Neural Network applied to the MNIST Database

Nicolas Gabrion

1 Introduction

The MNIST database is a famous tool for data analyst as it contains a huge number of digit images that is extremely useful to start working on computer vision. I downloaded this database from Kaggle and took part to the competition they organized around this database. That's why the scores I will mention in this report are the scores obtained on this platform. I used python and the library Keras for neural network.

2 Data

This database contains circa 70000 digits that are quite evenly distributed. As a consequence, we don't have to worry about balancing this repartition ourselves. The size of the image is 28x28 so every example is made of 784 pixels. The images are coded in gray level and the value is included between 0 and 255. However, in order to ease the gradient descent, we will normalize these values by divided them by 255. Now every value is incuded between 0 and 1.

3 Perceptron

We first implemented a perceptron neural network to solve the problem. Indeed, that is the easiest neural network and it still gives pretty good results.

3.1 Input and Output Layers

A perceptron is a simple succession of layers the input layer is made of 784 neurons (the size of each image) and the output layer is made of 10 neurons. Each output neuron i gives the computed probability for the input represent the digit i. Therefore, the structure is nearly defined.

3.2 Hidden Layers

We have to design the hidden layers. According to most articles on the topic, it appears that perceptron are working very well with 1 hidden layer. In addition,

the size of the hidden layer is usually included between the sizes of the output and input layers. Then, we will pick it between 10 and 784 and it will be our first parmater to try to improve the accuracy of our perceptron.

3.3 Model Parameters

We have a lot of paramaters to choose to run the algorithm. First, we have to set the number of epochs ie the number of iteration on the training data before predicting. This paramater is very significant because if it is too small, the algorithm won't be trained enough and the results could be average but if it is too big, the algorithm could need hours to give results. We will pick this number between 10 and 100 for now.

Then, we need to define the activation functions for every layer. We will use the most popular activation functions : ReLU for hidden layers and softmax for output layer.

The ReLU function is defined as $ReLU(x) = max(0, x)$ and it is used because it is a good way to simulate the behaviour of biological neurons. The softmax function is the most used function for output layers that needs to give a probability (ie for NN solving classification problems). $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ if the size of z is K.

4 Convolutional Neural Network

5 Parameters