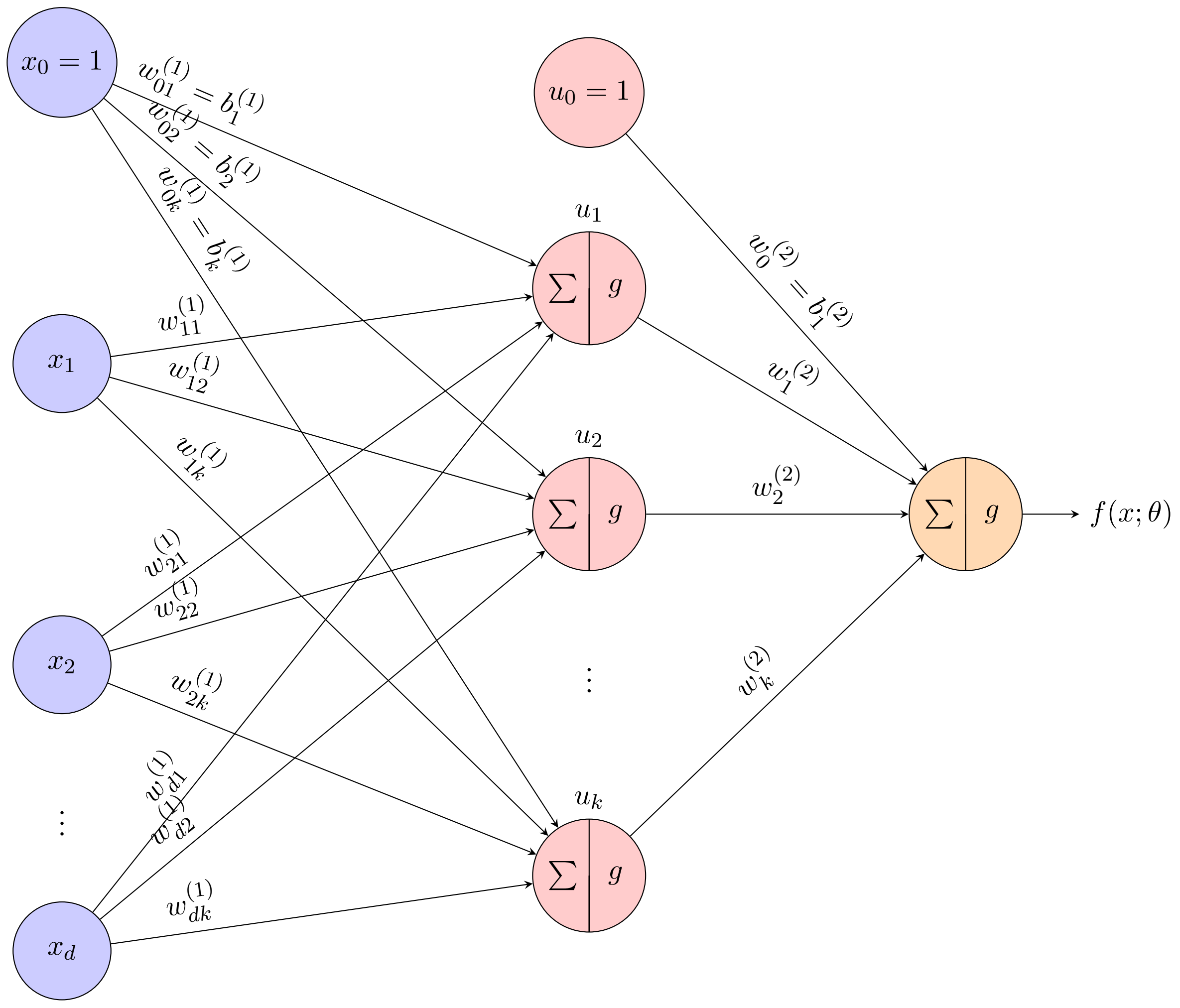

Figure 2: Architecture of a Multi Layer Feedforward Neural Network (FFNN) with One Hidden Layer of k Neurons

Forward Pass Equations and Dimensions

Input: Let the input be a vector $x \in \mathbb{R}^d$, and let us augment it with a bias node $x_0 = 1$, resulting in:

$$\tilde{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^{d+1}$$

Hidden Layer: We have k hidden neurons. Each hidden neuron receives all d inputs and a separate bias term:

$$\begin{aligned} a_1 &= \sum_{i=1}^{d} x_i w_{i1}^{(1)} + b_1^{(1)} \Rightarrow u_1 = g(a_1) \\ a_2 &= \sum_{i=1}^{d} x_i w_{i2}^{(1)} + b_2^{(1)} \Rightarrow u_2 = g(a_2) \\ &\vdots \\ a_k &= \sum_{i=1}^{d} x_i w_{ik}^{(1)} + b_k^{(1)} \Rightarrow u_k = g(a_k) \end{aligned}$$

where $g(\cdot)$ is the activation function (e.g., sigmoid, tanh, ReLU).

Augment Hidden Activations: Add bias unit $u_0 = 1$, so:

$$\tilde{u} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix} \in \mathbb{R}^{k+1}, \quad u_0 = 1$$

Output Layer: The final output is a linear combination of hidden layer outputs and an explicit bias:

$$z = \sum_{j=1}^{k} u_j w_j^{(2)} + b^{(2)}$$

$$f(x;\theta) = g(z)$$

Where:

- $W^{(2)} \in \mathbb{R}^k$
- $b^{(2)} \in \mathbb{R}$

Final Form: Using matrix-vector notation:

$$f(x;\theta) = g\left(W^{(2)^\top}\tilde{u} + b^{(2)}\right) = g\left(W^{(2)^\top} g(W^{(1)\top}\tilde{x} + b^{(1)}) + b^{(2)}\right)$$

- where $\theta = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\}$ are the parameters of the FFNN.
- First layer weight matrix $W^{(1)} \in \mathbb{R}^{k\times d}$:

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \dots & w_{1k}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \dots & w_{2k}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1}^{(1)} & w_{d2}^{(1)} & \dots & w_{dk}^{(1)} \end{bmatrix}$$

- Second layer weight vector $W^{(2)} \in \mathbb{R}^{k}$:

$$W^{(2)} = \begin{bmatrix} w_1^{(2)} \\ w_2^{(2)} \\ \vdots \\ w_k^{(2)} \end{bmatrix}$$

- First layer bias vector $b^{(1)} \in \mathbb{R}^{k}$:

$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_k^{(1)} \end{bmatrix}$$

- Second layer bias $b^{(2)} \in \mathbb{R}$

Dimensions:

- Input vector: $\tilde{x} \in \mathbb{R}^{d+1}$
- First layer weights: $W^{(1)} \in \mathbb{R}^{k\times d}$
- First layer bias: $b^{(1)} \in \mathbb{R}^{k}$
- Hidden activations: $\tilde{u} \in \mathbb{R}^{k+1}$
- Second layer weights: $W^{(2)} \in \mathbb{R}^{k}$
- Second layer bias: $b^{(2)} \in \mathbb{R}$
- Output: $f(x;\theta) \in \mathbb{R}$

Algorithm 2.1:

Simplified Forward Pass (No Augmentation)

```
1: **Input:** $x \in \mathbb{R}^{d+1}$
2: Compute hidden pre-activations: $a_j = \sum_{i=1}^{d} x_i w_{ij}^{(1)} + b_j^{(1)}$ for $j = 1$ to $k$
3: Compute hidden activations: $u_j = g(a_j)$ for $j = 1$ to $k$
4: Compute output pre-activation: $z = \sum_{j=1}^{k} u_j w_j^{(2)} + b^{(2)}$
5: Compute final output: $\hat{y} = f(x;\theta) = g(z)$
6: **Output:** $\hat{y} \in \mathbb{R}$
```

3 Training FFNN Parameters: Backpropagation and Optimization

Once the FFNN architecture is defined, we train the model by minimizing a loss function over the training dataset. The most commonly used loss for binary classification is the cross-entropy loss.

3.1 Gradient-Based Optimization

We aim to minimize the empirical loss (risk):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i;\theta))$$

Where $\theta = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\}$ and ℓ is typically the binary cross-entropy loss.

Gradient Descent (GD): Update parameters using the full dataset:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

Where η is the learning rate.

Stochastic Gradient Descent (SGD): Update parameters using a single example (x_i, y_i):

$$\theta \leftarrow \theta - \eta \nabla_\theta \ell(y_i, f(x_i;\theta))$$

Mini-batch Gradient Descent: Update parameters using a batch of m samples:

$$\theta \leftarrow \theta - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \ell(y_i, f(x_i;\theta))$$

3.2 Backpropagation Algorithm and Chain Rule in a Two-Layer Neural Network

This section elaborates on the backpropagation algorithm for training a two-layer feedforward neural network, consistent with the forward pass definitions established earlier. We will detail the computation of gradients using the chain rule, which is crucial for updating the network's parameters (weights and biases) to minimize a chosen loss function.

Initialization

At the beginning of the training process ($t = 0$), the parameters of our neural network are initialized randomly. These parameters include the weight matrix for the hidden layer ($W^{(1)}$), the bias vector for the hidden layer ($b^{(1)}$), the weight vector for the output layer ($W^{(2)}$), and the bias scalar for the output layer ($b^{(2)}$). Their dimensions are as follows:

- First layer weight matrix: $W^{(1)} \in \mathbb{R}^{k \times d}$
- First layer bias vector: $b^{(1)} \in \mathbb{R}^k$
- Second layer weight vector: $W^{(2)} \in \mathbb{R}^k$
- Second layer bias: $b^{(2)} \in \mathbb{R}$

Forward Pass (Recap)

Given an input vector $x \in \mathbb{R}^d$ (augmented with a bias $x_0 = 1$ to form $\tilde{x} \in \mathbb{R}^{d+1}$), the forward pass through the network is computed as follows:

1. **Pre-activation of the hidden layer:** The weighted sum of the inputs plus the bias for each hidden neuron is calculated:
$$a^{(1)} = W^{(1)\top}\tilde{x} + b^{(1)}$$
Here, $a^{(1)} \in \mathbb{R}^k$ is the vector of pre-activations for the k hidden neurons. Note that we are using $W^{(1)\top}$ here as in our final form equation, and $\tilde{x}$ includes the bias term. However, based on our initial hidden layer equations, the bias $b^{(1)}$ is added element-wise after the multiplication with the input x (not $\tilde{x}$). To be consistent with the majority of neural network literature and our final form, let's assume the bias is handled within the weight matrix by augmenting the input. If we strictly follow our initial equations, the pre-activation would be $a_i^{(1)} = \sum_{j=1}^d x_j w_{ji}^{(1)} + b_i^{(1)}$. For matrix notation, and aligning with the final form involving $\tilde{x}$, we should consider the weight matrix $W^{(1)}$ to be $k \times (d+1)$ and $\tilde{x}$ as the augmented input(i.e input $x_0 = 1$ w.r.t bias). However, our dimension for $W^{(1)}$ is $k \times d$. Let's proceed by treating the bias separately as in our hidden and output layer equations for clarity in the backward pass.
$$a_i^{(1)} = \sum_{j=1}^{d} x_j w_{ji}^{(1)} + b_i^{(1)} \quad \text{for } i = 1, \ldots, k$$
In vector form: $a^{(1)} = W^{(1)}x + b^{(1)}$, where $W^{(1)} \in \mathbb{R}^{k\times d}$, $x \in \mathbb{R}^d$, and $b^{(1)} \in \mathbb{R}^k$.

2. **Activation of the hidden layer:** An activation function $g(\cdot)$ is applied element-wise to the pre-activations to produce the hidden layer activations:
$$u = g(a^{(1)})$$
Here, $u \in \mathbb{R}^k$ is the vector of activations of the k hidden neurons.

3. **Pre-activation of the output layer:** The output layer receives the activations from the hidden layer, and a weighted sum with the output bias is computed:
$$z = W^{(2)\top}u + b^{(2)}$$
Here, $W^{(2)} \in \mathbb{R}^k$ is a column vector of weights connecting the hidden layer to the single output neuron, and $b^{(2)} \in \mathbb{R}$ is the output bias. The pre-activation z is a scalar.

4. **Activation of the output layer:** Finally, an activation function $g(\cdot)$ is applied to the output pre-activation to produce the network's output:

$$\hat{y} = g(z) = g(W^{(2)\top} u + b^{(2)})$$

The output $\hat{y} \in \mathbb{R}$ (assuming a single output neuron). For binary classification, g is often the sigmoid function, resulting in an output between 0 and 1.

Loss Function

For a single training example (x, y), where y is the true label, we use a loss function $L(y, \hat{y})$ to quantify the error between the predicted output $\hat{y}$ and the true label y. A common choice for binary classification is the binary cross-entropy loss:

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

Backward Pass: Gradient Computation

The core of backpropagation is the efficient computation of the gradients of the loss function with respect to each of the network's parameters. This is achieved by applying the chain rule of differentiation, propagating the error backward from the output layer to the input layer.

Gradient with respect to the output layer weights ($W^{(2)}$)

We want to compute $\frac{\partial L}{\partial W^{(2)}}$. Using the chain rule:

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W^{(2)}}$$

Let's compute each term:

1. **Derivative of the loss with respect to the output activation:**

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

2. **Derivative of the output activation with respect to the output pre-activation (assuming sigmoid $g(z) = \frac{1}{1+e^{-z}}$):**

$$\frac{\partial \hat{y}}{\partial z} = \frac{d}{dz} g(z) = g(z)(1 - g(z)) = \hat{y}(1 - \hat{y})$$

3. **Derivative of the output pre-activation with respect to the output layer weights:**

$$z = W^{(2)\top} u + b^{(2)} = \sum_{j=1}^{k} w_j^{(2)} u_j + b^{(2)}$$

Therefore, the derivative of z with respect to each element $w_i^{(2)}$ of $W^{(2)}$ is:

$$\frac{\partial z}{\partial w_i^{(2)}} = u_i$$

In vector form, $\frac{\partial z}{\partial W^{(2)}} = u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{bmatrix}$. Since L is a scalar and $W^{(2)}$ is a vector, $\frac{\partial L}{\partial W^{(2)}}$ will be a row vector (if we follow numerator layout convention) or a column vector (denominator layout). Consistent with most gradient descent updates where the gradient has the same shape as the parameter, we consider it a column vector here.

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

Simplifying the first two terms:

$$\frac{\partial L}{\partial W^{(2)}} = (\hat{y} - y)u$$

Gradient with respect to the output layer bias ($b^{(2)}$)

We want to compute $\frac{\partial L}{\partial b^{(2)}}$. Using the chain rule:

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b^{(2)}}$$

We already have $\frac{\partial L}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial z}$. Now we need $\frac{\partial z}{\partial b^{(2)}}$:

$$z = W^{(2)\top} u + b^{(2)}$$

$$\frac{\partial z}{\partial b^{(2)}} = 1$$

Putting it together:

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

Gradient with respect to the hidden layer weights ($W^{(1)}$)

We want to compute $\frac{\partial L}{\partial W^{(1)}}$. Using the chain rule:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial u} \frac{\partial u}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial W^{(1)}}$$

We already have $\frac{\partial L}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial z}$. Let's compute the remaining terms:

1. **Derivative of the output pre-activation with respect to the hidden layer activations:**

$$z = W^{(2)\top} u + b^{(2)} = \sum_{j=1}^{k} w_j^{(2)} u_j + b^{(2)}$$

$$\frac{\partial z}{\partial u_i} = w_i^{(2)}$$

In vector form, $\frac{\partial z}{\partial u} = W^{(2)} = \begin{bmatrix} w_1^{(2)} \\ w_2^{(2)} \\ \vdots \\ w_k^{(2)} \end{bmatrix}$.

2. **Derivative of the hidden layer activations with respect to the hidden layer pre-activations (assuming sigmoid $g(a_i^{(1)}) = \frac{1}{1+e^{-a_i^{(1)}}}$):**

$$u_i = g(a_i^{(1)})$$

$$\frac{\partial u_i}{\partial a_i^{(1)}} = g(a_i^{(1)})(1 - g(a_i^{(1)})) = u_i(1 - u_i)$$

This is an element-wise product. In vector form, $\frac{\partial u}{\partial a^{(1)}} = \text{diag}(u)(I - \text{diag}(u))$, where $\text{diag}(u)$ is a diagonal matrix with the elements of u on the diagonal, or more simply, an element-wise multiplication $u \odot (1 - u)$.

3. **Derivative of the hidden layer pre-activations with respect to the hidden layer weights:**

$$a^{(1)} = W^{(1)}x + b^{(1)}$$

$$a_i^{(1)} = \sum_{j=1}^{d} w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$\frac{\partial a_i^{(1)}}{\partial w_{ij}^{(1)}} = x_j$$

For the entire weight matrix $W^{(1)} \in \mathbb{R}^{k \times d}$, the derivative $\frac{\partial a^{(1)}}{\partial W^{(1)}}$ is a third-order tensor. However, when considering the gradient of the loss with respect to $W^{(1)}$, the dimensions will align. Let's look at the contribution to each $w_{ij}^{(1)}$:

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial u_i} \frac{\partial u_i}{\partial a_i^{(1)}} \frac{\partial a_i^{(1)}}{\partial w_{ij}^{(1)}}$$

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = (\hat{y} - y) \cdot w_i^{(2)} \cdot u_i(1 - u_i) \cdot x_j$$

In matrix form:

$$\frac{\partial L}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(2)} \odot (u \odot (1 - u)) \cdot x^\top$$

Here, $\odot$ denotes element-wise multiplication, and $x^\top$ is the transpose of the input vector (a row vector).

Gradient with respect to the hidden layer biases ($b^{(1)}$)

We want to compute $\frac{\partial L}{\partial b^{(1)}}$. Using the chain rule:

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial u} \frac{\partial u}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial b^{(1)}}$$

We have most of these terms already. The last term is:

$$\frac{\partial a^{(1)}}{\partial b^{(1)}} = \frac{\partial}{\partial b^{(1)}}(W^{(1)}x + b^{(1)}) = I$$

where I is the $k \times k$ identity matrix, implying that $\frac{\partial a_i^{(1)}}{\partial b_j^{(1)}} = 1$ if $i = j$ and 0 otherwise. Effectively, $\frac{\partial a^{(1)}}{\partial b^{(1)}}$ acts as selecting the appropriate element. Thus, in component form:

$$\frac{\partial a_i^{(1)}}{\partial b_i^{(1)}} = 1$$

Putting it together:

$$\frac{\partial L}{\partial b^{(1)}} = (\hat{y} - y) \cdot W^{(2)} \odot (u \odot (1 - u)) \cdot 1$$

Here, the multiplication by 1 is element-wise, resulting in:

$$\frac{\partial L}{\partial b^{(1)}} = (\hat{y} - y) \cdot (W^{(2)} \odot u \odot (1 - u))$$

Gradient Descent Update

Once the gradients are computed, the parameters of the network are updated using an optimization algorithm like gradient descent:

- Update rule for the output layer weights:

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial L}{\partial W^{(2)}}$$

- Update rule for the output layer bias:

$$b^{(2)} \leftarrow b^{(2)} - \eta \frac{\partial L}{\partial b^{(2)}}$$

- Update rule for the hidden layer weights:

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}}$$

- Update rule for the hidden layer biases:

$$b^{(1)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}}$$

where η is the learning rate, a hyperparameter that controls the step size of the updates.
This iterative process of forward pass, loss computation, backward pass (gradient computation), and parameter update is repeated for a number of epochs or until a satisfactory level of performance is achieved.

Backpropagation Algorithm

Algorithm 3.1:

```
1: Given a 2-layer NN, Initialize weights $W^{(1)}, W^{(2)}$ and biases $b^{(1)}, b^{(2)}$ randomly
2: for each training example $(x, y)$ do
3:     Forward pass:
4:         Compute hidden activations $u = g(W^{(1)\top} x + b^{(1)})$
5:         Compute output $\hat{y} = g(W^{(2)\top} u + b^{(2)})$
6:     Compute loss: $\mathcal{L}(y, \hat{y})$
7:     Backward pass:
8:         Compute gradients using the chain rule
9:     Update parameters:
10:        $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)} \leftarrow$ updated via gradient descent
11: end for
```

4 Backpropagation Example: 2-Layer NN

This document illustrates the backpropagation algorithm for a neural network with 2 input features, 2 hidden neurons, and 1 output neuron over three iterations. We use the sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$ and the binary cross-entropy loss $L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$. The learning rate η is set to 0.5.

Initialization

Initial weights and biases:

$$W^{(1)} = \begin{bmatrix} 0.1 & 0.4 \\ 0.2 & 0.5 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$
$$W^{(2)} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix}, \quad b^{(2)} = 0.9$$

Training example: $x = \begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix}$, $y = 1$.

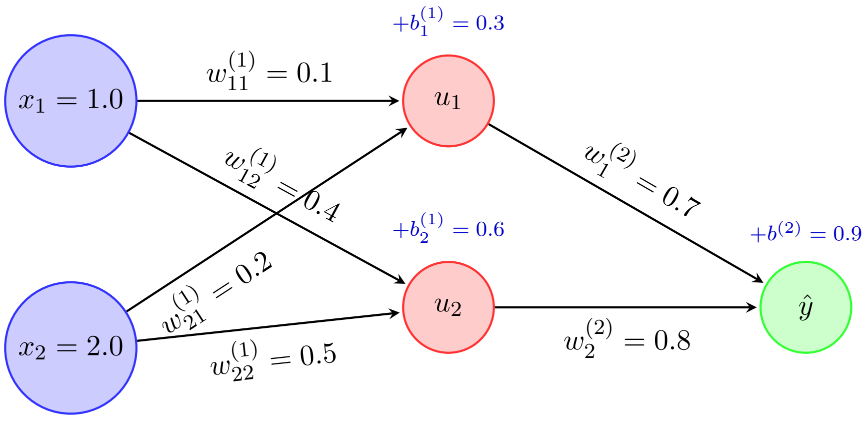

Figure 3: Example Architecture (2-2-1) of a Multi Layer NN

Iteration 1

Forward Pass

$$a^{(1)} = W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.1 & 0.4 \\ 0.2 & 0.5 \end{bmatrix}\begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix} + \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.8 \end{bmatrix}$$
$$u = g(a^{(1)}) = \begin{bmatrix} g(1.2) \\ g(1.8) \end{bmatrix} \approx \begin{bmatrix} 0.7685 \\ 0.8581 \end{bmatrix}$$
$$z = W^{(2)\top}u + b^{(2)} = \begin{bmatrix} 0.7 & 0.8 \end{bmatrix}\begin{bmatrix} 0.7685 \\ 0.8581 \end{bmatrix} + 0.9 \approx 2.1244$$
$$\hat{y} = g(z) = g(2.1244) \approx 0.8934$$

Loss:

$$L = -1 \cdot \log(0.8934) - 0 \cdot \log(1 - 0.8934) \approx 0.1122$$

Backward Pass

Gradients:

$$\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &= -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = -\frac{1}{0.8934} \approx -1.1193 \\
\frac{\partial \hat{y}}{\partial z} &= \hat{y}(1-\hat{y}) = 0.8934(1-0.8934) \approx 0.0953 \\
\frac{\partial z}{\partial W^{(2)}} &= u = \begin{bmatrix} 0.7685 \\ 0.8581 \end{bmatrix} \\
\frac{\partial z}{\partial b^{(2)}} &= 1 \\
\frac{\partial u}{\partial a^{(1)}} &= u \odot (1-u) = \begin{bmatrix} 0.7685(1-0.7685) \\ 0.8581(1-0.8581) \end{bmatrix} \approx \begin{bmatrix} 0.1779 \\ 0.1217 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial W^{(1)}} &= x^\top = \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial b^{(1)}} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\end{aligned}$$

Gradient of Loss w.r.t. parameters:

$$\begin{aligned}
\frac{\partial L}{\partial W^{(2)}} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W^{(2)}} = (-1.1193)(0.0953) \begin{bmatrix} 0.7685 \\ 0.8581 \end{bmatrix} \approx \begin{bmatrix} -0.08196 \\ -0.09153 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(2)}} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b^{(2)}} = (-1.1193)(0.0953)(1) \approx -0.1067 \\
\frac{\partial L}{\partial W^{(1)}} &= \left(\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial u} \odot \frac{\partial u}{\partial a^{(1)}} \right) x^\top \\
&= ((-1.1193)(0.0953) \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix} \odot \begin{bmatrix} 0.1779 \\ 0.1217 \end{bmatrix}) \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} \\
&\approx (\begin{bmatrix} -0.07466 \\ -0.08533 \end{bmatrix} \odot \begin{bmatrix} 0.1779 \\ 0.1217 \end{bmatrix}) \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} \\
&\approx \begin{bmatrix} -0.01328 \\ -0.01038 \end{bmatrix} \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} = \begin{bmatrix} -0.01328 & -0.02656 \\ -0.01038 & -0.02076 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(1)}} &= \left(\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial u} \odot \frac{\partial u}{\partial a^{(1)}} \right) \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&\approx \begin{bmatrix} -0.01328 \\ -0.01038 \end{bmatrix}
\end{aligned}$$

Parameter Updates

$$
\begin{aligned}
W^{(2)} &\leftarrow W^{(2)} - \eta \frac{\partial L}{\partial W^{(2)}} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix} - 0.5 \begin{bmatrix} -0.08196 \\ -0.09153 \end{bmatrix} \approx \begin{bmatrix} 0.7410 \\ 0.8458 \end{bmatrix} \\
b^{(2)} &\leftarrow b^{(2)} - \eta \frac{\partial L}{\partial b^{(2)}} = 0.9 - 0.5(-0.1067) \approx 0.9534 \\
W^{(1)} &\leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}} = \begin{bmatrix} 0.1 & 0.4 \\ 0.2 & 0.5 \end{bmatrix} - 0.5 \begin{bmatrix} -0.01328 & -0.02656 \\ -0.01038 & -0.02076 \end{bmatrix} \approx \begin{bmatrix} 0.1066 & 0.4133 \\ 0.2052 & 0.5104 \end{bmatrix} \\
b^{(1)} &\leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} - 0.5 \begin{bmatrix} -0.01328 \\ -0.01038 \end{bmatrix} \approx \begin{bmatrix} 0.3066 \\ 0.6052 \end{bmatrix}
\end{aligned}
$$

Iteration 2

Forward Pass

Using updated parameters:

$$
\begin{aligned}
a^{(1)} &= W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.1066 & 0.4133 \\ 0.2052 & 0.5104 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix} + \begin{bmatrix} 0.3066 \\ 0.6052 \end{bmatrix} \approx \begin{bmatrix} 1.2398 \\ 1.8312 \end{bmatrix} \\
u &= g(a^{(1)}) = \begin{bmatrix} g(1.2398) \\ g(1.8312) \end{bmatrix} \approx \begin{bmatrix} 0.7757 \\ 0.8618 \end{bmatrix} \\
z &= W^{(2)\top}u + b^{(2)} = \begin{bmatrix} 0.7410 & 0.8458 \end{bmatrix} \begin{bmatrix} 0.7757 \\ 0.8618 \end{bmatrix} + 0.9534 \approx 2.1885 \\
\hat{y} &= g(z) = g(2.1885) \approx 0.8993
\end{aligned}
$$

Loss:

$$
L = -1 \cdot \log(0.8993) - 0 \cdot \log(1 - 0.8993) \approx 0.1061
$$

Backward Pass

Gradients:

$$
\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &\approx -1.1120 \\
\frac{\partial \hat{y}}{\partial z} &\approx 0.0900 \\
\frac{\partial z}{\partial W^{(2)}} &= u \approx \begin{bmatrix} 0.7757 \\ 0.8618 \end{bmatrix} \\
\frac{\partial z}{\partial b^{(2)}} &= 1 \\
\frac{\partial u}{\partial a^{(1)}} &\approx \begin{bmatrix} 0.1745 \\ 0.1191 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial W^{(1)}} &= x^\top = \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial b^{(1)}} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\end{aligned}
$$

Gradient of Loss w.r.t. parameters:

$$\begin{aligned}
\frac{\partial L}{\partial W^{(2)}} &\approx \begin{bmatrix} -0.0749 \\ -0.0832 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(2)}} &\approx -0.1001 \\
\frac{\partial L}{\partial W^{(1)}} &\approx \begin{bmatrix} -0.0129 & -0.0258 \\ -0.0100 & -0.0200 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(1)}} &\approx \begin{bmatrix} -0.0129 \\ -0.0100 \end{bmatrix}
\end{aligned}$$

Parameter Updates

$$\begin{aligned}
W^{(2)} &\approx \begin{bmatrix} 0.7785 \\ 0.8874 \end{bmatrix} \\
b^{(2)} &\approx 1.0035 \\
W^{(1)} &\approx \begin{bmatrix} 0.1131 & 0.4257 \\ 0.2102 & 0.5204 \end{bmatrix} \\
b^{(1)} &\approx \begin{bmatrix} 0.3130 \\ 0.6102 \end{bmatrix}
\end{aligned}$$

Iteration 3

Forward Pass

Using updated parameters:

$$\begin{aligned}
a^{(1)} = W^{(1)}x + b^{(1)} &= \begin{bmatrix} 0.1131 & 0.4257 \\ 0.2102 & 0.5204 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix} + \begin{bmatrix} 0.3130 \\ 0.6102 \end{bmatrix} \approx \begin{bmatrix} 1.2775 \\ 1.8610 \end{bmatrix} \\
u = g(a^{(1)}) &\approx \begin{bmatrix} 0.7825 \\ 0.8653 \end{bmatrix} \\
z = W^{(2)\top} u + b^{(2)} &= \begin{bmatrix} 0.7785 & 0.8874 \end{bmatrix} \begin{bmatrix} 0.7825 \\ 0.8653 \end{bmatrix} + 1.0035 \approx 2.2488 \\
\hat{y} = g(z) &\approx 0.9042
\end{aligned}$$

Loss:

$$L = -1 \cdot \log(0.9042) - 0 \cdot \log(1 - 0.9042) \approx 0.1010$$

Backward Pass

Gradients:

$$\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &\approx -1.1059 \\
\frac{\partial \hat{y}}{\partial z} &\approx 0.0860 \\
\frac{\partial z}{\partial W^{(2)}} &= u \approx \begin{bmatrix} 0.7825 \\ 0.8653 \end{bmatrix} \\
\frac{\partial z}{\partial b^{(2)}} &= 1 \\
\frac{\partial u}{\partial a^{(1)}} &\approx \begin{bmatrix} 0.1701 \\ 0.1168 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial W^{(1)}} &= x^\top = \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} \\
\frac{\partial a^{(1)}}{\partial b^{(1)}} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\end{aligned}$$

Gradient of Loss w.r.t. parameters:

$$\begin{aligned}
\frac{\partial L}{\partial W^{(2)}} &\approx \begin{bmatrix} -0.0737 \\ -0.0816 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(2)}} &\approx -0.0950 \\
\frac{\partial L}{\partial W^{(1)}} &\approx \begin{bmatrix} -0.0125 & -0.0250 \\ -0.0097 & -0.0194 \end{bmatrix} \\
\frac{\partial L}{\partial b^{(1)}} &\approx \begin{bmatrix} -0.0125 \\ -0.0097 \end{bmatrix}
\end{aligned}$$

Parameter Updates

$$\begin{aligned}
W^{(2)} &\approx \begin{bmatrix} 0.8154 \\ 0.9282 \end{bmatrix} \\
b^{(2)} &\approx 1.0510 \\
W^{(1)} &\approx \begin{bmatrix} 0.1194 & 0.4382 \\ 0.2150 & 0.5298 \end{bmatrix} \\
b^{(1)} &\approx \begin{bmatrix} 0.3193 \\ 0.6150 \end{bmatrix}
\end{aligned}$$

Summary Table

Iteration	Predicted Output ($\hat{y}$)	Loss (L)	Weights ($W^{(1)}, W^{(2)}$)	Bias ($b^{(1)}, b^{(2)}$)
1	0.8934	0.1122	$W^{(1)} = \begin{bmatrix} 0.1 & 0.4 \\ 0.2 & 0.5 \end{bmatrix}, W^{(2)} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix}$	$b^{(1)} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}, b^{(2)} = 0.9$
2	0.8993	0.1061	$W^{(1)} \approx \begin{bmatrix} 0.1066 & 0.4133 \\ 0.2052 & 0.5104 \end{bmatrix}, W^{(2)} \approx \begin{bmatrix} 0.7410 \\ 0.8458 \end{bmatrix}$	$b^{(1)} \approx \begin{bmatrix} 0.3066 \\ 0.6052 \end{bmatrix}, b^{(2)} \approx 0.9534$
3	0.9042	0.1010	$W^{(1)} \approx \begin{bmatrix} 0.1131 & 0.4257 \\ 0.2102 & 0.5204 \end{bmatrix}, W^{(2)} \approx \begin{bmatrix} 0.7785 \\ 0.8874 \end{bmatrix}$	$b^{(1)} \approx \begin{bmatrix} 0.3130 \\ 0.6102 \end{bmatrix}, b^{(2)} \approx 1.0035$

Table 1: Summary of Parameter Optimization with Backpropagation over 3 Iterations

Next Lecture

The next lecture will cover the following topics:
(i) Backpropagation (contd)
(ii) RNN
(iii) CNN (Intro)

References

[1] D. Rumelhart, G. Hinton, and R. Williams, *Learning representations by back-propagating errors*, Nature, 1986.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 1998.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, *A neural probabilistic language model*, JMLR, 2003.

[4] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 2015.