Sapienza University of Rome

Optimization Methods for Machine Learning

Homework 1

Ivan Bergonzani, Michele Cipriano

December 2, 2017

1 Introduction

The aim of the homework is to train and compare different neural networks on a regression problem. In particular, the regression task is performed on the Franke's function building a dataset from it by sampling 100 random points (x^i, y^i) with noise, i.e. with $y^i = F(x^i) + \varepsilon^i$ where ε^i is a random number in $[-10^{-1}, 10^{-1}]$ and F is the Franke's function. The dataset has been split into a training set (70% of the dataset) and a test set (the remaining 30%).

Two architectures have been compared using different training methods and different hyperparameters. In particular, the multi-layer perceptron (MLP) and the radial basis function network (RBFN) have been trained with full minimization, two blocks method and decomposition method. In the full minimization, the training error is minimized w.r.t all the weights using the gradient descent algorithm. In the two blocks method, for the MLP the error is minimized via an extreme learning procedure and for the RBFN the error is minized by first selecting the centers through a clustering algorithm and then by solving a linear least squares problem. In the decomposition method the error is minimized by alternating a convex minimization w.r.t. the output weights and a non-convex minimization w.r.t. all the other weights.

The best result has been obtained with TODO, which has an error of **TODO** on the test set. The project has been developed in Python with TensorFlow and Numpy for the learning algorithms and the computation of the tensors.

2 Full Minimization

As mentioned before, the full minimization is done by using the gradient descent algorithm on the training error function, defined as:

$$E(\omega;\pi) = \frac{1}{2P}\sum_{p=1}^{P} \|f(x^p) - y^p\|^2 + \rho\|\omega\|^2 \tag{2.1}$$

where $\omega = (v,\ w,\ b)$ are the weights, $\pi = (N,\ \rho,\ \sigma)$ are the hyperparameters, P is the dimension of the training set and f is the function computed by the neural network.

Let's consider a shallow MLP with linear output:

$$f(x) = \sum_{j=1}^{N} v_j g\left(\sum_{i=1}^{n} w_{ji}x_i - b_j\right)$$

with n dimension of the input, N dimension of hidden layer, $v \in \mathbb{R}^N$ weights from the hidden to the output layer and g activation function, defined as:

$$g(t) = tanh\left(\frac{t}{2}\right) = \frac{1 - e^{-\sigma t}}{1 + e^{-\sigma t}}$$

The training has been performed over 15000 iterations on the training set with a learning rate $\eta = 0.001$. The hyperparameters have been tuned with a grid search over all the possible combinations of the hyperparameters in:

$$(N, \rho, \sigma) \in \Big\{25, 50, 75, 100\Big\} \times \Big\{10^{-3}, 10^{-4}, 10^{-5}\Big\} \times \Big\{1, 2, 3, 4\Big\}$$

selecting the network with the lowest error on the test set. The best result obtained an error of 0.008 on the training set and a mean squared error of 0.016 on the test set as it is also possible to see in table 5.1. The hyperparameters found are $N = 25$, $\rho = 10^{-5}$ and $\sigma = 3$. In figure 5.1 the plot of the function computed by the MLP found.

An analogous experiment has been done with the radial basis function network, defined as:

$$f(x) = \sum_{j=1}^{N} v_j \phi(\|x^i - c_j\|)$$

where $c_j \in \mathbb{R}^2$ is the center of the j-th hidden neuron, $v \in \mathbb{R}^N$ weights from the hidden to the output layer and ϕ is the activation function, defined as:

$$\phi(\|x - c_j\|) = e^{-(\|x - c_j\|/\sigma)^2}$$

with $\sigma > 0$. The training has been performed over 15000 iterations on the training set with a learning rate $\eta = 0.001$. The hyperparameters have been tuned with a grid search over all the possible combinations of the hyperparameters in:

$$(N, \rho, \sigma) \in \Big\{25, 50, 70\Big\} \times \Big\{10^{-3}, 10^{-4}, 10^{-5}\Big\} \times \Big\{0.25, 0.5, 0.75, 1\Big\}$$

selecting the network with the lowest error on the test set. The best result obtained an error of 0.007 on the training set and a mean squared error of 0.014 on the test set (see table 5.1). The hyperparameters found are $N = 50$, $\rho = 10^{-5}$

and $\sigma = 0.25$. In figure 5.1 the plot of the function computed by the RBFN found.

As already said before, both training procedures for MLP and RBFN have been performed using the gradient descent algorithm for 15000 iterations with a learning rate $\eta = 0.001$. This step has been implemented with TensorFlow's gradient descent optimizer, defined in `tf.train.GradientDescentOptimizer`. The training of the best network found took 23.343s for the MLP and 34.842s for the RBFN.

Studying all the experiments by comparing the difference between the training and the error set and the plot of the functions it's possible to see when overfitting and underfitting happen. It's interesting to notice that for both MLP and RBFN overfitting never happens, in fact for all the tests the difference between the training and the test error is so small that it can be neglected. On the other hand it's easy to have underfitting. This is due to the fact that the number of hidden neurons are not sufficient to make the network obtain a small error. Moreover, since the size of the training set is small, it's harder for the network the learn the Franke's function.

The RBFN performs slightly better than the MLP. The experiments didn't show an improvement of the performances with the use of early stopping algorithm.

3 Two Blocks Method

The idea, here, is to train the network by first setting up a subset of weights and then by solving efficiently the least squares problem obtained. This speeds up the training of the network obtaining better results than the previous section.

For what regards the MLP all the weights w_{ji} and the biases b_j, with $j \in \{1, ..., N\}$ and $i \in \{1, ..., n\}$, have been set up randomly. This reduces the minimization problem to the solution of:

$$\nabla_v E(\omega; \pi) = 0 \tag{3.1}$$

where the hyperparameters π are the values found in the previous section. Hence, the problem becomes a simple linear system:

$$\left(\frac{1}{2P} G^T G + \rho I\right) v = \frac{1}{2P} G^T y$$

where $G \in \mathbb{R}^{P \times N}$ is defined as:

$$G_{rc} = g\left(\sum_{i=1}^{n} w_{ci} x_i^r - b_c\right)$$

This method has been performed 10000 times obtaining an error of 0.003 on the training set and a mean squared error of 0.008 on the test set (table 5.1). In figure 5.1 the plot of the function computed by the MLP found.

An analogous procedure has been applied to RBFN. The centers c_j, with $j \in \{1, ..., N\}$ have been selected by using the K-means clustering algorithm. This allowed, as before, to reduce the problem to the solution of equation 3.1. The hyperparameters are the ones found in the previous section. Here G is defined as:

$$G_{ij} = \phi\left(\|x^i - c_j\|\right)$$

This method has been performed 10000 times obtaining an error of 0.001 on the training set and a mean squared error of 0.006 on the test set (table 5.1). In figure 5.1 the plot of the function computed by the RBFN found.

This second section has been implemented in Numpy, where the function `np.linalg.solve` has been used to solve the linear system. The best MLP is found in 2.67s, while the best RBFN is found in 13.67s. Each network, on average, is found in around 0.001s.

4 Decomposition Method

As already introduced before, this section describes the results obtained by a neural network trained using the two block decomposition method. More in details, the network is a full RBFN built using the best hyperparameters found in the first section, where a full minimization method is used. Here the centers and the output layer weights are updated alternatively in two consequent steps.

Given the initial guess of the centers using the K-means algorithm, the network is determined by solving the linear least squared problem (already discussed in the previus section) w.r.t. the output layer and by adjusting the centers through the gradient descent technique while keeping the weights v constant. These two steps are iteratively repeated for each epoch until the early stopping algorithm doesn't see improvement on the training error of at least 1e-5 over the last 100 epochs .

The network and the training implementation have been done using Tensorflow. In particular, the convex optimization of the weights v is done by solving the linear system of equations using the function `tf.matrix_solve_ls`, while the non-convex optimization of the centers c is performed using the gradient descent algorithm of TensorFlow, defined in `tf.train.GradientDescentOptimizer`.

The network obtained a training error of 0.002 after 4600 epochs making 4600 function evaluations and 4600 gradient evaluations before being blocked by the early stopping algorithm. The mean squared error is of 0.008 on the test set. The training took 19.807s. These results are listed and compared to the ones of the previous section in the table 5.1 .

5 Conclusion

The implementation of different learning strategies made it possible to compare not only the accuracy obtained by the different networks but also the amount of time that each network actually needs to converge to a solution.

It's interesting to notice that the extreme learning approach not only outperforms the full optimization method, but it also gives a solution in a very short period of time. TO CONTINUE.

Neural Network	N	σ	ρ	Training Error	MSE Test Set	Time
Full MLP	25	3	10^{-5}	0.008	0.016	23.343s
Full RBFN	70	0.25	10^{-5}	0.007	0.014	34.842s
Extreme MLP	25	3	10^{-5}	0.003	0.008	0.001s
Unsupervised c. RBFN	70	0.25	10^{-5}	0.001	0.006	0.001s
Two Blocks RBFN	70	0.25	10^{-5}	0.002	0.008	19.807s

Table 5.1: Results of the experiments described in the previous sections. The training error is computed with equation 2.1 on the training set. The MSE is computed on the test set. The last column shows the amount of time needed to train each specific network.

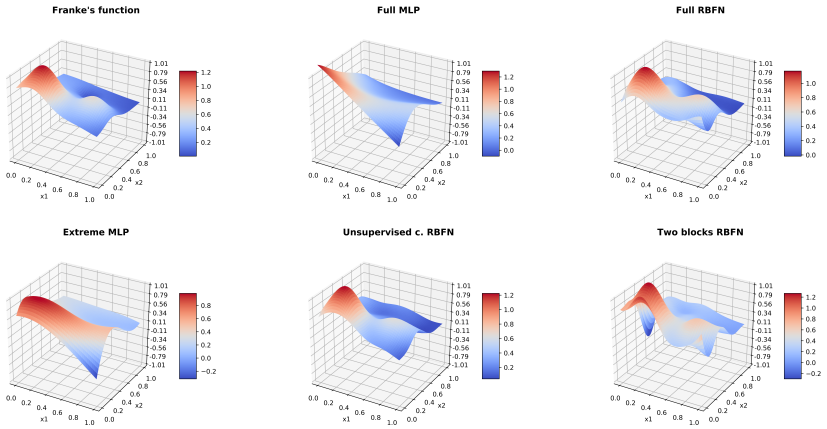

Figure 5.1: In order: the Franke's function and the functions obtained by Full MLP, Full RBFN, Extreme MLP, Unsupervised c. RBFN and Two Blocks RBFN. The hyperparameters used for each network are specified in table 5.1.