Homework 2

Anna Braghetto - 1205200

20/11/2019

NOTE: the model is trained **without transposing the images.**

Introduction

The aim of the proposed homework is to build, train and test a feedforward neural network with PyTorch that performs the handwritten digit classification. In particular, the structure and, in general, the hyperparameters of the considered neural network, are looked for by using the random search, implemented with the k-fold cross validation. Then, a smart method to establish when learning should stop is implemented.

Since, the needed computational power to perform the random search with many configurations is too high to use *CPU*, everything is perfomed with CUDA in *Google Colaboratory*.

Dataset

The MNIST dataset contains 60000 rotated and reflexed images, representing different handwritten digits, with the corresponding labels, from 0 to 9.

The dataset, stored in a *MATLAB* file, is loaded in the program and divided into training (80%) and test sets (20%) by using the function *train_test_split* provided by *sklearn.model_selection*. Furthermore, in order to assure that the separation into the two different sets is the same at different calls of the function, the seed is set to *1234*.

Model Search

Many architectures and hyperparameters are tested in order to define the best model for the classification of handwritten digits. Due to the high number of parameters, the search of the best model is carried out by performing a random search with a k-fold cross validation.

In particular, the activation function used is *Leaky ReLU* and the training is performed using as loss function the *Cross Entropy Loss* and the *Adam Optmizer*, whose weight decay, for L_2 regularization, is set to $5 \cdot 10^{-4}$. The early stopping permits to avoid the overfitting of the model by supervising the behaviour of the validation loss: if it increases for many consecutive epochs, the training stops.

Random Search

The hyperparameters of the models, tested with the random search, are the following:

- **Number of hidden layers**
  Three different number of hidden layers are considered: 1, 2 and 3.

- **Number of neurons per layer**
  The number of neurons per layers are left to vary from 100 to 600. Furthermore, in order to consider different architectures, two different way to initilialize the number of neurons are defined.

  [1] The number of neurons decreases with the depth of the network: the maximum value is chosen among *[300,400,500,600]* by using the random choice provided by numpy, while the minimum is set to *100*. Then, a *step* value is computed as $\frac{\text{max}-\text{min}}{\text{Number of hidden layers}}$. With the assigned *max* and *max-step*, the number of neurons is chosen between the two for the first layer; then, for the following layers, the two quantities iteratively decrease each by *step*.

  [2] The number of neurons is constant with the depth of the network: the maximum value is again chosen from the list *[300,400,500,600]* and the minimum is set to *max-100*. The number of neurons is chosen between max and min for all the layers.

- **Learning Rate**
  The learning rate is chosen among *[0.005,0.001,0.0005,0.0001]*. Furthermore, in order to reach the best configuration, a learning rate decay is considered. In particular the *ReduceLROnPlateau* provided by PyTorch is used: it permits to decay the learning rate by a an arbitrary factor γ when the validation loss does not decrease. The parameters tuned for this scheduler are the factor γ and a *cooldown*, which corresponds to the number of epochs to wait before applying again the scheduler itself.

- **Factor γ**
  Three different factors γ are considered: 0.5, 0.7, 0.9.

- **Cooldown**
  Three different numbers of epochs for *cooldown* are considered: 0, 50 and 100. In particular the lower values are used to observe how the scheduler works for fast decrease of learning rate while the greater ones analysed the behaviour of lazy decrease.[1]

Many configurations are defined by performing a random choice over all the possible values for each parameter defined above. Then, the k-fold cross validation is performed in order find the best configuration.

Cross Validation

The k-fold cross validation is performed on the training set, considering 50 different configurations.

The training set is divided into 4 folds: for each fold the neural network is trained in the union of the other 5 and validated in the selected one. The average validation loss is an estimate of the error for the selected hyperparameters and the best structure corresponds to the model with the least average validation loss.

The best configurations is defined by:

Architecture	Learning Rate	Factor γ	Cooldown
(784, 480, 132, 10)	0.001	0.9	0

Table 1: Best model.

Final Training

The final training is perfomed for the best parameters. In order to avoid overfitting and to use the scheduler learning rate decay, the training set is divided into two subsets: one training itself (80% of the data) and one to validate the model (20% of the data).

The behaviour of the validation loss is shown in Fig. 1.

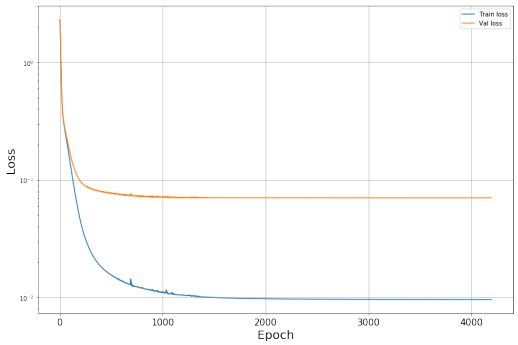


Figure 1: Behaviour of the validation loss of the final training. It is possible to observe an initial peaks due to an high initial learning rate: the scheduler used for the learning rate decay is essentially to reach the convergence, otherwise the oscillatory behaviour would be present up the final epoch. Furthermore, the learning stops before the total number of epochs (10000), thanks to the early stopping.

The accuracies obtained in training and in the test set are 99.6% and 97,9% respectevely. Furthermore, the confusion matrices for both the sets are shown in Fig. 2.

Receptive Fields

The receptive field of some hidden neurons are determined and then shown for the layers.

[1] By using an high number of epochs for the cooldown, it is observed an oscillatory behaviour of the validation loss.

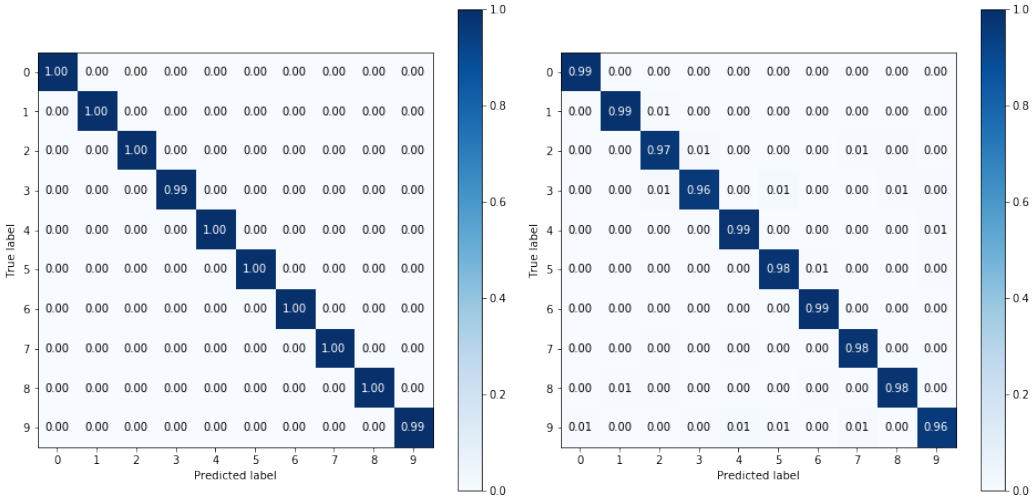


Figure 2: Confusion matrices for training (on the left) and test (on the right) sets. It is possible to observe that, for the training set, the percentage of well classified digits is really high (99%) instead, for the test set, it decreases: since the difference is lower than 3% , the model is not overfitting.

Projection of the weights

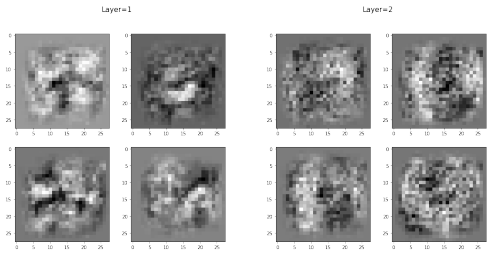
Considering the best configuration, the receptive field for each layer (hiddens and output) are computed thanks to the weights of the neural network, as follows.

$$RF_{1,j} = W_{1,j} \qquad RF_{2,j} = W_{2,j}W_1 \qquad RF_{3,j} = W_{3,j}W_2W_1$$

where W_i is the matrix of weights that connect the layer (i-1)-th to layer i-th and $W_{i,j}$ is the vector of weights that connect the layer (i-1)-th to the neuron j$-th$ in layer i$-th$.

In Fig. 3 it is possible to observe the receptive field for 4 random neurons for the two hidden layers.

Figure 3: Receptive field for 4 random neurons in the first (on the left) and the second (on the right) hidden layer, respectevely. It is really hard to understand what the neurons actually learn: single patterns are not highlithed in the images. This is probably due to the fact that the regularization used to train the model is L_2 and not the sparsity L_1, that would have improved the feature visualization.



In Fig. 4 it is, instead, possible to observe the receptive field for the 10 neurons of the output layer.

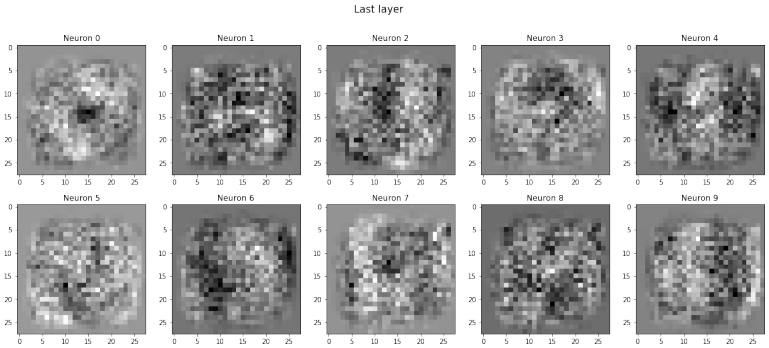


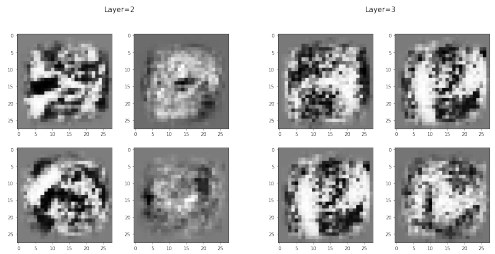
Figure 4: Receptive field for the 10 neurons in the output layer. In the last layer it is expected that the features learned by the neurons have a pattern similar to the different digits: for the first neuron (number 0), in fact, it is possible to recognize the shape of the number 0 (black circle in the center of the receptive field.) but, for the others, it is really difficult to highlight the shapes of the relative numbers.

Advanced Feature Visualization

An improvement for the feature visualization is performed by perform thee gradient descent over the image pixels to maximize the neuron's activation.

In Fig. 5 it is possible to observe the results obtained for 4 random neurons in the two hidden layers.

Figure 5: Receptive field for 4 random hidden neurons in the first (on the left) and the layer (on the right), respectevely. As observed in Fig 3, it is really hard to understand what the neurons actually learn. Neverthless, comparing the two blocks of images, it is possible to notice how the constrast is highlighted in the advances method: this characteristic permits to underline better the features.



In Fig. 6 it is, instead, possible to observe the results for the 10 neurons of the output layer.

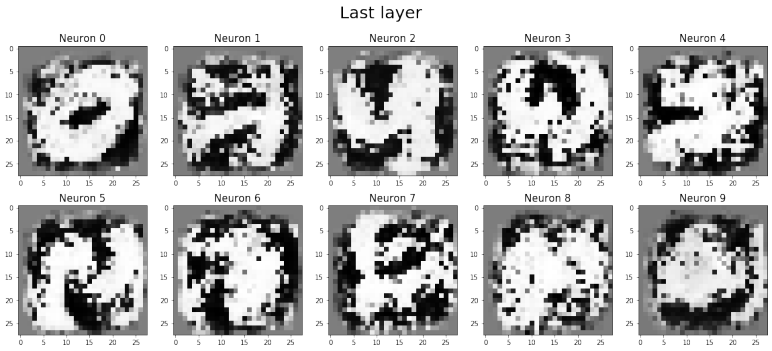


Figure 6: Receptive field for the 10 neurons in the output layer. Comparing these images with the ones in Fig. 4, it is possible to observe how the shapes of the digits are more visible: not only the shape for the number 0 is highlighted but it is also possible to recognize well the digit 2, 3, 5 and 7.

The more advanced method for feature visualization is actually working better than the *reprojection of the weight* performed above.