Neural Networks and Deep Learning :

Homework 1: Supervised deep learning

Regression + Classification

Prepared by : Norhen Abdennadher

Prof : Dr. Alberto Testolin

Academic year 2021-2022

I. Overview:

This homework is about learning how to implement and test simple neural network models for solving supervised problems. It is divided in two tasks.

- **Regression task**: the regression model will consist in a simple function approximation problem.
- **Classification task**: the classification model will consist in a simple image recognition problem, where the goal is to correctly classify images of handwritten digits (MNIST).

We will first discover the regression dataset and explain the solution provided for the regression problem. Then, we will explore the famous handwritten MNIST handwritten digits dataset and build a classifier for it.

We will use Pytorch for these two tasks.

II. Regression task:

Regression predictive modeling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). A continuous output variable is a real-value, such as an integer or floating-point value.

Our dataset consists of 100 data points of only one feature and the values are between -5 and 5 for the x coordinate and between -4 and 8 for the y coordinate.

	input	label
count	100.000000	100.000000
mean	0.535661	2.826761
std	3.511726	1.962990
min	-4.915863	-3.742970
25%	-3.255942	1.677844
50%	0.037716	2.663394
75%	4.143882	3.727911
max	4.977516	7.199304

Figure1: Dataset details

We can visualize these 100 data point and we can clearly see the distribution of our small dataset in figure2.

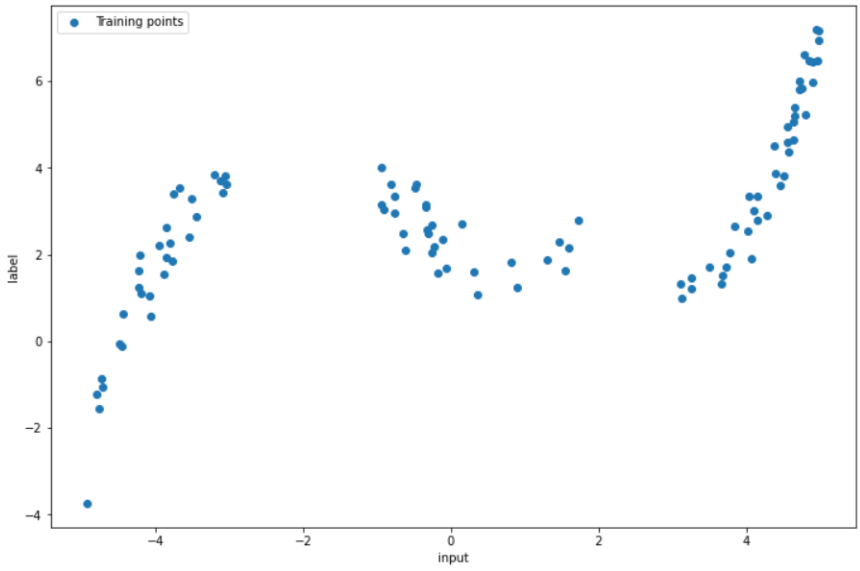


Figure2: Training data visualization

We need to define the used network for the regression task, it is composed of:

- Two fully connected linear layers, first one of size 128 and the second of size 256.
- An output layer, followed by the sigmoid activation function.
- Adam optimizer
- Learning rate of 0.01
- MSE loss

To get out the best of our model and to make sure that isn't overfitting, we used cross validation with 3 folds while training the network.

The dataset in this case is divided into 3 folds:

- **Model1**: Trained on Fold1 + Fold2, Validated on Fold3 : Training loss=0.2429488 and validation loss=0.4173703
- **Model2**: Trained on Fold2 + Fold3, Validated on Fold1 : Training loss=0.13003038 and Validation loss= 0.58391374
- **Model3**: Trained on Fold1 + Fold3, Validated on Fold2 : Training loss=0.1681982 and Validation loss=0.36142215

We can pick the best model which is the 3rd model, below we can see the training and validation loss variation for the 3rd model:

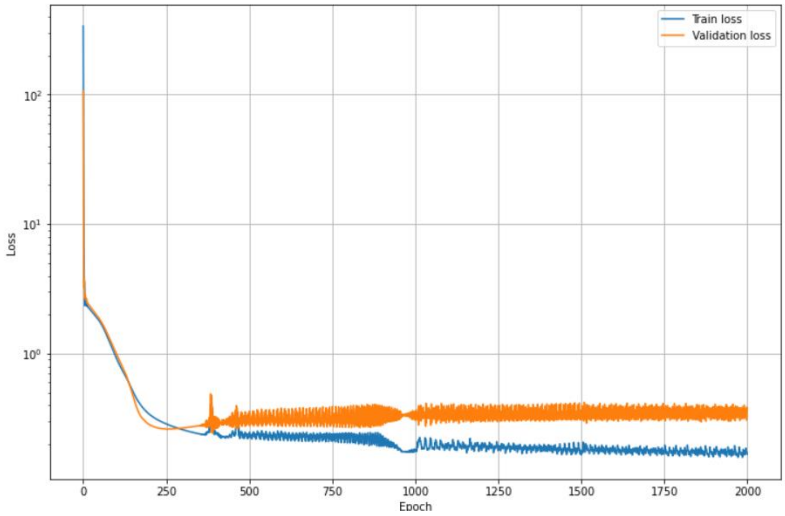


Figure3: Training and validation loss for 3rd fold

We can also visualize the original distribution and the predicted points while using the weights of the 3rd model:

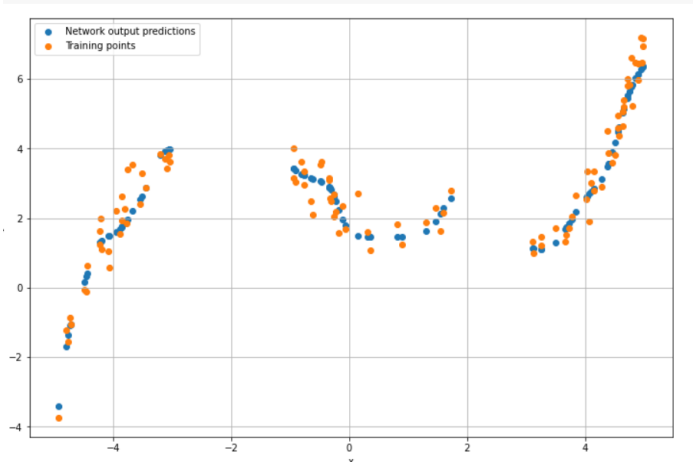


Figure4: Training points and network output predictions

The testing set is composed of 100 data points:

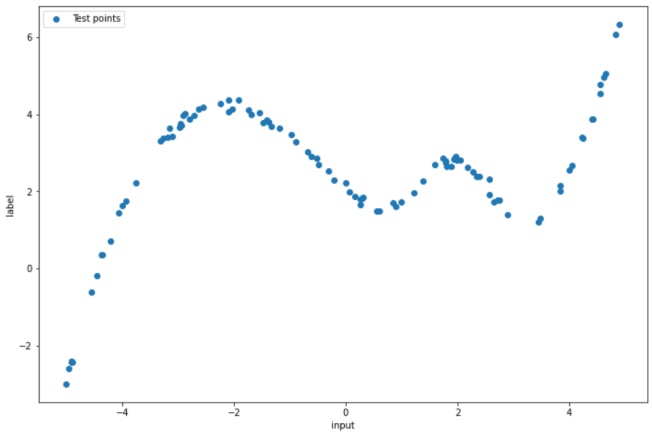


Figure5: Test dataset

We tested the 3 obtained models on the testing set and we measured the loss for each of them. The obtained results are:

```
### MODEL 1 ####
AVERAGE TEST LOSS: 0.20464715361595154
Network initialized
### MODEL 2 ####
AVERAGE TEST LOSS: 0.26183021068573
Network initialized
### MODEL 3 ####
AVERAGE TEST LOSS: 0.12449510395526886

Best model is 3 with less training loss
```

In order to pick the number of units for each hidden layer and over hyperparameters such as learning rate, we used grid search on the following hyper parameters with 3-fold validation:

- Number of units for first hidden layer: 64,128
- Number of units for second hidden layer: 128,256
- Learning rate: 0.001, 0.01

We defined the grid search space, and we did the training on all combinations:

```
Space=[(64,128,0.001),(128,256,0.001),(64,128,0.01),(128,256,0.01)]
```

Training loss 1st combination: 0.6651532

Validation loss 1st combination: 0.37044778

Training loss 2nd combination: 0.37973607

Validation loss 2nd combination: 0.4255147

Training loss 3rd combination: 0.14336367

Validation loss 3rd combination: 0.3561332

Training loss 4th combination: 0.17349274

Validation loss 4th combination: 0.37197986

So the best combination is:

- Number of units for first hidden layer: 64
- Number of units for second hidden layer: 128
- Learning rate: 0.01

And the test loss is : 0.102

III. Classification task:

In this part we'll build a simple convolutional neural network using PyTorch and train it to recognize handwritten digits using the MNIST dataset.

MNIST handwritten digits dataset contains 70,000 images of handwritten digits: 60,000 for training and 10,000 for testing. The images are grayscale, 28x28 pixels, and centered to reduce preprocessing and get started quicker.

We used k-fold cross validation technique to test the model in the training phase in order to validate the fact that our model isn't overfitting and whether it will generalize its capabilities on unseen dataset or not. The training set is divided into 10 folds and at each time we are going to select only one validation set for testing and keep the 10 left partitions for the training process.

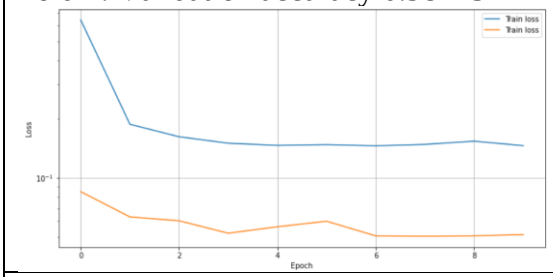
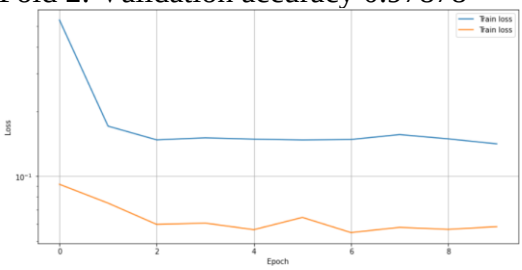
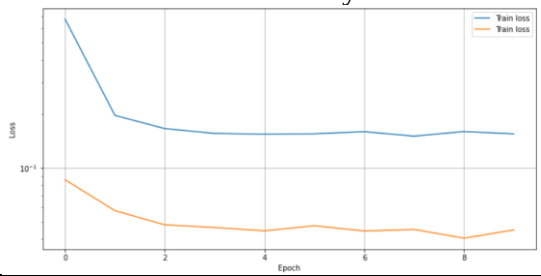
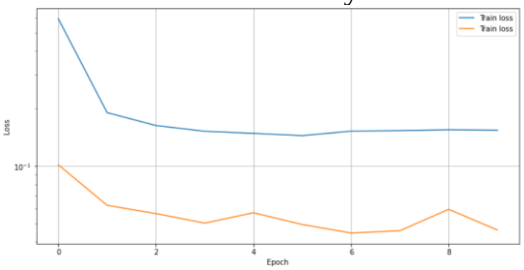
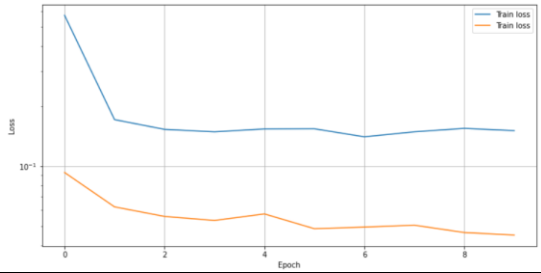
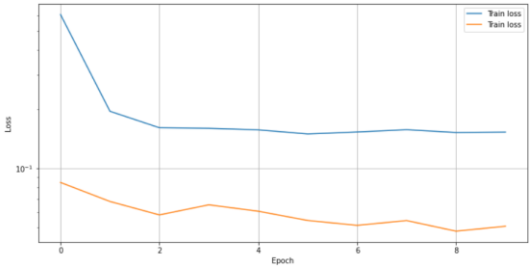
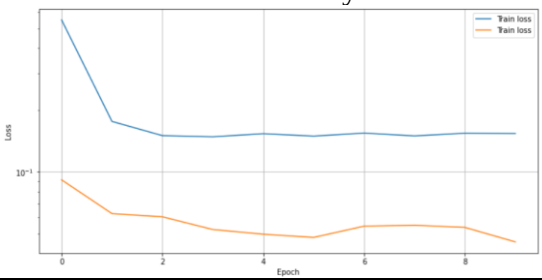
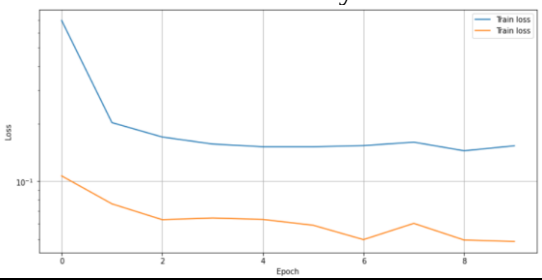
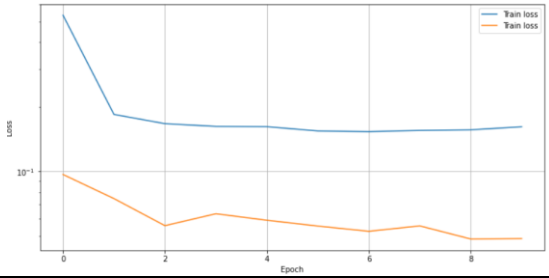
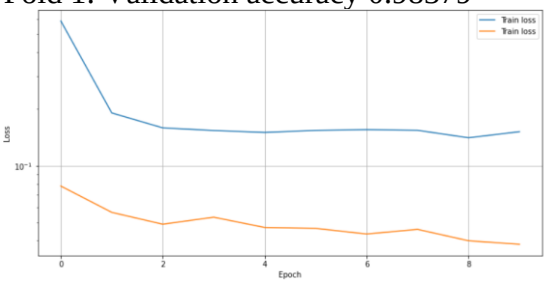
Now let's go ahead and build our network. We'll use three convolutional layers followed by two fully connected (or linear) layers.

- Three convolutional layers with kernel size equal to 5, with ReLU as activation function and followed by a dropout layer as a means of regularization.
- A fully connected layer followed by rectified linear units ReLU and a dropout.
- An output fully connected layer followed by Softmax as activation function.

In PyTorch, a nice way to build a network is by creating a new class for the network we wish to build. The forward() pass defines the way we compute our output using the given layers and functions.

Note: While using a GPU for training, we should send the network parameters to the GPU using. It is important to transfer the network's parameters to the appropriate device before passing them to the optimizer, otherwise the optimizer will not be able to keep track of them in the right way.

During the training, we'll keep track of the progress with some printouts. In order to create a nice training curve during the training process, we also created lists for saving training and validation losses with the cross validation. The table reports the obtained results:

Fold 1: Validation accuracy 0.98125 	Fold 2: Validation accuracy 0.97878 
Fold 1: Validation accuracy 0.98387 	Fold 1: Validation accuracy 0.98068 
Fold 1: Validation accuracy 0.98186 	Fold 1: Validation accuracy 0.98041 
Fold 1: Validation accuracy 0.98179 	Fold 1: Validation accuracy 0.97895 
Fold 1: Validation accuracy 0.98131 	Fold 1: Validation accuracy 0.98379 

We kept track of the testing loss and accuracy of each model:

```
Result for model1
Average test loss: 0.041622698307037354
Model Accuracy:0.984%
Result for model2
Average test loss: 0.04018806666135788
Model Accuracy:0.983%
Result for model3
Average test loss: 0.03984498232603073
Model Accuracy:0.984%
Result for model4
Average test loss: 0.044787995517253876
Model Accuracy:0.982%
Result for model5
Average test loss: 0.04010253772139549
Model Accuracy:0.985%
Result for model6
Average test loss: 0.038741447031497955
Model Accuracy:0.984%
Result for model7
Average test loss: 0.037113092839717865
Model Accuracy:0.985%
Result for model8
Average test loss: 0.0419965535402298
Model Accuracy:0.983%
Result for model9
Average test loss: 0.03924034163355827
Model Accuracy:0.983%
Result for model10
Average test loss: 0.03375449404120445
Model Accuracy:0.986%
```

Model 10 is the best one in terms of accuracy and generalization capabilities.

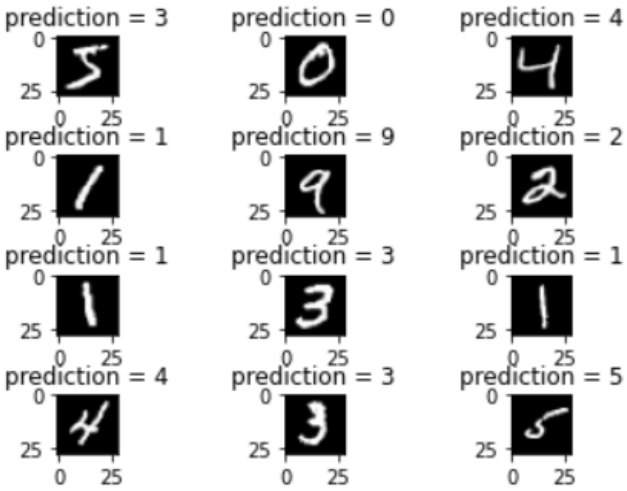


Figure 6: Correctly predicted samples using Model 10

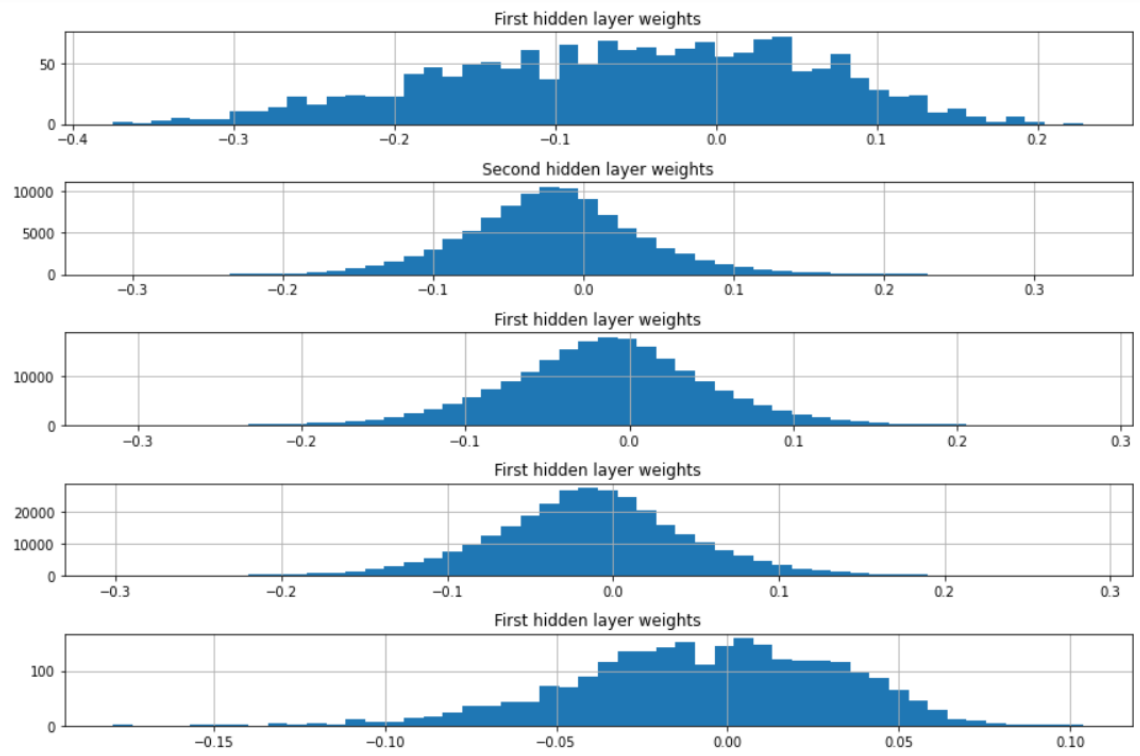


Figure 7: weights histograms for 10th model

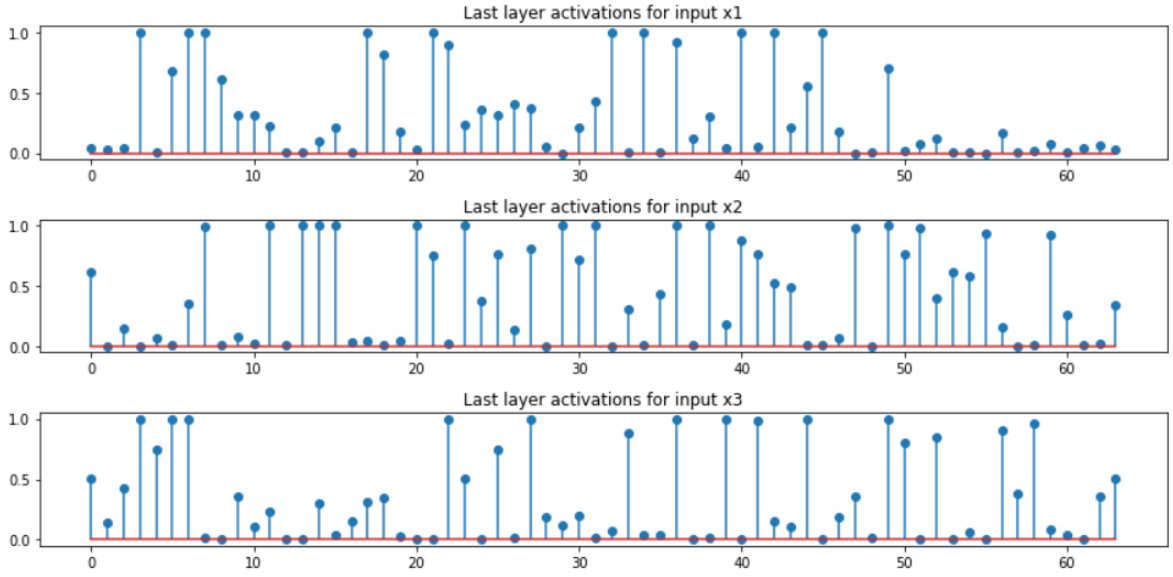


Figure 8: Last layer activation profile for 3 different inputs