Using neural nets to recognize handwritten digits

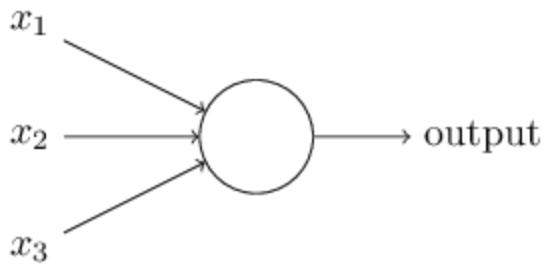
Shivansh Rao
Department Of Electronics & Communication Engineering
Delhi Technological University

Abstract

This project aims on implementing a neural network that learns to recognize handwritten digits. It is just a basic project and uses no special neural network libraries. But this naive project can recognize digits with an accuracy over 96 percent, without human intervention.

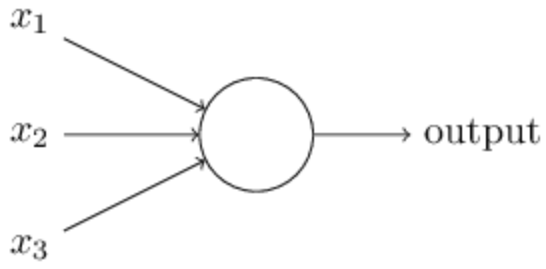
Perceptrons

A perceptron takes several binary inputs, $x_1, x_2, \ldots$, and produces a single binary output:



In the example shown the perceptron has three inputs, x_1, x_2, x_3. In general it could have more or fewer inputs. Rosenblatt proposed a simple rule to compute the output. He introduced *weights*, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some *threshold value*. Just like the weights, the threshold is a real number which is a parameter of the neuron.

Sigmoid Neurons

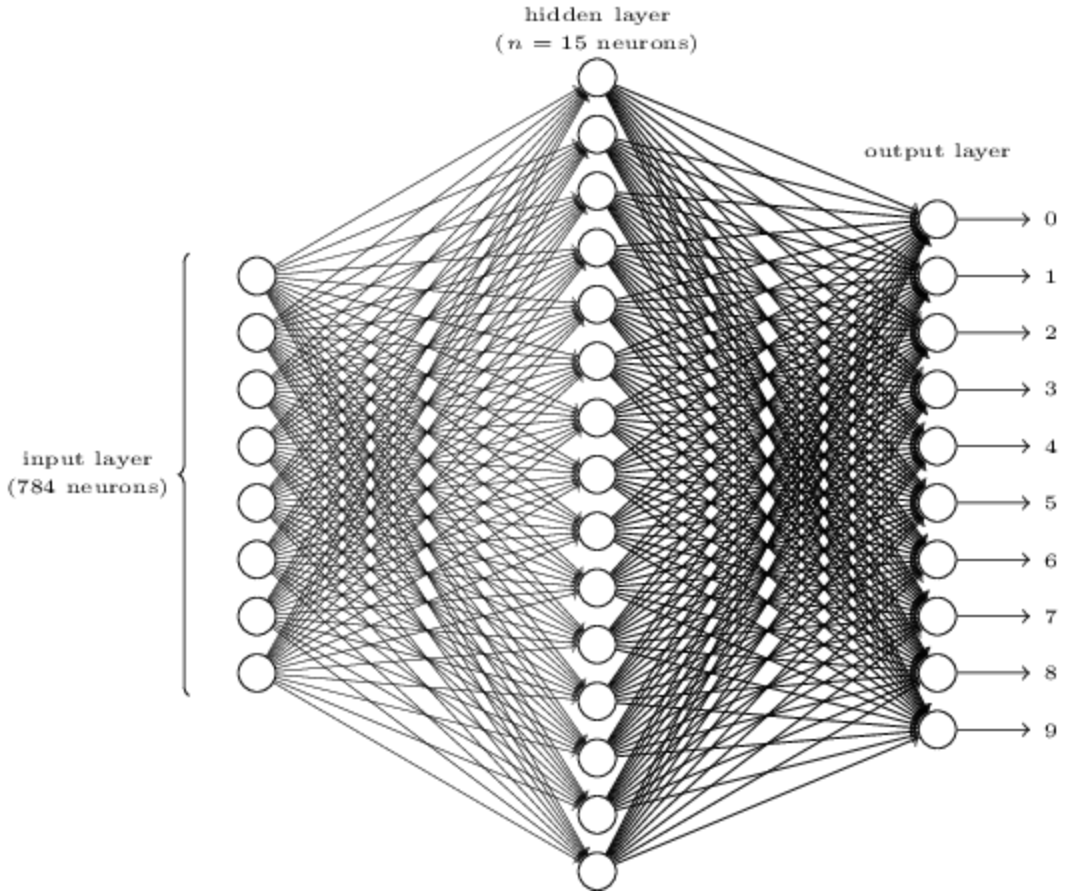


Just like a perceptron, the sigmoid neuron has inputs, $x_1, x_2, \ldots$. But instead of being just 0 or 1, these inputs can also take on any values *between* 0 and 1. So, for instance, 0.638… is a valid input for a sigmoid neuron. Also just like a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, \ldots$, and an overall bias, b. But the output is not 0 or 1. Instead, it's $\sigma(w \cdot x + b)$, where σ is called the *sigmoid function*, and is defined by:

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

Simple network to identify handwritten digits

To recognize individual digits we will use a three-layer neural network:



The input layer of the network contains neurons encoding the values of the input pixels. Training data for the network consists of many 28 by 28 pixel images of scanned handwritten digits, and so the input layer contains 784=28×28 neurons.

Learning with gradient descent

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

By repeatedly applying this update rule we can "roll down the hill", and hopefully find a minimum of the cost function. In other words, this is a rule which can be used to learn in a neural network.