# Neural Networks Homework 1: Supervised Learning

Manuel Guatto; email: manuel.guatto@studenti.unipd.it
Mat: 2022574

February 11, 2022

## 1 Introduction

The homework is about the Supervised Learning and it is articulated in two main tasks:

1. Regression
    - the regression task will consist in a function approximation problem
2. Classification
    - the classification task will consist in a simple image recognition problem, where the goal is to correctly classify images of Zalando's article images (Fashion MNIST)

The main goal of the homework is to compare different kind of models based on different optimizers, some regularization techniques, and also different network architectures (e.g. mlp classifier, cnn classifier). In particular to carry out the homework we will follow the following points:

- Implement basic regression and classification tasks
- Explore advanced optimizers and regularization methods
- Optimize hyperparameters
- Implement CNN for classification task
- Visualize weight histograms, activation profiles and receptive fields

## 2 Regression Task

### 2.1 Introduction

The goal is to train a neural network to approximate an unknown function:

$$f : R \longrightarrow R$$
$$x \longrightarrow y = f(x)$$
$$network(x) \approx f(x)$$

As training point, we only have noisy measures from the target function: $\hat{y} = f(x) + noise$

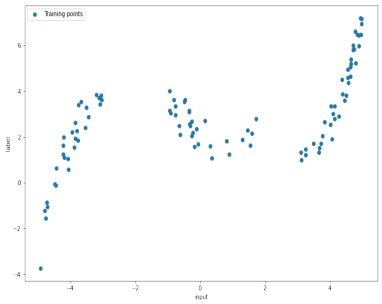


Figure 1: Training Points

## 2.2 Define the model, comparison between optimizers and regularization methods

First of all we define the neural network model with some Linear layers, in particular we define, beside the input and output layers, two hidden layers. The first neural network we define presents the following parameters:

| Layer | # of neurons |
|---|---|
| Ni | 1 |
| Nh1 | 128 |
| Nh2 | 256 |
| No | 1 |

Table 1: Neurons for each layer initial network

As loss function we will use the Mean Square Error. For the optimizer we will use as first the Stochastic Gradient Descent (SGD) then we will test the network with the SGD with momentum and Adam. K-fold cross validation was implemented to compare the various models. The learning rate is 1e-3 and the momentum is fixed at 0.5 and we train the model for 2000 epochs. The k-fold function returns the training, the validation errors and the generalization gap. The results are the following:

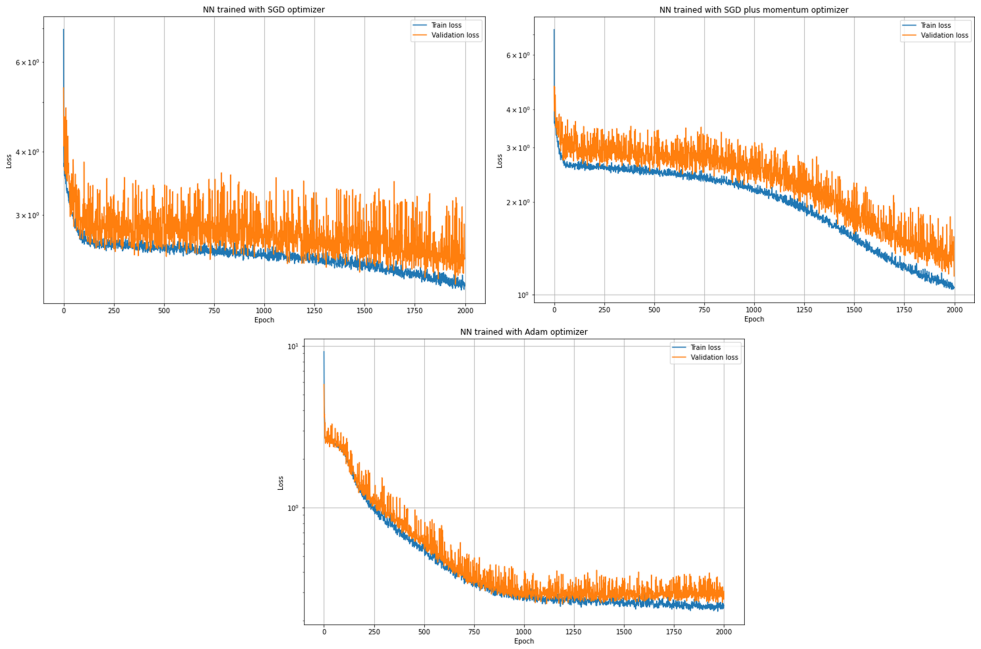


Figure 2: Training Validation plot of different optimizers

| Optimizer | Training Error | Validation Error | Generalization Gap |
|---|---|---|---|
| *SGD* | 2.49938 | 2.53977 | 0.04039 |
| *SGD + momentum* | 1.28711 | 1.84355 | 0.98978 |
| *Adam* | 0.55009 | 0.27588 | 0.27420 |

Table 2: Results of optimizer comparison

As expected from theory we have an improvement in terms of both training and validation error applying the momentum to the SGD, but the best result is achieved by the Adam optimizer that converges to a minimum faster w.r.t. the others two optimizers, therefore for the next comparisons we will use the Adam optimizer. Trying to reduce the overfitting and as a consequence the generalization gap we will use some regularization techniques, in particular we will exploit:

- Dropout regularization,
- l2 regularization.

Below we can find the plots of the train validation error of both the regularization methods.

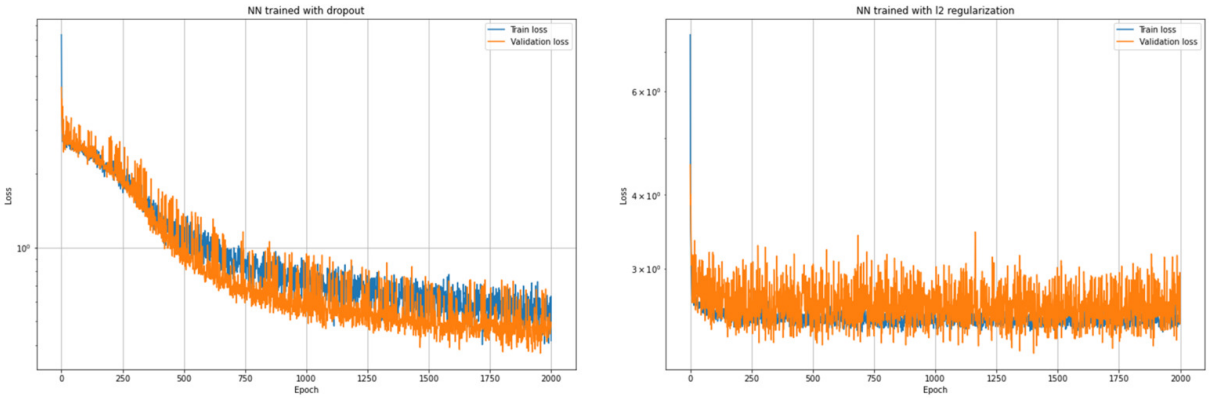


Figure 3: Training Validation plot of different regularization methods

| Regularization method | Training Error | Validation Error | Generalization Gap |
|---|---|---|---|
| *Dropout* | 1.00285 | 0.47041 | 0.53244 |
| *l2* | 2.46826 | 2.58581 | 0.11754 |

Table 3: Results of regularization methods comparison

Looking to the two regularized methods we can see that both the training and validation error of the l2 regularized model is greater than the previous cases but this can follow to the fact that we have added to the loss the l2 norm of the weights. The dropout version of the model is worst than the Adam without regularization and also this was an expected result since some of the neurons were "tuned-off" during the training, but we still want to try to improve the dropout model by tuning the hyperparameters.

## 2.3 Tuning hyperparameters

To tune the hyperparameters we used the Optuna framework with the k-fold function to train the various models and the validation error to select the best one. The parameters that we have take care of are the number of neurons in the hidden layers and the learning rate. In the optimization of the hyperparameters the models are trained for 1000 epochs. The hyperparameters tested were:

| # Trial | Nh1 | Nh2 | Lr | Training Error | Validation Error |
|---|---|---|---|---|---|
| 1 | 81 | 67 | 0.05770 | 1.08428 | 0.75428 |
| 2 | 250 | 147 | 0.00333 | 0.79022 | 0.46494 |
| 3 | 216 | 92 | 0.00255 | 0.86584 | 0.44307 |
| 4 | 100 | 99 | 0.00532 | 0.90186 | 0.48945 |
| 5 | 90 | 254 | 0.01422 | 0.88946 | 0.33547 |

Table 4: Hyperparameters tested

Training the model with dropout and the new parameters for 2000 epochs we obtained the following result:

| # epochs | Nh1 | Nh2 | Lr | Training Error | Validation Error | Generalization Gap |
|---|---|---|---|---|---|---|
| 2000 | 90 | 254 | 0.01422 | **0.69977** | **0.59994** | **0.09983** |

Table 5: Model with tuned hyperparameters result

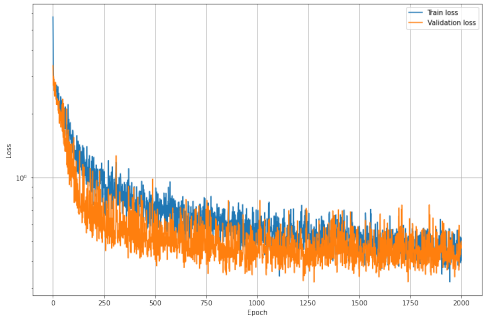


Figure 4: Train and Valid error after tuning hyperparams

Here we can see that the training error is similar to the one obtained with the standard model with the Adam optimizer, but the generalization gap is lower. We could reduce more the generalization gap implementing the early stopping since the network was starting to overfitting as shown in Figure 4.

Testing this model the **test error was: 0.18235** and overlapping the network output to the testing points we obtained the plot in Figure 5. In Figure 6 we can see the comparison between the histograms of different network architectures and in Figure 7 we can see the activation of the last layer of our final model.

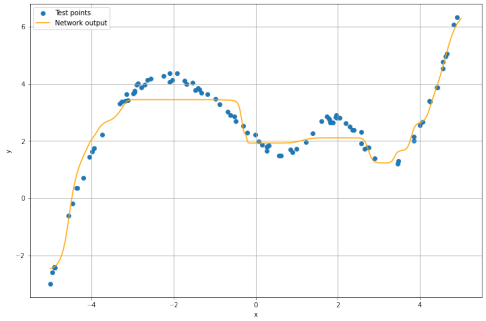


Figure 5: Overlapping between testing points and network output function

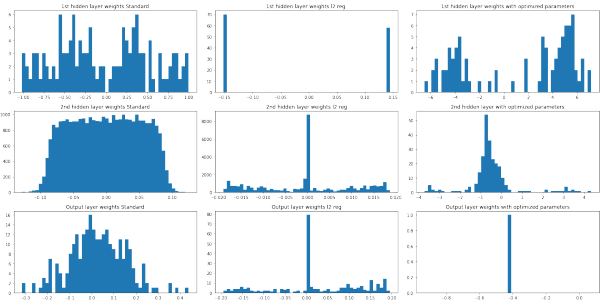

Figure 6: Weights histograms of different network architectures

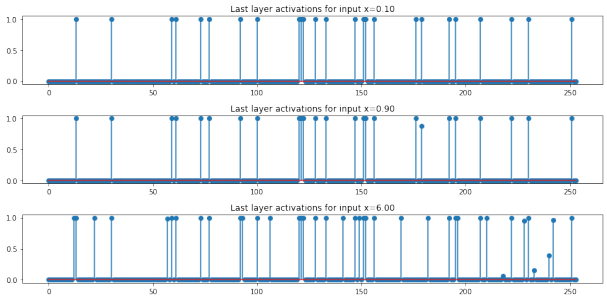


Figure 7: Layer activation after tuning hyperparams

# 3 Classification Task

## 3.1 Introduction

The goal is to train a neural network that maps an input image (from fashionMNIST) to one of the ten classes (multi-class classification problem with mutually exclusive classes) and explore different optimizers, acivation functions, network architectures, analyzing the effect of different regularization methods.

## 3.2 Define the mlp classification model and compare optimizers

First of all we define the neural network model with some Linear layers, in particular we define, beside the input and output layers, two hidden layers. In particular the first neural network we define present the following parameters:

| Layer | # of neurons |
|---|---|
| Ni | 28*28 |
| Nh1 | 512 |
| Nh2 | 128 |
| No | 10 |

Table 6: Neurons for each layer initial network

As loss function we will use the Cross Entropy Loss. For the optimizer we will use as first the Stochastic Gradient Descent (SGD) then we will test the network with Adam. K-fold cross validation was implemented to compare the various models. The learning rate is 1e-3 and we train the model for 40 epochs. The k-fold function returns the training and the validation errors. The results are the following:

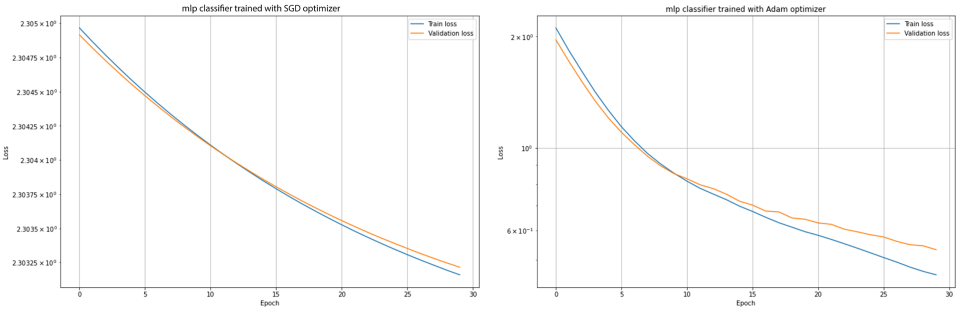


Figure 8: Training Validation mlp classifier with SGD and Adam

| Optimizer | Training Error | Validation Error |
|---|---|---|
| *SGD* | 2.30390 | 2.30321 |
| *Adam* | 0.84072 | 0.53168 |

Table 7: Comparison between optimizers in classification task

## 3.3 Tuning Hyperparameters

As for the regression task we tune the hyperparameters with the Optuna framework. Also this time we try to select the best number for the hidden neurons and learning rate. In this case we will train the possible optimized models for 10 epochs.

| # Trial | Nh1 | Nh2 | Lr | Training Error | Validation Error |
|---|---|---|---|---|---|
| 1 | 113 | 365 | 0.00851 | 1.31224 | 1.25849 |
| 2 | 206 | 235 | 0.08381 | 2.34144 | 2.34798 |
| 3 | 109 | 172 | 0.03855 | 2.26293 | 2.32453 |

Table 8: Hyperparameters tested for mlp classifiers

Training the model with the hyperparameters tuned and the Adam optimizer we can see that the result is better than the previous result without the tuning, in fact both training and validation error are lower w.r.t the mlp adam classifier with the initial number of neurons in the hidden layer.

| # epochs | Nh1 | Nh2 | Lr | Training Error | Validation Error |
|---|---|---|---|---|---|
| 30 | 113 | 365 | 0.00851 | **0.44873** | **0.39867** |

Table 9: Model trained with tuned hyperparameters result

Testing the networks accuracy with the test set we obtain the following result.

| Optimizers | Tuned Hyperparameters | Accuracy |
|---|---|---|
| SGD | No | 10.0000 % |
| Adam | No | 82.3400 % |
| Adam | Yes | 86.5500 % |

Table 10: Mlp classifiers accuracy

## 3.4 Define CNN model and evaluate the regularization techniques

Now we define a CNN network with two convolutional layers followed by one linear layer. The function activation used in the network are the ReLU for the convolutional layers and for the first linear layer instead we used the Softmax activation function. The architecture can be resumed as:

| Layer | In_Features | Out_Features | Kernel | Stride | Padding |
|---|---|---|---|---|---|
| 1st Convolutional | 1 | 16 | 5 | 1 | 2 |
| 2nd Convolutional | 16 | 32 | 5 | 1 | 2 |
| 1st Linear | 1568 | 10 | / | / | / |

Table 11: CNN architecture

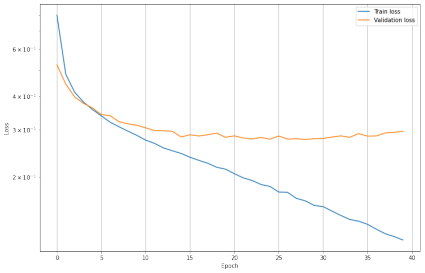


Figure 9: CNN train and valid loss

Training 40 epochs the CNN model we obtained the following results in terms of training and validation error. The plot is shown in Figure 9.

| Network | Training Error | Validation Error |
|---|---|---|
| CNN | 0.23830 | 0.29526 |

Table 12: CNN Training and validation loss

Trying to regularized the model we added batch normalization for each of the convolutional layer keeping the same architecture. From Figure 10 and also from 9 we can see that both network are overfitting therefore we have tried to implement also the early stopping regularization (Figure 10).

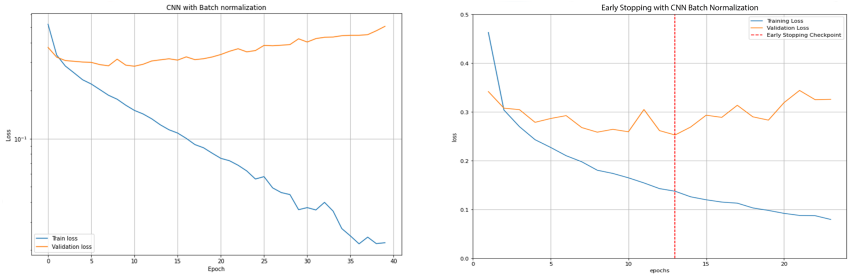


Figure 10: Batch Normalization and Early Stopping

Checking the accuracy we have:

| CNN type | Early Stopping | Accuracy |
|---|---|---|
| CNN | No | 89.8900 % |
| CNN + Batch Norm | No | 88.9000 % |
| CNN + Batch Norm | Yes | 90.6900 % |

Table 13: CNN classifiers accuracy

## 3.5 Plot weights, Filters, Convolutional Layers

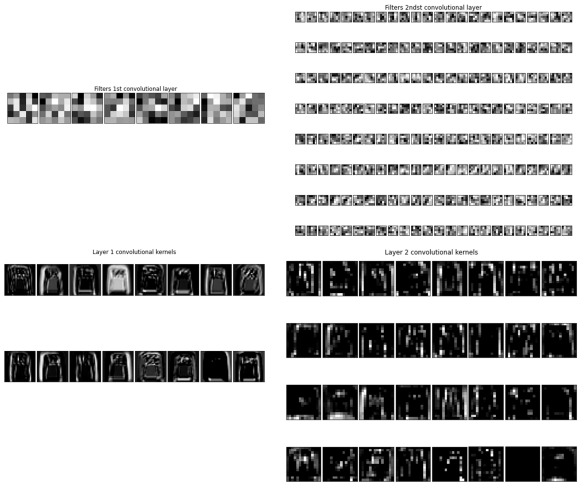


Figure 11: Filters and Convolutional Layers

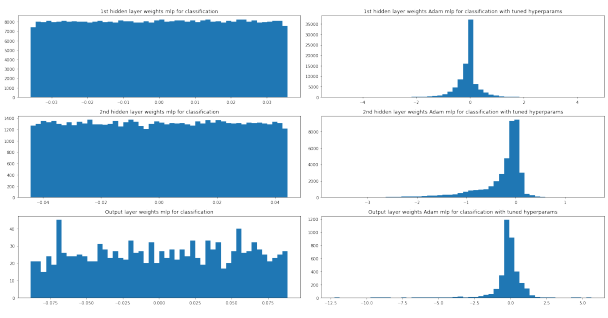

Figure 12: Weights historgam of MLP classifiers