Neural Networks and Deep Learning

Homework I, Simone Trevisan, 1238612

December 29, 2020

1 Regression task

The aim of this exercise is to build a feed forward neural network able to learn a given unknown function from a set of noisy samples. The training set includes 100 samples: the presence of 'dark' intervals makes the task harder for a machine learning algorithm, but we can take advantages from our human brain that excels on finding patterns for low dimensional data. The test set is also of 100 samples, but they are much less noisy.

1.1 Layout

Given a trivial task, it might be more interesting to find the most efficient model rather than a huge model with unnecessary high performances. Thus, a big model having three hidden layers with 128 neurons is generated, then the capacity is gradually decreased as long as the output matches our expectation. This strategy is applicable only due to the low dimensional of the problem, even with a 3-D function it might be much more challenging for our brain to predict the function. The process ended up with two hidden layers with 16 and 64 neurons.

The ReLu activation function speedup the training and avoid saturation, but it suffers a discontinuity in 0 and it results in a 'segmented' output trajectory. Instead, a Scaled Exponential Linear Unit (SeLu, fig.1), is used in the first layer. in the second layer the sigmoid function turned out to work good.

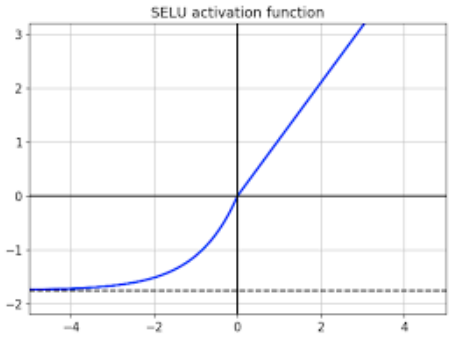


Figure 1: Scaled Exponential Linear Unit activation function

The final model has only 1k parameters, that is a very low value for a deep neural network.

1.2 Optimization and generalization methods

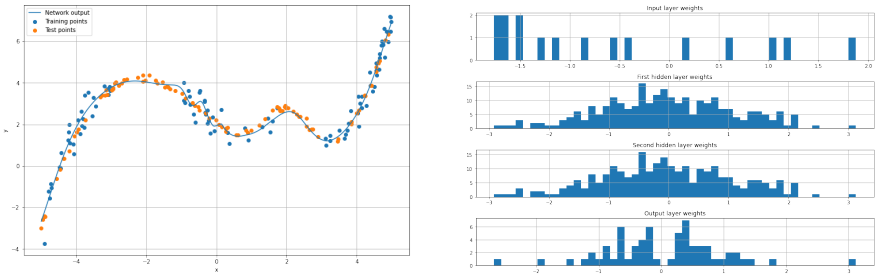
The noise in the training data can easily lead to overfitting, thus generalization strategies are applied. First, it is possible to keep weights small using L2-norm to penalize big ones. The amount of penalization is tuned together with other hyperparameters. To improve generalization and speedup the training, input samples are arranged in mini-batches.

Hyperparameters (learning rate, weight decay and mini-batch size) are firstly tuned using a random search and comparing the result versus human prediction, then they are refined using k-fold cross validation with k=5. The values taken into account are $lr = [5e-4, 4e-4, 6e-4]$, $wd = [13e-4, 15e-4, 17e-4]$, $bs = [50, 100]$. For each combination of hyperparameters, 5 neural network are trained (each with a different validation set). The minimum MSE reached by each network is averaged to give the configuration performance. The winning values are lr=4e-4, wd=15e-4, and bs=50.

1.3 Results

The final model is trained using the whole training dataset for 50000 epochs. The training loss is 0.19 while on the test set (less noisy), the network achieve

a MSE of 0.38. In fig.2a the model output is compared versus the training and test samples: it is able to reconstruct the 'dark' intervals, but between -1 and 0 it is overfitting the training data. Also, from fig.2b, it is possible to conclude that L2 norm failed to shrink the weights around 0. Thus more can be done from generalization point of view.



(a) Output (line) vs Training samples (blue dot) vs Test samples (orange dot).

(b) Weights distributions among layers .

Figure 2

2 Classification task

The aim of this exercise is to develop a neural network able to recognize handwritten digits between zero and nine. The dataset used for training, validation and test is a customized MNIST dataset of 70.000 digits. Layout, data preprocessing, generalization methods, hyperparameters tuning and results will be analyzed in the following sections.

2.1 Data preprocessing

The dataset was divided in a training set of 48.000 samples, a validation set of 12.000 samples, and a test set of 10.000 samples. To enhance the generalization of the algorithm and avoid the risk of overfitting, a data augmentation was performed: a copy of the train set was randomly rotated, translated and sheared. Finally, images are normalized with $mean = 0.1307, sd = 0.3081$, which is known from literature to provide better results.

Finally, the 96.000 samples training set is ready for the processing.

2.2 Layout

To perform image classification, Convolutional Neural Networks (CNN) are widely used in the literature due to their good performance. The input is a 28x28 pixels image representing the digit and the output in a convolutional layer can be seen as the outputs of neurons, everyone looking at its own small portion (kernel size) of the entire picture.

Three different level of depth are trained and tested versus the validation set: as show in fig.3, the configurations with three and two hidden layers performs are very similar, thus the less computational expensive is selected.

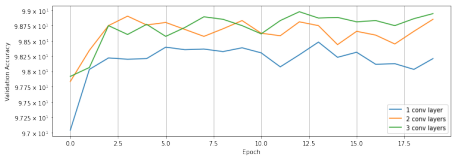


Figure 3: Comparison between different depths for CNN model

The number of convolutional filters is tuned as well, exploring different solutions. This time there are not remarkable differences (fig.4) between the solutions. The selected model has 32 filters in the first hidden layer and 64 in the second one.

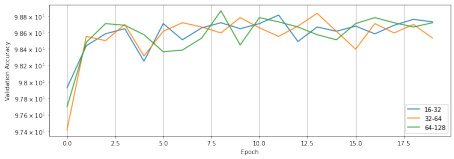


Figure 4: Comparison between different number of filters for CNN layers

The features extraction step is followed by a classification step, performed by a fully connected layer. Among the tested number of neurons, the larger one turned out to give better results (fig.5).

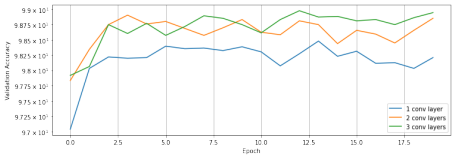


Figure 5: Comparison between different number of neurons for dense layer

The ReLU activation function is used at each layer but the last one, where a logaritmic softmax function is use to classify the digits. The loss is computed using negative log-likelihood (log-softmax + NLLLoss = Cross Entropy Loss).

The final model has around 3 million trainable parameters.

2.3 Optimization and generalization methods

To avoid overfitting and find more generic features, generalization methods are applied. First, dropout is implemented by pytorch function to the last convolutional layer. In figure 6, the results from three values of dropout probability are shown.

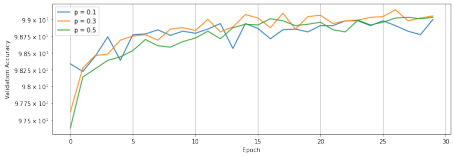


Figure 6: Comparison between different dropout probabilities

To speed up the learning and, again, generalize the network, data are collected in batches, which are used as input. Different batches size are tried (16,64,128) and the analysis shown that size=128 is the best solution.

To avoid unbalanced input due to high weights and to speedup the training, batch normalization is applied after each layer (results in fig.7. Also, in literature it is suggested that using stride=2 instead of a max pool layer, could increase the capacity of the model making the downsampling process learnable. It is tested in this network but it degraded the performance (fig.8).

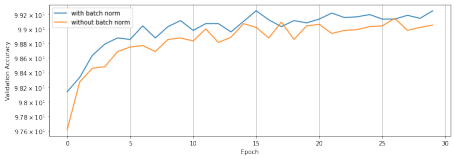


Figure 7: Comparison between with and without batch normalization

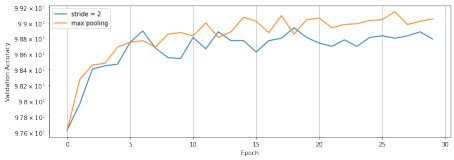


Figure 8: Comparison between stride=2 and max pooling

2.4 Results

Results are obtained running the algorithm on a Google CoLab backend with GPU.

An high accuracy is reached after few epochs (fig. 9), then the algorithm continues its learning generalizing the model thanks to the large dataset. After 50 epochs the model is tested and it reaches an overall accuracy around 99.3%.

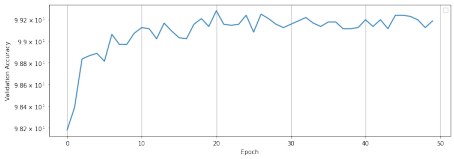
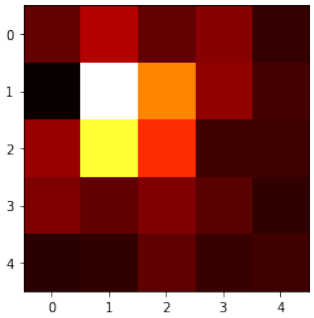
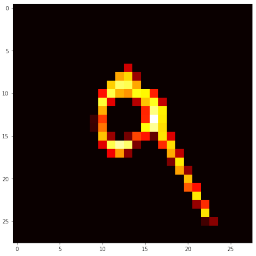


Figure 9: Accuracy values over the epochs

Methods to explore the codified features and neurons activation are also implemented for the convolutional layers. In figure 10a, a filter from the first layer is shown. In figure 10b the activation corresponding to the digit '9' is shown.



(a) Sample filter from the first convolutional layer



(b) Convolution output + activation of sample filter

Figure 10