Supervised Deep Learning

Marco Ballarin 1228022
marco.ballarin.6@studenti.unipd.it

Abstract

In this report we will review the regression task on a polynomial function of unknown order and a classification task on the MNIST dataset. We will review with particular attention the use of optimizers, regularization methods, hyperparameters optimization and feature visualization for the networks. The code related to this work will be provided as an attachment and as a link to a github repository in the Appendix.

1 Introduction

In this report we will tackle two different problems using the supervised deep learning framework. This means that we will have datasets D formed of couples $\{(x_i, y_i)\}_{i=1,\dots,N}$, where x_i are the samples and y_i the labels. The report will be divided into four main Sections:

1. Introduction, where we introduce our work;
2. Regression, where through a deep neural network we try to predict the behavior of a curve;
3. Classification, where through a convolutional neural network we try to classify the handwritten digits from 0 to 9 using the MNSIT dataset;
4. Appendix, where we will add images that are not fundamental for the flow of the report, but may be useful.

2 Regression

2.1 Methods

For the regression task it is better to use a reduced number of hidden layers, and in particular we choose to use 2 of them. Our network will so be composed of:

- An input layer of one neuron;
- A first hidden layer oh N_h neurons;
- A second hidden layer of $2 \cdot N_h$ neurons;
- An output layer of one neuron.

To avoid vanishing problems in the gradient we will use a rectified linear unit (ReLU) as activation function. This simple topology in the network is due to the fact that this problem is "simple" from the neural networks point of view, and adding too many parameters we will be easily led to overfitting. We will so make an extensive use of regularization procedures to avoid these problems. In particular, we will use two main methods:

1. dropout, which consists of randomly eliminate neurons during the training with probability p_{drop}.
2. L2 regularization, which consists of adding a penalty in the loss proportional to the 2-norm of the parameters, i.e. weights and biases. We stress that in pytorch this method is enabled not in the loss function but in the optimizer, using the `weight_decay` keyword.

We will try two different optimizers:

1. The stochastic gradient descent with momentum, which is the really basic algorithm for optimization. The addition of the momentum, i.e. the memory of previous steps, really helps to go out of local minima and speeds up convergence;

2. the Adam algorithm, adaptive moment estimator, a more advanced algorithm that uses estimations of first and second moments of the gradient to adapt the learning rate of each weight of the neural network.

The amount of data available for this task is quite limited, we will so apply a cross-validation technique to perform the hyper parameters tuning. This method consists of splitting the training test into k different folds and then, for each hyper parameters set, train the network on the union of $k-1$ folds, performing the validation on the one which is missing. This approach ensure also better performances, since the validation set is different for each of the iterations, and so we cannot end up with a model that perfectly fits one given validation set. After the hyper parameters choice is done, we will train the model over the full training test.

We will then optimize the number of hidden units in the layers of the network, the learning rate and the number of epochs needed for the training procedure.

Lastly, we will visualize the network features: in particular we will plot the weights histograms and the activation profiles.

2.2 Results

The best performing network is reported in Figure 1.

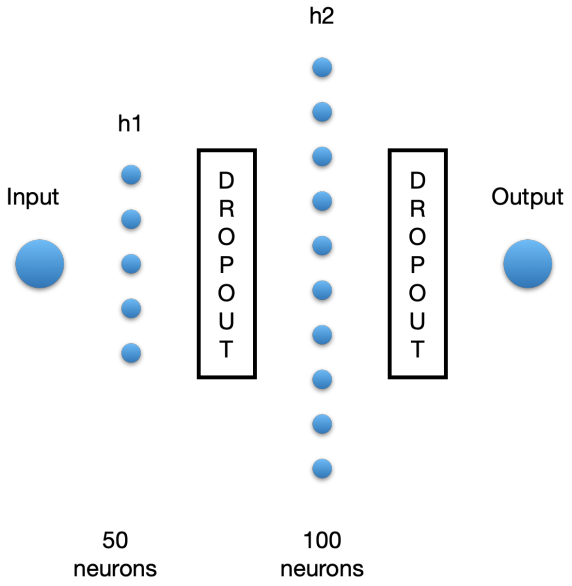

Figure 1: Final network for the regression task.

3 Classification

3.1 Methods

In the classification task we have to develop a network which classifies the MNIST dataset, i.e. the dataset containing handwritten digits from 0 to 9. This dataset is quite exhaustive, and so we will not need to apply the Cross-validation technique. The size of this dataset present nevertheless another challenge: the training time can become too long. For this reason we really need to finely optimize the number of epochs we use in this procedure. One possible solution would be, as in the previous case, use them as a hyper parameter. However, we will waste a lot of computational

resources in this search. We so decided to apply a technique that is called early stopping: if the validation loss l_v does not diminish sufficiently in a given number of epochs then we kill the training loop. In this way we can get rid of poorly initialized networks and have an overall speed up of the process. In particular, calling $\vec{l}^{(t)} = (l_v^{(1)}, l_v^{(2)}, \dots, l_v^{(t)})$ the vector containing all the losses at iteration t, with a maximum number of iteration T we stop the process if:

$$\bar{l} - l_v^{(t)} < 0.01 \quad \text{with} \begin{cases} \bar{l} = \frac{1}{t-\tau} \sum_{i=\tau}^{t} l_v^{(i)} \\ \tau = \min(t, T/10) \end{cases} \tag{1}$$

The other main difference with respect to the regression task is that the dataset is composed by images. For this reason we decided to use two convolutional layers at the beginning of the network. These layers make use of filters, i.e. feature maps that are applied along all the image with the same weights. We will use filters of dimension 3×3. This means that we will several small 3×3 feature maps that will enhance the same details along all the figure. The remarkable example in this framework are filters which enhance edges. We will analyze them in Section 3.2.

Through a random search we will optimize the following parameters:

- The dropout;
- The learning rate;
- The optimizer, choosing from stochastic gradient descent and Adam;
- the strength of the L2 regularization,

To further enhance the dataset we will apply a random rotation of 45ř and some Gaussian noise. To help the network in the learning phase we will also normalize the inputs, using the apposite pytorch transformation.

3.2 Results

4 Appendix