2 A typical neural network for solving the MNIST database

Contents

Abstract

The MNIST database (Mixed National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database contains 60,000 training images and 10,000 testing images. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23 % (in a paper by the original creators of the database, they used a support vector machine to get an error rate of 0.8%).[1]

2.1 Initial example

A typical neural network to solve the MNIST task might have

- ***784 inputs*** (pixels) connected to ***1,000 neurons***, which are in turn connected to ***10 output targets*** (one for each digit).
- Each layer is *fully connected* to the layer above (so each input pixel has a set of ***weights*** that is connected to every neuron in the layer above).

This architecture can do quite well on handwritten digit recognition, but is orders of magnitude smaller than the human brain.

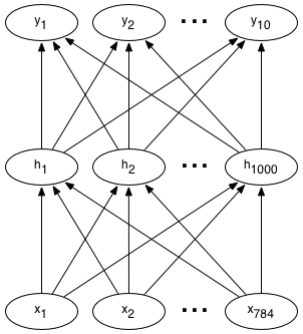


Figure 2.1: A graphical representation of the network described above, where x_i are the inputs, h_j are the hidden neurons and y_k are the output class variables. The arrows represent the weights; the first layer connects 784 pixels to $1,000$ hidden neurons (units) using $784 \times 1,000 = 784,000$ weights (typically represented as a $784 \times 1,000$ matrix). The next layer connects $1,000$ hidden units to 10 output labels using $10 \times 1,000 = 10,000$ weights; thus this network has a total of $10,000 + 784,000 = 794,000$ weights.

Remark. The number of neurons to use in the hidden layer is just a heuristic.

2.2 A more detailed example[2]

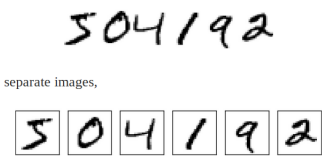
Remark. "I have included this description here for consitency, but it includes more technical details that can be found in the later sections of the course."

[1] Details from Wikipedia
[2] `http://neuralnetworksanddeeplearning.com/chap1.html#a_simple_network_to_classify_handwritten_digits`

2.2.1 Setup

We can split the problem of recognising handwritten digits into two sub-problems:

1. First, we would like a way of breaking an image containing many digits into a sequence of separate images, each containing a single digit. Humans solve this segmentation problem with ease, but it's challenging for a computer program to correctly break up the image.



2. Once the image has been segmented, the program then needs to classify each individual digit.

We focus on the step 2 for the remaining of this subsection.

Consider using the following neural network

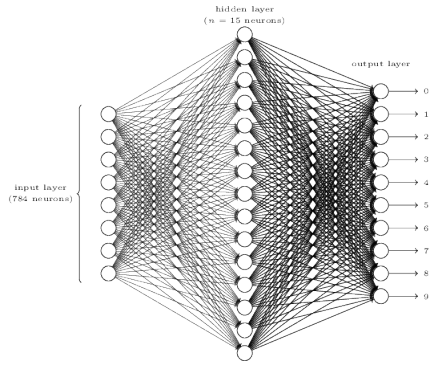


Figure 2.2: **The input layer** of the network contains neurons encoding the values of the input pixels. As before, the input layer contains $784 = 28 \times 28$ neurons (for simplicity we have omitted most of the 784 input neurons in the diagram). The input pixels are greyscale, with a value of 0.0 representing white, a value of 1.0 representing black, and in between values representing gradually darkening shades of grey.
The second (or hidden) layer contains any number of neurons n, and we'll experiment with different values for n. The example shown illustrates a small hidden layer, containing just $n = 15$ neurons.
The output layer of the network contains 10 neurons. We number the output neurons from 0 through 9, and figure out which neuron has the *highest activation value* (indicating that the input has been classified by the algorithm as this).

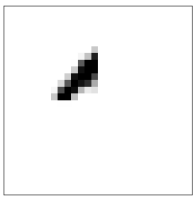
2.2.2 Why use 10 output neurons?

A seemingly natural way of doing this is to use just 4 output neurons, treating each neuron as taking on a binary value, depending on whether the neuron's output is closer to 0 or to 1.

The ultimate justification is empirical: we can try out both network designs, and it turns out that, for this particular problem, the network with 10 output neurons learns to recognise digits better than the network with 4 output neurons.

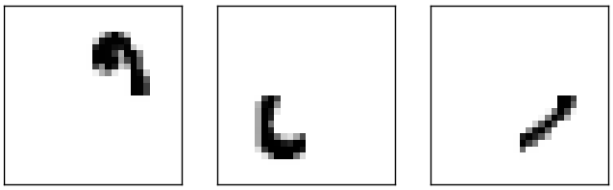
To understand why we do this, it helps to think about what the neural network is doing from first principles:

- Consider first the case where we use 10 output neurons.
  - The first output neuron is trying to decide whether or not the digit is a 0.
  - It does this by weighing up evidence from the hidden layer of neurons.
  - What are those hidden neurons doing? Well, just suppose for the sake of argument that the first neuron in the hidden layer detects whether or not an image like the following is present:

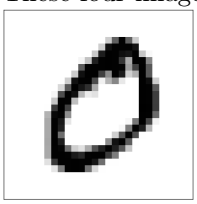


It can do this by *heavily weighting input pixels which overlap with the image*, and only lightly weighting the other inputs.

– In a similar way, suppose for the sake of argument that the second, third, and fourth neurons in the hidden layer detect whether or not the following images are present:



– These four images together make up the 0 image:



So if all four of these hidden neurons are firing then we can conclude that the digit is a 0.

Note. That is not the only sort of evidence we can use to conclude that the image was a 0 - we could legitimately get a 0 in many other ways (say, through *translations* of the above images, or *slight distortions*).

Remark. Supposing the neural network functions in this way, if we had 4 outputs, then the first output neuron would be trying to decide what the most significant bit of the digit was and there is no easy way to relate that most significant bit to simple *shapes* like those shown above.

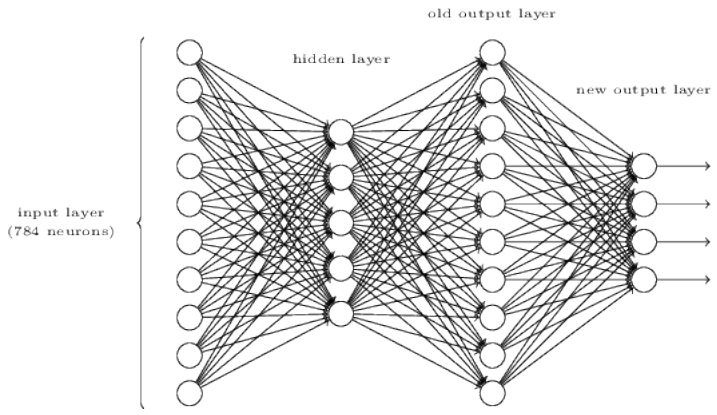


Figure 2.3: There is a way of determining the bitwise representation of a digit by adding an extra layer to the three-layer network above. The extra layer converts the output from the previous layer into a binary representation.