Neural Networks*

March 13, 2025

Contents

*This lecture notes were written by Nail Ibrahimli and were heavily inspired by and adapted from:
- Mickael Nielsen. Neural Networks and Deep Learning

1 Introduction

In these lecture notes we introduce neural networks by focusing on the problem of recognizing handwritten digits. Although the human visual system is remarkably adept at recognizing digits, this task poses significant challenges for computers. Neural networks, which learn from training examples, provide an effective solution. Throughout these notes, you will learn how to implement a neural network that recognizes handwritten digits with high accuracy, without any human intervention. This lecture notes serve for understanding key neural network components: perceptrons, sigmoid neurons and the learning algorithm known as stochastic gradient descent. By the end, you will have a foundational grasp of multi-layered perceptron and its relevance in modern applications.

2 Perceptrons

A **perceptron** is an early model of an artificial neuron developed by Frank Rosenblatt in the 1950s and 1960s. It takes several binary inputs and produces a single binary output based on a weighted sum. Each input x_j is multiplied by a corresponding weight w_j, and the result is compared against a threshold. In precise terms, a perceptron computes:

$$\text{output} = \begin{cases} 0, & \text{if} \quad \sum_j w_j x_j \leq \text{threshold}, \\ 1, & \text{if} \quad \sum_j w_j x_j > \text{threshold}. \end{cases} \tag{1}$$

A Decision-Making Example

Imagine deciding whether to attend a cheese festival in Gouda based on three factors:

1) Is the weather good?

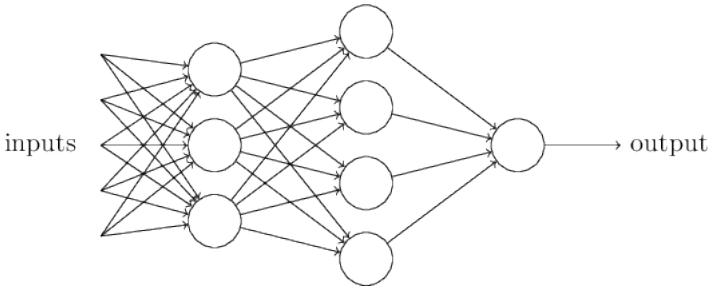
2) Do your friends want to accompany you?

3) Is the venue easy to reach?

We encode these factors as binary variables x_1, x_2, and x_3. If you love cheese so much that you would attend regardless of your friends' interest or the commute, but would never go if the weather is bad, you might assign weights $w_1 = 6$, $w_2 = 2$, $w_3 = 2$ and choose a threshold of 5. In this case, the output is 1 (go to the festival) if the weighted sum exceeds the threshold, and 0 otherwise.

Multi-Layer Perceptrons

Perceptrons can be combined into layers. In a multi-layer network, the output of one layer serves as the input to the next. For example, the first (input) layer might make simple decisions based on raw data, while subsequent layers combine these decisions into more abstract and complex conclusions:



Note that even if the diagram shows multiple arrows emerging from a perceptron, each perceptron still produces a single output that is distributed as input to several subsequent neurons.

Notational Simplifications

The condition $\sum_j w_j x_j >$ threshold can be written more compactly by:

1) Expressing the weighted sum as a dot product: $w \cdot x = \sum_j w_j x_j$.

2) Moving the threshold to the left side of the inequality by introducing the **bias** $b \equiv -$threshold.

Thus, the perceptron rule becomes:

$$\text{output} = \begin{cases} 0, & \text{if} \quad w \cdot x + b \leq 0, \\ 1, & \text{if} \quad w \cdot x + b > 0. \end{cases} \tag{2}$$

The bias can be interpreted as a measure of how easily the perceptron "fires" (i.e., outputs a 1). In the remainder of these notes, we will use the bias formulation exclusively.

3 Sigmoid Neurons

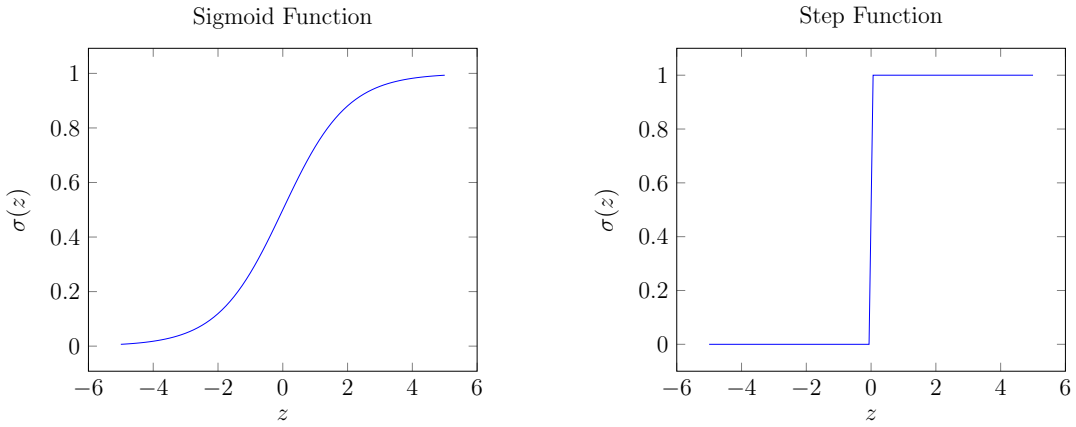
While perceptrons are useful for basic decision-making, their binary nature makes them unsuitable for gradual learning. A slight change in a weight or bias in a perceptron can flip the output abruptly from 0 to 1. In contrast, a **sigmoid neuron** uses a smooth activation function to produce outputs that vary continuously between 0 and 1. Its output is given by

$$\sigma(w \cdot x + b),$$

where $\sigma(z)$ is the sigmoid function defined as

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

This function approximates the behavior of a perceptron: for large positive z, $\sigma(z) \approx 1$, and for large negative z, $\sigma(z) \approx 0$. However, because the transition is smooth, small changes in weights and biases yield small changes in the output. It is an essential property and requirement for gradient-based learning.

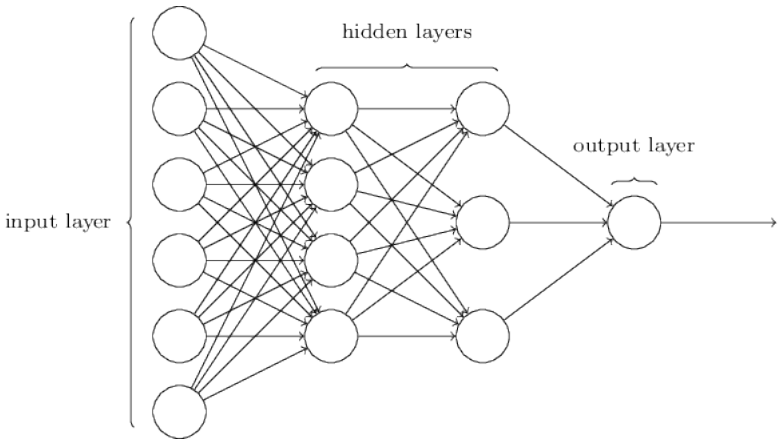


4 The Architecture of Sigmoid Neural Networks

A typical neural network is organized into layers:

- The **input layer** receives the raw data (e.g., pixel values from an image).
- One or more **hidden layers** process the input into higher-level features.
- The **output layer** produces the final prediction (e.g., the digit class).

The following diagram illustrates a network with these three layers:



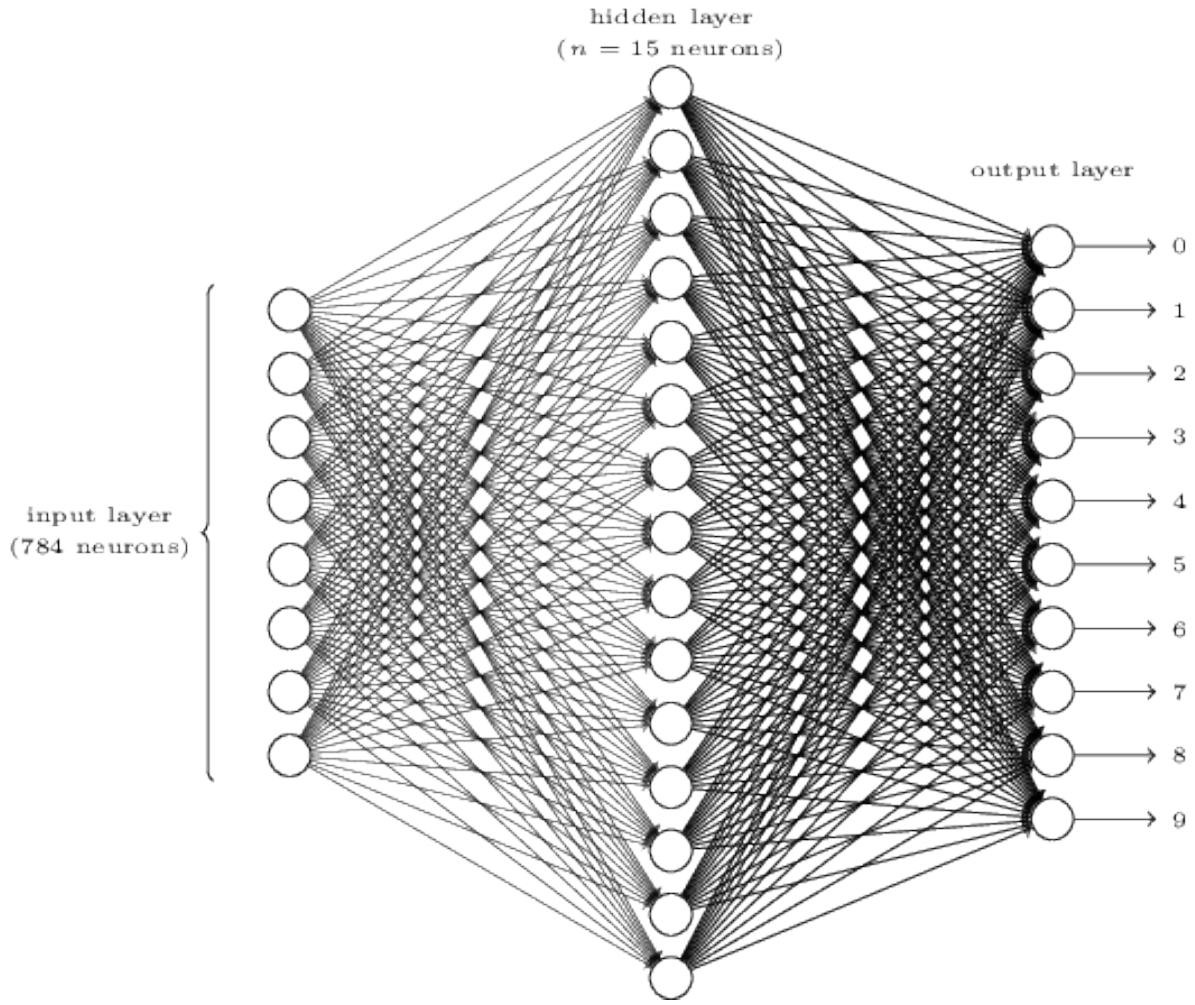
5 A Simple Network for Classifying Digits

We now apply these ideas to the task of handwritten digit classification. The problem can be divided into two parts:

1) **Segmentation:** Breaking an image containing several digits into individual digit images. As shown in paper figure.

2) **Classification:** Determining which digit (0–9) is depicted in each segmented image.

Here, we focus on classification. We will use a three-layer neural network:



- The **input layer** has 784 neurons, corresponding to the 28 by 28 pixels of a digit image.
- The **hidden layer** contains n neurons (e.g., $n = 15$ in our example), which can be adjusted to experiment with network complexity.
- The **output layer** consists of 10 neurons, each representing a digit from 0 to 9. The network's prediction is given by the neuron with the highest activation.

6 Learning with Gradient Descent

To train the network, we use a training set. Here, we utilize the MNIST dataset containing tens of thousands of handwritten digit images. Each training input is represented as a 784-dimensional vector x, where each component corresponds to a pixel's gray value. The desired output for an input image depicting the digit 6, for example, is a 10-dimensional vector

$$y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T.$$

Cost Function

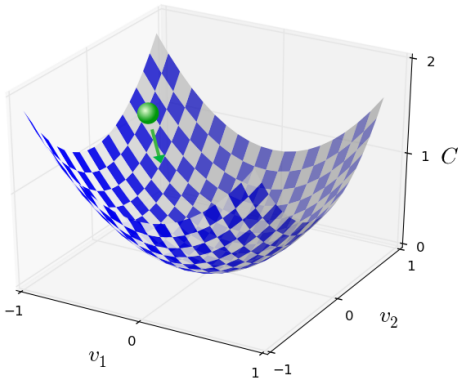
To measure how well the network performs, we define the quadratic cost function:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2, \tag{4}$$

where:

- w denotes all the weights in the network,
- b denotes all the biases,
- n is the number of training examples,
- a is the output vector produced by the network when input x is given.

Gradient Descent Overview



The goal is to adjust w and b to minimize $C(w,b)$. Consider a function $C(v)$ of variables $v = (v_1, v_2, \ldots, v_m)^T$. A small change Δv causes a change in C given approximately by

$$\Delta C \approx \nabla C \cdot \Delta v, \tag{5}$$

where the gradient is

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T. \tag{6}$$

To ensure ΔC is negative (i.e., to move downhill in the cost landscape), choose

$$\Delta v = -\eta \nabla C, \tag{7}$$

with a small positive parameter η called the **learning rate**. Then, the update rule is

$$v \to v' = v - \eta \nabla C. \tag{8}$$

Application to Neural Networks

For the weights w_k and biases b_l, the update rules become:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}, \tag{9}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{10}$$

By iteratively applying these updates, the network gradually "rolls down" the cost function landscape toward a minimum.

Stochastic Gradient Descent

Computing the gradient over the entire training set can be computationally expensive. **Stochastic gradient descent** (SGD) approximates the true gradient by using a small, randomly selected subset (mini-batch) of the training data. If $X_1, X_2, \ldots, X_m$ denote the mini-batch, then we approximate:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}, \tag{11}$$

and update the weights and biases accordingly:

$$w_k \to w_k - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial C_{X_j}}{\partial w_k}, \tag{12}$$

$$b_l \to b_l - \frac{\eta}{m} \sum_{j=1}^{m} \frac{\partial C_{X_j}}{\partial b_l}. \tag{13}$$

One complete pass through the training set is called an **epoch**. After each epoch, the process is repeated with a new randomly selected mini-batch, allowing the network to gradually learn the appropriate weights and biases.