Neural Networks and Deep Learning
Homework 1: Supervised Deep Learning

Giulio Zani

February, 2021

Contents

1 Regression Task

1.1 Introduction

For this task, we were given two files, one containin the train dataset and the other one the test datset. Both files containing a list of (x, y) couples.

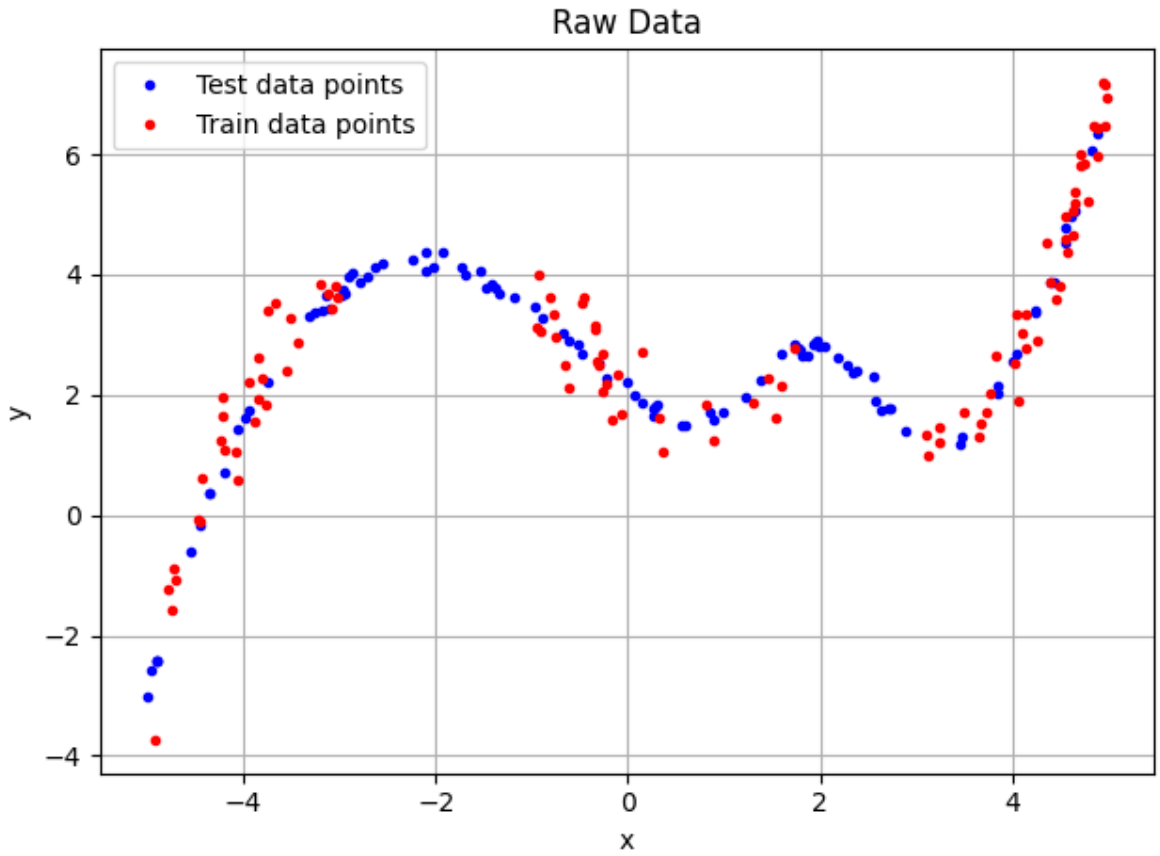

Figure 1: Raw data given for regression task

As one may notice from the figure, the challenging part of task consists of ensuring that the model correctly guesses the function in those regions not represented by train data points (red in Figure??)

1.2 Method

For this task I have tried out a variety of settings by hand. This is the network structure:

```
Net(
  (fc1): Linear(in_features=1, out_features=100, bias=True)
  (fc3): Linear(in_features=100, out_features=1, bias=True)
)
```

I have used a sigmoid activation function, without activation function in the output layer since this is a regression task. As an optimizer I have used vanilla stochastic gradient descend algorithm (SGD). I have used a cross-validation training setup. Given that training for this task didn't take long, I decided to progressively keep track of the test loss as well (Figure**??**)

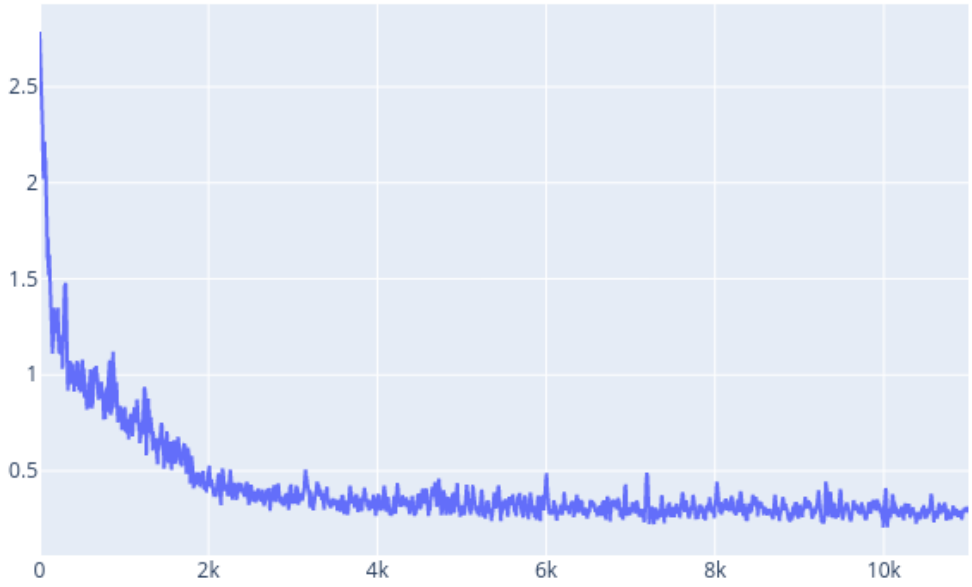

Figure 2: Validation loss as a function of the epoch (smoothed)

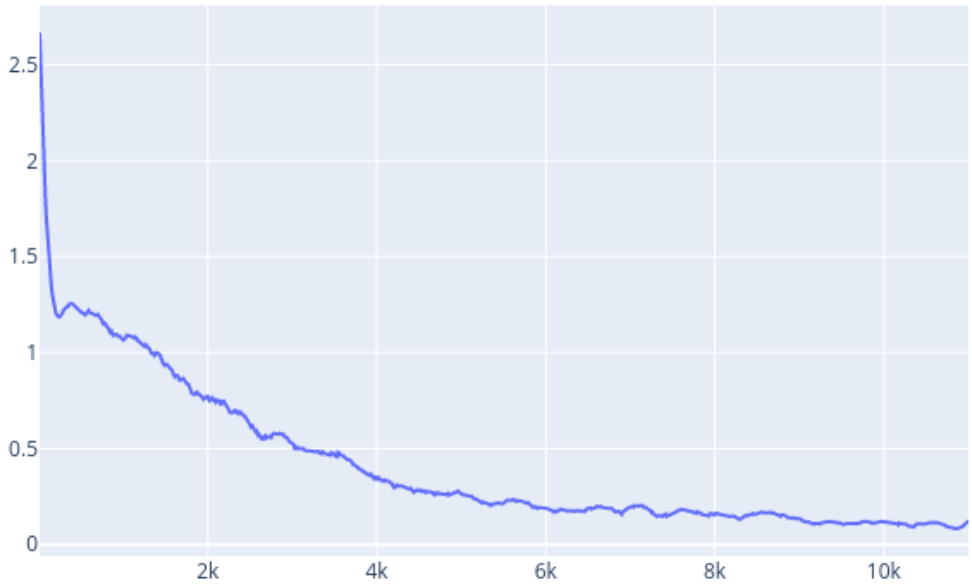

Figure 3: Test loss a function of the epoch (smoothed)

As one can see validation loss and test loss are both (roughly) monotonically decreasing, which is a good sign that the model is not overfitting. The model was trained for 1200 epochs, I have picked this number simply because I have observed that training for longer did not improve loss. The cross-validation is of 10 folds and a batch size of 9. As a loss funciton I have picked MSELoss, which implements mean square error (squared L2 norm) between each element in the input x and target y. Code can be run with the following command:

```
python -m deep_learning_hw1.regression_task
```

1.3 Result

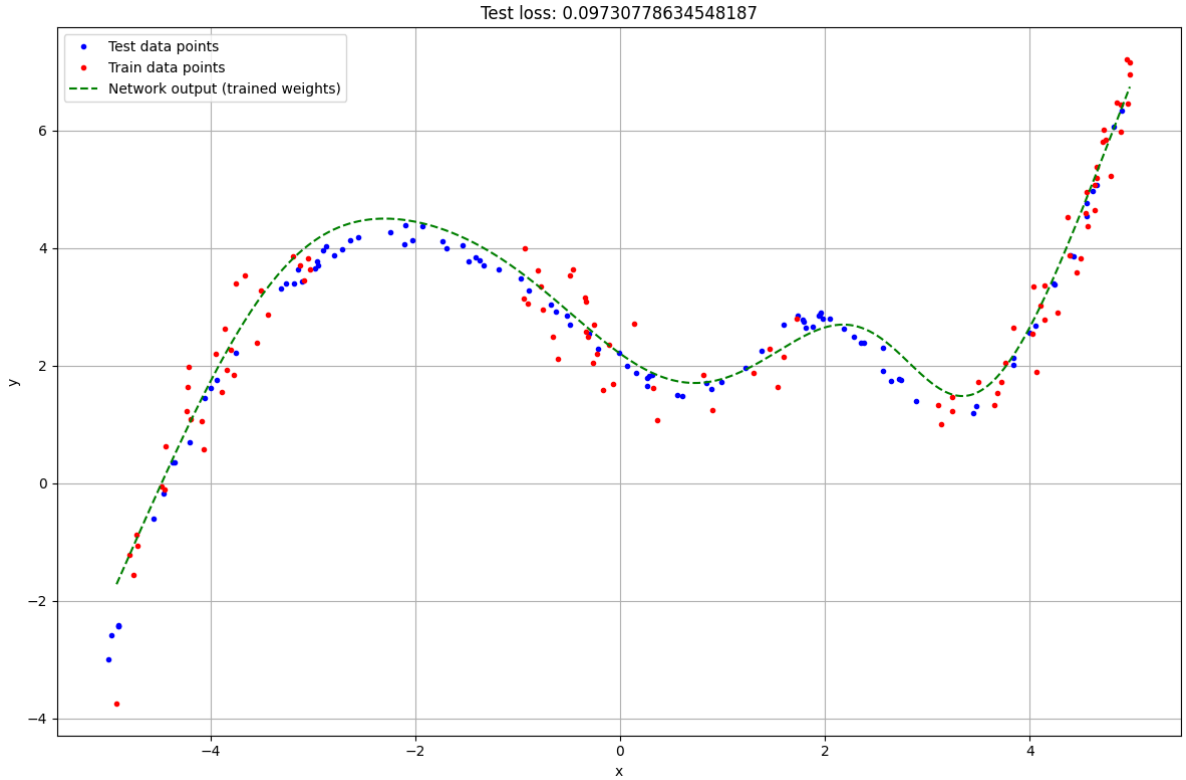

Figure 4: Trained model plotted against raw data

The mean final test loss was of 0.097. As one can see from Figure?? the model has approximated pretty well the function even in those regions that were not included in the training set.

2 Classification Task

2.1 Introduction

In this taks we were given a dataset (MNIST) composed of images of handwritten digits. The deep model's objective is to learn to correctly classify the image to the corresponding image. Images are grayscale, 28x28 pixel.

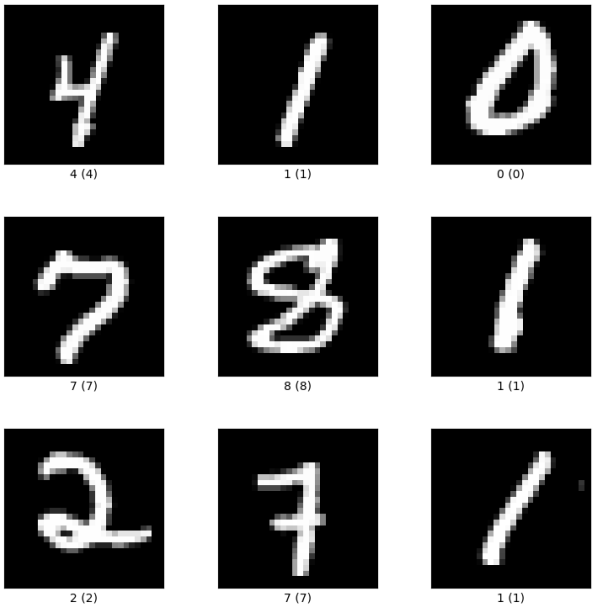

Figure 5: Raw data given for regression task

While the regression task can be better solved via methods that don't use deep learning, such as polynomial regression, this one canonically requires deep models.

2.2 Method

Due to the relative small size of the input images, a simple, fully connected feed-farward ANN would suffice to solve the task pretty well. However I wanted to experiment with convolutional neural networks (CNNs). I have started out with a tutorial I have found on the internet.

```
Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

I mainly fine-tuned this model by hand. I chose the dropout of 0.25 because it seems like this is a canonical value for CNNs. I have used ReLU activation function. I choose to use Adadelta optimizer because I have seen it in a tutorial, so I read about it and decided to used it. I have found it to be beneficial with respect to vanilla SGD.

2.2.1 Optimizing Hyperparameters with Genetic Algorithm

Genetic algorihms (GAs) are widely used to optimize hyperparameters, network structure and, more recently, deep learning algorihms. I have implemented a very simple tools to optimize hyperparameters using a GA. The algorithm performs both crossover and mutation. I meant to build this as a tool which I would use also in other exercises (such as homework 2), therefore, the tool doesn't assume anything about the nature of the problem, rather, it delegates the implementation of a few functions, namely: "mutate", "get random genome" and "get fitness". In this particular problem to compute an organism's fitness, I have run a training for one epoch with the given genome/parameters and returned the test loss (which actually would be the fitness' inverse). I am a little concerned about whether computing the fitness based only of a one-epoch training could introduce a bias

in hyperparameters search, however I had no choice due to limited hardware availability. The initial population size is of 30, and I have run this for 28 generations. The hyperparameters over wich I run it were the following:

$$\text{batch size} \in \{10, ..., 128\}$$

$$\text{lr} \in [1e-8, 1.0]$$

$$\text{gamma} \in [0, 1.0]$$

$$\text{weight decay} \in [1e-7, 1e-4]$$

The evolved hyperparameters are:

$$\text{batch size} = 10$$

$$\text{lr} = 0.419$$

$$\text{gamma} = 0.8366$$

$$\text{weight decay} = 7.511239e-05$$

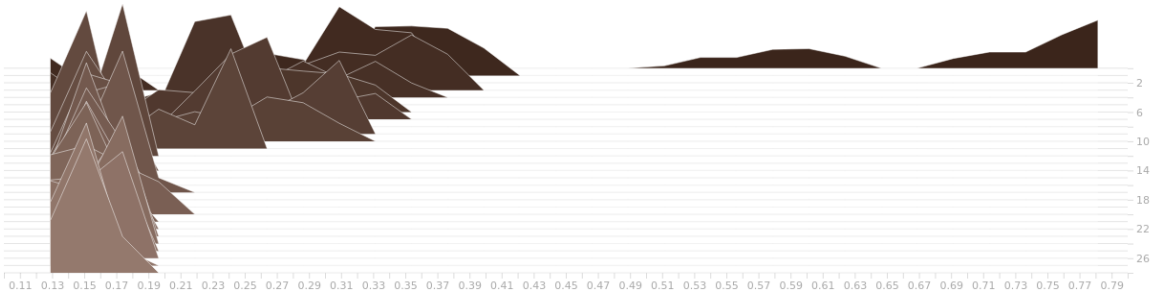

Figure 6: Distribution of 30% best individuals after each generation. One-epoch loss on y axis.

With more time and computational resources it would have been interesting to explore network structure optimization.

2.2.2 Training and network analysis

Here I have traced the loss during training. In the second plot I have taken the test set because my first version was not using a (cross)validation set.

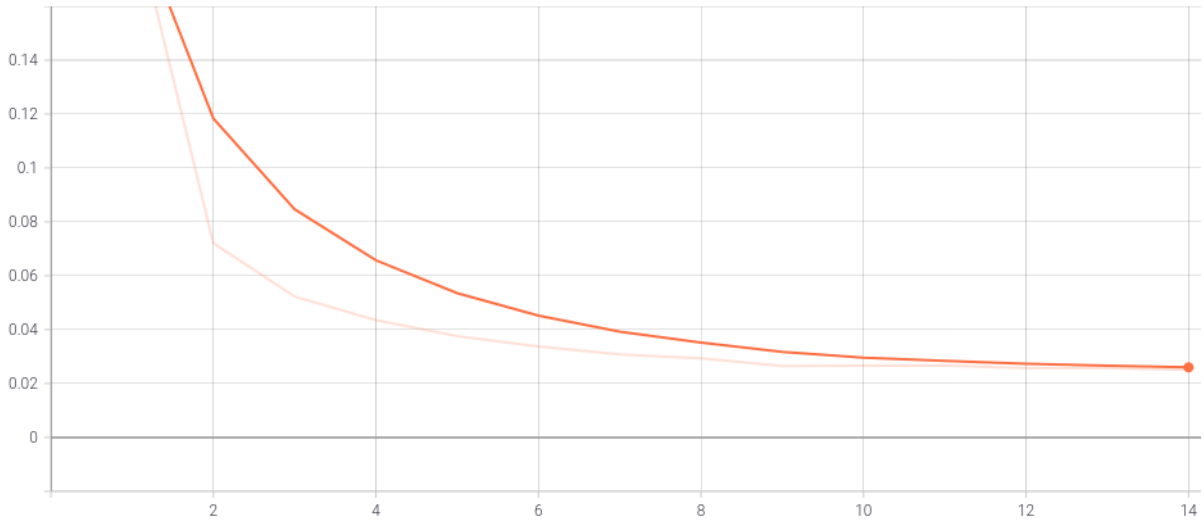

Figure 7: Training loss as a function of the epoch

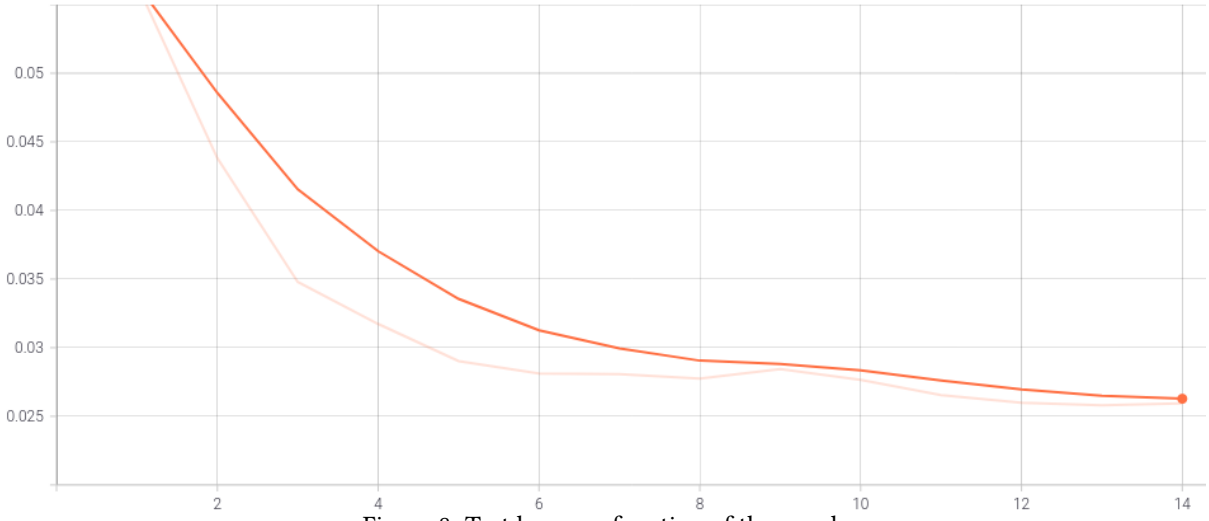

Figure 8: Test loss as a function of the epoch

I have used a pytorch hook to plot the activation of the firs conolutional layer. And with tensorboard I have plotted its weights distribution over the epoch.

Figure 9: Activation of network's first convolutional layer (one image per channel)

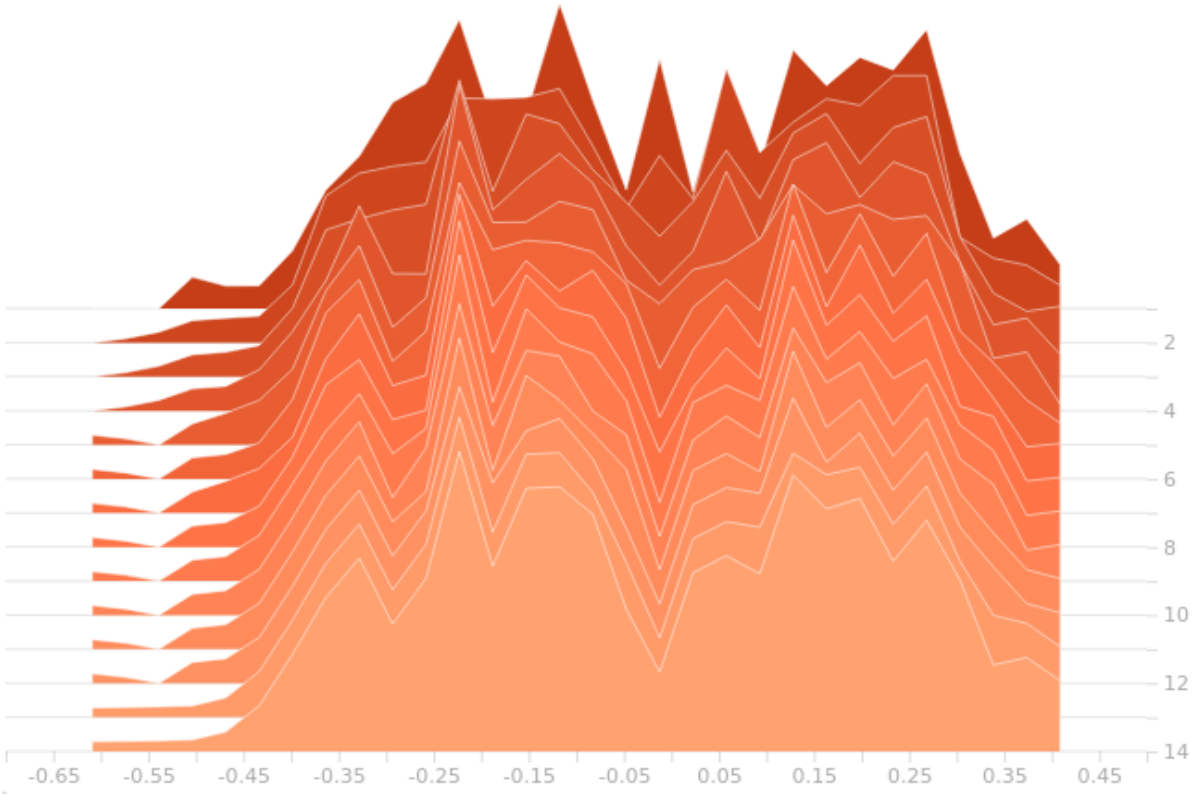

Figure 10: First convolutional layer weights distribution at every epoch

2.3 Result

In the final test set the average loss was of 0.0268 and the accuracy of 9917/10000=99.17%