# Regression and classification tasks: supervised deep learning

Lorenzo Mancini 2019098

January 18, 2022

**Abstract**

Deep learning methods can be very useful for solving supervised deep learning tasks. In this homework we investigare the behaviour of deep neural networks applied to a regression problem and a classification one. We optimize the performances of such models using the random grid search for hyperparameters. Finally we show weight histograms and receptive fields.

## 1 Introduction

In this homework we build two neural networks in order to solve two different tasks:

- the first task is a **Regression** problem: the aim is to learn an unknown function from a noisy small dataset;
- the second task is for **Classification**: the aim is to build a classifier for the FashionMNIST dataset which containt images of Zalando's articles.

## 2 Regression

### 2.1 Dataset

We are provided with a training dataset containing 100 labeled elements which are supposed describe an unknown function where some noise has been added. Obviously, we also have a test set with other 100 elements needed for testing the accuracy of our model after the training.

$$\hat{y} = f(x) + \text{noise}.$$

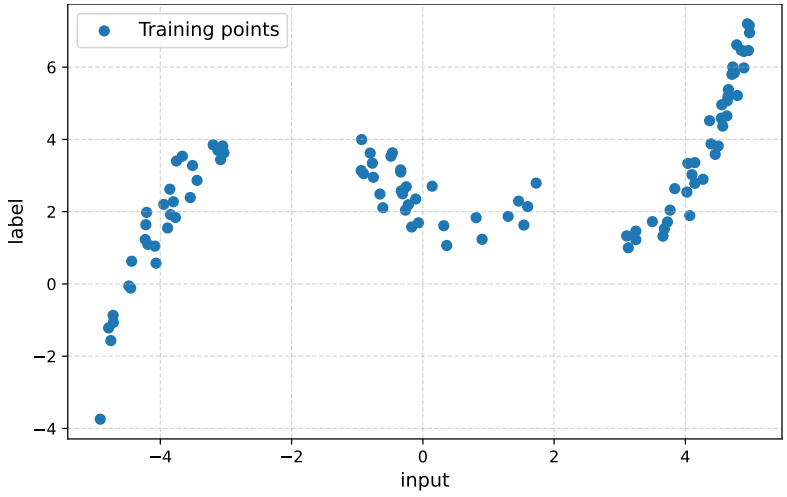



Figure 1: Plot of training points.

## 2.2 Methods

In order to solve our regression task we start with a simple architecture of a Neural Network composed by 3 hidden layers with $10, 20, 10$ neurons per layer respectively. Different architectures gave very similar results, furthermore the number of neurons should be not too small and not to high, $< 30$, since the problem is quite simple. For this reason we're not going to include the number of neurons into the hyperparameters to be optimized. In order to find the right set of hyperparameter, we exploit the tecnique of random search in which the network is trained with a random combination of hyperparameter for a choosen number of trials. The space of the hyperparameters is as follows:

- **Optimizers**: Adam, SGD or RMSprop;
- **Learning rate**: log-uniform between $10^{-4}$ and $10^{-1}$;
- **Batch size**: 4, 8, 10 or 12;
- **L2**: in the loss function we add a factor that multiplies the sum of the weights squared. The factor is chosen from a log-uniform distribution between $10^{-5}$ and $10^{-1}$;
- **Activation function**: ReLu, LeakyReLu, Sigmoid or Tanh.

The performances are evaluated using the cross-validation method with $k = 5$ folds. The random search consists of 300 trials. As loss function we consider the Mean Squared Error.

## 2.3 Results

From the random search process it results that the best optimizer to be used is Adam whereas the activation function can be one between ReLu or LeakyRelu (both show good performances). Regarding the regularization, best performance are reached with L2 regularization or even without any regularizer. A good learning rate can be computed between 0.001 and 0.008. Thus, we repeat the training for 150 epochs with learning rate equal to 0.006, Adam optimizer, LeakyRelu activation and 0.0002 of L2 regularization.
Here we show the results of the training.

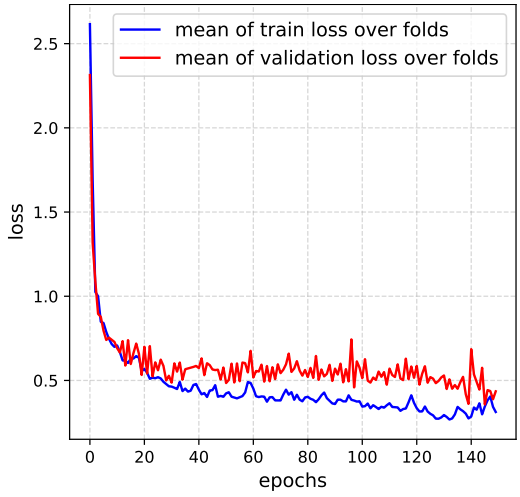



Figure 2: Plot of train and validation loss obtained with best hyperparameters found.

As can be seen, the validation and training error reach a loss of $\sim 0.3$, which is quite good considering that the training samples are noisy.
The network output is the following:

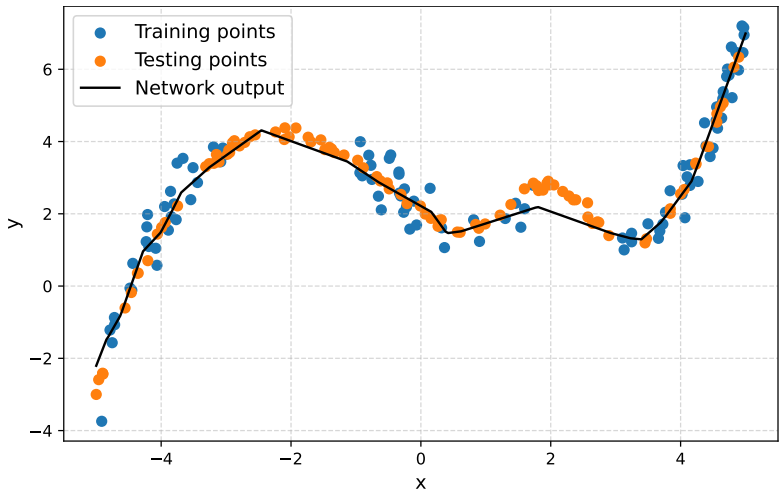



Figure 3: Plot of training points (in blue), test points (in orange) and the output of our netwokr.

Finally, we can compute the accuracy of our model using our test set. It results that the test loss is $\sim 0.112$ which is lower than the training one: this is reasonable since our test set is less noisy than the training one.

## 2.4 Activations and histograms

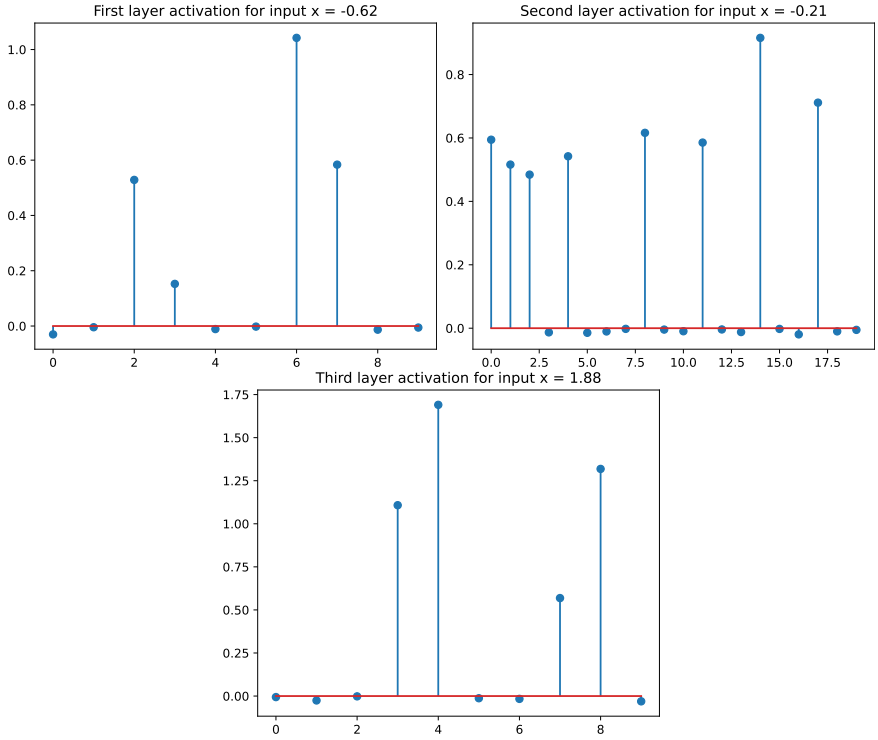



Figure 4: Activation of the three hidden layer for random inputs.

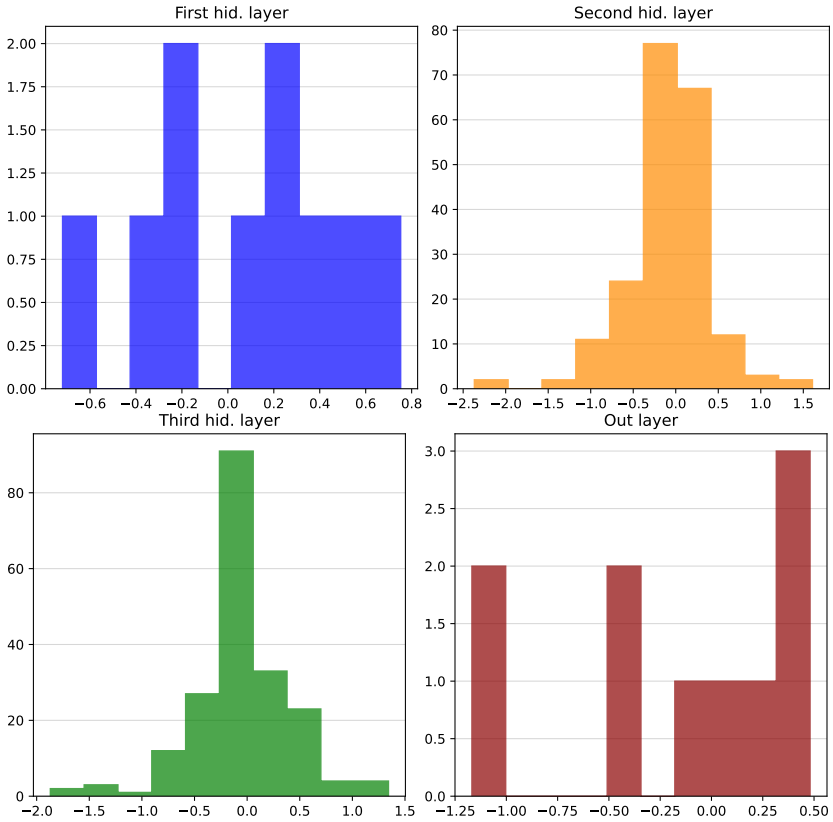



Figure 5: Weight histograms for the three hidden layers and the output layer.

# 3 Classification

## 3.1 Dataset

Fashion MNIST is a dataset that contains Zalando's articles as 28x28 grayscale images. It is divided into the training set which contains 60000 items and a test set which contains 10000 items. The images can be grouped in ten classes. Here is an example of an item of the training set:

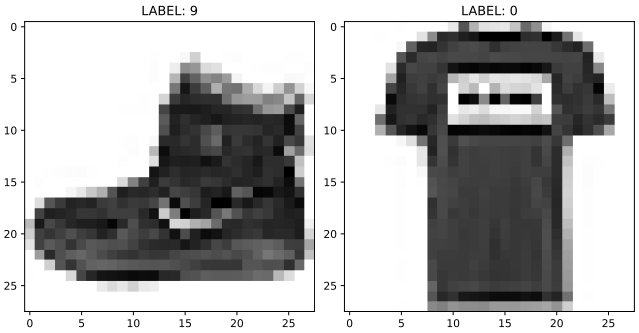



Figure 6: Example of two training items with the corresponding labels.

## 3.2 Methods

The structure of the network is the following:

- First convolutional layer with 32 filters of size 5, stride 1 and padding 0;
- Max pool layer with kernel size of 2;
- Second convolutional layer with 64 filters of size 5, stride 1 and padding 0;

- First f.c. linear hidden layer of 128 neurons.
- Output layer of 10 neurons.

The activation function is the ReLu.
Following the procedure of the regression task, we apply a Random Grid Search technique in order to find a satisfying range for the hyperparameters. We run the training with random hyperparameters for 50 trials. This time, in order to save some computational time, we do not make use of the cross-validation setup: one could've used the same chunk of code of the regression task for the training loop, but here it would've taken too long since the dataset is bigger. The hyperparameters space is as follows:

- **Optimizers**: Adam, SGD or RMSprop;
- **Learning rate**: log-uniform between $10^{-4}$ and $10^{-1}$;
- **Batch size**: 64, 128, 256;
- **L2**: in the loss function we add a factor that multiplies the sum of the weights squared. The factor is chosen froma log-uniform distribution between $10^{-5}$ and $10^{-1}$;

As loss function we use the Cross Entropy loss.

### 3.3 Results

It results that the best performances are achieved using the Adam optimizer with L2 regularization. Thus, the network is trained again for 100 epoch with the best hyperparameters found:

- batch size of 128;
- Adam optimizer;
- learning rate equal to 0.0002.

We also add some dropout (0.3) in order to avoid overfitting.

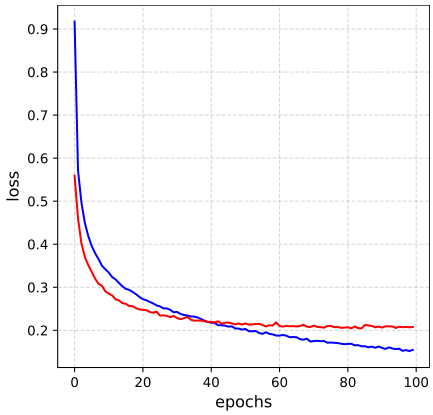
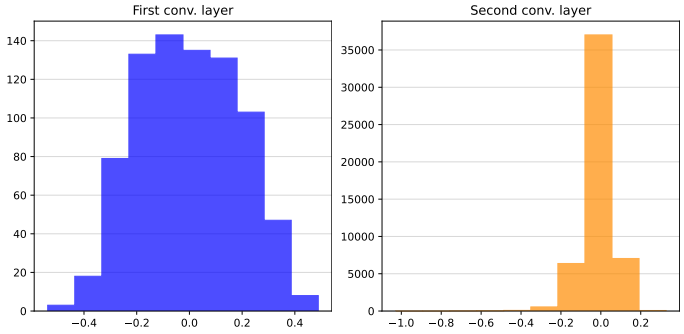



Figure 7: Training (in blue) and validation (in red) loss for the classification task.



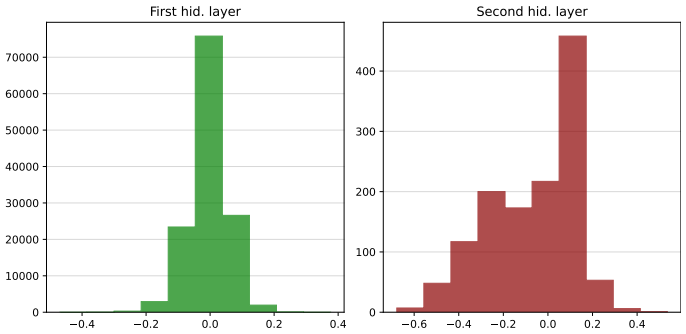



Figure 8: Weight histograms of the various layers.

Finally, we can try to visualize the output of the first convolutional layer (the same procedure could be applied also to the second convolutional layer).

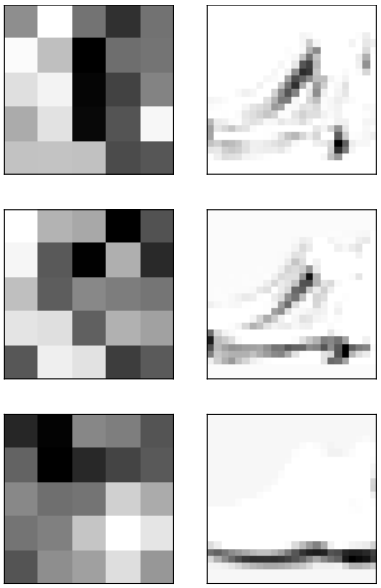


Figure 9: Output of the first convolutiona layer to the first element of the training set. We show the results of three filters that seem to highlight edges.

The model reaches an accuracy of $\sim 92\%$ over the test set. Here we show the confusion matrix.

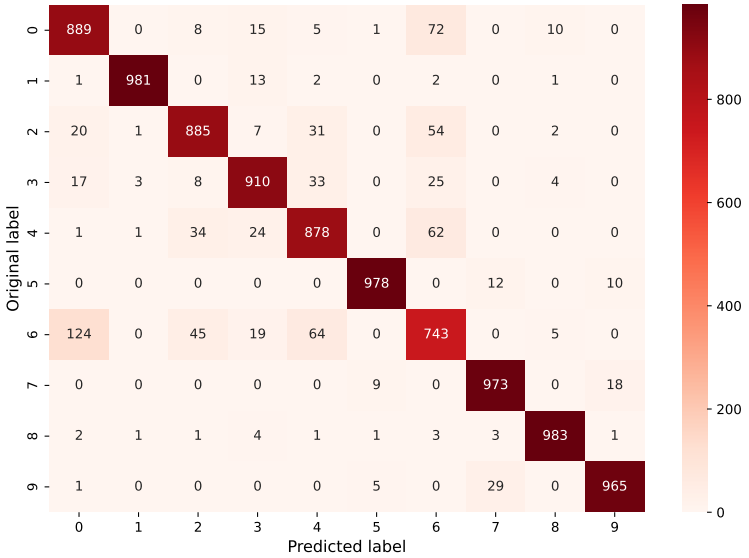



Figure 10: Confusion matrix of the classification task.