Optimization Methods for Machine Learning

Assignment # 2

1536242 - Paolo Tamagnini

21 November 2016

1 Question 1

1.1

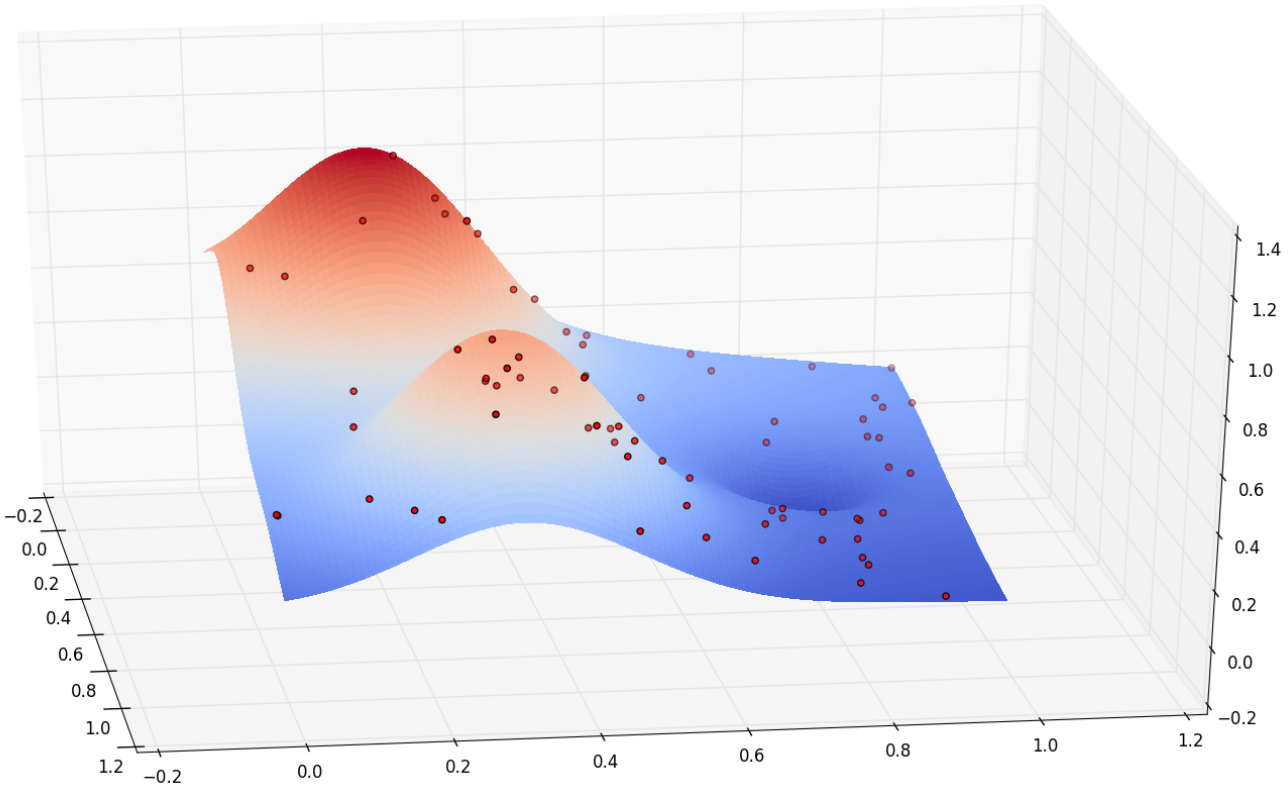
- I chose the logistic function as my activation function.
- To choose c and N, I tried many different combinations before running the optimization. A problem investigating this issue is the huge amount of time required in optimizing with a large N. For this reason I tried just all the values between 1 and 10. For c instead the values 0.1, 0.5, 1, 2, 5 and 10. Another problem is that as N increase the parameters vector v, w and b increase in size and since we are picking their values at random each optimization would have every time a completely different starting point. To avoid this I keep the values for the neurons already present in the network and I pick at random at each iteration just the initial values of the new neuron that is added as N increase. By doing so, the starting value of the variables will change just for the newly added components for the new neuron.
  At the end of every optimization, computed over the train error, I was checking the value of the error of the test instances predictions. By increasing N the optimization achieved on the train error is better and better, but the error test doesn't decrease in the same way. Indeed by increasing the number of neurons N in the hidden layer we are able to cover more closely all the different cases found in the train set. Theoretically by setting the number of hidden neurons N close to the number of train instances P we would be able to fit perfectly the train set. This is related to the interpolation capability of the neural networks and to the Pinkus' Theorem. This explains why the error over the train decreases as we increase N.
  In the train data-set we have certain relations between the features and the output that are exceptional. As we increase N too much we tend to make those exceptions, sort of general rules true over every data point with such relations. That is why as we increase N too much the test error start to increase, as the network is confusing the test data as part of the interpolated exceptions found in the train set.
  Indeed the best combination which gives the lowest train error is (N, c) = (10, 1), while for the test error is (N, c) = (6, 0.5). This is just by trying all N between 1 and 10 and probably the train error would have decreased even more if we would have kept on increasing N, while the test error would have probably increased.
  The value c instead is giving good results as long as it is around 1. It never happen that any of the two errors were optimized by an higher c because in that case the activation function $g(t)$ would have been really steep. By having a really steep $g(t)$ the values flowing in the network from the hidden layer to the output, considering the weights v equal to 1, would have been in most cases close to 1 or close to 0 and rarely in between. This almost binary behaviour would have not described well the curvilinear shape of the Franke's function $f(x)$ and that is why $c = 1$ gives good results.
  Below we see first the best combination (N, c) that gives the best train error and then the one for the test error. Beneath them there are the relative parameters found in the optimization.

```
the best N and c for train are:
N:  10
c:  1

with train error: 0.003230647141205
with test error: 0.092642947086

v:  [  3.11036801 -52.59511297  -0.19771761  17.15457736   1.58991819
```

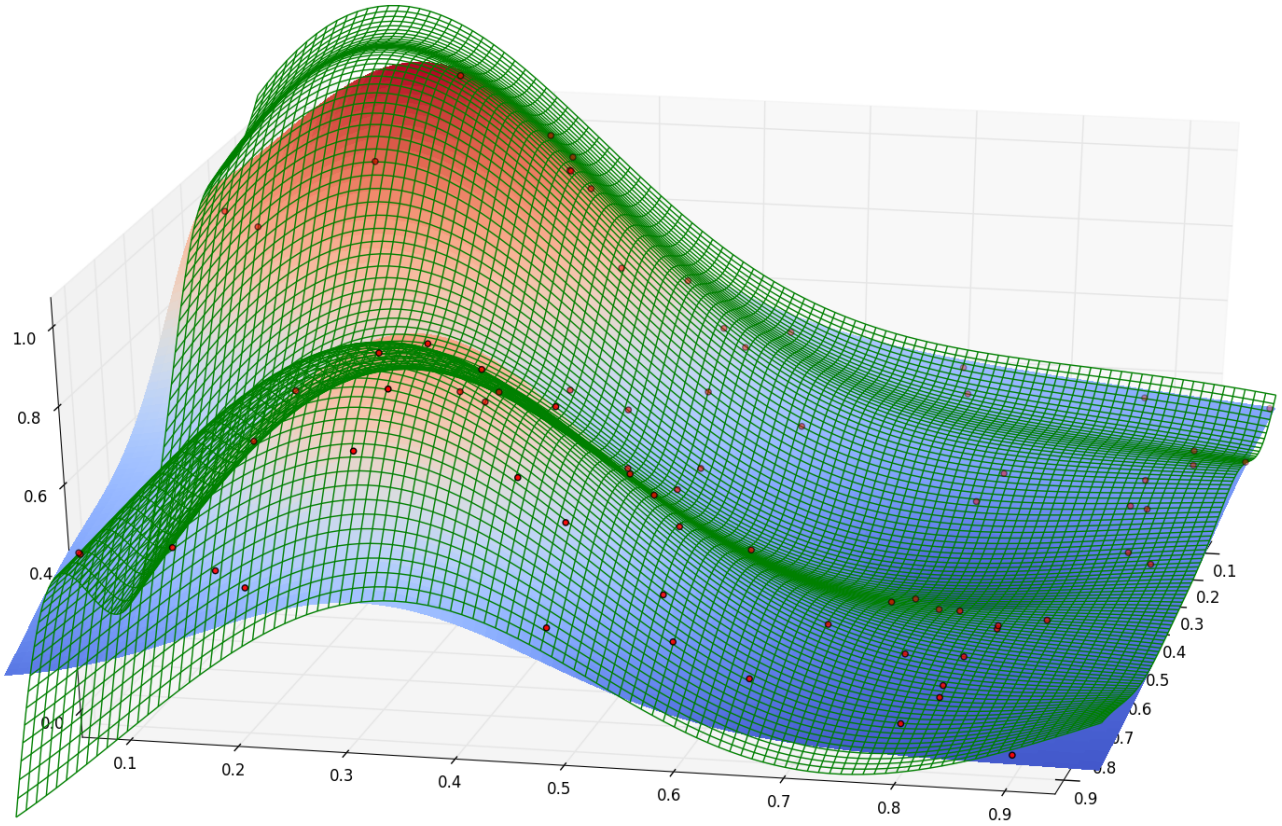
```
  51.80781689    0.07215543    1.70883109  -22.8072586     0.25316919]

w:  [[   -2.11648263     5.46520308]
    [  -10.5738753     -1.33147826]
    [-414.07973251  -125.88030477]
    [    0.94084045     7.81835423]
    [    3.69979041    12.3945232 ]
    [  -10.7133163     -1.30467579]
    [   62.89200015   -27.80703701]
    [   -1.55634654     5.28510741]
    [    0.92312397     7.06095126]
    [  -68.48800567   120.40725786]]

b:  [   0.49731848    5.34025693   96.34899403   -5.04408381   -7.15161368
   5.36632484  -30.45836796   -4.30763404   -4.39146198   25.0752904 ]

_____________________________________

the best N and c for test are:
N:  6
c:  0.5

with train error: 0.0216271646971
with test error: 0.0374336076242

v:  [  -6.00324357   11.22223938    6.21618339   -4.07314761    4.13608812
    -11.8798632 ]

w:  [[   -24.39359506    -5.73277184]
    [15.40313868   -16.06174653 ]
    [-24.29766189   -4.66968074 ]
    [  -3.59030463    15.54316305]
    [   -3.94143586  17.21038357]
    [15.66531799   -13.95922685  ]]

b:  [  14.65662368   -9.18767862   13.62281412   -3.10855269   -1.46299136
    -10.12076475]
```

- The optimization routine used is called *BFGS* from Broyden–Fletcher–Goldfarb–Shanno algorithm. Such optimization technique is an iterative method for solving unconstrained nonlinear optimization problems and it falls within the category of quasi-Newton methods. I give some starting values for the parameters to optimize which I take at random between 0 and 1. This technique computes iteration after iteration an approximation of the hessian matrix to find a path towards the minimum.
- The errors computed were already displayed in the previous points. Anyway we are referring to the ones where the test error is smaller ($N = 6$, $c = 0.5$): error over test = 0.037433, error over train = 0.021627.

- The plot of the approximation of $y = f(x_1, x_2)$ has 3 dimensions. For this reason is really hard to perceive compare to one in 2 dimensions. The following in plot is the $f(x)$ with the generated points on it.



The requested plot instead is the following, where we see an additional green wire-frame, which represents the computed approximation $\hat{y}(x)$. It looks like the approximation then is matching the surface in multiple colors which represents $f(x)$. We remember that the ground truth is $y(x) = f(x) + \epsilon$ with $\epsilon \sim U(0, 10^{-2})$.



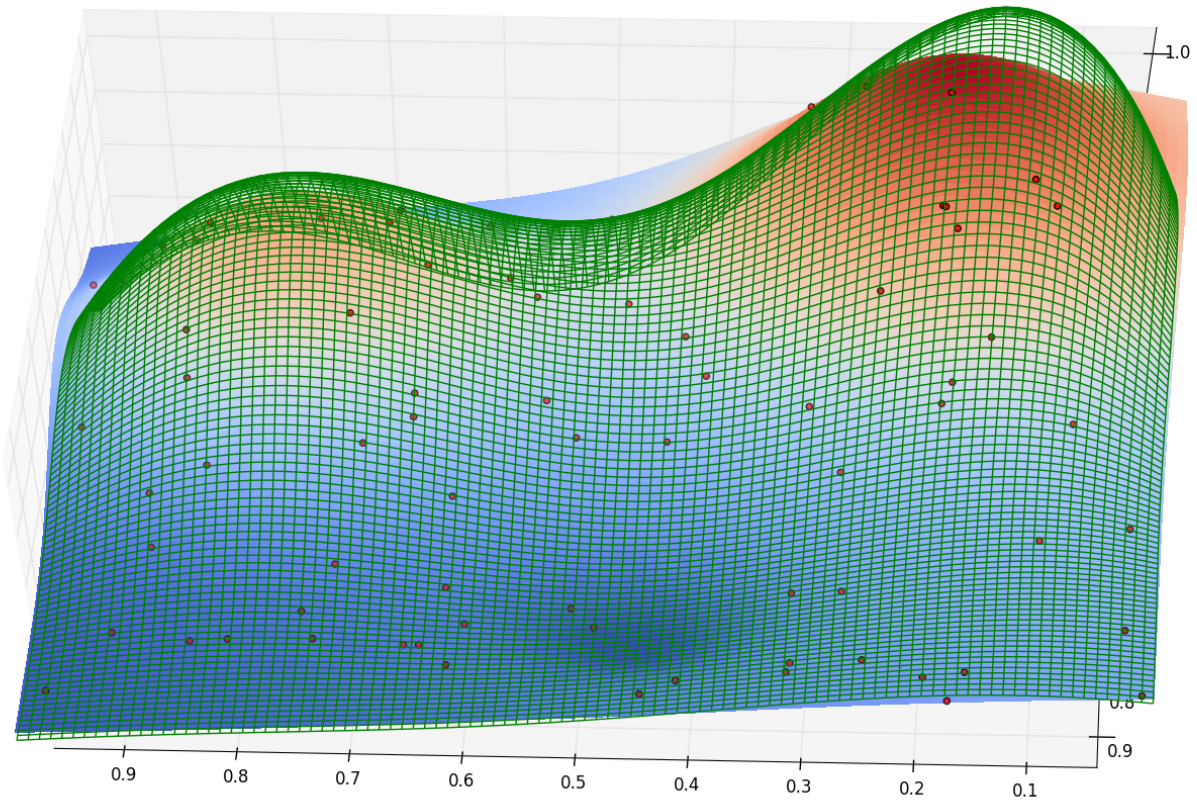
1.2

- I chose the Gaussian function.
- I chose $\rho = (\rho_1, \rho_2) = (10^{-4}, 10^{-4})$. The role of the regularization coefficients is to weight the contribution of the regularization terms that prevent over-training.
- Similar to the case of the previous question, I try many different combination of N and σ before optimizing the error function. For higher N I was expecting to have the train error always decreasing, but what happen was instead that I got the same value $N = 7$ for optimizing both errors. This is probably related to the presence of the regularization terms that prevent over-fitting, something we did not have in the MLP case.
  Suitable values of σ are 0.5 and 1, but 0.5 is way more frequent. Indeed for σ=0.5 we have better result both for train and test error. As σ is closer to 0 the Gaussian RBF function becomes thinner and most of the values returned by the function will be either 0, when x^i is far from c_j, or 1, when x^i is close to c_j, and rarely in between. Instead if σ grows bigger the Gaussian RBF function becomes thicker and the values returned by the function will be many times between 0 and 1 and not as much 0 or 1. This should not surprise us since σ is the standard deviation of the Gaussian function. Apparently $\sigma = 0.5$ is a trade-off between those situations and this is why the error are smaller mostly with such value.
  Below you can see the results of the iterations performed where N was varying between 1 and 10 and σ was taking values within 0.1, 0.5, 1, 2, 5 and 10. The chosen combination is the one that gives the smallest test error $(N, \sigma) = (7, 0.5)$.

```
the best N and sigma for train and test are:

N:  7
c:  0.5

with train error: 0.0544675571879
with test error: 0.0779825959537

w:  [-3.30135769 -2.47045723 -5.53939177 -5.1731324   4.89775938  6.11583218
    1.27749701]

c:  [[ 1.2204531   0.13225388]
    [ 0.3626939   0.44698984]
    [-0.37017004  0.00739228]
    [ 0.48979913 -0.08666898]
    [ 0.81539142  0.11820637]
    [ 0.12358006  0.04019551]
    [ 0.26733293  0.65817032]]
```

- Same optimization routine as before (*BFGS*). This means that, in this supervised selection of centers and training of the model, I decided to optimize without using the gradient equations, but using instead a quasi-newton method over the original error function.
- The optimization chosen is the one that gives the lowest test error while minimizing the train error. Therefore, with $N = 7$ and $\sigma = 0.5$, we find train error $= 0.05446$ and test error $= 0.07798$.

- As before the green grid is the approximation and the surface is the Franke's function.



- In the following table we can compare the performance and quality of the optimizations MLP and RBF.

	err_train	err_test	time (s)	#_steps	N	c	sigma
MLP	0.0216	0.0374	51	32069	6	0.5	/
RBF	0.0544	0.0779	57	18274	7	/	0.5

The results show that I achieved a bit more quality in MLP but more performance in RBF. This because the errors for RBF are not small as the MLP ones, but its number of iterations is smaller in value and the time is about the same.

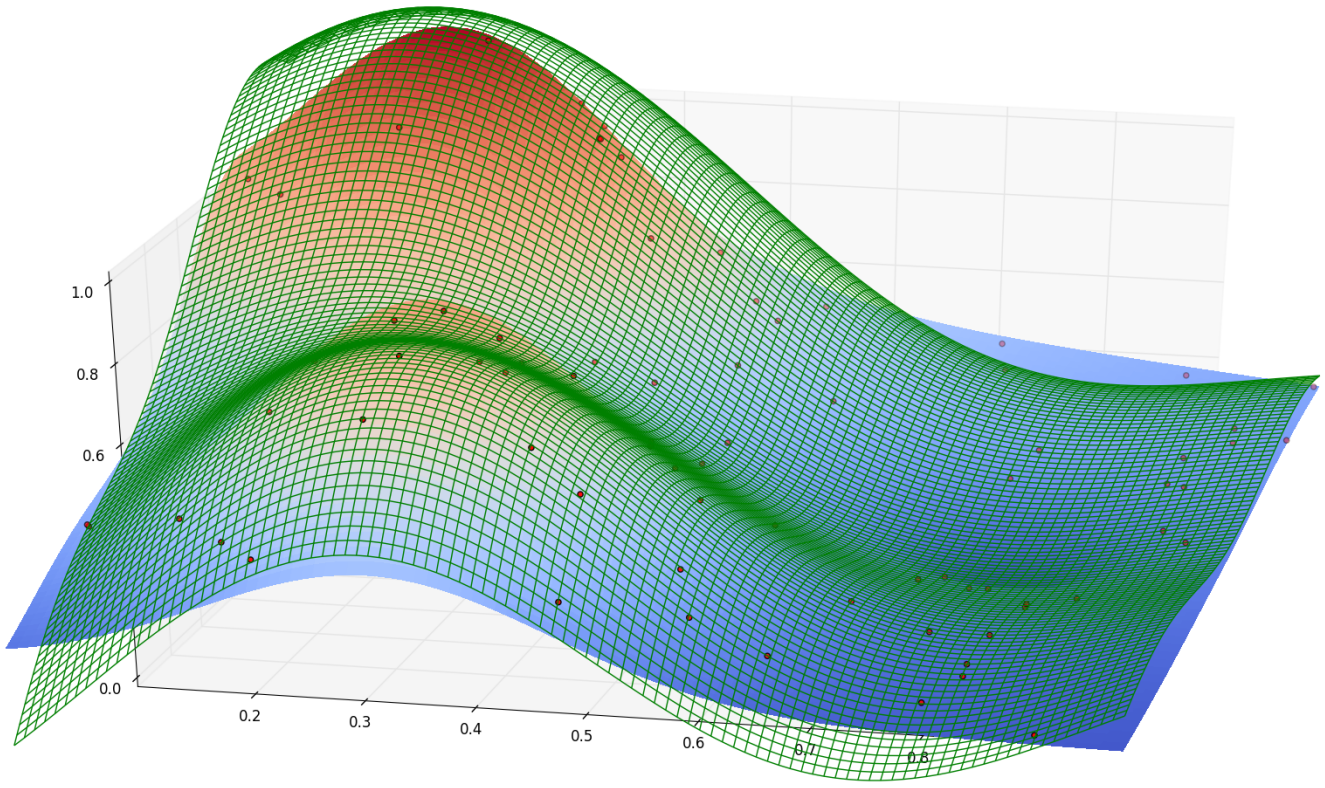
2 Question 2

2.1

- I chose again the Gaussian function.
- I chose again $\rho = 10^{-4}$.
- I couldn't find a decent error with $N = 7$ so I iterated again for bigger N. The computation is much faster having only N variables in the vector of w. I tried every N up to 20 and found that $N = 16$ was the one giving the least test error. The found σ is always 0.5.
  Before the whole cycle 20 centers were drawn from the train data-set. As the N value increases the c matrix $N \times 2$ is appending values from this first sample 20 values long. This means that the random picking of centers for already existing neurons is not performed at each iteration. I noticed that the outcome really differs with the set of centers chosen before the cycle.

```
the best N and sigma for test error are:
N:  16
c:  0.5

with train error: 0.104094190677
with test error: 0.0670733971351

w:  [ -5.39163561   3.81825606  -7.61111776   2.90520475   3.90878747
   3.16907804  -0.55341968  11.35550854   5.21691999  -0.02336278
  -2.23324039  -3.75896968  -3.25350231   0.07012029   4.29658572
 -11.14515709]

c picked at random from x_train:

[[ 0.96445484  0.2187121 ]
 [ 0.25310327  0.90666448]
 [ 0.65215889  0.36389164]
 [ 0.65952319  0.50016333]
 [ 0.70336316  0.55476796]
 [ 0.91726472  0.79950299]
 [ 0.87620059  0.13064917]
 [ 0.84176062  0.32411291]
 [ 0.86012217  0.44274793]
 [ 0.72364345  0.6958806 ]
 [ 0.08266571  0.53331288]
 [ 0.54535388  0.47365237]
 [ 0.31823751  0.91948818]
 [ 0.28295876  0.03707649]
 [ 0.20881193  0.34904034]
 [ 0.89346791  0.58858039]]
```

- I still used *BFGS*.
- The optimization chosen is the one that gives the lowest test error while minimizing the train error. Therefore, with $N = 16$ and $\sigma = 0.5$, we find train error = 0.10409 and test error = 0.06707.
- As before the green grid is the approximation and the surface is the Franke's function.


1.0
0.8
0.6
0.0
0.2
0.3
0.4
0.5
0.6
0.7
0.8