Neural Networks and Deep Learning - Homework #1

Chioccarello Stefano - 2011656
stefano.chioccarello.1@studenti.unipd.it

November 16, 2021

This report describes the first homework of the Neural Networks and Deep Learning course. The homework is divided in two parts which are explained in detail in the following.

1 Regression Task

1.1 Introduction

In the first part of the homework, a regression task has been assigned. The goal is to train a neural network to approximate an unknown function:

$$f : \mathbb{R} \to \mathbb{R}$$
$$x \mapsto y = f(x)$$
$$\text{network}(x) \approx f(x)$$

For this task, a multi-layer neural network has been employed in order to approximate the behaviour of the given function starting from the labelled data points.
In addition to that, a random search technique with a K-Fold cross-validation method was exploited, in order to find the best hyperparameters for the network, as well as the application of optimizers and regularization methods to improve convergence of the network.

1.2 Method

The data set consists of 100 data points for both the training set and the test set. The data points belong to the interval $[-5, 5]$.
The neural network used for this regression task consists of two hidden layers with $Nh1 = [20, 50, 100]$ for the first one and $Nh2 = [100, 200, 300]$ for the second one, the output layer, in addition to a dropout layer applied at the end of each hidden layer with probability sampled from a uniform random variable between $[0, 0.25]$. The initial setting for the

number of neurons was of 50 for the first layer and of 100 for the second one. To begin with, the training of the network was performed for 250 epochs. As far as the loss function is concerned, the one used for this task is the Mean Square Error, as it fits best the regression task since it measures the L2 norm between the outputs of the network and the targets. The activation function chosen is the ReLU function, in order to prevent the vanishing gradient problem, which occurs less with respect to other activation functions such as the Sigmoid or the Tanh.

Moreover, in order to further improve the convergence of the gradient descent and promote generalization of the network, as well as avoid overfitting, the use of regularization and optimization techniques was explored, namely:

- Stochastic Gradient Descent with momentum = 0.9 and learning rate sampled from a uniform distribution between $[1e-2, 1e-4]$ and weight decay sampled between $[1e-3, 1e-5]$
- Adam optimizer with same learning rate choice and weight decay as before

The hyperparameters selected with the random search technique included also the batch size of the training data loader $[4, 10, 16, 32]$, as well as the number of epochs of training $[100, 250, 500]$. A total of 500 randomly generated networks were tested with a 5-fold cross validation, from which the best model was found by picking the lowest average validation loss. Finally, histograms and activation fields were plotted and analyzed.

1.3 Results

In the following are shown the main results coming from the regression task. The best model given by the random search was the one with the following parameters:

Num epochs	batch size	lr	dropout	weight decay	optimizer	Nh1	Nh2
250	4	0.00035	0.1638	0.00025	SGD	100	100

Table 1: Best parameters from the random search

Such parameters gave an average training loss of 0.38 and test loss of 0.17. Here it is reported the plot of the approximated function based on the predictions of the network, together with the training and the test points. As one can observe (Figure 1), the network is able to fit very well the test points in the intervals $[-5, -3]$, $[-1, 2]$, $[3, 5]$, while in the others, namely where the training points are missing, it is less accurate.

However, in general it can be said that the regression task is well achieved with this kind of framework. To conclude, the histograms and weight profiles were plotted (Figure 2, 3, 4).

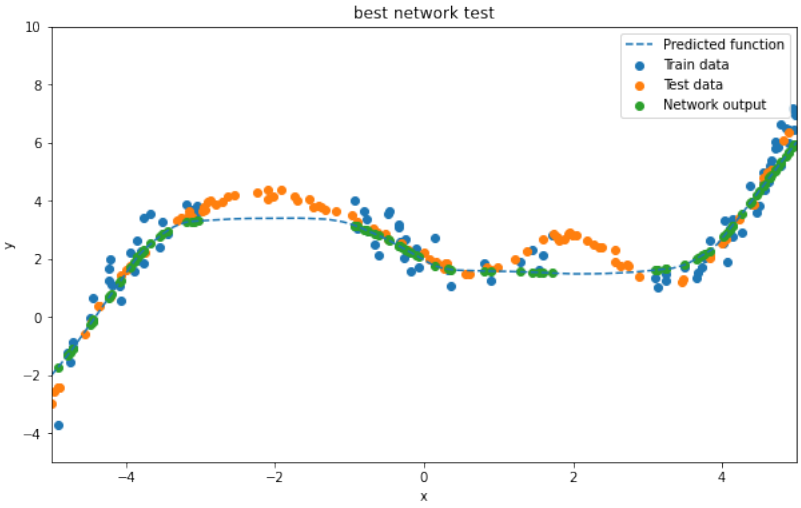

Figure 1: Network output with the best parameters

It can be observed that in the second hidden layer the weights are peaked around zero due to the effect of regularization. However the same does not apply for the output layer, which means that in such layer the weight decay has not played a significant role in determining the network output. In conclusion, the activation profiles are reported below.

2 Classification Task

2.1 Introduction

For the second part of the homework, the goal was to train a network to perform classification on images taken from *FashionMNIST* dataset, which is a collection of Zalando's article images. The input data are black and white images and they are divided into 10 classes of articles.
In this case, a simple multi-layer linear model was initially employed for the classification of the images, and then more complex structures such as convolutional architectures were added to the network in order to increase the performances. Moreover, also in this second part the use of optimizers and regularizers was explored, in addition to a hyperparameter-tuning section with random search as well. In conclusion to that, the exploration of the activation profiles of the convolutional layers was carried out.

2.2 Method

As far as the dataset is concerned, the Fashion MNIST dataset is a gathering of Zalando's article images. Each image is 28×28 pixels, and there is a single channel only with 10 classes

(t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, boot). The dataset is already normalized between 0 and 1, hence the only necessary transformation to apply to it is to map the images into tensors. (More on transforms...)
The basic classifier consists of three hidden layers. The number of input units is 784 (=28x28), while the hidden neurons are respectively 256, 128 and 64, then finishing with 10 units for the output layer, corresponding to the 10 classes of the articles. After each hidden layer, a dropout with probability uniformly sampled between $[0, 0.25]$ was added. This basic classifier was trained with the following parameters:

Batch size	**Epochs**	**Optimizer**	**Learning Rate**	**Activation Function**
64	20	SGD	$1e-2$	ReLU

Subsequently, a more advanced network was employed, all together with more efficient optimizers and regularizers. In particular, a Convolutional Neural Network (CNN) was defined in the following way:

- Two Sequential layers, each one consisting of the following layers:
  - 3x3 convolutional layer with padding= 1, stride= 1
  - Batch Normalization layer
  - Activation function (ReLU)
  - 2x2 Max Pooling layer, with stride = 2
- Dropout layer with probability p= 0.25
- 2 fully connected layers to reduce the number of outputs
- Output layer with 10 units, one for each class.

After that, the exploration of the more advanced library *Pytorch-lightning* in combination with *Optuna* was performed, in order to do some hyperparamer tuning in a different way with respect to the regression task. The CNN was reported into a *Pytorch-Lightning* class, and with it the tuning of the optimizers was performed. The tuning of the hyperparameters with the convolutional network was very highly demanding from a computational point of view, hence a reduced version of the optimization by means of *Optuna* was implemented. Only the optimizer and its variables were taken in consideration, namely:

- *SGD* or *Adam* optimizer
- learning rate uniformly sampled from $[1e-5, 1e-3]$
- weight decay uniformly sampled from $[1e-6, 1e-3]$
- momentum uniformly sampled from $[0.6, 1]$

2.3 Results

In order to measure the performances of the classifier, the network was tested on the test set, and for each class the accuracy in percentage is reported. With the basic classifier with linear layers only the results shown in Table 2 were achieved, with a validation loss of 0.4317. This model achieved an average accuracy of 88% of classification.

T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
83.10%	96.50%	69.80%	83.90%	84.60%	93.60%	75.70%	97.50%	97.00%	94.30%

Table 2: Accuracy with basic classifier

On the other hand, with the deeper convolutional architecture described before, the performances on the test set were the one shown in Table 3, with an average accuracy of 91.1%.

T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot
76.70%	97.90%	85.40%	91.40%	85.00%	98.60%	78.20%	92.70%	98.30%	98.70%

Table 3: Accuracy with CNN classifier

The validation loss in this case was of 0.1265.

As one can observe, the implementation of the convolutional layers has improved the performances of the classifier. One problem that however arises in both architectures is that the objects that are more frequently wrongly classified are the one belonging to the "T-shirt" and "Shirt" classes. This is understandible since they are two similar objects, and the CNN allows to get better results in both of them.

From the optimization part with *Optuna*, the following parameters were chosen as the best ones:

- Optimizer: *Adam*
- learning rate = 0.0040
- weight decay = $4.33e-5$
- momentum = = 0.6037

With such configuration, the loss computed on the test set was of 0.3612

The number of trials was reduced to 5 for computational purposes, however with more power at disposal one can explore more trials and find better configurations of the parameters.

Finally, also the receptive fields of the convolutional layers are reported in Figures 5 and 4 for the first and second layer respectively. It can be observed how the first layer is responsible of detecting the main part of the image related to the object to be classified, while in the second layer the activations are related to the singular features, like lines or edges or single points. The activation profiles in the second layer are also downsized with respect to the first one, due to the *Max Pooling* layer. Finally, in Figures 7 and 8 one can observe the filters of the convolutional layers.

3 Appendix

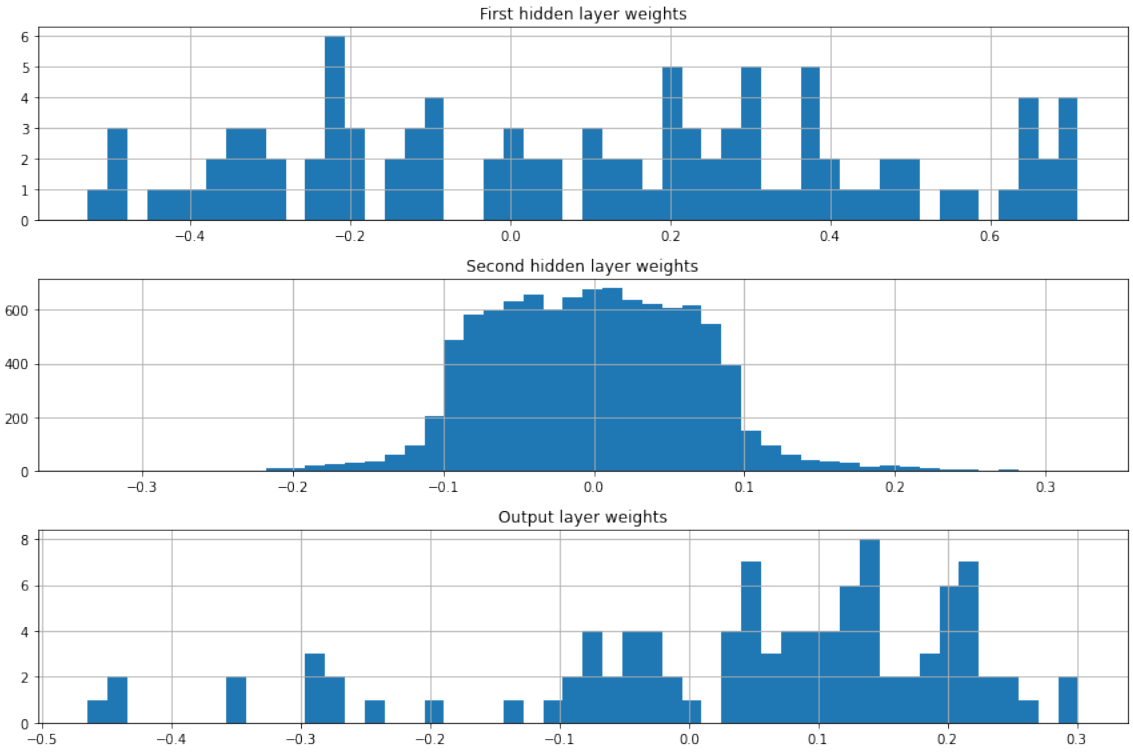

Figure 2: Network weight histogram

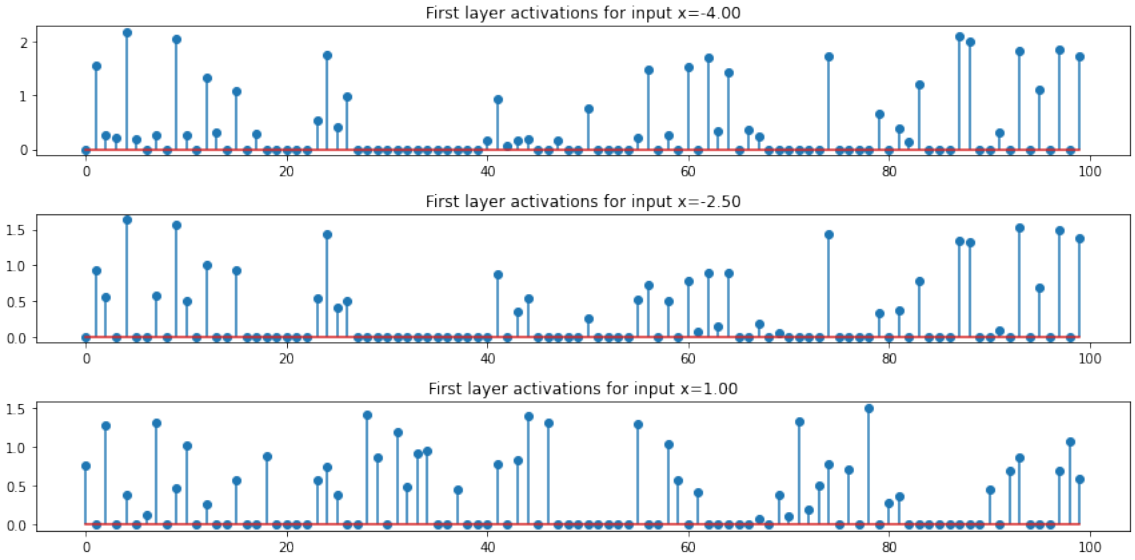

Figure 3: Activation in the first layer

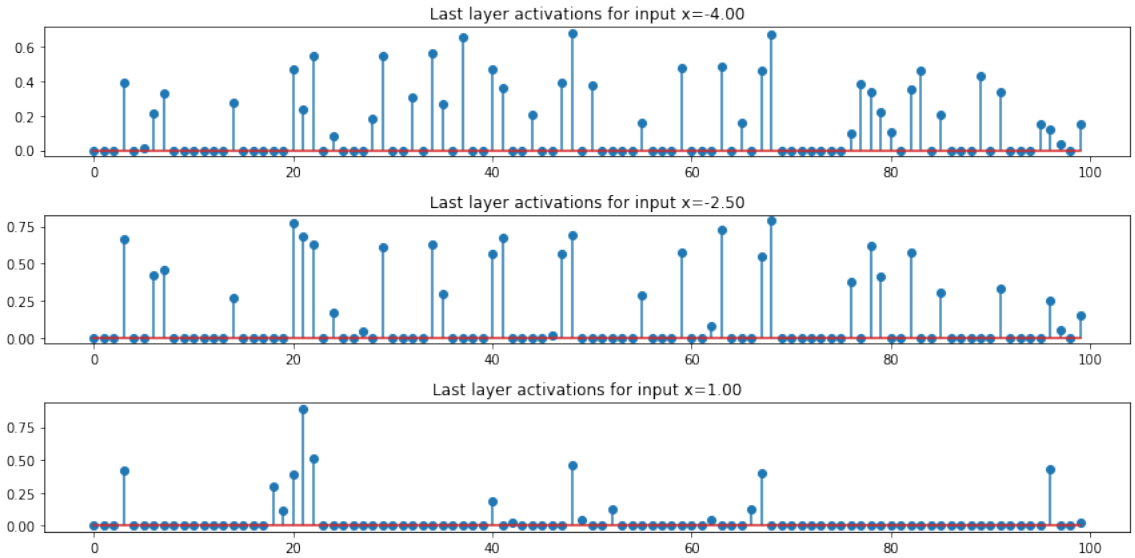

Figure 4: Activation in the second layer

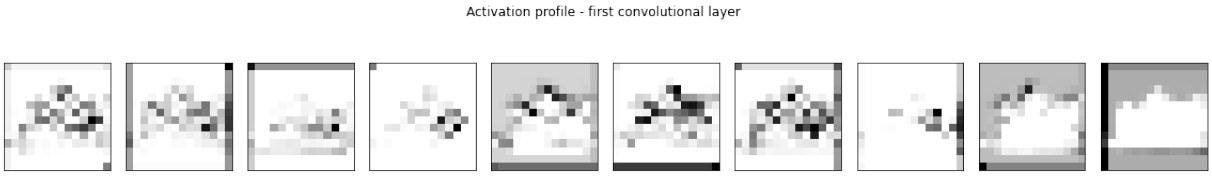

Figure 5: Activation in the first convolutional layer

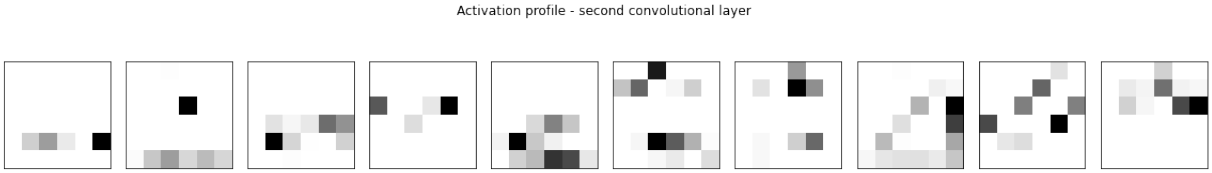

Figure 6: Activation in the second convolutional layer

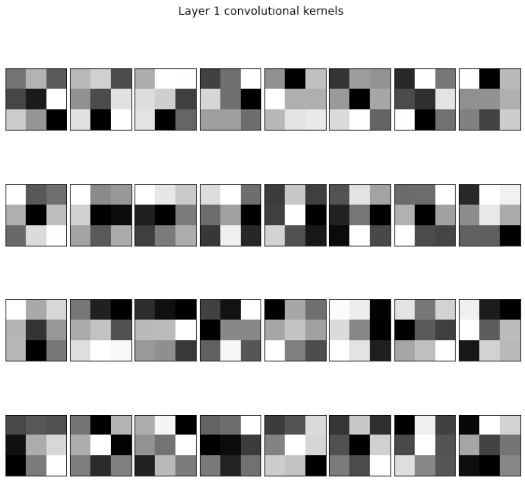

Figure 7: Kernels of the first layer

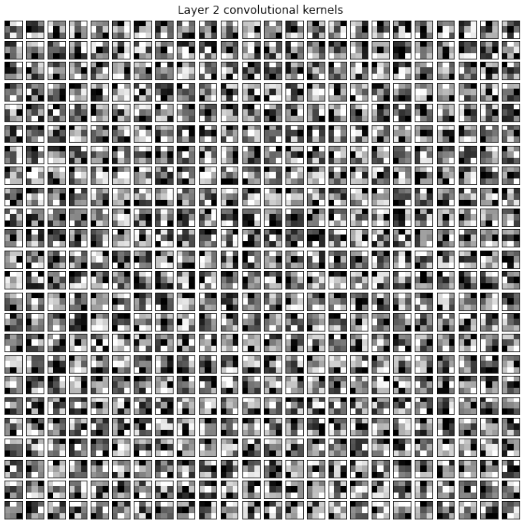

Figure 8: Kernels of the second layer