Objectives

- Neural Network Architecture
- Feedforward Networks
- Backpropagation Algorithm
- Cost Function Minimization
- Chain Rule Application

1 Neural Network Architecture

1.1 Feedforward Neural Networks

A neural network consists of multiple layers of neurons where connections between neurons do not form cycles. Information moves in only one direction - forward - from the input nodes, through the hidden nodes, to the output nodes.

We organize our network into L layers:

- Input layer (indexed as $l = 1$)
- Hidden layers (indexed as $l = 2, \ldots, L - 1$)
- Output layer (indexed as $l = L$)

Each layer contains a set of neurons (nodes). As shown in our notes:

- Now we have a total of L layers
- Each neuron is connected to neurons in adjacent layers

1.2 Indices and Notation

As seen in our notes, we use superscript (l) to denote the layer number:

- $x^{(l)}$ indicates indices
- i, j, k are used as indices

For a network with n input neurons, m hidden neurons, and k output neurons:

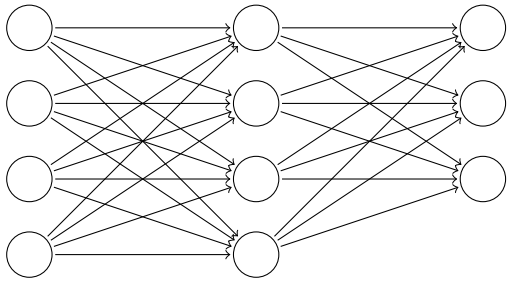

Figure 1: A feedforward neural network with 3 layers ($L = 3$)

- Each neuron in a layer is connected to every neuron in the next layer
- All of these connections have weights associated with them

1.3 Matrix Representation

Every neuron in the hidden layer is connected to neurons from the previous layer. We can represent the connections using matrix notation:

For a neuron i in layer l, we compute its activation $a_i^{(l)}$ as:

$$a_i^{(l)} = \sigma\left(\sum_{j=1}^{n^{(l-1)}} w_{ji}^{(l)} a_j^{(l-1)} + b_i^{(l)}\right) \tag{1}$$

Where:

- $w_{ji}^{(l)}$ is the weight connecting the j-th neuron in layer $l-1$ to the i-th neuron in layer l
- $b_i^{(l)}$ is the bias for the i-th neuron in layer l
- σ is the activation function
- $n^{(l-1)}$ is the number of neurons in layer $l-1$

In matrix form, we can write this as:

$$\mathbf{a}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2}$$

Where:

$$\mathbf{a}^{(l)} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n^{(l)}}^{(l)} \end{bmatrix}, \mathbf{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \dots & w_{1n^{(l-1)}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \dots & w_{2n^{(l-1)}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{(l)}1}^{(l)} & w_{n^{(l)}2}^{(l)} & \dots & w_{n^{(l)}n^{(l-1)}}^{(l)} \end{bmatrix}, \mathbf{b}^{(l)} = \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n^{(l)}}^{(l)} \end{bmatrix} \tag{3}$$

2 Cost Function Minimization

2.1 Objective Function

Our objective is to minimize the cost function C. As noted in our class:

$$\text{obj: minimize } C \tag{4}$$

The cost function is typically defined as:

$$C = \frac{1}{2}\sum_j (a_j^{(L)} - y_j)^2 \tag{5}$$

Where:

- $a_j^{(L)}$ is the output of the j-th neuron in the output layer
- y_j is the corresponding target value (ground truth vector)

2.2 Backpropagation Algorithm

To minimize the cost function, we need to compute its gradient with respect to the weights and biases. The backpropagation algorithm allows us to do this efficiently.

2.2.1 Forward Pass

The forward pass computes the activations of all neurons in the network:

$$z^{(l)} = w^{(l)} \cdot a^{(l-1)} + b^{(l)} \tag{6}$$

$$a^{(l)} = \sigma(z^{(l)}) \tag{7}$$

Where σ is the activation function.

2.2.2 Backward Pass

In the backward pass, we compute the error terms for each neuron and propagate them backward through the network.

The error term for an output neuron j is defined as:

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = \text{sensitivity} \tag{8}$$

As noted in our calculations:

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = (a_j^{(L)} - y_j) \cdot \sigma'(z_j^{(L)}) \tag{9}$$

For a neuron in an earlier layer, we have:

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)} \cdot \sigma'(z_j^{(l)}) \tag{10}$$

2.3 Chain Rule Application

The core of backpropagation is the application of the chain rule from calculus. As our notes indicate, we apply the chain rule to compute:

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \frac{\partial C}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} \cdot a_j^{(l-1)} \tag{11}$$

Similarly, for the bias:

$$\frac{\partial C}{\partial b_i^{(l)}} = \frac{\partial C}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial b_i^{(l)}} = \delta_i^{(l)} \tag{12}$$

2.4 Weight Update

Once we have computed the gradients, we update the weights and biases:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial C}{\partial w_{ij}^{(l)}} \tag{13}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial C}{\partial b_i^{(l)}} \tag{14}$$

Where α is the learning rate.

3 Summary

To summarize what we've covered:

- Neural networks consist of layers of interconnected neurons
- Each connection has an associated weight
- The forward pass computes the output of the network
- The cost function measures the error between the network's output and the desired output
- Backpropagation uses the chain rule to compute the gradients of the cost function efficiently
- Gradient descent updates the weights and biases to minimize the cost function