NEURAL NETWORKS
Assignment 4 of Machine Learning I, A.Y. 2021/2022

Marco Macchia
DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
Università Degli Studi di Genova

This report describes the fourth assignment of Machine Learning I, what were the requirements and what are the obtained results.

I. Introduction

NEURAL NETWORKS are a set of simple algorithm that are able to recognize patterns. Their name (and also structure) is inspired by the human brain.

Neural networks are commonly used in many applications, e.g. prediction of stock market, weather forecasting and image recognition. When a person tries to unlock his own phone using his face, for example, almost always the smartphone implements a neural network to recognize the allowed face and unlock the phone.

II. Theory of Neural Networks

A. Mathematical theory

Neural network is made of *neurons*. A neuron is a single base unit that takes an input, computes a decision and gives an output. The neuron can receive inputs either from outside the neural network or from another neuron, and can also send his output to the outside of the neural network of to another neuron.

Each neuron based his computation on a set of parameters $\mathbf{w}$ called *weights*, that slightly changes during all the learning process. The *weight update* (this is the name of the procedure) follows this formula

$$\mathbf{w}_{l+1} = \mathbf{w}_l + \Delta\mathbf{w}_l \tag{1}$$

Let $\mathbf{x}$ be the inputs of the neural network. The net input r can be defined as

$$r = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{d} w_i x_i \tag{2}$$

Let a be the *activation value*, i.e. the output of the neuron. a is computed as follows

$$a = f(r - \theta) \tag{3}$$

where r is the net input, θ is a threshold (an additional weight) and $f()$ is the activation function. In particular, $f()$ can correspond to many functions, e.g. *Heaviside step, sigmoid, hyperbolic tangent* or *softplus*.

The correction factor $\Delta\mathbf{w}_l$, used in formula 1, can then be computed as

$$\Delta\mathbf{w}_l = \eta\delta_l\mathbf{x}_l \tag{4}$$

where η is a parameter and $\delta_l = \frac{1}{2}(t_l - a_l)$ (t is the target of $\mathbf{x}_l$).

B. Multiple-layers Neural Network

When there are more the one set of neurons operating at the same *depth*, then the neural network has *multiple layers*. The simplest multi-layer NN is a neural network which has one *input* layer, one *hidden* layer and one *output* layer.

The *input* layer takes $\mathbf{x}$ and spreads it to the *hidden* layer. The *hidden* layer takes data from the *input* layer and spreads his calculations to the *output* one, that gives the data to the outside.

The *hidden* layer is called in this way because it is not visible by the outside of the neural network. The more hidden layers there are, the better the classification is. However, many *hidden* layers means many more computational power required, so it is important to find the correct balance between *performance* and *layers number*.

C. Autoencoder

An *autoencoder* is a type of neural network. It has two main features:
- an *autoencoder* has the same amount of output units between input and output layers, while the hidden layer has less neurons;
- the *autoencoder* is trained using the same pattern as both the input and the target, so that it is required to replicate the input as output.

The most important thing about an *autoencoder* then are the values of the hidden layer, because observations from the same class will results in very similar weight $\mathbf{w}$. At first sight the *autoencoder* can be seen as a *classifier*. However this is not the case, because it does not receive any *target*, so it is just unable to *classify* something.

III. The Assignment

The given assignment consist of three tasks:
- **Task 0:** Neural networks in Matlab
- **Task 1:** Feedforward multi-layer networks
- **Task 2:** Autoencoder

A. Task 0: Neural networks in Matlab

The first task simply ask to follow a tutorial for dealing with Neural Networks in Matlab. For this purpose, it is require to install and run the *Deep Learning Toolbox*. The toolbox can be executed using the command *nftool*, which toggle the toolbox interface.

In order to solve an input-output problem, it is required to load a data set. The toolbox provided some examples taken from the *UCI Machine Learning Repository*. For this example, the body fat set is used. The sample are then divided in three different subsets:

- *training* subset (70% of the entire set),
- *validation* subset (15% of the entire set),
- *test* subset (15% of the entire set).

Note that *validation* set is useful to validate that the network is generalizing and to stop training before overfitting (i.e. the NN is trained *too well*). With the overfitting, the network is well trained on recognizing what he has already seen, but may struggle on recognizing data that he has never seen.

The toolbox then train a neural network and examine the result, giving the possibility of plotting the regression, the performance, the training state and the error histogram.

B. Task 1: Feedforward multi-layer networks

The second task asks again to follow a tutorial. In this case it is required to solve a patter recognition problem using a two-layer feed-forward network. In particular, the hidden layer has a *sigmoid* transfer function, while the output layer has a *softmax* transfer function.

Also in this case the set is divided in three subsets (with the same percentage as before, 70%,15%,15%). For this tasks, two example sets were used, *glass* set and *iris* set.

After selecting the set, in the toolbox (which can be executed using the command *nprtool*) it is possible to select the amount of *hidden neurons*, i.e. the number of neurons of the *hidden* layer (default is 10).

When the number of hidden neurons is defined, the toolbox creates and trains a neural network with the given input set. After the NN execution the toolbox is capable of plotting the confusion matrices and the *ROC* (Receiving Operating Characteristic).

C. Task 2: Autoencoder

The third task asks to implement an *autoencoder* in Matlab. The used set is the *MNIST* set already used in Assignment 3, and is made of 60.000 handwritten digit (from 0 to 9) stored in a 28x28px grayscale images.

From the *MNIST* set, it is required to extract only two classes (i.e. digits) at the time, and give them as a unique input set to the autoencoder. Since the set is quite large, from the two class sets a fixed amount of observation are randomly extracted (default is 250 per class set), in order to speed up the execution.

The two selected classes however are not random. For this task, with the purpose of highlighting the autoencoder performance, some pair of digits are quite easy to discern, while others may be much more difficult. The pair of classes are:

$$classes = \begin{pmatrix} 1 & and & 8 \\ 3 & and & 8 \\ 1 & and & 7 \\ 5 & and & 6 \end{pmatrix} \tag{5}$$

An autoencoder can be initialized and trained in Matlab using the functions *trainAutoencoder()* and *encode()*.

trainAutoencoder() requires as inputs the data set and the number of hidden nodes. For this simulation, in order to have some viewable data, the number of hidden nodes has been set to 2. Note that the *Deep Learning Toolbox* uses a different notation as in Machine Learning I course, so it is necessary to transpose all the data before continuing. The function returns an *autoencoder variable*.

encode() is then used to encode the data, so that they are ready to be plotted. *encode()* requires two inputs, which are the *autoencoder* and the data set. It returns the encoded data.

Finally the program plots the output of the two *hidden neurons*, one per axes. The data are plotted using the *plotcl()* function, provided into the Machine Learning I course. The function plots the data using different shapes and colours, that corresponds to the class which the observation belongs to.

If the two sets of different shapes and colours are distant from each other, it means that the autoencoder has learned well. Otherwise if the two sets are mixed, it means that the learn process were more confused, and the two class aren't indistinguishable.

IV. Results

A. Task 1

Regarding the second task, the feed-forward network can be analyzed using the confusion matrices. The figure 1 shows the result obtained using the glass set and 10 hidden neurons. The overall performance is good, with an accuracy of 93.9%. It struggled a bit with the validation set, with the accuracy dropped to 84.4%.

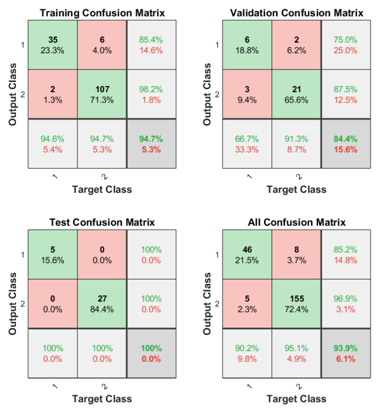

Fig. 1: Glass set, with 10 hidden neurons

The figure 2 shows the result obtained using the glass set and 25 hidden neurons. The results are better than what we obtain with 10 neurons, with an accuracy of 99.1%.

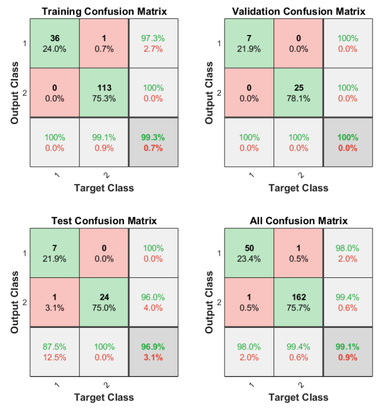

Fig. 2: Glass set, with 25 hidden neurons

The figure 3 shows the result obtained using the iris set and 10 hidden neurons. Here there are three classes instead of two. The accuracy percentage in this case is 97.3%. Also here the neural network struggled with the validation set: the accuracy dropped to 87%.

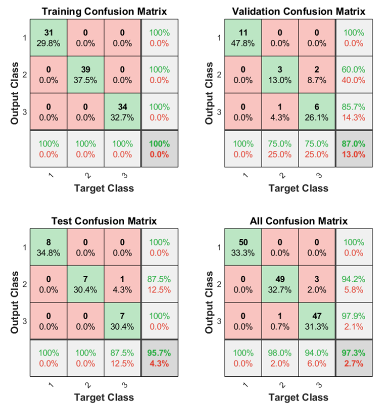

Fig. 3: Iris set, with 10 hidden neurons

Finally the figure 4 shows the result obtained using the iris set and 25 hidden neurons. The results are quite similar and comparable to the case with 10 hidden neurons. The overall accuracy is 98%.

B. Task 2

Regarding Task 2, we can study the performance of the *autoencoder* using the graph plotted with the *plotcl()* function.

In the figure 5 are represented the performance when the input is made by observations belonging to class 1 (orange triangles) and 8 (red dots). The two classes are generally speaking very distant from each other (apart of a single red dots that is very close to the class 1 region, meaning that an 8 is very similar to a 1), meaning that the two classes are quite easily distinguishable.

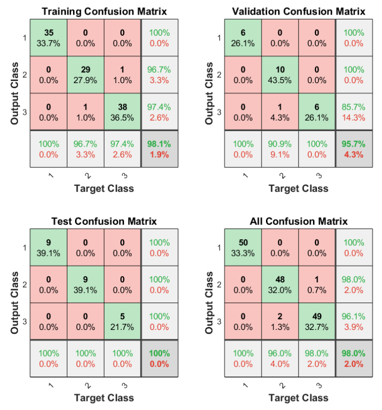

Fig. 4: Iris set, with 25 hidden neurons

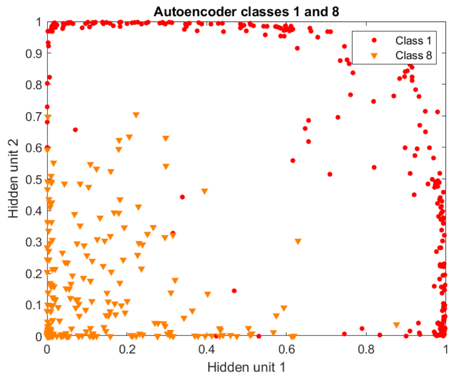

Fig. 5: Autoencoder with classes 1 and 8

The figure 6 produces a different output than the previous one. At this time the autoencoder had to deal with different observations of the digits 3 (orange triangles) and 8 (green dots), which can be very similar in many cases. The figure clearly shows that the two digits are not easily distinguishable, and it is highlighted by the fact that there are many threes inside the top-left region (populated by eights), and the middle region of the graph shows that many images can't be understood easily.

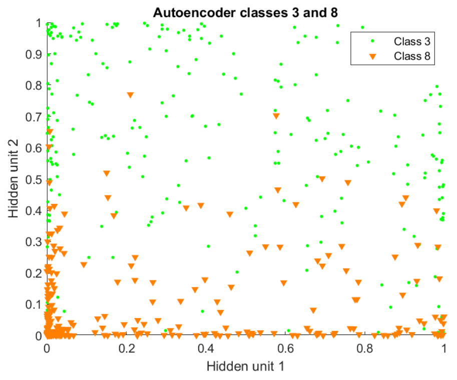

Fig. 6: Autoencoder with classes 3 and 8

The figure 7 shows the outputs of the hidden neurons when the input correspond to a set of 1 (red dots) and 7 (blue triangles). Also in this case there are some seven that are interpreted as one and vice-versa. However, the data appears more ordered than before, meaning that the difference between 3 and 8 is less then the difference between 1 and 7.

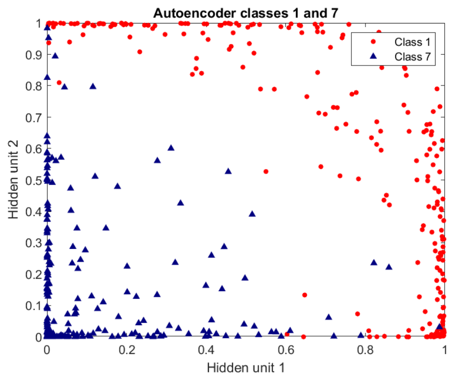

Fig. 7: Autoencoder with classes 1 and 7

The last figure shows the results of a set made by observations belonging to classes 5 (dark red squares) and 6 (green diamonds). Here we can see something different than in previous cases, because many images representing the 6 seems to be similar to images that represents 5, but not vice-versa. Generally speaking, this could mean that some 6 are similar to 5, but almost ever a 5 is distinguishable from a 6.

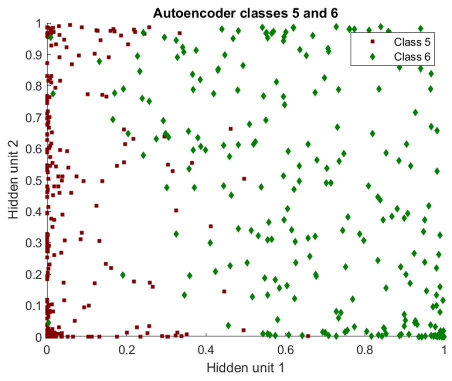

Fig. 8: Autoencoder with classes 5 and 6