SUPERVISED DEEP LEARNING

Matteo Bortoletto, 1142935

June 23, 2021

1 Introduction

In supervised learning the goal is to learn a function that maps each possible input to a certain output. To do that, we need one label associated to each input pattern. The two main forms of supervised learning are:

- *regression*, in which we train a neural network to approximate a function;
- *classification*, in which we want to correctly classify some inputs.

In this work we consider both tasks. In particular: for regression, we use a deep neural network to approximate behaviour of a function using a training set and a test set composed of 100 points; for classification, we use a convolutional neural network to perform multi-class classification of handwritten digits using the MNIST dataset.

1.1 Feed-forward deep neural networks

For the regression task we will use a deep feed-forward neural network, which is an artificial deep neural network wherein connections between the nodes do not form a cycle. Consequently, the information moves in only one direction (i.e. forward) from the input layer, through the hidden layer and to the output layer.

1.2 Convolutional neural network

For the classification task we will use a convolutional neural network (CNN), which is an architecture most commonly applied to analyze visual imagery. In fact, convolutional neural networks are able to reduce images into a form which is easier to process, without loosing critical features. A CNN is structured in the following way:

- the input is an image with a certain number of pixels and color channels (usually three, RGB);
- then we have a *feature learning* part, which is composed of convolutional and pooling layers. In a convolutional layer a *filter* (also called kernel) shifts over the image with a certain *stride* length, every time performing a matrix multiplication with the portion of image which is covering. This allows to extract the high-level feature of the image. To keep the original dimension of the image we can use *padding*, adding layers of zeros to the image. In the pooling layer the size of the convolution is reduced. In this work we will use MaxPooling, keeping the maximum value for each patch of the feature map.
- finally, we have the *classification* part, in which the final output of the feature learning part is flattened and fed to a fully connected neural network.

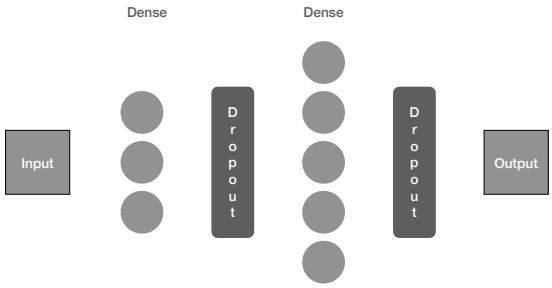


Figure 1: Feed Forward Neural Network architecture used for the regression task.

2 Methods

2.1 Regression

Our network is composed of the following layers:

1. An input layer with one neuron (corresponding to the x value);
2. A hidden layer with N_h neurons;
3. A second hidden layer with $2N_h$ hidden layers;
4. An output layer with one neuron (corresponding to the y value).

A scheme of the architecture is shown in Figure 1. In order to avoid gradient vanishing, we use a Rectified Linear Unit (ReLU) activation function:

$$\text{ReLU}(x) = \max(0, x) \tag{1}$$

The main advantages of using this kind of activation function is that it is computationally efficient thanks to its particular form. Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply using comparisons, additions and multiplications. It also contributes to sparse activation. Finally, the ReLU guarantees a better gradient propagation with respect to the sigmoid, for example.
The optimizers we consider are two: the Stochastic Gradient Descent (SGD) with momentum and the Adaptive Moment Estimator (Adam). We choose a L2 regularization, adding a term in the loss that penalizes the norm of weights and biases.
The hyper-parameters that we want to optimize are the dropout probability, the learning rate, the number of hidden units for the first layer, the optimizer and the L2 regularization value. In order to do that, we use a *cross validation* procedure with $k = 4$ folds. This allows us to better exploit the data, since it is quite scarce. Furthermore, even if training is fast, we use *early stopping*, stopping the training if the validation loss does not decrease sufficiently in a certain number of consecutive epochs (called patience). In this way, if a network does not perform well because it has a poor initialization in terms of hyper-parameters, it will be discarded in a fast way.
After finding the best parameters, we train the model using the full training set.

2.2 Classification

The network for classification is composed of the following layers:

- a first convolutional layer with 1 input channel (because the input is a gray-scale image), 16 output channels, size of the convolving kernel 3×3, stride 1 and padding 2;

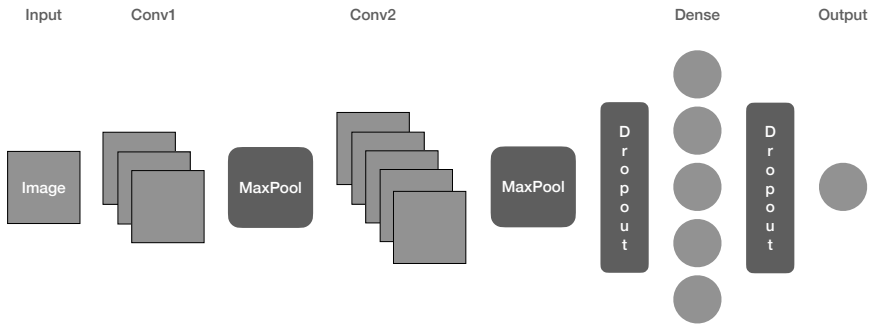


Figure 2: Convolutional Neural Network architecture used for the classification task.

- a max pooling layer of size 2×2;
- A second convolutional layer with 16 input channels, 32 output channels, size of the convolving kernel 3×3, stride 1 and padding 2;
- another max pooling layer of size 2×2;
- a fully connected layer with 128 neurons;
- an output layer connected layer with 10 neurons.

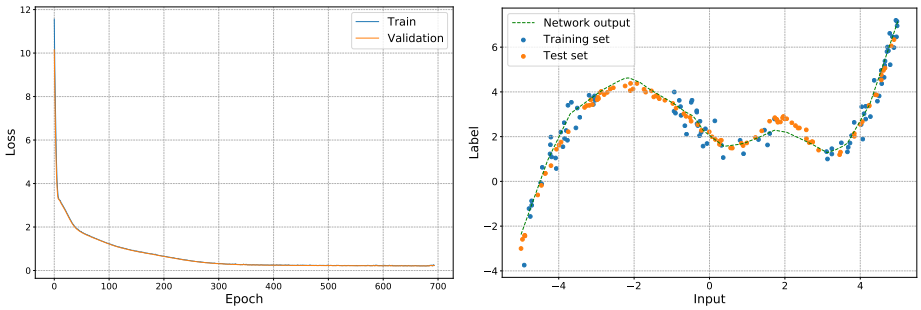
In the passage from the second max pooling layer to the fully connected layer, the output is flattened and dropout is applied (if set to a value different from zero). The activation function we use is, again, the ReLU. A scheme of the network is shown in Figure 2.
The CNN has 10 output neurons, one for each MNIST digit (from 0 to 9). The "predicted digit" is the index corresponding to the maximum one.
Unlike the regression case, here the main problem is that the training time is quite long. To overcome this problem, the early stopping procedure is crucial.
The hyper-parameters we want to optimize are the dropout, the number of epochs (even if we use early stopping, we still use it as a parameter), the learning rate, the optimizer (SGD or Adam) and the L2 regularization value. Given the fact that the training time is long, we find the best values with a random search (instead of a cross validation procedure).
The data is normalized and we also apply some transformations, like a rotation of the image and the addition to some gaussian noise.

3 Results

3.1 Regression

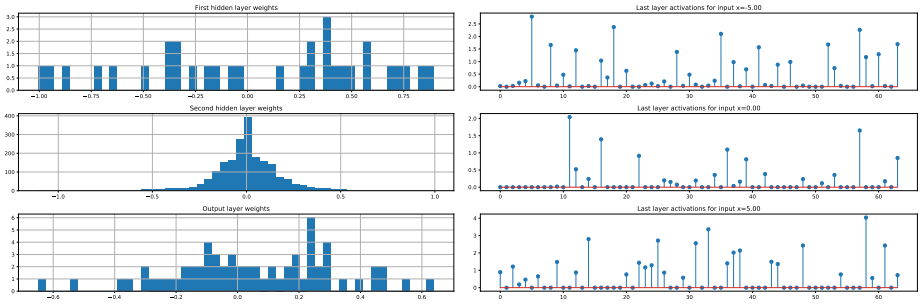
The results of the cross validation procedure are the following:

- Dropout probability: 0;
- Learning rate: 0.001;
- Number of hidden units for the first layer: 32;
- Optimizer: Adam;
- L2 regularization value: 10^{-5}.



(a) Training and validation losses for best parameters. (b) Training points, test points and model prediction.

Figure 3: Network performances.



(a) Weights histograms. (b) Activation profiles.

Figure 4: Network analysis for the regression task.

The plot of train and validation losses is shown in Figure 3a, while the performances on the test set are shown in Figure 3b (green dashed line). The test loss is approximately 0.105.
The peculiarity of the training set is that it leaves out the two local maxima that we see in the test set around $x = -2$ and $x = 2$. As we can see from Figure 3b, the network is able to the first maximum quite well, but the second one is a bit lower.
Figure 4a shows the histogram of the weights. As we can see, the weights module is always reasonably small. Figure 4b shows the activation profiles. Notice that there are groups of neurons that are always activated and other groups that only respond to specific parts of the curve.

3.2 Classification

The results of the random search are the following:

- Dropout probability: 0.2;
- Number of epochs: 50;
- Learning rate: 0.0001;
- Optimizer: Adam;
- L2 regularization value: 10^{-5}.

The plot of train and validation losses is shown in Figure 5. The accuracy on the test set is very good, and amounts to 0.9927.
Figure 6a shows the histogram of the weights and tells us that the weights are reasonably small. Figure 6b shows the activation profiles. Notice that the activations are consistent with the labels.

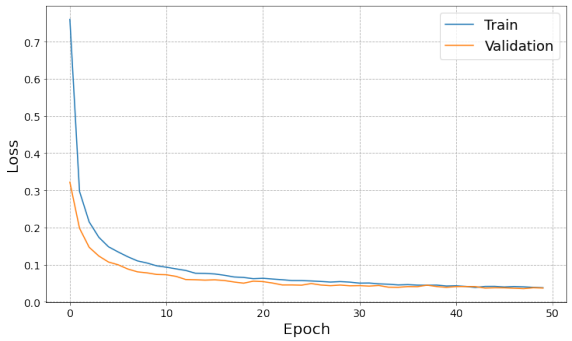
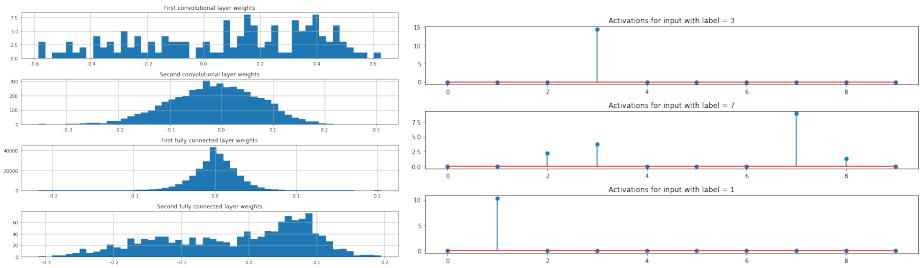


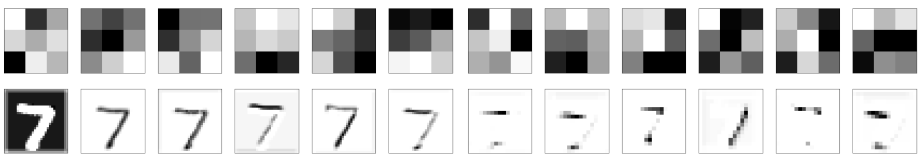
Figure 5: Classification loss function.



(a) Weights histograms. (b) Activation profiles.

Figure 6: Network analysis for the classification task.

Figure 7 shows some of the filters and the activation profiles of the two convolutional layers. From Figure 7a we can see very clearly that the result is the digit we give as input. If we look at Figure 7b we see that the result is less clear. It seems that the second filter focuses more on the borders of the digit, acting like a kind of "edge-detector".



(a) First convolutional layer. (b) Second convolutional layer.

Figure 7: Some filters and corresponding activations for the two convolutional layers.