Homework 1 - Supervised Deep Learning

Michele Puppin ID 1227474 michele.puppin@studenti.unipd.it

(Dated: August 26, 2021)

I. INTRODUCTION

In this homework we will implement and test simple neural network models for solving supervised problems. Firstly, we will consider a **regression task** that consists in a simple function approximation problem and we will use a deep feed-forward neural network. Secondly, we will consider a **classification task** that consists in a simple image recognition problem, where the goal is to correctly classify images of handwritten digits (MNIST). Here we will use a convolutional neural network. In particular, advanced optimizers and regularization methods will be implemented and learning hyperparametwrs will be tuned using appropriate search procedures.

These models are implemented in Python using the PyTorch framework.

II. REGRESSION TASK

1. Methods

In the regression task, the goal is to train a neural network to approximate an unknown function $f : \mathbb{R} \to \mathbb{R}$:

$$x \to y = f(x),$$

so that network$(x) \approx f(x)$.

As training set, 100 noisy measures from the target function are provided:

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

A deep feed-forward neural network is implemented. In this architecture processing units are organized in layers and every unit in a layer is connected with all the units in the previous layer, each connection is weighted. The information moves in only one direction: from the input node, through the hidden nodes and to the output node. No cycles or loops are considered in the network. This allows to exploit Back-Propagation algorithm to adjust connection weights. We use a deep network, namely a network with a number of hidden layers grater than one. Indeed, the depth of the network allows to learn the mapping of more complex functions, with respect to shallow architectures.

Specifically, the network is composed by:

- an input layer with one neuron (one dimensional input);
- a first hidden layer with N_h neurons;
- a second hidden layer with $2 \times N_h$ neurons;
- an output layer with one neuron (one dimensional output).

The number of hidden layers is limited due to the relatively simple task that the network has to tackle. In fact, reducing the number of hidden layers is an easy way to promote generalization and to reduce computational cost. Rectified Linear Unit (ReLU) activation function is choosen:

$$\text{ReLU}(x) = \max(0, x).$$

Indeed it helps preventing gradient vanishing problem, results to be computationally convenient and favours sparse activation. Moreover, use L2 regularization technique to favour generalization. We consider two optimizers: the basic Stochastic Gradient Descent (SGD) and the Adaptive Moment Estimator (Adam), that is an extension to SGD. We also consider *momentum*, which allows to speed up learning and helps not getting stuck in local minima. Furthermore, in both hidden layers, we implement dropout; this technique helps to avoid the network to rely excessively to a single neuron and promotes generalization. Finally, we make use of early-stopping technique, a form of regularization used to avoid overfitting that stop training at the point when performances on a validation dataset start to degrade.

The network structure and the techniques presented so far are characterized by parameters, that we call hyperparameters, that define the model and can be tuned in order to get optimal performances. We use the following hyperparameters:

- the number of neurons in hidden layers $N_h \in \{16, 32, 64, 128\}$;
- the learning rate $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$;
- the L2 regularization parameter $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$;
- the dropout probability $p_{drop} \in \{0.0, 0.10, 0.15, 0.20\}$;
- finally, the otimizer is choosen between SGD and Adam.

Since we use early-stopping, the number of epochs in the training procedure is not explored as hyperparameter, just a maximum number of epochs is imposed.

For the sake of model selection, due to the small size of the training set, we implement a K-fold cross validation procedure, that consists in a resampling procedure to create fresh training and validation sets. We use 4 folds. Each time we test a random combination of hyperparameters, since a full grid search would result to be too computationally expensive. Once we find the best performing hyperparameters, we run the training procedure over the whole training set and then we test the performances over the test set. We use Mean Square Error (MSE) loss function.

2. Results

The best parameters resulting from model selection using K-fold cross validation are:

- the number of neurons in hidden layers $N_h = 128$;
- the learning rate $\eta = 10^{-4}$;
- the L2 regularization parameter $\lambda = 10^{-5}$;
- the dropout probability $p_{drop} = 0.15$;
- the optimizer is Adam.

Training and validation profile for training with best hyperparameters is plotted in Fig. 1a. Early-stopping procedure stops the training before 400 epochs. The trained model is tested over the test dataset and we get a test error of 0.1278. The predictions produced for the test set are plotted in Fig. 1b. The function is characterized by two local maxima that are, on porpouse, not described by the training set. We can see that the model is able to predict accurately the first local maximum but fails in predicting the second one.

Eventually, in Fig. 2a we can observe weights histograms. We see that, thanks to the use of regularization, weights modules are always contained and never explodes. In Fig. 2b activation profiles are shown. We can see that activation is sparse and varying the input we can see that some inputs produce stronger activations.

In conclusion, the network architecture chosen is able to perform the regression task with acceptable performance. The optimization techniques implemented proved to be effective.

III. CLASSIFICATION

1. Methods

In the classification task the goal is to train a neural network to correctly classify the inputs; we use the MNIST dataset composed by images of handwritten digits from 0 to 9. Since the inputs are images, we implement a Convolutional Neural Netwrok (CNN). In this architecture, convolutional layers are used in order to abstract the image to a feature map, also called an activation map. This layers acts similarly to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only from its receptive field and shares its set of connection weights with other neurons. Thanks to this it is possible to analyze larger inputs such as high resolution images. In particular, the network is composed by:

- a first convolutional layer with one input channel (the image is in gray scale) and 16 output channels, we use a 3×3 filter (kernel) that slides over the image with a stride of 1 and a padding of 2;
- a first maximum pooling layer of 2×2;
- a second convolutional layer with 16 input channels and 32 output channels, we use a 3×3 kernel with a stride of 1 and a padding of 2;
- a second maximum pooling layer of 2×2;
- a fully connected layer with 128 neurons;
- an output layer with 10 neurons (one for each of the possible classes).

We use again dropout in the flattened output of the second maximum pooling layer. As for the regression task, we use all the techniques introduced above and we use a random grid search to tune the hyperparameters, which are the following:

- the learning rate $\eta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$;
- the L2 regularization parameter $\lambda \in \{10^{-4}, 10^{-5}, 10^{-6}\}$;
- the dropout probability $p_{drop} \in \{0.0, 0.10, 0.15\}$;
- the epoch number between $20, 40, 60$;
- finally, the otimizer is choosen between SGD and Adam.

A K-fold corss validation is not needed due to the large dataset. Once we find the best performing hyperparameters, we run the training procedure over the whole training set and then we test the performances over the test set. We choose cross entropy loss function.

Some transformations are randomly selected between the addition of Gaussian noise, image rotation and image distortion. Moreover inputs are normalized to enhance learning.

2. Results

The best parameters resulting from model selection using random grid search are:

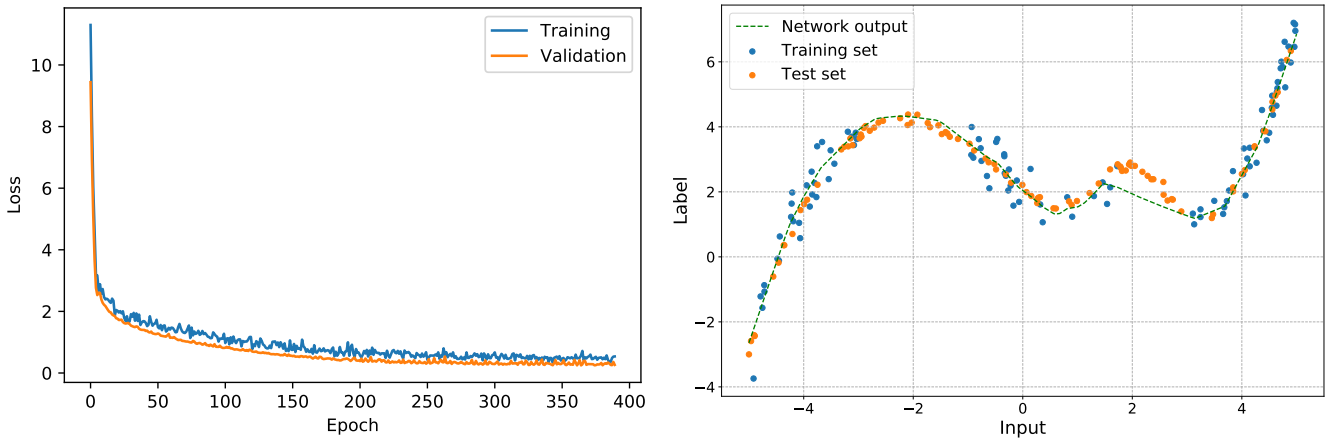
- the learning rate $\eta = 10^{-4}$;
- the L2 regularization parameter $\lambda = 10^{-4}$;
- the dropout probability $p_{drop} = 0.15$;
- the epochs number is 60
- the optimizer is Adam.

Training and validation profile for training with best hyperparameters is plotted in Fig. **3**a. Early-stopping procedure never stopped the training before the maximum number of epochs set. The resulting accuracy measured on the test dataset is 0.9927 and the confusion matrix obtained is shown in Fig. **3**b

In Fig. **4**a we can observe weights histograms. Again we see that, thanks to the use of regularization, weights modules are always contained and never explodes. In Fig. **4**b activation profiles are shown for the output layer. We that, as expected, the mostly activated neuron is the one corresponding to the correct label for the input. Eventually, in Fig. **5** we plot some examples of 3×3 filter and the corresponding activation profiles for the first convolutional layer and the second convolutional layer. From the activations we can see that in the first layer digits are still recognizable by humans, meaning that this layer encodes the shape of the digit. In the second layer instead shapes are no longer distinguishable, this layer encodes deeper features.

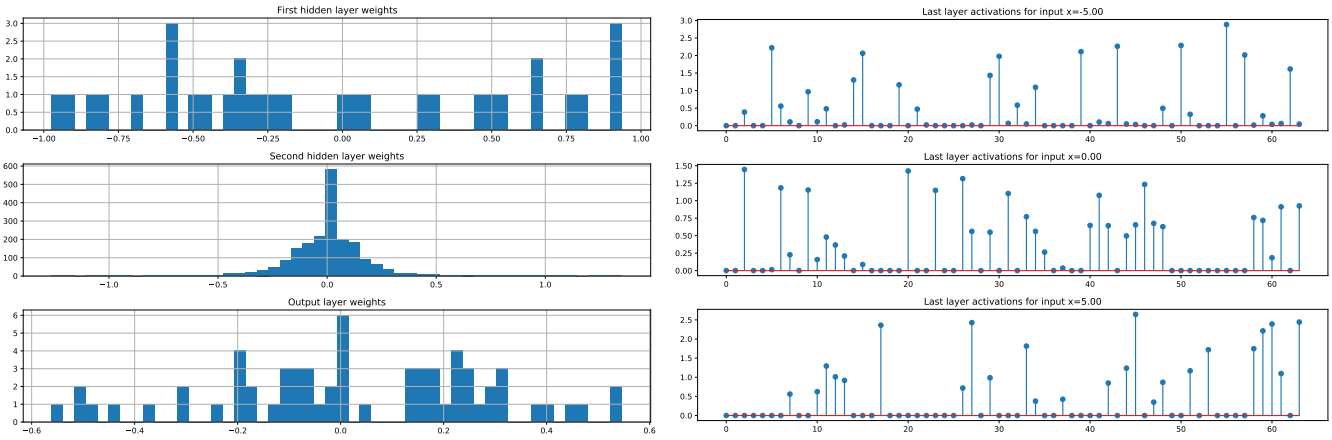
In conclusion, the chosen model is able to perform a classification task over a dataset of handwritten digits with a very good accuracy.

IV. APPENDIX



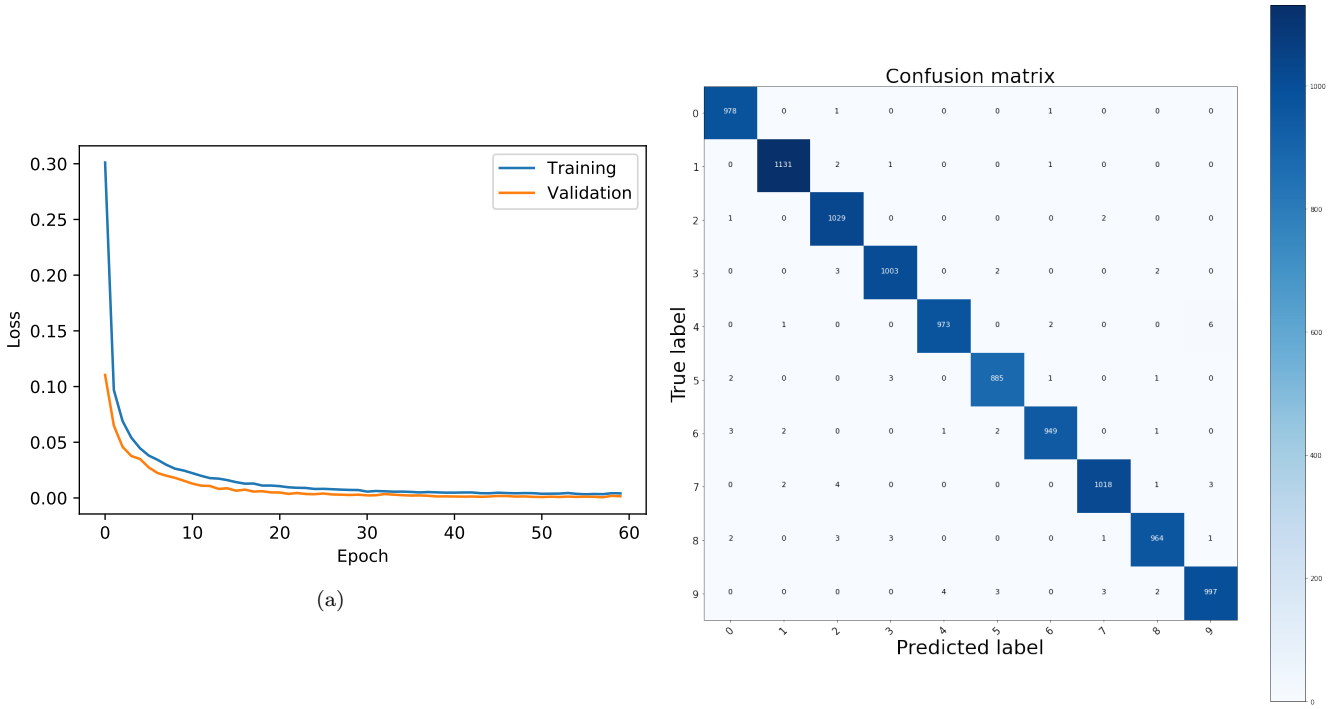
(a) (b)

FIG. 1 Regression task. (a) Training and validation loss profile for training epochs. (b) Predictions produced by the model trained with best hyperparameters selected.



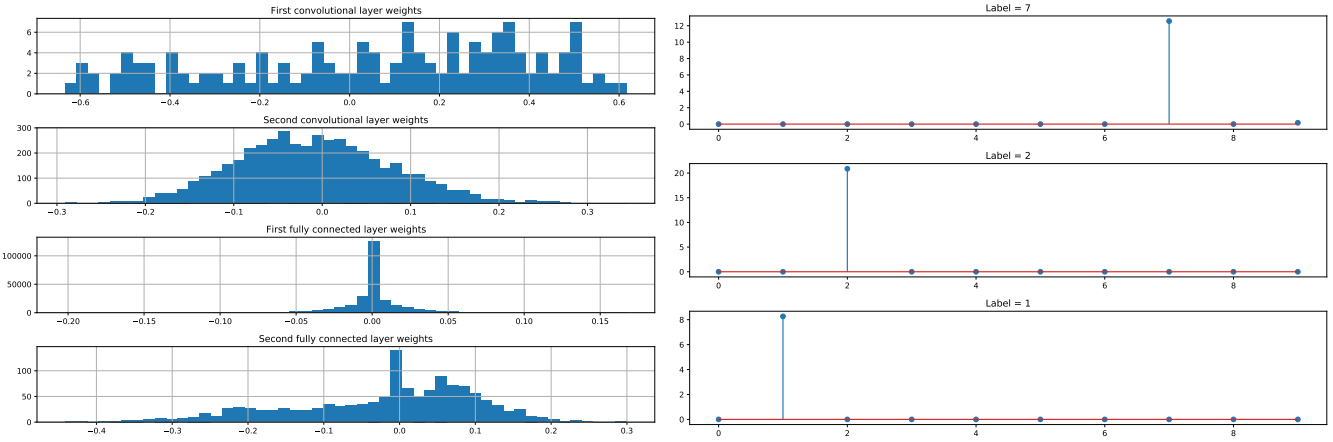
(a) (b)

FIG. 2 Regression task network analysis. (a) Weights histograms. (b) Activation profiles.



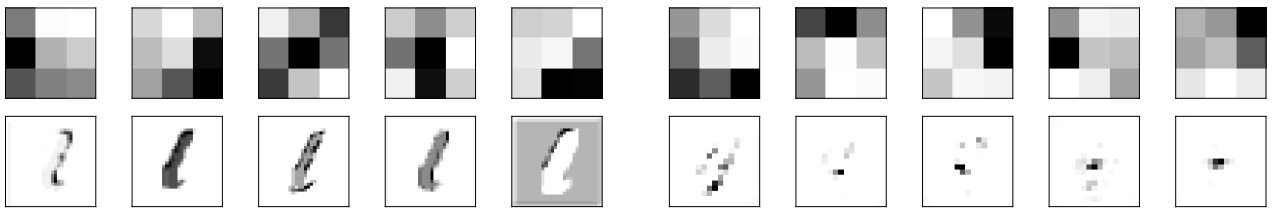
(a) (b)

FIG. 3 Classification task. (a) Training and validation loss profile for training epochs. (b) Confusion matrix for the test dataset.



(a) (b)

FIG. 4 Classification task network analysis. (a) Weights histograms. (b) Activation profiles.



(a) (b)

FIG. 5 Classification task. Examples of 3×3 filter (top) and corresponding activation profiles (bottom) for (a) first convolutional layer and (b) second convolutional layer.