Application of Supervised Deep Learning techniques

Delfina Iriarte (1231682)

[1]University of Padova, Physics of Data

Abstract. *In this work, supervised learning techniques such as regression and classification are performed in* `Pytorch`*. In particular, regression was study on a given data set whereas classification was performed on the Digit* MNIST *Dataset. Hyper parameter optimization was applied for both method as well as cross validation. Moreover, weights histograms, feature filters and the activation profiles of the layers were studied.*

1. Introduction

In supervised learning, both a set of inputs and the corresponding desired outputs are imposed during training, so the network is essentially given the answer. The main tasks consider within the context of supervised learning are classification of inputs into two categories and function approximation (or regression), in which the output of a network unit was trained to approximate a specified function of the input.

In this work, regression on a given data set was performed in PYTROCH which was a library available in PYTHON. Since the dataset was small, cross validation was implemented. Furthermore, hyper parameter optimization was also applied. On the other hand, classification on the Digit MNIST Dataset was performed. As in the regression task, fine tuning of the hyper-parameter was performed. Finally, for both cases the weights histograms, feature filters and the activation profiles of the layers are studied.

2. Method

In this section, the model implementation of the regression and the classification task are presented.

2.1. Regression Model

The task of the model was to predict a polynomial curve given. The network architecture done for the regression task was simply composed by three fully connected layers: an input layer of one neuron and output NH1, a hidden layer of input NH1 and output NH2 and a output layer of input NH1 and output of one neuron. Furthermore, the rectified linear activation function (Relu) was used in order to overcome vanishing gradient problem, allowing models to learn faster and perform better in comparison of the Sigmoid for example. However, the main drawback of this activation function is that it might lead to the `Dying ReLU problem` and this is why studying the weights activations is indeed important.

Additionally, regularization methods to avoid overfitting were used. In particular, Ridge regression (known as well as L_2) regularization methods were used by using the `weight_decay` flag provided by PYTORCH, which adds *squared magnitude* of coefficient as penalty term. Moreover, randomly units were eliminate during the training by using DROPOUT with probability `drop`.

Since the data available was limited, dividing the dataset into Train and Validation sets may cause some data points with useful information to be excluded from the training procedure, therefore CROSS VALIDATION techniques were applied by using the SKLEARN function called `GridSearchCV`. This approach ensure also better performances, since the validation set was different for each of the iterations, and so we cannot end up with a model that perfectly fits one given validation set. In particular the number of folds set were`k_folds = 4`. In order to tune the number of epochs, **Early stopping** was also implemented by using the library `skorch`. This techniques consist on tracking the validation loss; if it does not diminish sufficiently in a given number of epochs then the training loop is break.

Two different optimizer were used: the stochastic gradient descent and the Adam algorithm in which the last is a **second order optimization method**. As already mentioned, hyperparameter optimization were performed by implementing the `GridSearchCV` function. The parameters consider were the following ones:

1. Number of neurons `Nh1`:$[20, 40, 50, 75]$
2. Number of neurons `Nh2`:$[20, 40, 50, 75]$
3. Dropout `drop`:$[0.001, 0.01, 0.1, 0.5]$
4. Learning Rate `lr`:$[1e-4, 1e-2]$
5. Weight Decay `weight_decay`:$[1e-3, 1e-4, 1e-5]$
6. Optimizer `optimizer`:$[optim.SGD, optim.Adam]$

Afterwards, the model was trained with the best hyper-parameters. Finally, weights histograms and the activation profiles of the layers were studied, which is a easy task done with PYTORCH, and as already mentioned, are important to understand if we are having problems with the network.

2.2. Classification Model

The classification task was performed on the MNIST dataset which contains $60,000$ small square 28×28 pixel images of handwritten single digits between 0 and 9. The data is divided into train and test by using the `train_test_split` provided by SKLEARN with a test size given by default.

The task was to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively. The data was normalized at the beginning in order to improve convergence of the SGD by simply re-scaling it.

The architecture model of the classification task consist on two convolution layers that uses filters. Figure 1 presents the model architecture done for the classification task for the optimal hyper-parameters.

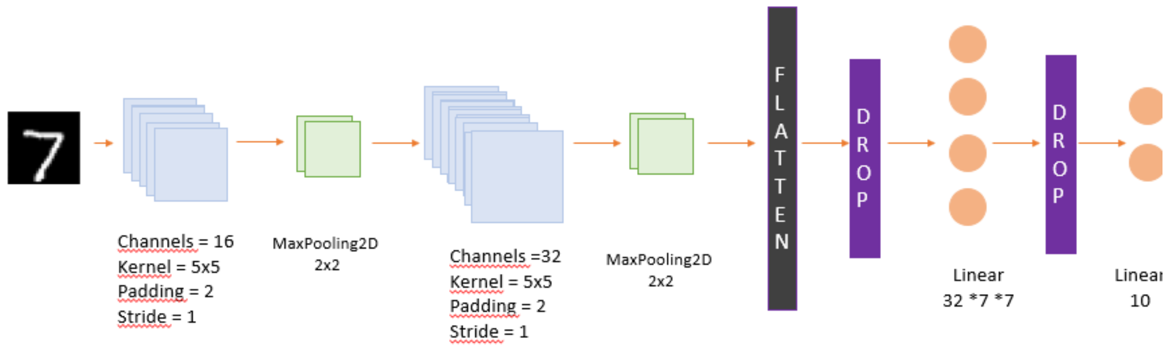


Figure 1: Best hyperparameters architecture

Regardless of the size of the data, which was quite exhaustive, a Cross validation technique was applied in the same manner as explained in Section 2.1 but with a `k_fold = 3`. Similarly, `Early stopping` and hyperparameter optimization was also implemented:

1. Number of neurons `Nh1`:$[8, 16, 32, 64]$
2. Number of neurons `Nh2`:$[8, 16, 32, 64]$
3. Dropout 1`dropout_rate`:$[0.01, 0.1, 0.5]$
4. Dropout 2`dropout_rate2`:$[0.01, 0.1, 0.5]$

5. Learning Rate `lr`:$[0.001, 0.0001, 0.00001]$
6. Weight Decay `weight_decay`:$[1e-4, 1e-3, 1e-2]$
7. Optimizer `optimizer`:$[optim.SGD, optim.Adam]$

As in the regression task, fine tuning of the hyper-parameter was performed, and the weights histograms, feature filters and the activation profiles of the layers are studied. The loss function used was Cross entropy which is a common choice for multi-classification tasks.

3. Results

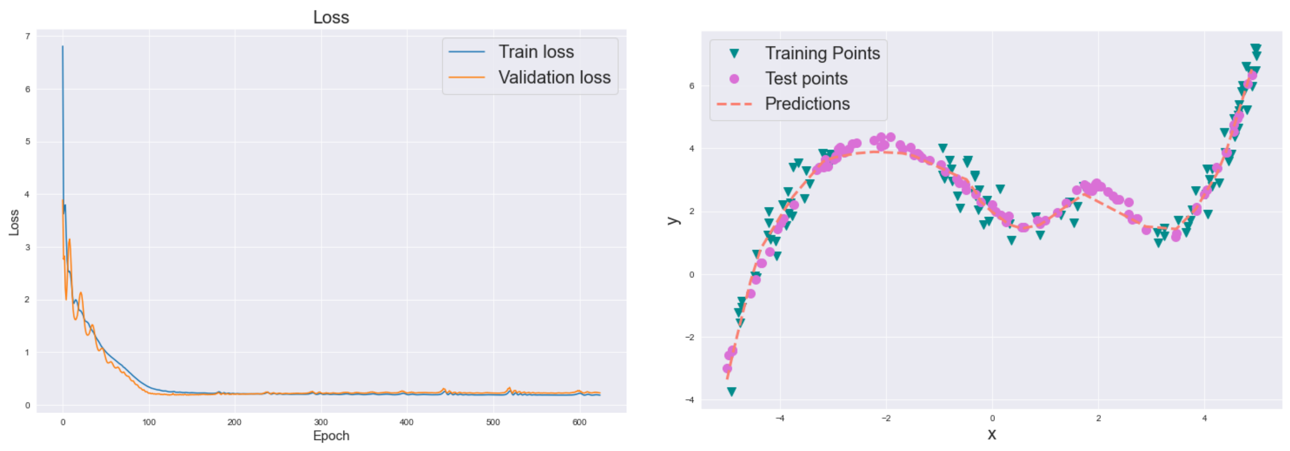
In this section, the results obtained for each supervised techniques are presented and discuss.

3.1. Regression

The best network architecture obtained correspond to the following hyper-parameters:

1. Number of neurons `Nh1`:50
2. Number of neurons `Nh2`:75
3. Dropout `drop`:$[0.1]$
4. Learning Rate `lr`:$1e-2$
5. Weight Decay `weight_decay`:$1e-3$
6. Optimizer `optimizer`:$optim.Adam$ which was expected since generally it is better than SGD.

Figure 2a presents the evolution of the training and validation losses per epoch. After 200 epochs, it can be observed that the model was not able to learn more. Recall that the total number of training was 3000 but due to early stopping that was set to start after 600 epochs, it can be observed that the training stop. Figure 2b presents the final prediction of the model. First of all, it should be point out that the splitting of the test and train dataset was not optimal since around $x = -2$ and $x = 2$ there are missing points in which the model is not able to reproduce correctly. Nevertheless, it can be observed that the model was able to capture most of the range consider.



(a) Training and Validation Loss (b) Model prediction

Figure 2: Results obtained for the Regression task.

Figure 5a presents the weights histograms of the layers used in the model. It can be observed that there are not any vanishing or exploding weights (which is a typical problem of the Relu activation function as already mentioned) and therefore it can be concluded that there are lying in an acceptable range. The activation profiles of the layers are shown in Figure 5b and it can be seen that all neurons are being used which is expected since we are not observing any strange behaviour of the weights as well.

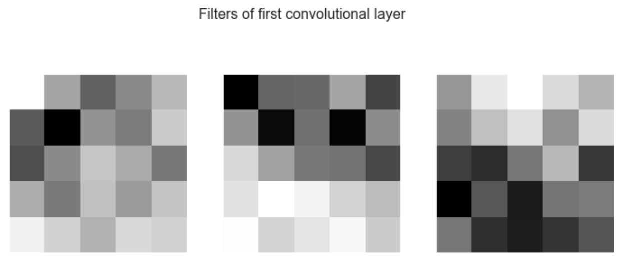
3.2. Classification

Figure 1 presents the best hyperparameters found for the model leading to a `Train Loss : 0.038`, `Val Loss : 0.04` and `Test Loss : 0.012`. In particular the optimal hyper-parameter obtained were $Nh1$: 16, $Nh2$: 64, $dropout\_rate$: 0.5,$dropout\_rate2$: 0.1, $optimizer$: $optim.adam.Adam$, lr : 0.001, $weight\_decay$: 0.0001.

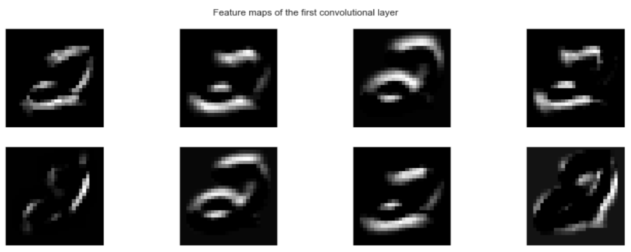
The evolution of the validation and the training losses are shown in Figure 6a. The number of epochs are low since as already mentioned `early stopping` was applied. For the train loss it was obtained 0.038, for the validation 0.04 and for the test loss 0.01. Therefore, the model while training was not overfitting the data. Figure 6b presents the comparison between the output of the predicted one and the exact one as a Confusion matrix. It can be seen that indeed the model was able to classify almost all of the samples correctly. In particular, it can be observed that the number five was mostly mistaken by the number three, and the number nine with seven. Nevertheless, the accuracy obtained for the model was indeed good.

Figure 5a and Figure 7b presents the weights histogram and the activation profile of the layers. As before a reasonable behaviour of them can be observed.

The filters of the first convolutional layer and its activation profiles are shown in Figure 3. It was not so clear which are the features by observing the filter. However, when observing the activation profile of the first convolutional layer, a clear digit can be observed.



(a) 5 × 5 kernels of the filters of the first convolutional layer



(b) activation profile of the first convolutional layer.

Figure 3: Results obtained for the Classification task.

Finally, the activation profile of the second convolutional layer was shown in Figure 4 where a decomposition of the digits, such as the edges can be observed.

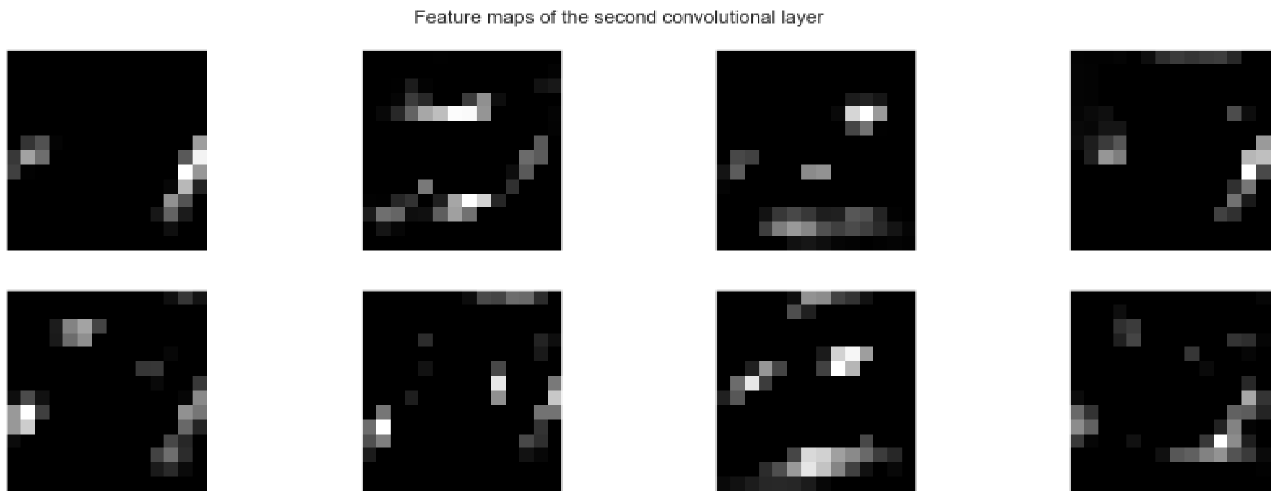
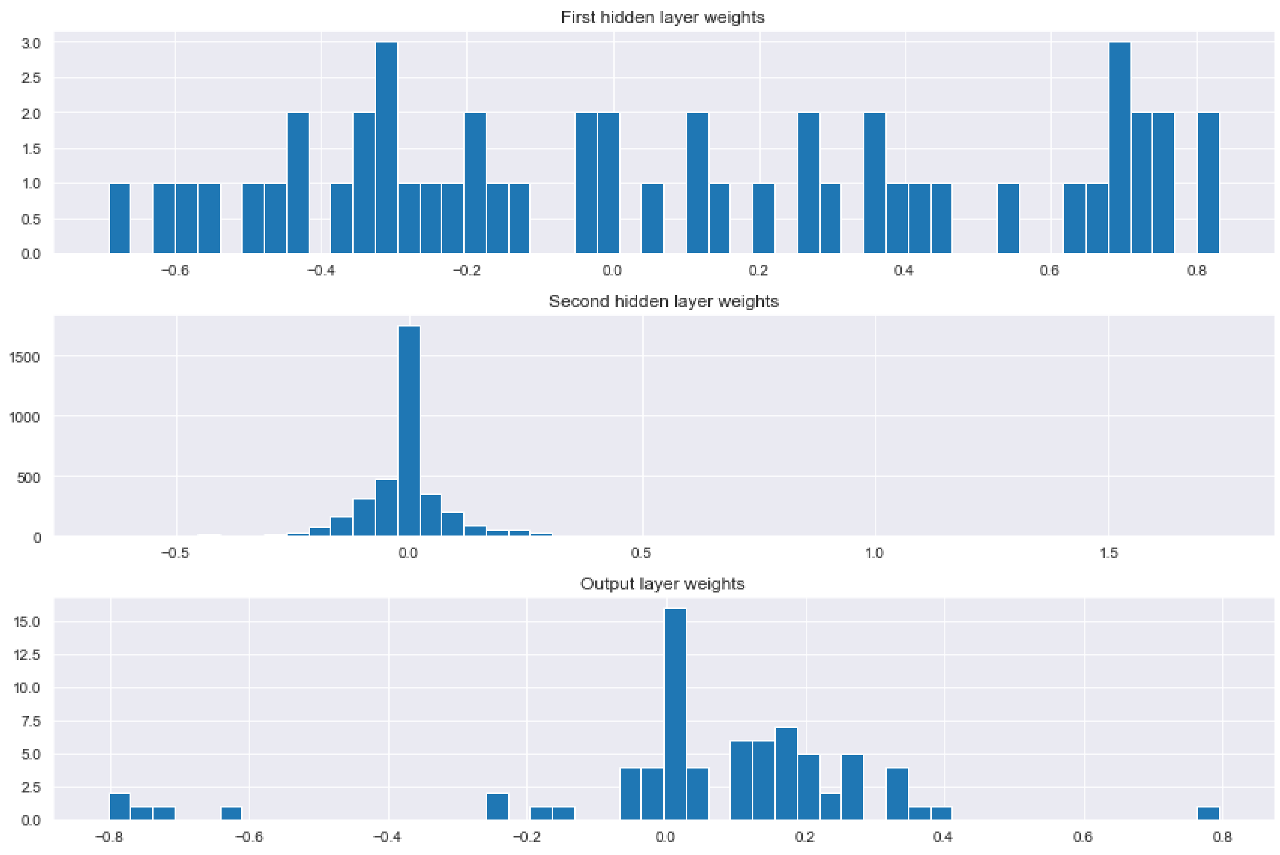


Figure 4: Activation of the second convolutional layer.

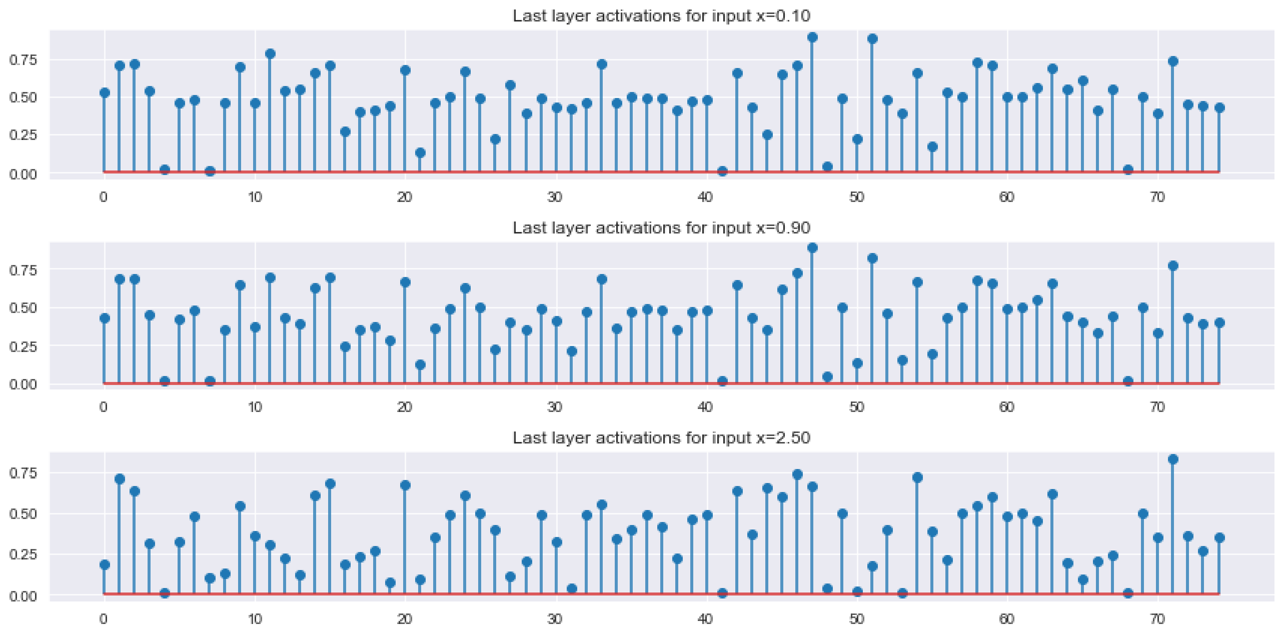
4. Conclusion

In conclusion, supervised learning techniques has been successfully apply in the dataset of our study. Even tough it was not perfect, the models are able to achieve have accuracy in both cases. In particular one of the main drawback of the regression model was the already Train-Test split given, since it doesn't allow to train the model correctly. Regarding the classification, some small minor mistaken are achieve between numbers.

5. Appendix

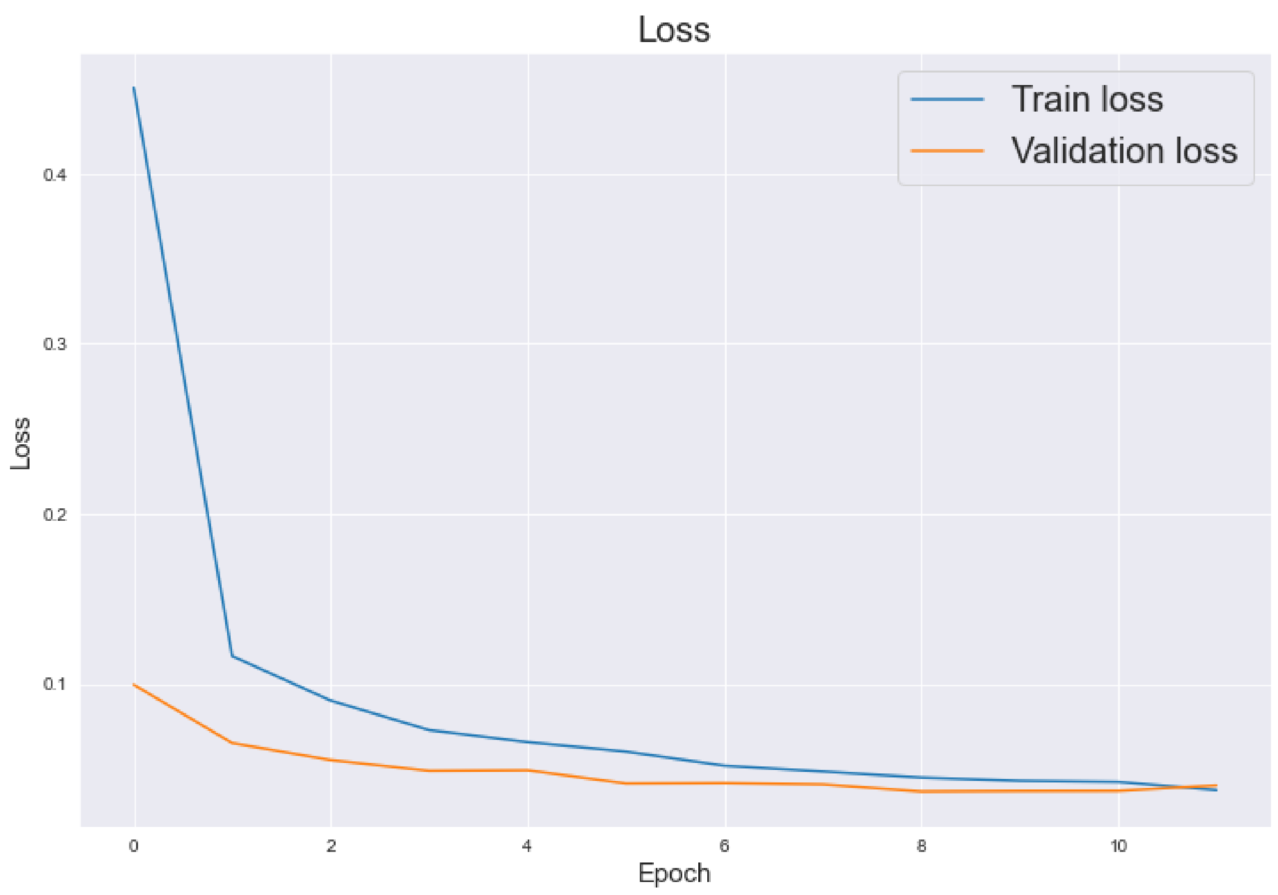


(a) Weight histogram for the layers of the network.

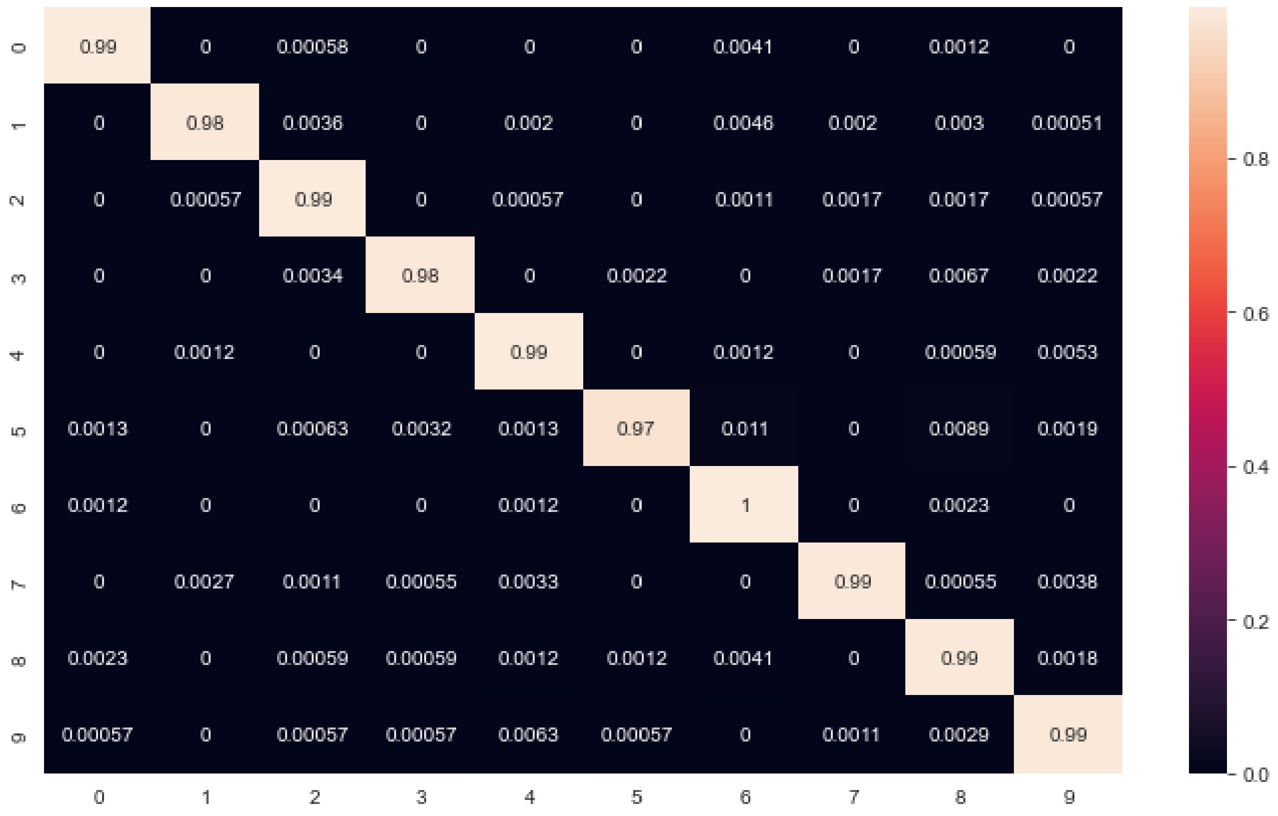


(b) Activation profile for the layers of the network.

Figure 5: Results obtained for the Regression task.



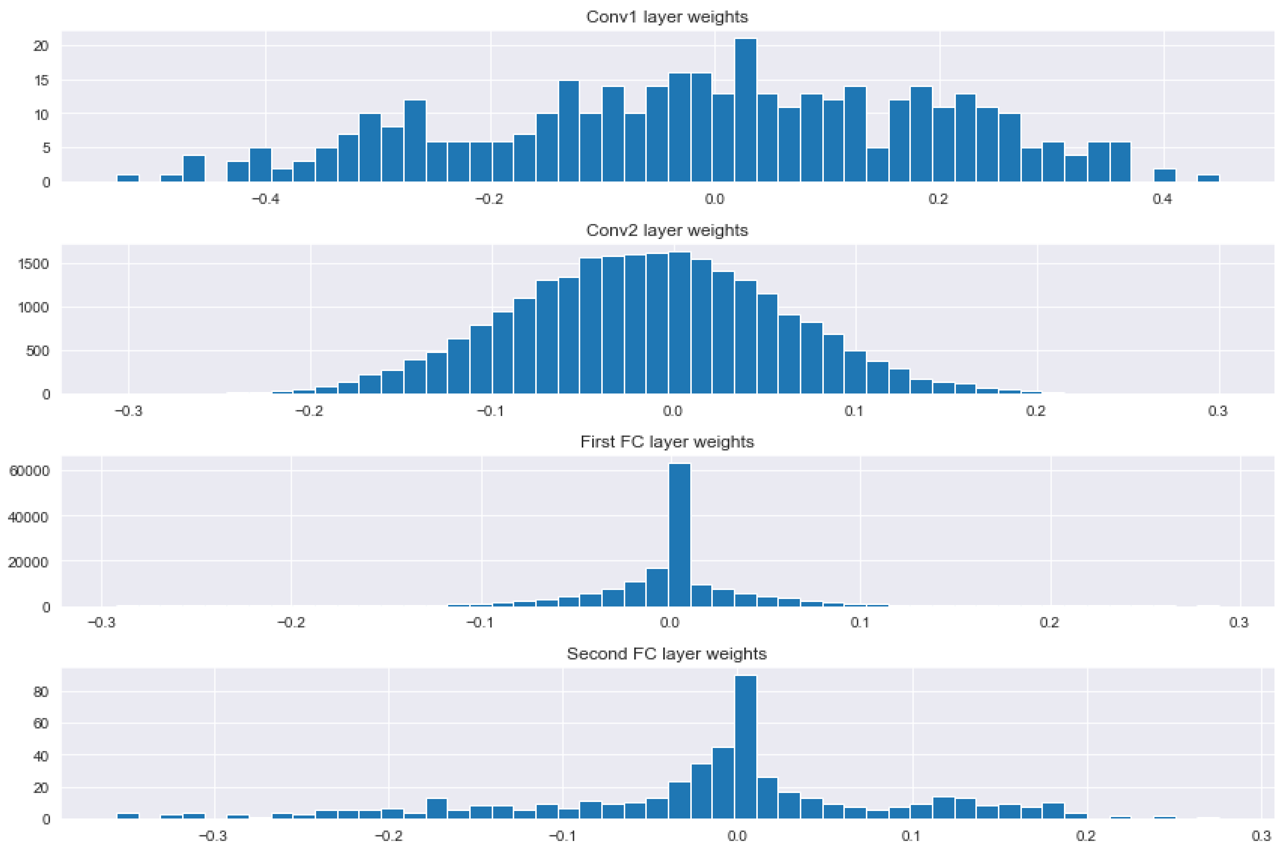
(a) Evolution of the training loss and the validation loss per epoch.



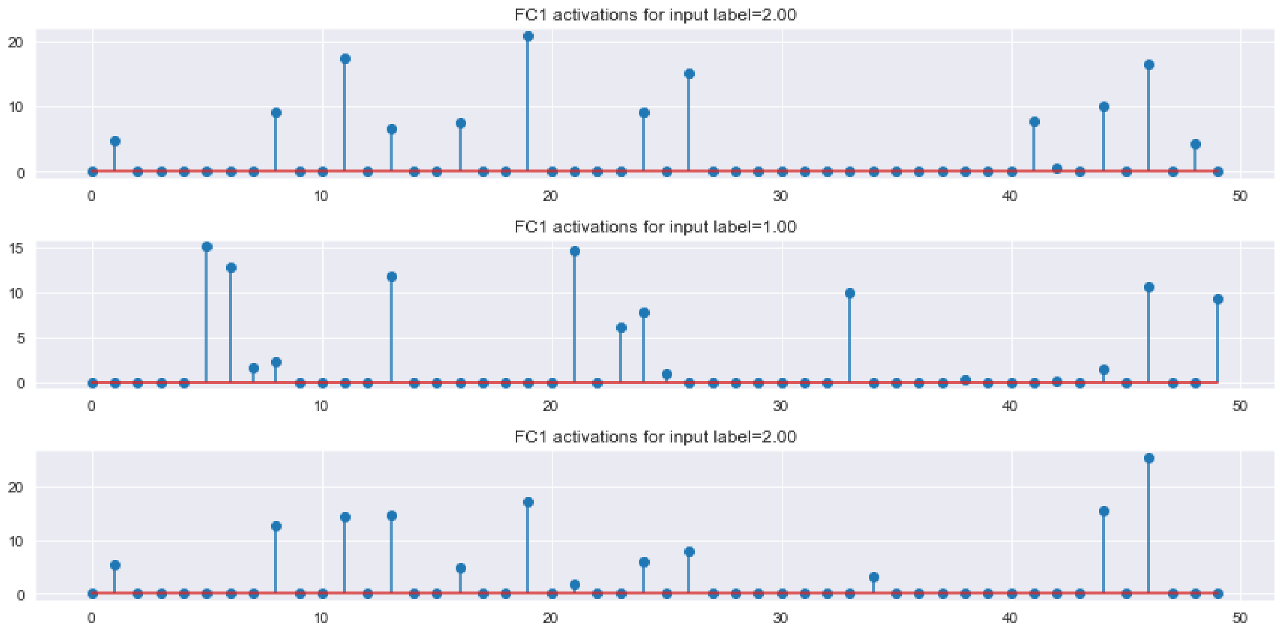
	0	1	2	3	4	5	6	7	8	9
0	0.99	0	0.00058	0	0	0	0.0041	0	0.0012	0
1	0	0.98	0.0036	0	0.002	0	0.0046	0.002	0.003	0.00051
2	0	0.00057	0.99	0	0.00057	0	0.0011	0.0017	0.0017	0.00057
3	0	0	0.0034	0.98	0	0.0022	0	0.0017	0.0067	0.0022
4	0	0.0012	0	0	0.99	0	0.0012	0	0.00059	0.0053
5	0.0013	0	0.00063	0.0032	0.0013	0.97	0.011	0	0.0089	0.0019
6	0.0012	0	0	0	0.0012	0	1	0	0.0023	0
7	0	0.0027	0.0011	0.00055	0.0033	0	0	0.99	0.00055	0.0038
8	0.0023	0	0.00059	0.00059	0.0012	0.0012	0.0041	0	0.99	0.0018
9	0.00057	0	0.00057	0.00057	0.0063	0.00057	0	0.0011	0.0029	0.99

(b) Confusion matrix with respect to the exact values and the predicted ones.

Figure 6: Results obtained for the Classification task.



(a) Weight histogram of the layers of the classification task.



(b) Activation profile of the layers of the classification task.

Figure 7: Results obtained for the Classification task.