Università degli Studi di Padova

Homework #1 Supervised Deep Learning

Student: *Beatrice Segalini* – 1234430

Course: *Neural Networks and Deep Learning* – Professor: *A. Testolin*

1. Introduction

In this homework, the implementation and testing of simple neural network models for solving supervised problems is displayed, specifically with the purpose of analysing two different tasks: regression and classification.

Firstly, a Fully Connected Network (FCN) with 2 hidden layers is implemented, with the aim of approximating the behaviour of an unknown scalar function $f : \mathbb{R} \longrightarrow \mathbb{R}$, $x \mapsto f(x)$, given as training points only some noisy measures from the target function: $\hat{y} = f(x) + noise$.

For the supervised classification task, a more deep and complex network is coded: in fact, in this case two convolutional layers will be applied, followed by two other fully connected ones. The Convolutional Neural Network (CNN) so developed is then applied and its hyperparameters tuned so that it can solve an image recognition problem, where the goal is to correctly classify images of handwritten digits (from the MNIST database), assigning to each of them the proper label (i.e. the correct digit they represent).

A random search (with a K-Fold cross validation method for the regression task) is performed to tune the main hyperparameters of both networks, also implementing more advanced optimiser and regularisation methods.

2. Regression

The training data for this task consists in 100 points, which intentionally do not cover the entire function domain: in fact, there are no samples belonging to the intervals $x \in [-3, -1]$ and $x \in [2, 3]$. In this way, it is possible to actually test how good are the generalising abilities of the final network.

2.1. Methods

For this task, the FCN general structure is defined by the `Net` Pytorch class, and has the following features:

- a linear layer with `Ninp` $= 1$ input unit and `Nhi1` $\in [20, 50, 100]$ output units;
- a dropout layer with dropout probability `drop` (sampled uniformly in the $[0, 0.25]$ interval);
- a linear layer with `Nhi1` $\in [20, 50, 100]$ input units and `Nhi2` $\in [100, 200, 300]$ output units;
- another dropout layer with the same dropout probability as the previous one `drop` ;
- an output layer with `Nhi2` $\in [100, 200, 300]$ input units and `Nout` $= 1$ output unit;

The user-defined class also includes the main learning steps, i.e. training and validation steps, training loop (`fitting`) and test procedure (`testing`). The ReLU function is chosen as activation function to prevent the "vanishing gradients" problem, namely the significant increase in the error back-propagation phase due to the magnitude of the gradient of the activation function. The ReLu function is less subject to this issue, given its linear behaviour with respect to other, more steep functions, such as the sigmoid one.

The loss function chosen is the Mean Squared Error (MSE), that measures the squared L2 norm between each element of the input and target tensors.

Moreover, to further improve the model and mainly to prevent overfitting, the following regularisation and optimisation techniques are studied and tested:

- Stochastic Gradient Descent (SGD) optimiser, with L2 regularisation parameter (weight decay) `reg_param` sampled from a log-uniform distribution in the interval $[10^{-5}, 10^{-2}]$, momentum parameter set to 0.9 and learning rate `learning_rate` sampled from a log-uniform distribution in the interval $[10^{-5}, 10^{-3}]$;
- Adam optimiser, with the same sampling methods for `learning_rate` and `reg_param` .

Finally, for the same purpose different batch sizes are considered (10, 16, 32 and 64) and also different maximum number of training epochs (100, 250 and 500).

500 different randomly generated networks are tested with a 5−fold cross-validation method defined in the function `cross_valid` . The best network is then picked selecting the one with the lowest average validation loss, to reward the model that arrives faster to convergence. This specific model is therefore trained and its fitting process is recorded and saved in a `.gif` file, to be finally tested with the "complete" test dataset (which fills also the gaps for $x \in [-3, -1]$ and $x \in [2, 3]$).

In conclusion, to get more insight about the network learning process and about each layer features, histograms of the weights and activation profiles will be reported and analysed.

2.2. Results

The best model to fit the data for the regression task is reported in Table (1).

`Nhi1`	`Nhi2`	`batch_size`	`dropout`	`num_epochs`	`optimizer`	`learning_rate`	`reg_param`
20	100	10	0.031	500	Adam	0.00986	0.00016

Table (1) – Best network parameters for the regression task, chosen by evaluating the average validation loss with a 5−fold validation process.

The network trained with this parameters leads to an average loss of the testing procedure equal to 0.14068002 and to the predicted function reported in Figure (1).

One can observe that the network does not generalise well the data tendency in the gaps of the training set. In fact, the model cannot predict the first local maximum of the function around $x = -2$, which appears as "cut" when looking at the blue dashed prediction line, and, despite behaving better for the second missing interval (around $x = 2.5$), it still cannot completely grasp the true data trend. Furthermore, one can notice that some fluctuating behaviour can be outlined around $x = -4$ and $x = 0$: this might be due to some form of over-fitting.

To understand better the learning process of this network, analysing the weight histograms and the activation profiles could help. The weight histograms reported in Figure (6) show that, due to the effect of weight decay regularisation, the second hidden layer and the output layer's weight histograms are peaked around zero. The activation profiles are displayed in (4), (5) for both the first hidden layer and the second one for inputs $x \in [-4, -2, 0, 2.5]$. It can be seen that, for the first hidden layer, only a few of the neurons are actually activated, in particular, for $x = -2$, which is in the middle of the first "gap", only 2 out of 20 are active. Differently it happens for the last hidden layer for which the activation profile for $x = -2$ is the one with more active neurons.

In conclusion, one could state that broadly speaking this network was able to infer the general tendency of the data, although not performing particularly well with an incomplete dataset. However, the results achieved are considered acceptable.

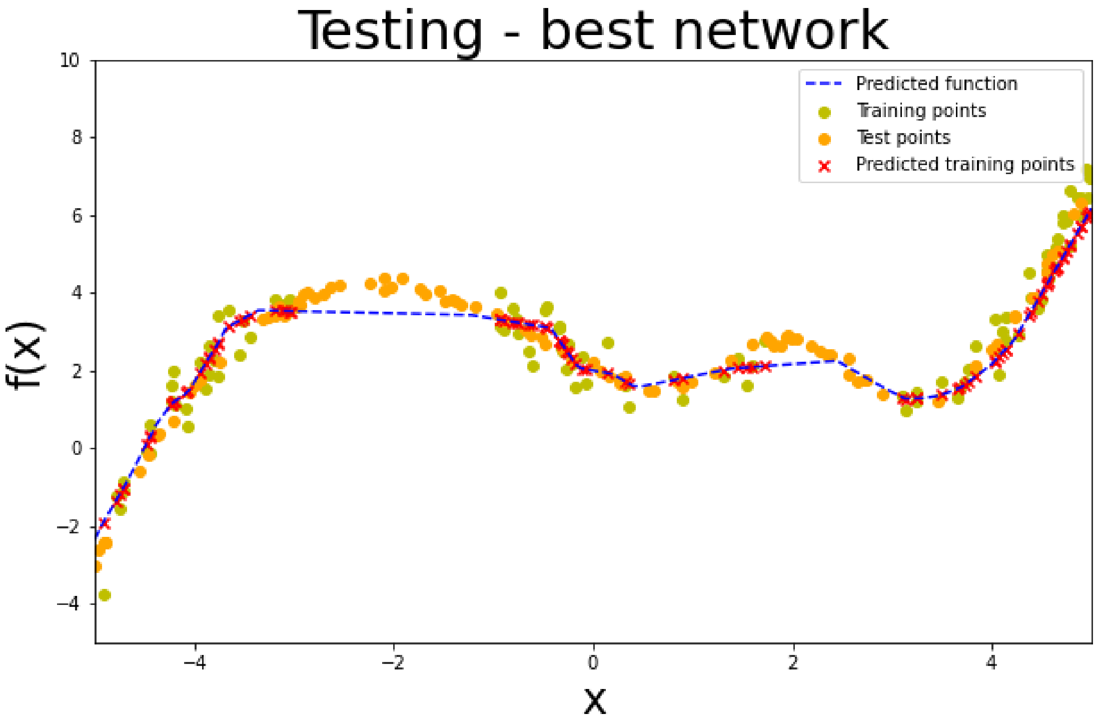

Figure (1) – Final results after training the network with the optimal parameters estimated after the random search. Both the training and test points are displayed, together with the predictions made by the network.

3. Classification

The goal of this section is to code a network able to perform a *supervised classification* task on the MNIST database of hand-written digits, and therefore to be able to divide into the 10 classes (digits from 0 to 9) the samples included in the dataset. The database is composed of 28 × 28 pixels images representing digits, labelled from 0 to 9. The training set includes 60000 samples, while the test one 10000. When the datasets are imported, thanks to the possibility to apply transformations directly when downloading the data provided by Pytorch, two basic transformations are performed:

- `ToTensor()`, to convert to Pytorch tensors;
- `Normalize((0.1307,), (0.3081,))` to normalise data mean providing mean and standard deviation respectively.

3.1. Methods

First of all, it is important to state that, given the big number of samples in the dataset, in this case the K-Fold cross-validation method will not be implemented. However, for the hyper-parameters optimisation and the model selection, the training set will be divided into two chunks via the `random_split` function: 80% of it (48000 samples) will be used for the training process, while the other 20% is used to validate each model tested in the random search.

Considering the nature of the samples (i.e., the fact that they are images) the network employs two convolutional layers and two max pooling layers to infer the images properties by producing feature maps and then to reduce the images sizes, to decrease the computational cost of the learning process.

More specifically, the network structure is the following:

- one convolutional layer with 1 input channel, `n_channels` output channels, a 3 by 3 kernel, padding and stride set to 1 to keep the image size equal to 28 × 28 pixels;

- a max pooling layer which halves the picture sizes ($28 \times 28 \longrightarrow 14 \times 14$);
- a 2-dimensional dropout layer with dropout probability `pdro2d`, sampled uniformly in $[0, 0.25]$;
- another convolutional layer with `n_channels` input channels, 2·`n_channels` output channels, 5 by 5 kernel, padding= 0 and stride= 1, that reduces the size of the images to 10×10 pixels;
- another max-pooling layer which halves again the image sizes ($10 \times 10 \longrightarrow 5 \times 5$);
- one fully connected layer with $5 \times 5 \times 2$·`n_channels` = 50·`n_channels` input units and 100 outputs;
- another dropout layer (1 dimensional) with dropout probability `pdro1d`, sampled uniformly in $[0, 0.25]$;
- one fully connected layer with 100 inputs, 10 outputs (corresponding to the 10 labels).

The number of convolutional channels `n_channels` is an integer chosen in the $[3, 11]$ interval. Each layer is activated by a ReLU activation function but the last one: in fact, for handling the production of class probabilities, the `CrossEntropyLoss()` loss function combines the LogSoftmax activation function and the negative log likelihood loss into a single class and therefore performs the final classification.

In addition, during the random search also some regularisation and optimisation methods are investigated, in particular the optimiser is chosen between:

- SGD, with L2 regularisation parameter (weight decay) `reg_param` sampled from a log-uniform distribution in the interval $[10^{-5}, 10^{-2}]$, `momentum` parameter sampled uniformly in $[0.5, 0.95]$ and learning rate `learning_rate` sampled from a log-uniform distribution in the interval $[10^{-5}, 10^{-3}]$;
- Adam optimiser, with the same sampling methods for `learning_rate` and `reg_param`.

Given the size of the dataset, the training process can be quite computationally demanding and significantly time consuming, thus an *early stopping* procedure is implemented in the main network class, in order to stop the training after the validation loss has reached a certain stability, in order to avoid useless iterations of the training loop. In particular, when the difference in absolute value between the validation loss computed at a certain iteration and the average loss of the last 10 epochs is less than 0.005, the training loop is interrupted and the learning stops.

After having performed the random search over 100 different CNN structures, the one with the minimum final validation loss (`final_net2`) is chosen as best model and tested, evaluating the accuracy in the classification task over the test set as the percentage of correctly classified samples.

Finally, convolutional filters and activation profiles are plotted to gain more knowledge about the learning process.

3.2. Results

The best network's parameters found during the random search are displayed in Table (2).

`n_channels`	`pdrop1d`	`pdrop2d`	`optimizer`	`momentum`	`learning_rate`	`reg_param`
10	0.05445	0.14743	SGD	0.88737	0.00241	0.00028

Table (2) – Best network (`final_net2`) parameters for the classification task.

The final model is able to correctly classify 9887 samples out of 10000, achieving an accuracy of 99%.

In Figure (7), the filters of the first convolutional layer can be found. As expected, they are `n_channels` $= 10$, 3×3 images representing the weights of each convolutional kernel. Similarly, in Figure (8), the second layer filters are displayed: this time, given the initial setting of the net, the images are 5×5 and there are `n_channels` $\times 2\cdot$ `n_channels` $= 200$ of them. Broadly speaking, it is very difficult for an external viewer understand only from looking at these pictures how the network learns.

It is, on the other hand, more intuitive to grasp which features are more easily detected by the network if the activation profiles are analysed. In Figure (2), each channel shows clearly the input digit (a 6), in different shades of grey, meaning that the first convolutional layer grasps already the main features of the input digit. The second layer, displayed in Figure (3), instead focuses in images decomposition, highlighting the single features of the input image, e.g. the edges. One also could notice that the pictures size in this last figure is, as expected, 10×10, given the kernel sizes and the down sampling operated by the Max-Pooling layer.

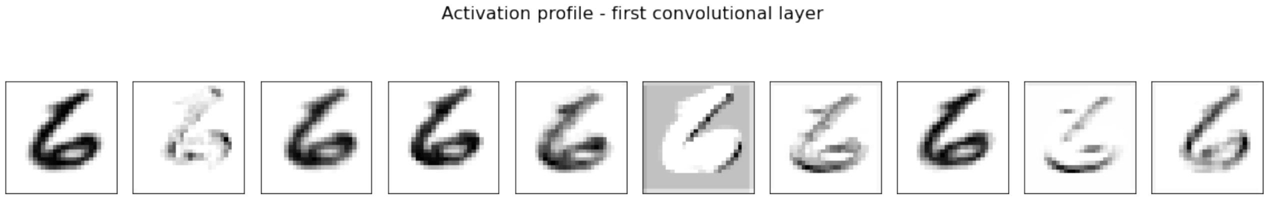

Figure (2) – Activation profiles for the first convolutional layer. The shape of the input digit (6) can be clearly outlined.

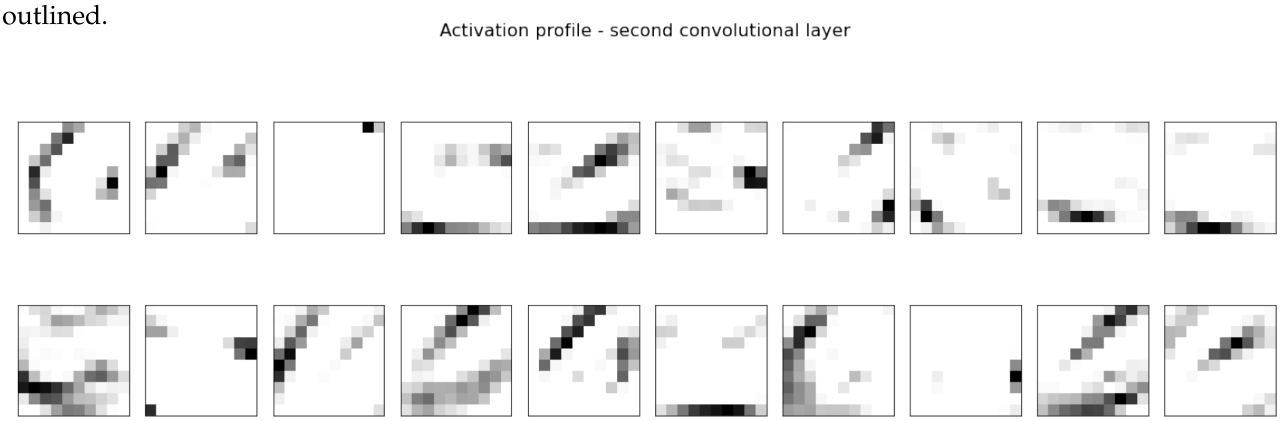

Figure (3) – Activation profiles for the first convolutional layer. The input digit is decomposed and its features are enhanced and detected by the convolutional channels.

4. Conclusion

In conclusion, one could state that both the architectures implemented and optimised worked properly for the assigned tasks.

For regression, the network is able to properly infer the function behaviour, despite proving to be not too flexible with an incomplete training set. To improve this issue, a more deep network may be implemented in the future, and maybe reducing the impact of regularisation processes could help in making the network less rigid.

Concerning the classification, the network achieved brilliant results, correctly classifying almost 99% of the test samples. However, the ability of detecting digits despite noise or when subject to transformations (rotation, mirroring...) has not been tested, therefore making the network effective only in a limited set of conditions.

Appendix

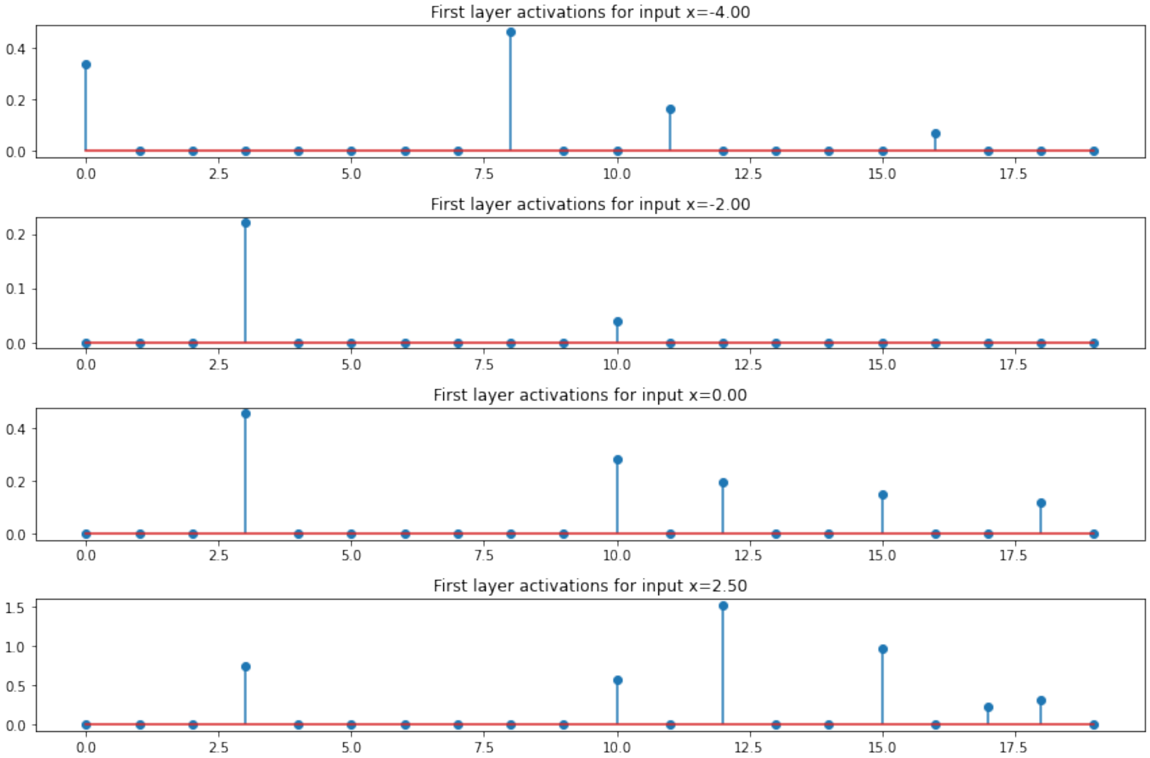

Figure (4) – Regression FCN activation profile of the first hidden layer, for inputs $x = [-4, -2, 0, 2.5]$

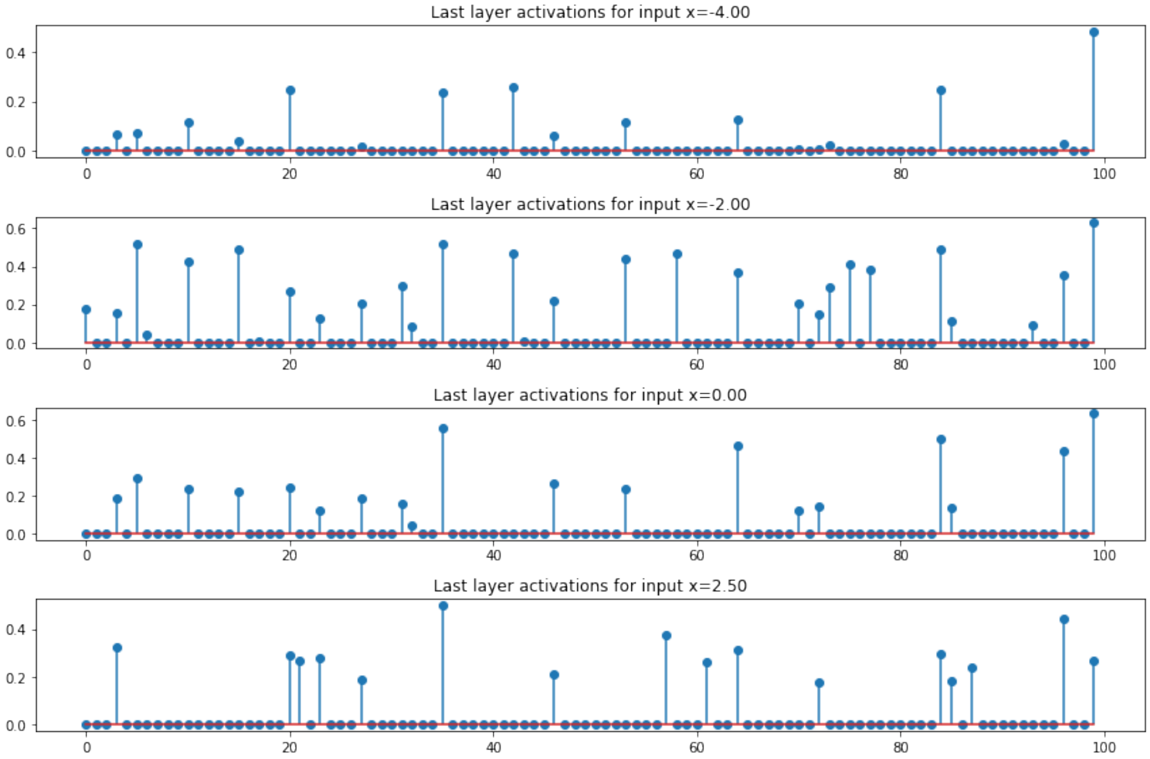

Figure (5) – Regression FCN activation profile of the second hidden layer, for inputs $x = [-4, -2, 0, 2.5]$

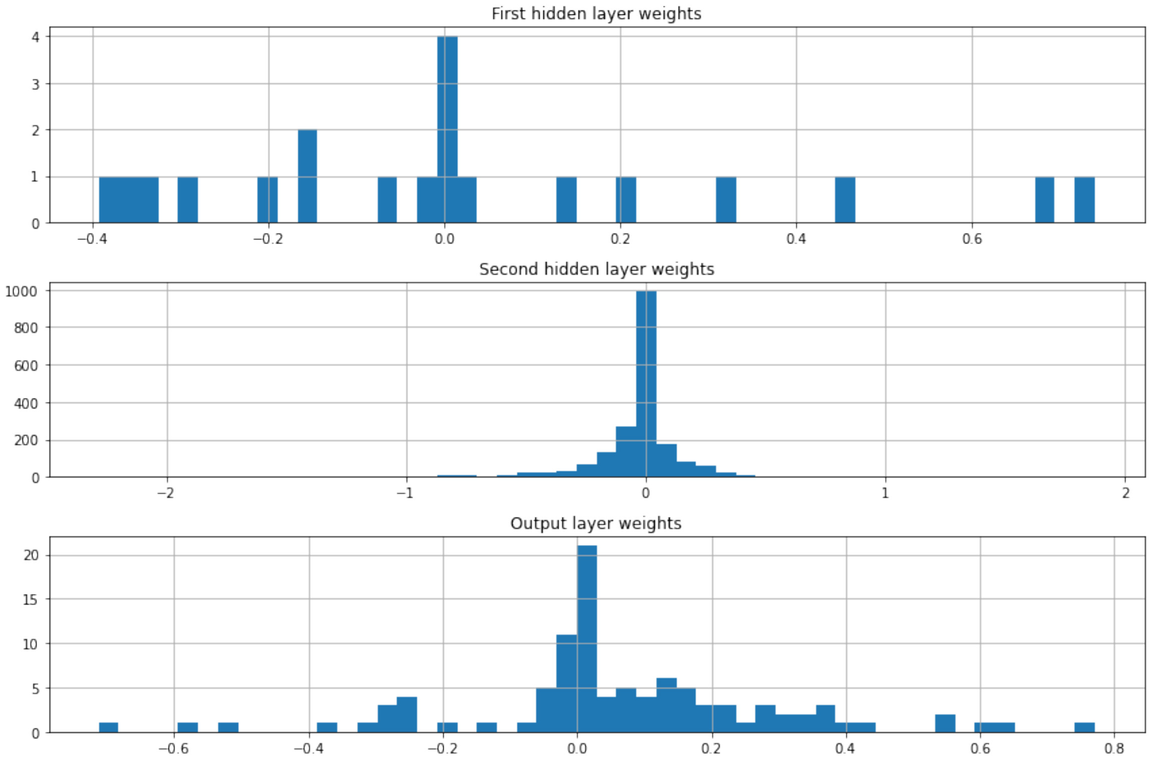

Figure (6) – Regression FCN weights reported in histograms.

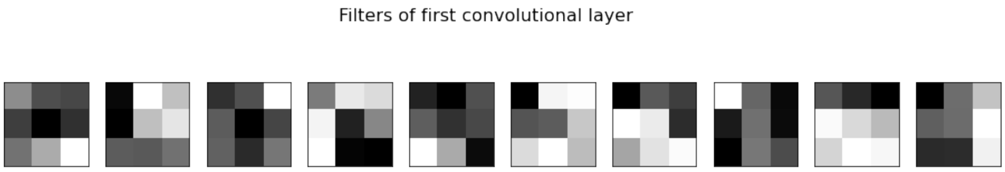

Figure (7) – First convolutional layer filters for the CNN optimised for the classification task.

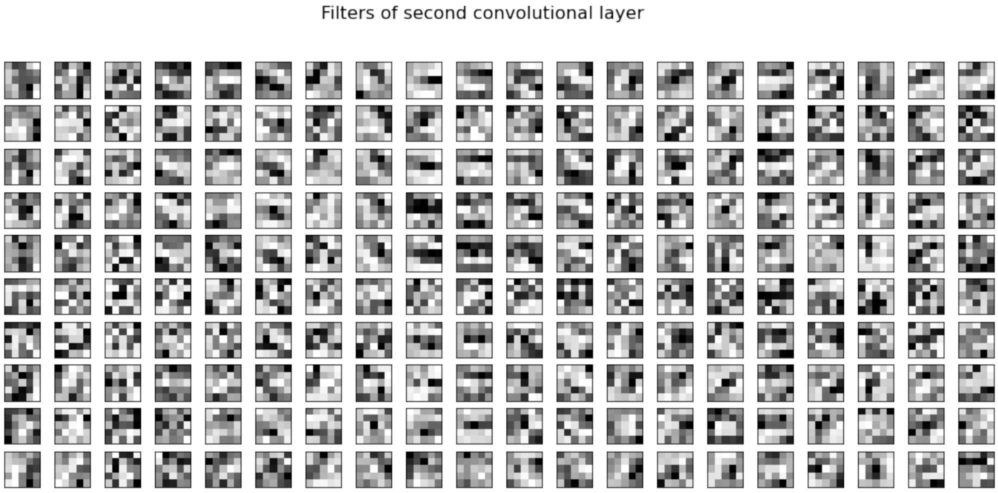

Figure (8) – Second convolutional layer filters for the CNN optimised for the classification task.