NNDL Homework 1
Supervised Deep Learing

Enrico Convento – ID: 2023572

February 7, 2022

1 Introduction

In this homework, two supervised learning tasks have been considered. In particular, the first part is associated with regression, which consist of approximate a function from data, while the second part is associated to a classification task based on the FashionMINST dataset, in which we want to classify each image provided as accurately as possible. The structure of the report is divided in two parts, the first introduces the methods used, while the second one is used to describe the results that have been obtained. Each section is split in one part associated to the regression task and another one linked to the classification one.

2 Methods

2.1 Regression

In this first task we analyze the regression task, in particular we are provided with two datasets, one for training and one for testing. The analysis is structured in the following way, first a simple fully connected network was implemented with the class *Net*, it consists of only two hidden layers. Through this network, an analysis of the various components that compose the learning process was performed. The class *Training* manage all the training process, from the single epoch run to the losses plot. The class *KFoldCrossValidator* implements Kfold cross validation which is a statistical method to evaluate the performance of a model, often used when the data is not too much to improve the data efficiency, this is used together with *Optuna* to perform different hyperparameters studies.
The parameters that have been studied are resumed in the following list:

- learning rate
- early stopping
- momentum
- momentum vs learning rate
- L2 regularization
- dropout
- optimizers

The analysis of the regression task goes on with the implementation of other structures, in particular the number of layers is changed from 2 to 3 and 4. The implemented networks, i.e. the classes *Net3*,*Net3_d*,*Net4* and *Net4_d*, have been tested and in conclusion, once the best structure for this task has been found, for such network is done the visualization of the weights and the associated activations.
In the end, an automatic optimization process was performed via *Optuna*. Such analysis is performed using the base class *Net_d* and in this case the parameters that have been tested are:

- number of neurons for each layer
- dropout probabilities
- learning rate
- weight decay

2.2 Classification

Before starting to describe specifically how the classification was performed, we begin by describing the dataset provided. The Fashion MINST dataset is made up of 70.000 28x28 grayscale images, of this dataset 60.000 are used for training, in particular 48.000 (80%) associated with training and 12.000 (20%) for validation and the remaining 10.000 for the test part. The *ListDataset* class was created to handle this type of dataset. The classification has been done using more than one model, in the following, the methodologies applied will be introduced. The classification task is divided into three main sections. In the first section a simple network is implemented and with it a small study on the basic parameters have been preformed, in particular the analysis focus on the learning rate, the weight decay and the optimizers, also in this case the tool *Optuna* were used. The implemented network is a fully connected network made up of two layers. The training is managed by the class *Training* as for the regression part, in particular it allows you to set parameters such as the optimizer and batch size easily. After the training the accuracy, the confusion matrix are observed for this network and then the associated weights and activations are displayed. As a last step, the *activation-maximization* was implemented for the network. The second section deals with a convolutional neural network composed of three convolutional layers and three linear layers, which undergoes the same analysis as before. The last part deal with automatic optimization performed with *Optuna*, the parameters that are modified are :

Number of convolutional layers	2,3 or 4
Number of channels	4 to $4 * 2^j$ (j : convolutional layer number)
Kernel size/ stride / padding	2 to 6 / 1 to 4 / 0 or 1
Number of nodes for the first linear layer	16 to 256
Number of nodes for the second linear layer	4 to 128
Learning rate/ weight decay	10^{-5} to 10^{-1}/ 10^{-7} to 10^{-1}

The overall study is composed of 50 trial, the number of trials is high because is very likely that some net structure randomly generated are not valid so in those cases the trail is pruned and use more trial permits to get a good study.

3 Results

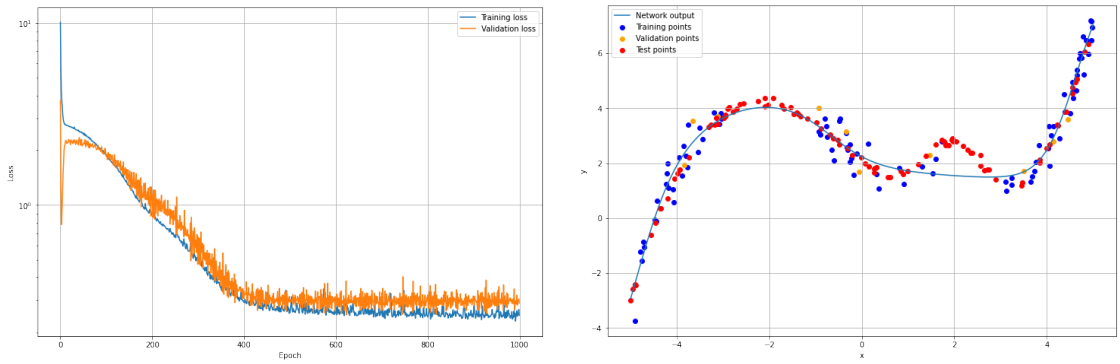
3.1 Regression

The first network implemented as introduced in Sec.2.1 has been trained 1000 epochs, the associated training and validation loss can be observed in Fig.3.1(a), in particular the final loss obtained is around the 0.25 for the training and 0.29 for the validation. The results are good but not satisfactory enough, as it can be observed in Fig.3.1(b), the model is able to fit the data, but the predictions' capability are not so good.

3.1.1 Parameters analysis

In this section will be analyzed the results obtained after performing the parameters study.

Learning rate The first test is performed on the learning rate, that is the quantity that controls the speed with which the model adapts, the results show that as from the theory, the use of a



(a) *Training and validation loss* (b) *Test plot*

Figure 1: First model result

learning rate that is too high can cause the model to converge too quickly to a suboptimal solution or leading to instability, whereas a learning rate that is too small can cause the process to get stuck into suboptimal points, from the study perform with *Optuna* the best results are obtained when the parameter is inside $[1e-5, 1e-1]$.

Early stopping The first regularization method used is the *early stopping* which is a method to avoid overtraining, essentially the training is stopped when no improvement is observed for n consecutive epochs. To observe this in the study performed a very small patience term was used because of the very short training performed which is of 100 epochs, obviously this is not totally correct, however it is done to observe the effect of this method. From what obtained, it can be observed that the methods stops the training, whereas there is no more improvement.

Momentum Another very interesting factor, especially when using gradient descent and derivatives as optimizers, is the *momentum* which is a parameter that helps them get out of local minima points. There is one main drawback, that is that too much of momentum may create issues associated to stability, in the minimization process oscillations are crated and if they grow in magnitude they can make the process diverge. From the study performed, it seems that an approximate range from which to draw the value of the momentum is $[0.2, 0.8]$, higher values tend to deteriorate the results.

Momentum vs Learning rate A study was also carried out to observe the results by varying the momentum and the learning rate at the same time, these two parameters are strongly linked as both affect the speed of learning, however the composed effect of the two may not be good, as low values of the two could lead to difficulties in convergence, while high values could lead to strong instability. This is confirmed by the results, which find the optimal values within the ranges highlighted above.

Weight decay Another regularization method is the L2 regularization, implemented in a neural network through the *weight_decay* parameter. In a nutshell the idea came from the fact that large weights in a neural network are a sign of overfitting, this technic update the loss function that is optimized during training taking into account the magnitude of the weights, penalizing high weights. The results are remarkable in this case, the range of values for which there were good results is more or less $[1e-4, 1e-1]$.

Dropout Another technique used is the *dropout*, basically during training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. The consequent effect is that each update to a layer during training is performed with a different "view" of the configured layer. In this case you have to modify the network structure by adding dropout layers, this has been implemented in the class *Net_d()*, where two dropout layers have been added.

The range of optimal values for dropout probabilities seems to be $[0.2, 0.8]$.

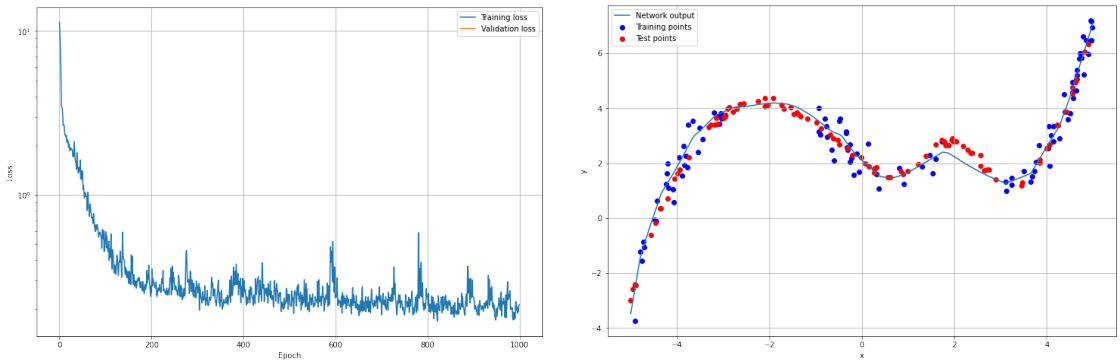
Optimizers A study has been done associated with optimizers, in particular is being performed a comparison between the following optimizers:

- **SGD**, which is a variant of the standard Gradient Descent algorithm that introduce noise in order to do not get stuck into suboptimal point
- **Momentum SGD**, which is a variation of the SGD that tries to tackle the high variance problem in the standard SGD
- **RMS prop**, which a popular optimizer used, it is an extension of GD.
- **Adam**, which is one of the most performing optimizers up to now, it is based on the first and second moments of the gradients

The analysis performed tries to find the best optimizer by providing the best intervals previously obtained associated with the various parameters, obviously the analysis should be done trying to make the best tuning associated with each optimizer. The optimizer that gives the best results is Momentum SDG, however the results are very similar even using the others.

3.1.2 Different network structures

In the following other networks have been tested, in particular the structures used have three and four hidden layers implemented in the classes *Net3* and *Net4* respectively, and the corresponding versions with dropout *Net3_d* and *Net4_d*. The results put the network *Net3* in the first place, this is consistent with the theory, since there are not too many data available, it follows that a simple network is often preferred w.r.t. more complex models. As regards this, a final test was performed by training the network on the entire training dataset. The results are shown in Fig.2 below, the final test loss value is 0.08 which is pretty good w.r.t. the first solution result in which it is about 0.24. It can be notice that now the model is able to generalize well. The image in Fig.2(b) give an intuition about the good result obtained.



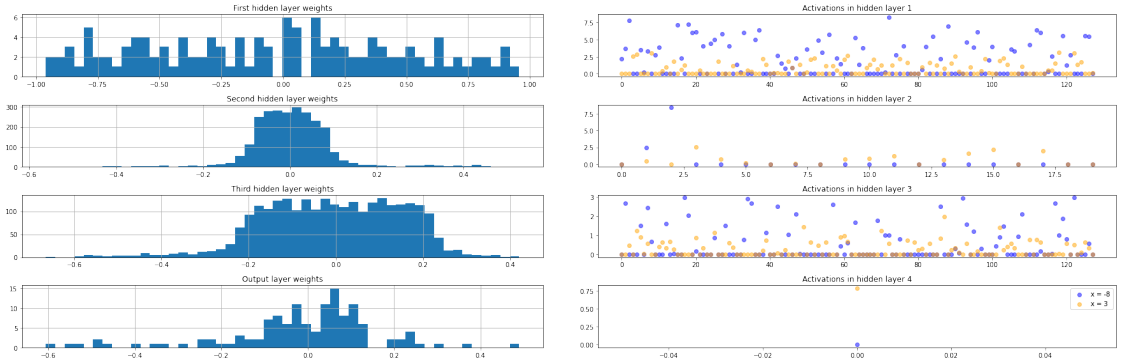
(a) *Final training loss* (b) *Test plot*

Figure 2: Best net results

Furthermore, the weights and activations plots were performed for this network, in particular in Fig.3(a) the weights are observed while in Fig.3(b) the activations using as test values $x_1 = -8$ and $x_2 = 3$.

3.1.3 Optuna optimization

As last thing, an automatic tuning was performed with *Optuna* trying to obtain a better network, however, although the results are good, it does not reach the levels of the previous network, in this case a two-layer network with dropout is used in which also the number of neurons of the layers is tested to try to obtain the best result as possible.



(a) *Weights* (b) *Activations*

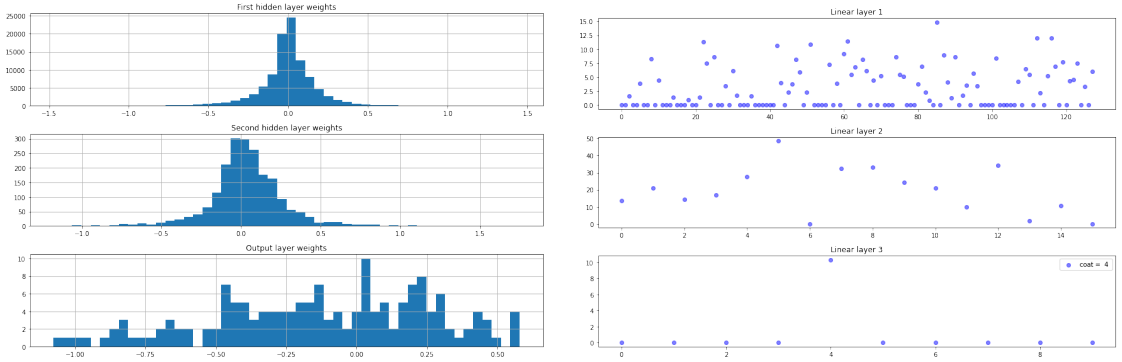
Figure 3: Network visualization

3.2 Classification

In this section are reported the results associated to the classification task.

3.2.1 Fully connected

First, start by analyzing the results obtained with the fully connected network. In Fig.6(a) are shown the results obtained through the learning process, in particular it is shown the confusion matrix associated to this neural network obtained over the test dataset. The accuracy associated to the model is about 85% but tuning a little the parameters the accuracy can be increase to get results similar to the best network. However, the fact that the accuracy is lower in this case is plausible since the information that a fully connected network is able to extrapolate from an image is relatively small compared to a convolutional one that will be used later. To have a visualization of the results, a simple test has been performed, and it is observed that the network is able to classify the object in question with high probability.



(a) *Weights* (b) *Activations*

Figure 4: Network visualization

In Fig.4 are shown respectively the weights associated to the fully connected network and the activations corresponding to a coat image, it is interesting to observe that the neuron associated to the 'coat' fires in this case as desired.

Activation maximization Activation Maximization is a method to visualize how a neural network view something, it aims to maximize the activation of a certain neuron. During the 'standard' training, one would iteratively tune the weights and biases of the network such that the error, or loss, of the neural network is minimized across training examples in the data set. On the contrary, activation maximization flips this around: after the classifier has been trained, we want to iteratively

find the parts of the data that the model thinks belongs to a given class. In a nutshell, activation maximization find inputs that return an output with the highest confidence. The implementation

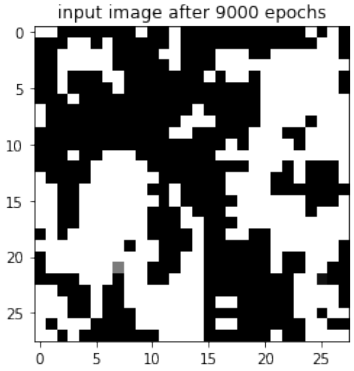


Figure 5: Activation maximization result

works as follows:

1. First it is needed to freeze the weights associated to the trained model, otherwise they will change during the activation maximization process
2. In order to deal with the activations is used the hook function (in the code this is done for the output layer)
3. The function *random_image* is used to build a starting image which is basically composed of noise, which compose the starting point of the algorithm.
4. The main algorithm optimizes the network input instead of the network parameters, which are frozen. Actually the code records the activation associated to the last layer and maximize the activation which is used as the error term in a 'standard' training.

The result is shown in Fig.5, in particular this is the image that maximize the activation associated to the 5th output neuron.

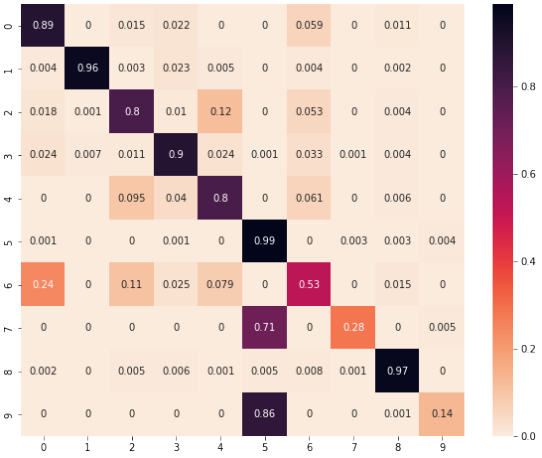
3.2.2 Convolutional network

As said in Sec.2.2 the second approach used to solve the classification task regards a convolutional network. In this case the accuracy reached after the training is about 89%, so a little higher w.r.t. the fully connected one. The associated confusion matrix can be observed in Fig.6(b). Also in this case the weights associated to the network have been plotted, in this case each convolutional layer is composed by different kernels which are plotted as images, and the linear layers are plotted as histograms, those plots can be observed in Sec.A of the Appendix. Similarly, the activations have been plotted starting from an initial image, in particular the image of coat, as before, also those can be founded in the Sec.A of the Appendix, such activations can be observed starting from the first convolutional layer to the output one, where it can be observed that the neuron associated to the coat class fires also in this case. Activation maximization has been performed also in this case, the results are not reported since they are similar to the one presented for the fully connected case.

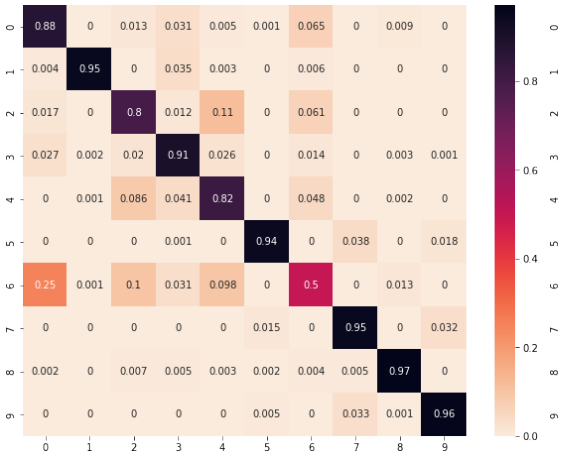
3.2.3 Optimization and best results

In order to improve the results a little, it has been tried to add some regularization in particular the dropout is used, two dropout layers are added in the fully connected part of the convolutional network. The results in terms of loss seems to be better, in fact it goes from an average loss of about 1.3 to 0.3 by comparing it with the previous networks. In order to try to push the performance to the maximum, the network has been subjected to tuning with Optuna in particular, almost all the parameters that make it up are changed. Being a convolutional network, the structure generated

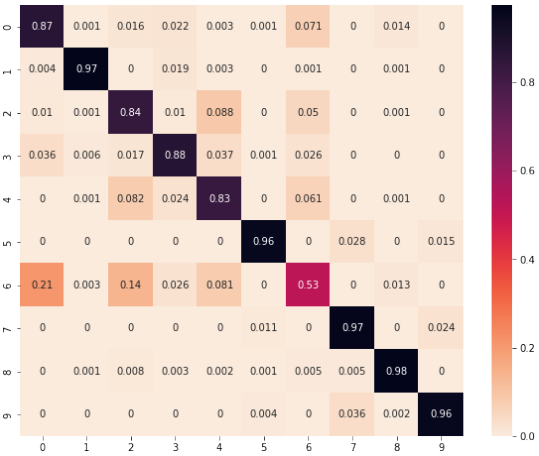
randomly by Optuna may not be valid, many trials have been pruned. In conclusion, an optimal network has been obtained which reaches an accuracy of 88%. The associated confusion matrix can be observed in 6(c)



(a) *Fully connected*



(b) *Convolutional*

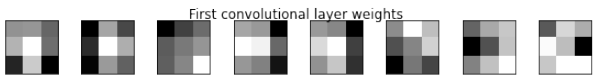


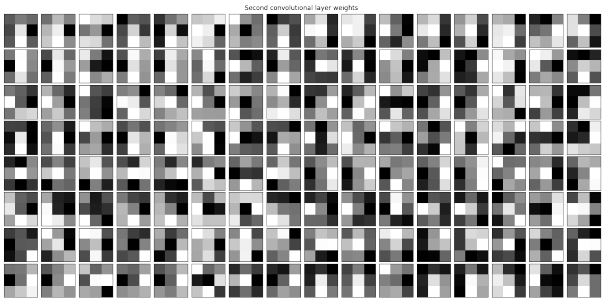
(c) *Best net*

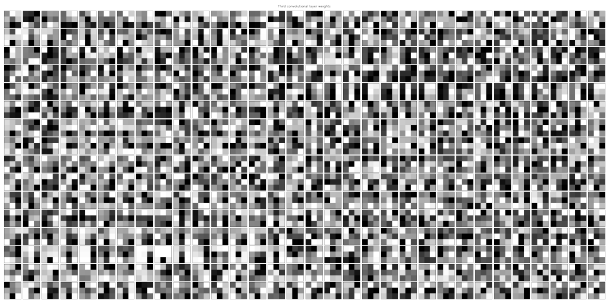
Figure 6: Confusion matrix

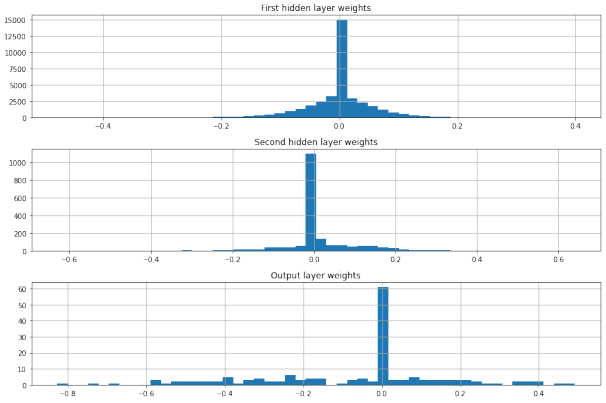
A Convolutional network

The weights associated to the convolutional network are shown below.









The activations associated to the convolutional network are shown below.