UNIVERSITÀ
DI SIENA
1240

Dipartimento di ingegneria dell'informazione e scienze
matematiche
Computer and Automation Engineering

Robotics and Automation

Handwritten classification of MNIST dataset with TensorFlow

Professor: **F. Scarselli**

Student: **Francesco Vigni**

A.A. 2017-2018

Introduction

This report describes the work done with the MNIST dataset, and TensorFlow® framework. The dataset provides two different sets, the first one contains 60K examples of handwritten digits from 0 to 9, labeled as trainingset; the second one contains other 10K examples of handwritten digits, labeled as testingset.
The idea is to train a neural network to let it learn the association between a handwritten digit and its real numerical value.

Figure 1: examples of digits

Chapter 1

The Neural Network

Neural networks with multiple hidden layers can be useful to solve classification problems with complex data, such as images. Each layer can learn features at a different level of abstraction. However, training neural networks with multiple hidden layers can be difficult in practice, due to the computational time of the simulation. One way to effectively train a neural network with multiple layers is by training one layer at a time. We can achieve this by training a type of network known as an autoencoder, in an unsupervised fashion, for each desired hidden layer, and training another type of network known as softmax as the output layer, and in the final step, stack the layers together to form a deep network, which is trained one final time in a supervised fashion to increase accuracy. We can summarize the procedure as follows.

1. Training the first autoencoder, named *autoenc1*, that applies a function $f : \mathbb{R}^d \to \mathbb{R}^d : f(x) = x$ and $x \in \mathbb{R}^d$ going throught a hidden layer of dimension $k = 100$.

2. Training the second autoencoder, named *autoenc2*, that applies the function $g : \mathbb{R}^k \to \mathbb{R}^k : g(z) = z$ and $z \in \mathbb{R}^k$ going throught a hidden layer of dimension $h = 50$.

3. Training the last layer, named *softmax1*, that applies the function $f : \mathbb{R}^h \rightarrow \mathbb{R}^m : i(s) = y$ and $s \in \mathbb{R}^h$ and $y \in \mathbb{R}^m$.

4. Now it is possible to remove the output layers of the networks *autoenc1* and *autoenc2* in order to be coherent with the dimensions; eventually it is possible to stack together the three networks, as *autoenc1*, *autoenc1*, *softmax1* in order to create the network in Fig. 1.1.

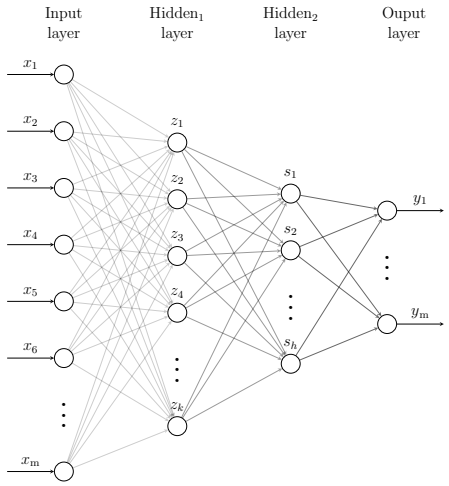


Figure 1.1: Architecture of deep Neural Network

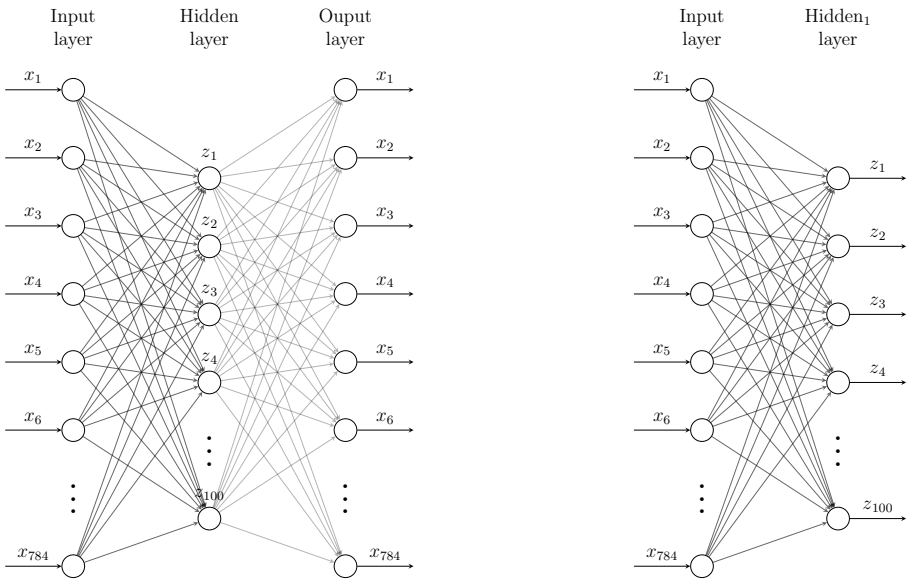
The whole network obtained is now applying a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ where m is the dimension of the output (10) or numbers of classes of our problem, one class per each digit.

The final architecture obtained (Fig. 1.5) is a 3 layer full-connected network, most commonly known as deep neural network, because of the numbers of hidden layers between the input and the output layer.

1.1 Autoencoders

The first autoencoder *autoenc1* applies a function in order to learn $f(x) = x$; in other words, it is trying to learn an approximation to the identity function. The

training in this unsupervised experiment 1.2a, is an iterative learning process in which data are presented to the network one at a time, and the weights values w_{ij} associated with the connection between layers are adjusted each time. The status of a learning process can be evaluated with Epochs[1] or with the Mean Square Error between the target and the outputs of the NN (*MSE*). During this learning phase, the network learns by adjusting the weights, so it can be able to predict the correct pattern of input samples. After having trained the net in fig. 1.2a, the output layer is removed in order to create fig. 1.2b, in doing so we obtain a compressed version of the input which is a smaller representation of the data.



(a) Full architecture

(b) Removed output layer

Figure 1.2: Autoencoder 1

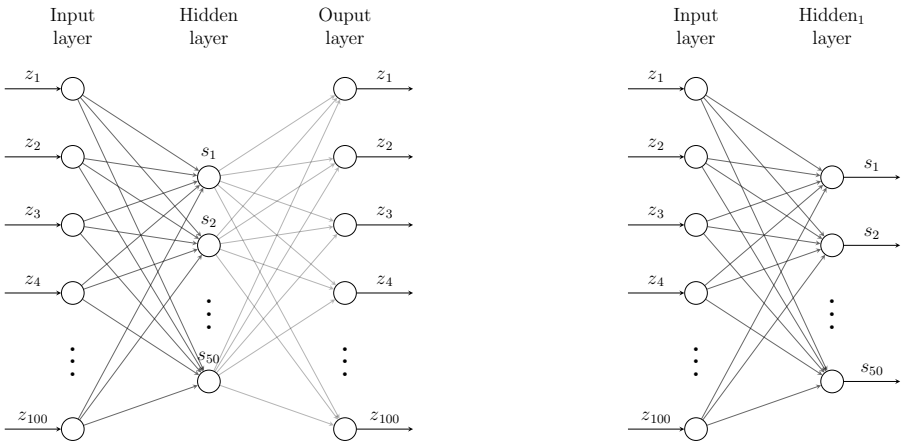
In *autoenc1* and *autoenc2* the number of neurons in the hidden layer are less than the size of the input, thus the autoencoder learns a compressed representation of the input. The hidden layer and the output layer in autoencoders are applying

[1]One epoch is one forward pass and one backward pass of all the training examples

the function 1.1 where $i \to j$ is a forward connection between node $i \in L_k$ and node $j \in L_{k+1}$; overmore N_k is the number of neuron in layer k that are linked with neuron j with a connection labeled with weight w_{ij}.

$$f_j(a_j) = \frac{1}{1 + e^{-a_j}} \qquad a_j = \sum_{i=1}^{N_k} x_i \cdot w_{ij} \tag{1.1}$$

The second autoencoder applies a function $g(z) = z$ similarly to the previous description. We first train the network 1.3a, then we can remove the output layer and obtain 1.3b; now we have even a smaller representation of the data (vector size=50).



(a) Full architecture

(b) Removed output layer

Figure 1.3: Autoencoder 2

1.2 Softmax layer

The softmax layer in Fig. 1.4 is so called because its layer applies the softmax function eq. $i_j(s) = \frac{e^{s_j}}{\sum_{k=1}^{m} e^{s_k}}$, very useful in probabilistic multiclass classification problems, because it simultaneously ensure that:

- the outputs of the network are confined to the interval (0,1)
- the sum over all the outputs is equal to one.

In doing so, this layer is actually returning probabilistic values of the class belonging to our inputs. The training process takes as inputs the output of the NN 1.3b, and uses the same logic of learning as the other: back propagation as training algorithm and gradient descend as optimality algorithm.

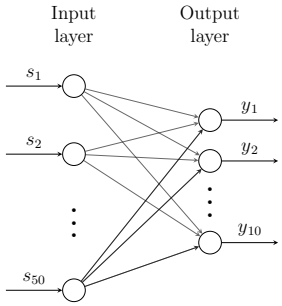


Figure 1.4: Softmax layer

At this point we have 3 trained neural network, and the next step is to stack together 1.2b, 1.3b and 1.4. Eventually we have obtained our deep neural network Fig. 1.5 that takes as inputs digit images, reordered as vectors, and returns values as the probability of assign to the right class ω the pattern x fed as input.

This Neural Network contains the values w_{ij} that the previous processes have found, but in order to have better results, we train the network in Fig. 1.1 with as inputs, the images (each of them is saved in a 28x28 matrix, then we reorder them so that we obtain a vector of 784 elements per each image) and as outputs, the target labels provided by the dataset.

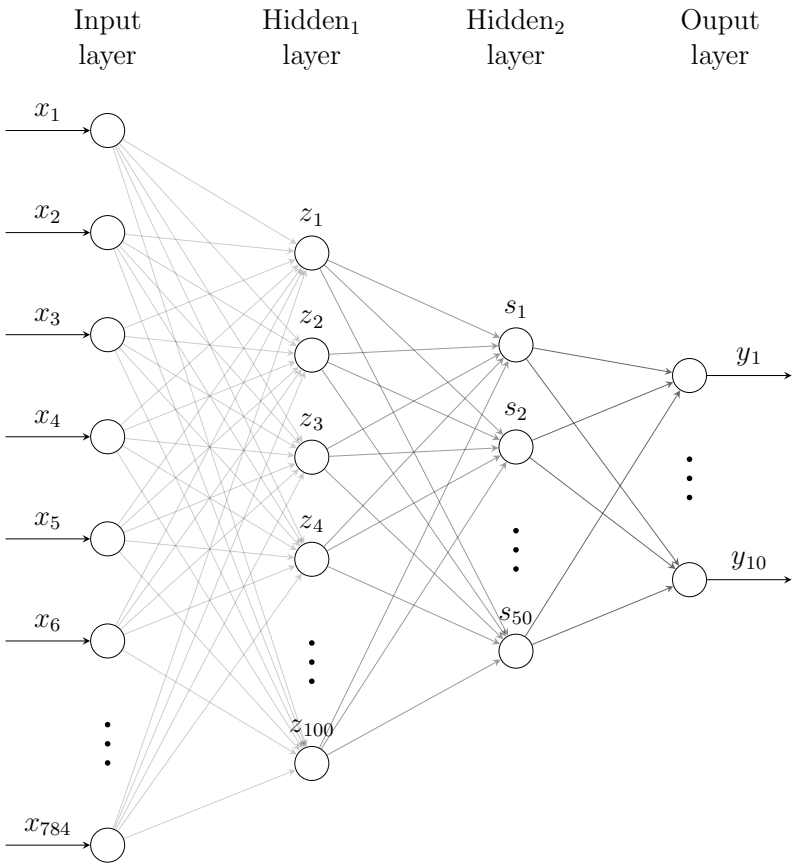


Figure 1.5: Deep Neural Network

Results

The deep neural network 1.1 is now fully trained. In order to evaluate the accuracy of the pattern recognition we need explore the net with the testing set data. The confusion matrix Fig. 1.6 computed on the outputs of the network fed with testing images, is a great indicator of the accuracy of our network.

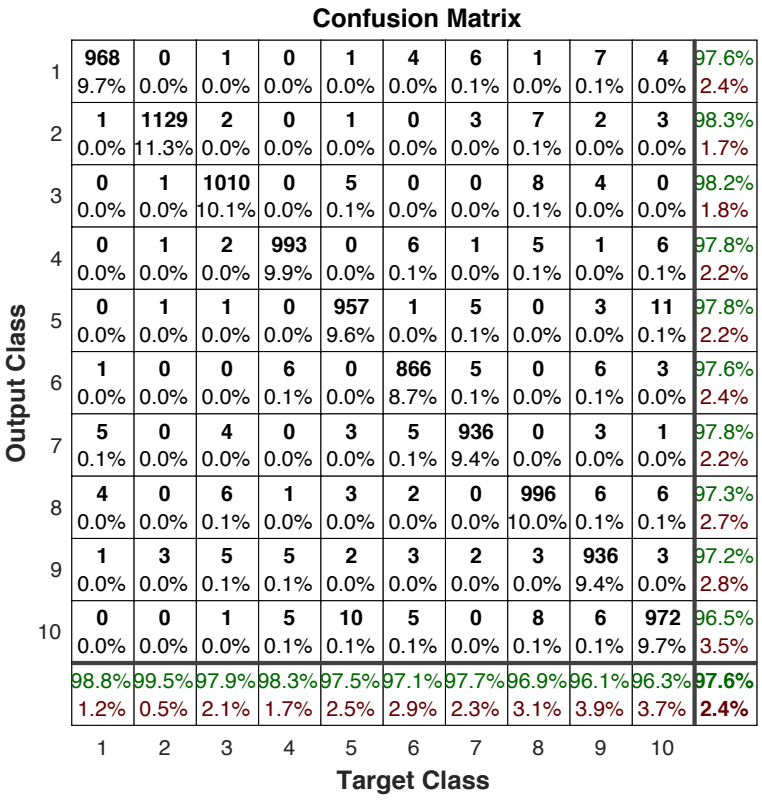


Figure 1.6: Confusion Matrix of testing set

Conclusion

This project applies learning techniques to MNIST handwritten dataset. As we can see in the previous confusion matrix the accuracy of the final work is 97.6%. The overall idea is to train *autoenc1*, *autoenc2* and *softmax1* once per time and to crop the nets in order to have coherents dimension between network interconnections. At the end of this process we stack all the partial neural network together and the deep neural network come to life.

The satisfaction behind this project can be experimented by running the file "MNIST_drawsim.m" which is a matlab function that allows the user to draw a digit and returns the correct digit value 97,6 times over 100.