Neural Network and Deep Learning - A.A. 2021/22

Assignment 1

Supervised deep learning: Regression and classification tasks

Fontana Francesco 2026924

22 March 2022

Abstract

Deep learning methods can be very usefull for solving supervised deep learning tasks. In this homework we investigare the behaviour of deep neural networks applied to a regression and a classification problem. A random grid search for hyperparameters has been used in order to optimize the performances of such models. Finally we show weight histograms and receptive fields.

1 Introduction

Supervised Learning is a framework of machine learning in which a model learns "by example", inferring useful patterns from a set of inputs (features) paired with their desired output (labels). Formally, given a labelled dataset $D = (x_i, y_i)_{i=1,...,N}$ supervised learning trains a model $f : x \to y$ such that $f(x_i) \approx y_i, \forall i = 1, ..., N$. This is done by iteratively minimizing a *loss function*, which quantifies the difference (error) between predictions $f(x_i)$ and known labels y_i. For neural networks, the minimization is achieved by the *backpropagation* algorithm, along with an *optimizer* such as Stochastic Gradient Descent.

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 do a function approximation i.e. learn an unknown function from a noisy small dataset;
- the second task is for **Classification**: the aim is to build a multi-class 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 samples (x, y) which are supposed to describe an unknown scalar 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. Practically, the model is trying to approximate a function of the following form:

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

Learning a single smooth scalar function is usually an easy task. In this case, there are a few specifics of the dataset that make it more challenging:

- Samples are few (only 100)
- The training dataset is biased, in the sense that it does not cover uniformly the function's domain. In fact, two regions around $x = \pm 2$ are left out (fig. 1)

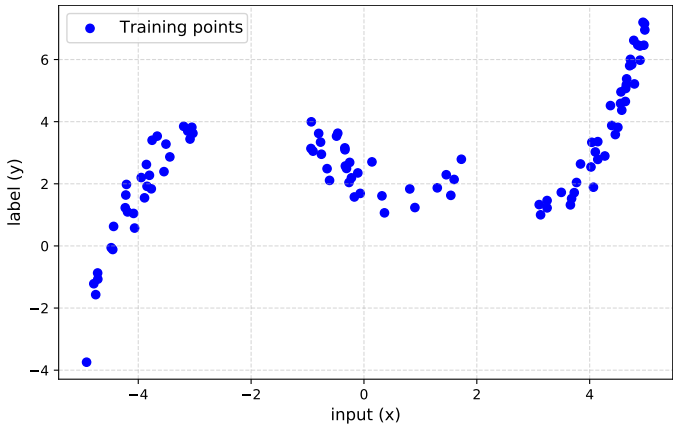

Figure 1: Graphical plot of the training points

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 respectively 10, 15, 10 neurons. 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 hyperparameters, we exploit the tecnique of random search in which the network is trained with a random combination of hyperparameters for a choosen number of trials (in this case set to 300). The space of the hyperparameters is as follows:

- **Optimizer:** 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* .

To mitigate the dataset's small size, cross-validation with $k = 5$ folds is used to estimate generalization performance.

The number of epochs has been set to 150 for each random hyperparameter's choice while it was increased to 500 for the last training with the best parameters. As loss function we've considered the *Mean Squared Error* (MSE).

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 without regularization but similar results are achieved also with L2 regularization; instead a good learning rate is in the range $[0.005, 0.016]$.

Thus, we repeat the training for 500 epochs with learning rate equal to 0.010, Adam optimizer, LeakyRelu activation function, without L2 regularization and with dropout value equal to 0.005. Here we show the results of the training.

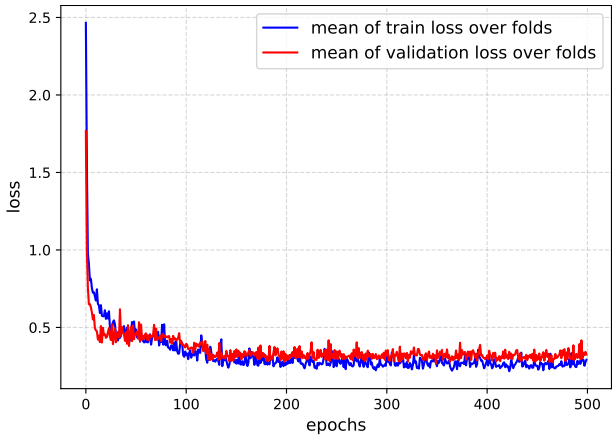

Figure 2: Plot of train and validation loss obtained with best hyperparameters of random grid-search.

As can be seen, the validation and training error reach a loss of ~ 0.27, which is quite good considering that the training samples are noisy. The output of the network is shown below in fig.3:

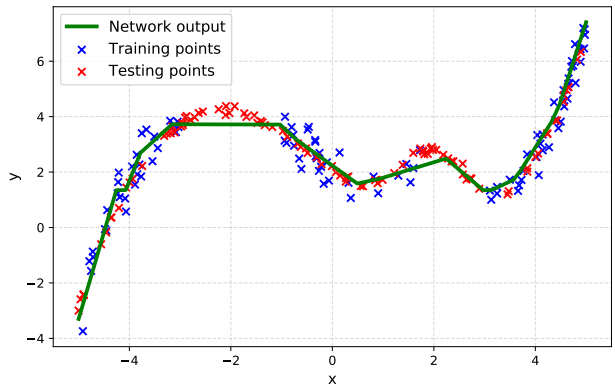

Figure 3: Plot of training points (in blue), test points (in red) and the output of the network (green line).

Finally, we can compute the accuracy of our model using our test set. It results that the test loss is ~ 0.14 which is lower than the training one: this is reasonable since our test set is less noisy than the training set.

2.4 Activations and histograms

In this section we show some plots and histograms about the hidden layers; in particular the output of each neuron is shown in fig.4. Note how the complexity is higher when we are moving to the deeper layers. At the beginning, neurons can only differentiate low/high inputs, but already at the second layer they begin to specialize to model local maxima. In layer 3 the output of several neurons already looks like the fitted function, while others do not activate at all. This opens the possibility of removing neurons,"pruning" the network, and reducing its computational cost while not affecting its performance.

Other plots are reported in the *Appendix* section, i.e. some histogram plots of all the weights of each layer (fig.9) and a plot of the activation' strenght of each single neuron in responce of a random input (fig.10).

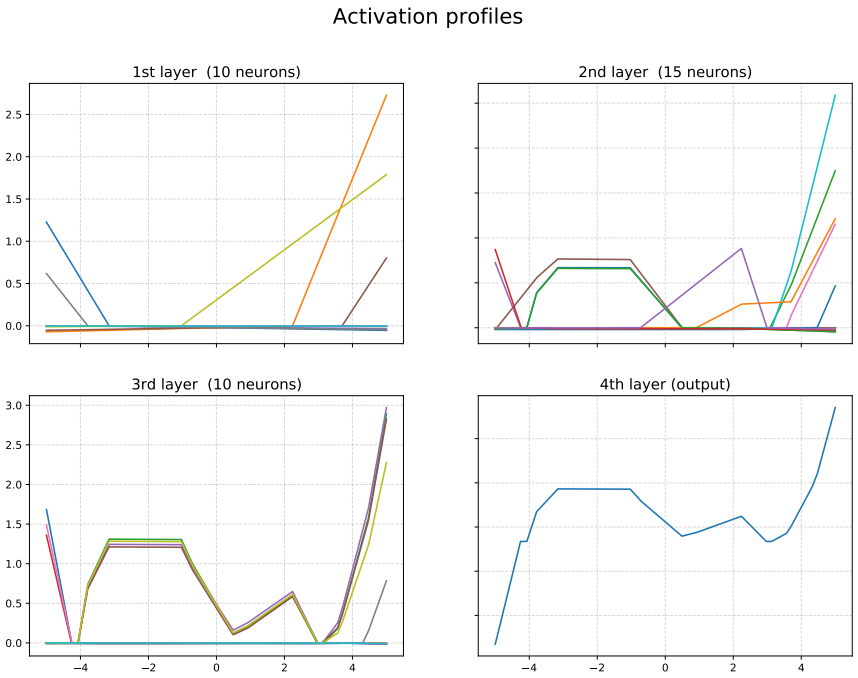

Figure 4: Activation profiles for all layers in the network

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 10 classes. Here is an example of an item of the training set:

3.2 Methods

The structure of the network is the following (fig.6):

- First convolutional layer with 32 filters of size 5, stride 1 and padding 0;

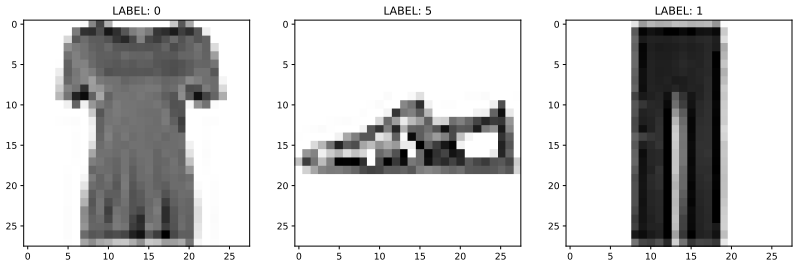

Figure 5: Example of 3 training images with the corresponding labels from MNIST dataset.

- Max pool layer with kernel size of 2;
- Second convolutional layer with 64 filters of size 5, stride 1 and padding 0;
- First fully connected linear hidden layer of 128 neurons.
- Output layer of 10 neurons.

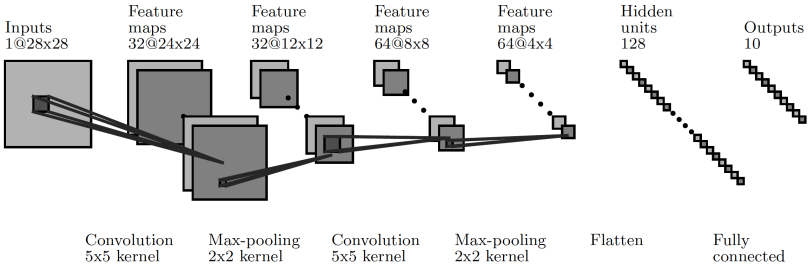

Figure 6: CNN Architecture. Image drawn using the script at gwding/draw_convnet.

The same activation function has been used in all the layers and corresponds to the *ReLu* function. As in the previous regression task, we apply a *Randoms Grid-Search* technique in order to find a satisfying range for the hyperparameters and this has been done 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 far bigger. The hyperparameters space is as follows:

- **Optimizer:** 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 from a log-uniform distribution between 10^5 and 10^1

Finally, the loss function we've used is the *Cross Entropy loss*.

3.3 Results

It results that the best performances are achieved using the Adam optimizer without regularization. Given that we have to note that the grid-search evaluates each choice of hyperparameters over just 9 epochs, to avoid taking too long. Since overfitting is not likely to happen immediately, the search suggests not to use regularization. But, at the end, one would like to train the model for longer, to maximize its performance. In that case, some amount of regularization must be considered. We also add some dropout (0.2) in order to avoid overfitting. Thus, the network is trained again for 100 epochs with the following hyperparameters:

- batch size = 64;
- Adam optimizer;
- learning rate = 0.0002.
- dropout = 0.2

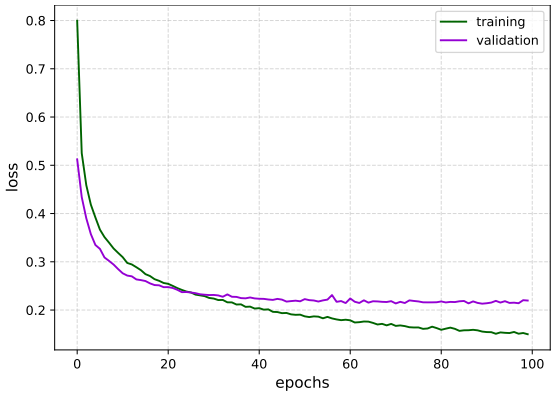

Figure 7: Training (in green) and validation (in purple) loss for the classification task.

Finally, we can try to visualize the output of the first convolutional layer in fig.8. Also in fig.11(*Appendix*) are plotted the histograms of the weight for the various hidden layers and the output layer.

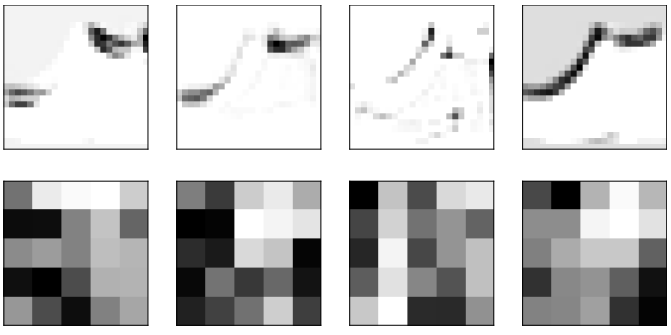

Figure 8: Output of the first convolutional layer to a random sample of the training set. We show the results of 4 filters that seem to highlight edges.

In the end, when we apply the model over the test set, it reaches an accuracy of $\sim 92,2\%$ and in fig. 12 (*Appendix*) there is the plot the corresponding confusion matrix.

4 Appendix

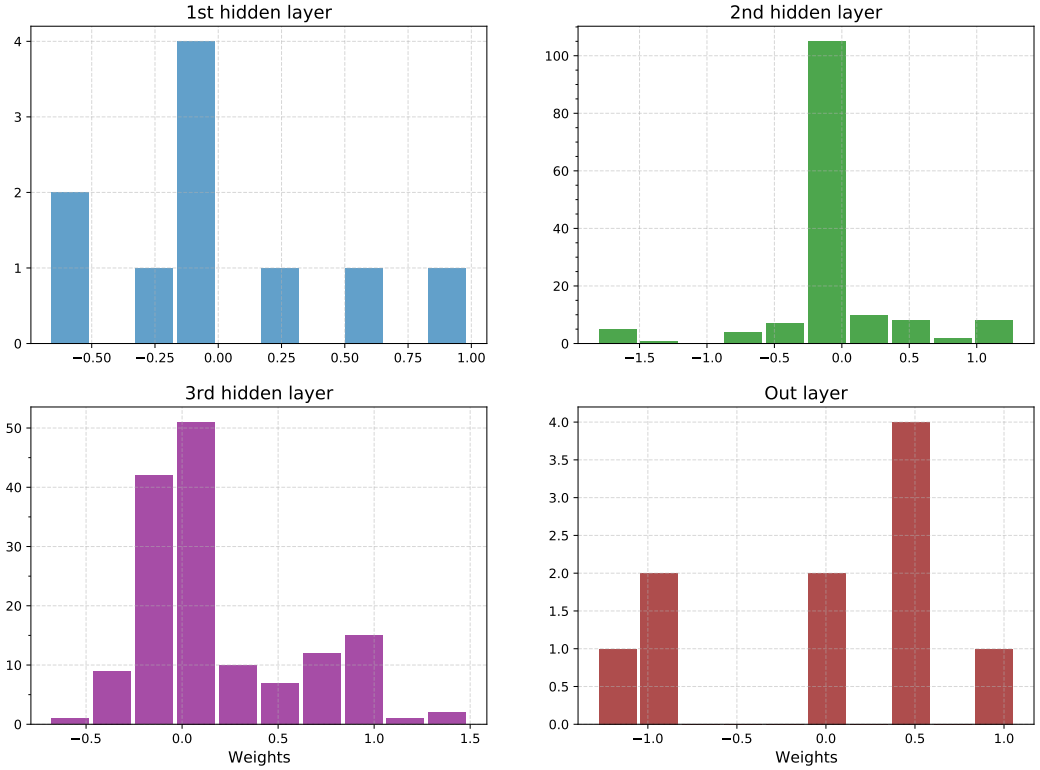

Figure 9: *Regression*: Weight histograms for the three hidden layers and the output layer.

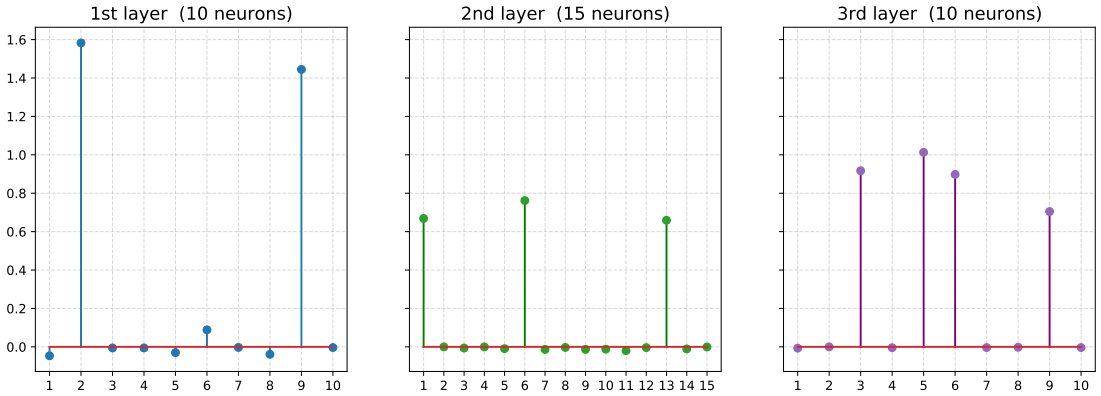

Figure 10: *Regression*: Activation of the three hidden layer for random inputs ($x = 4.05$)

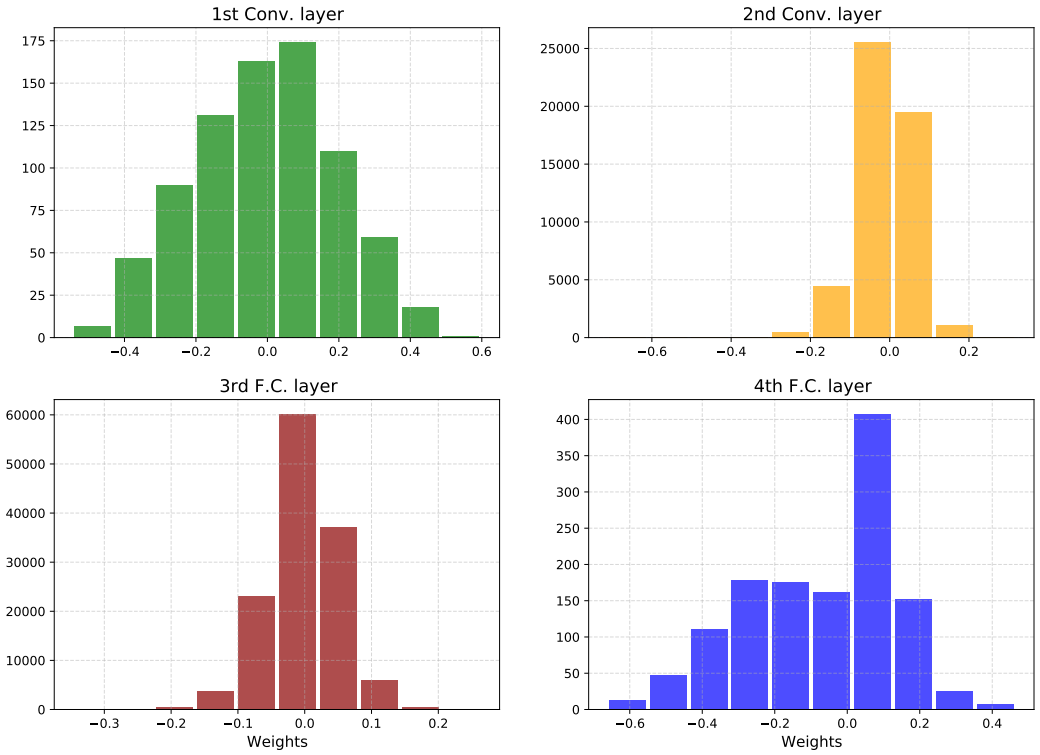

Figure 11: *Classification:* Weight histograms for the various hidden layers and the output layer.

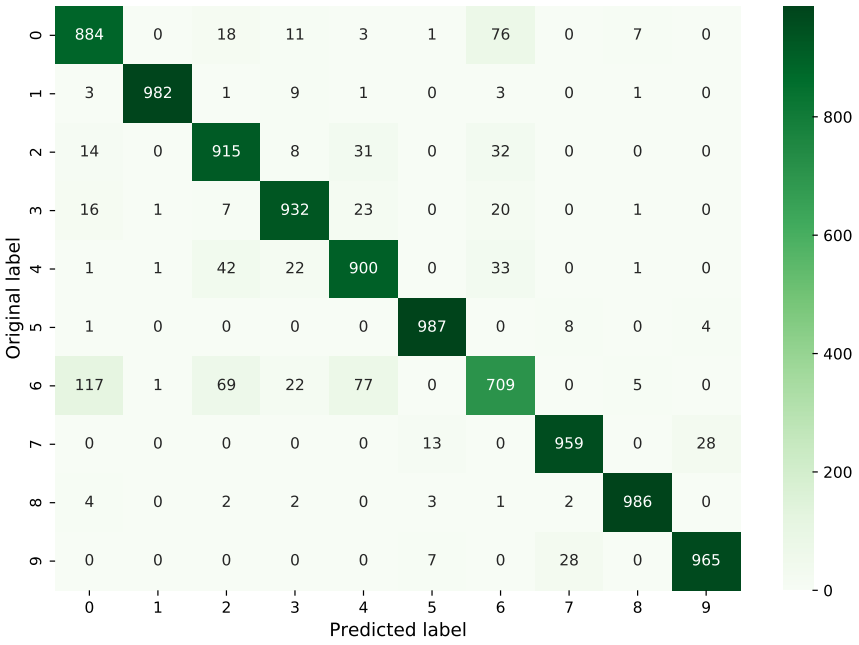

Figure 12: *Classification:* Confusion matrix of the classification task with the last model.