Assignment # 1 - MLP and Generalized RBF Network

Group members: Conti Emmanuele and Rossini Valerio

In this first homework we will use the Python’s library called TensorFlow and we consider a shallow FFN that provides the model f(x), where f(x) is the Franke’s function.

The data set is obtained by sampling on 100 random points the function and adding a uniform noise, i.e. = f ( ) + ε. After that we divide our data set into a training set (that contains the 70% of our data) and a test set (that contains the remaining 30%). We have done this creating two functions (all the functions are written in “Class”): the first called “Franke\_function” where we write simply the Franke’s function and the second called “generateTrainingTestSets” that allows us to split the data set in x\_train, x\_test, y\_train and y\_test. About the implementation of the shallow FFN, for the choice of the optimum parameters, through a grid search we vary the number of neurons N, the parameters ρ – that it varies in the range – and σ:

* N assumes the values 2, 5, 10, 15, 20, 30, 50 and 70
* ρ assumes the values 0.0001, 0.0005 and 0.00001
* σ assumes the values 0.5, 1, 2.5 and 5

These values will be used subsequently in the implementation of the MLP and RBF.

Then we implement the regularized training error function, that it is written in the following way:

We have used a function for computing it, called “loss\_func”, that take in input the y predicted, y observed, (i.e. the weights from input to hidden), (i.e. the weights from hidden to output), (i.e. the bias), and P (i.e. the total number of the samples) and returns the error. For making the function as general as possible, the parameters and are already set to zero; we have done this because if we want to implement for example the regularized training error function for a RBF network in the second part of the exercise 1, we don’t need to consider the parameters and . In that case we are only interested in the weights from hidden to output. Other main functions are the “predict” and “predict\_RBF” that return respectively the y predicted for the MLP and the y predicted for the RBF. We have these two distinct functions for the prediction of y because these latter are computing in different ways:

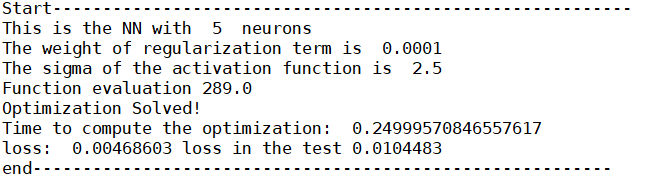
* for the MLP, we have:
* for the RBF, we have:

For us is useful build a “class” called “NeuralNetwork”. If we focus on that class, we can find two methods fixed for all the exercises and one that varies in every exercise.

The two-fixed method are:

* a function called “\_\_init\_\_”, where we give the number of the samples (i.e. the number of samples that going to be propagated through the network), number of features, number of hidden layers, number of output, number for neurons for layer, the batch size, the epochs, the display\_step (i.e. every how many epochs print the loss) and the learning rate. Inside the method, we define input and output matrices (respectively x and y) like placeholder
* a function called “run”, that runs the computational graph in a tf.Session (a Session object is an object that encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated). For every setting of “\_\_init\_\_” we print a message that say us that the optimization is finished, the time to compute the gradient, loss function for the training set and the loss function for the test set. We also print, every display\_step, the number of epoch and the average loss (cost) associated.

An example of all the things that we print for N equal to 5, ρ equal to 0.0001 and σ equal to 2.5 is the following:



About the computation of the gradient, we will use the momentum. momentum helps the stochastic gradient descent to navigate along the relevant directions and softens the oscillations in the irrelevant. It simply adds a fraction of the direction of the previous step to a current step. This achieves amplification of speed in the correct direction and softens oscillation in wrong directions.

The function called “plot3D” give us, according to the values that we pass in input, the following plot:

* a plot for the values of x\_train and y\_train (i.e. the values in the training set of x and y that we are observing)
* a plot for the values of x\_train and y\_p\_final (i.e. the values in the training set of x that we are observing, and the y predicted)
* a plot for the values of x\_test and y\_p\_t\_final (i.e. the values in the test set of x that we are observing, and the y predicted)

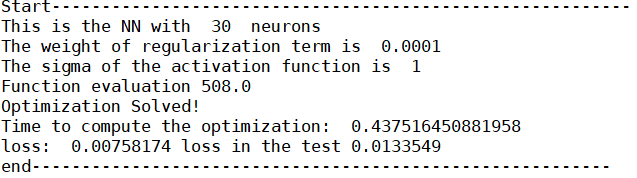
Another function that give us another plot is “scatterY” that returns:

* a plot for the values of y\_train and y\_p\_final (i.e. the values in the training set of y observed and y predicted)
* a plot for the values of y\_test and y\_p\_t\_final (i.e. the values in the training set of y observed and y predicted)

1. Full minimization
   1. MLP

For the implementation of the multilayer perceptron we need to consider, in addition to the functions written above, also the activation function. The activation function chosen for this assignment is the hyperbolic tangent, that we have written in a function called “tanh”. The method that varies inside the class “NeuralNetwork” is the function called “MLP\_config”, where we define the weights from input to hidden, weights from hidden to output and the bias and we recall two previous functions written before, i.e. “predict” and “loss\_func” (these functions are computed two times: one time for the training set and another time for the test set). Then we compute the gradient, using tf.train.MomentumOptimizer as optimization routine for solving the minimization problem.

For the best N (equal to 30), best ρ (equal to 0.0001), and best σ (equal to 1) we have:

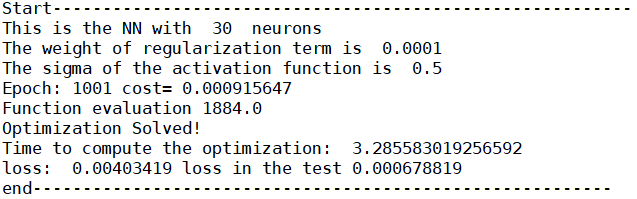


So, as we can see, the best loss function for the training set is equal to 0.00758174 and the best loss function for the test set is equal to 0.0133549. The function evaluations and gradient evaluations are equal to 508.

1.2 RBF

In this other point of the exercise 1, we are going to consider an RBF network.We have created appositely two functions for this exercise: “predict\_RBF” that gives us the y predicted and “gaussian\_function” in which we write the gaussian function. We need to implement the regularized training error function of the RBF network and minimize it respect to the weights from hidden to output and the centers. We have done this using always the function “loss\_func” that is the same used before but in this case, we need to consider only the parameter . The method that varies inside the class “NeuralNetwork” is the function called “RBF\_config\_1”, where we define only the weights from hidden to output and the centers, and we recall “predict\_RBF” and “loss\_func” that are computed two times: one time for the training set and another time for the test set. tf.train.MomentumOptimizer is the optimizer used for computing the gradient.

For the best N (equal to 30), best ρ (equal to 0.0001), and best σ (equal to 0.5) we have:



So, as we can see, the best loss function for the training set is equal to 0.00403419 and the best loss function for the test set is equal to 0.000678819. The function evaluations and gradient evaluations are equal to 1884.

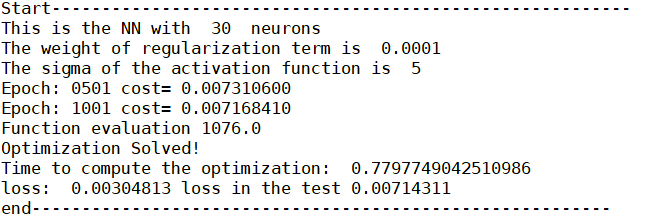
1. Two block methods

2.1 MLP with two block methods

In this exercise we are going to fix randomly the values of the parameters and and we want to minimize the quadratic convex training error of . We have generated randomly the values of parameters and choosing the best combination of these ranges: Wessels and Barnard, Le Cun, Yam and Chow, Haffner, Nguyen & Widrow. At the end we choose Yam and Chow range

The method that varies inside the class “NeuralNetwork” is the function called “MLP\_twoblock\_config”, where we define the weights from input to hidden, weights from hidden to output and the bias and we recall two previous functions written before, i.e. “predict” and “loss\_func”. If we work using the training set, the function “predict” takes in input the same parameters used in the first exercise (point 1.1), while the function “loss\_func” has different parameters in input, respect to the other exercise, because we don’t need to consider and when we minimize the loss function. tf.train.MomentumOptimizer is the optimizer used for computing the gradient.

For the best N (equal to 30), best ρ (equal to 0.0001), and best σ (equal to 5) we have:



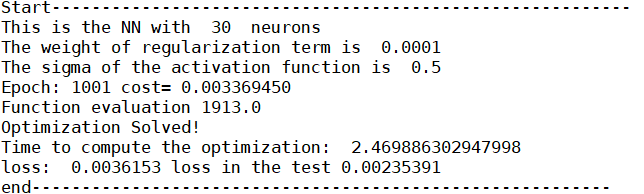
So, as we can see, the best loss function for the training set is equal to 0.00304813 and the best loss function for the test set is equal to 0.00714311 (in the exercise 1.1 instead the best loss function for the training set is equal to 0.00758174 and the best loss function for the test set is equal 0.0133549). So, we can confirm that the loss function of the training set and the loss function of the test set, using two block methods, are better than the previous point.

2.2 RBF with two block methods

In this point we have used a clustering procedure, called k means, for the selection of the centers, instead of selecting the centers randomly on the P points of the training set. We have done this creating a function called “getCenters”. Then inside the function “RBF\_config\_unsupervised” we define only the weights from hidden to output and when we recall the function “predict\_RBF” and “loss\_func”.

tf.train.MomentumOptimizer is the optimizer used for computing the gradient.

For the best N (equal to 30), best ρ (equal to 0.0001), and best σ (equal to 0.5) we have:



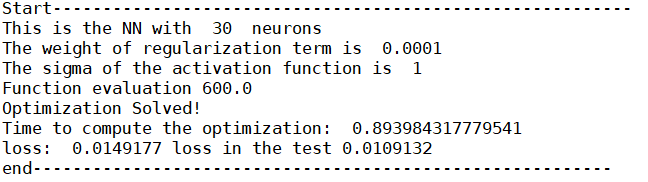
So, as we can see, the best loss function for the training set is equal to 0.0036153 and the best loss function for the test set is equal to 0.00235391.

1. Decomposition method

In this point we alternate the convex minimization with respect to and the non-convex minimization with respect to and for the MLP network (all these operations are made inside the function “MLP\_decomposition\_config”).

The optimization algorithm stops in our case when the norm of the gradient is less or equal to 1e-3.

For the best N (equal to 30), best ρ (equal to 0.0001), and best σ (equal to 1) we have:



So, as we can see, the best loss function for the training set is equal to 0.0149177 and the best loss function for the test set is equal to 0.0109132.

The number of outer iterations (number of subproblems solved) is equal to 1200, the number of function evaluation is equal to 600, the number of gradients evaluation is equal to 1200, and the computational time needed to get it is approximately equal to 0.90 seconds.

1. Comparison among all the implemented methods

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ex | FFN | settings | | | Training error | Test error | Optimization time |
| Q1.1 | Full MLP | N = 30 | ρ = 0.0001 | σ = 1 | 0.0075 | 0.0133 | 0.4375 |
| Q1.2 | Full RBF | N = 30 | ρ = 0.0001 | σ = 0.5 | 0.0040 | 0.0006 | 3.2855 |
| Q2.1 | MLP with two block methods | N = 30 | ρ = 0.0001 | σ = 5 | 0.0030 | 0.0071 | 0.7797 |
| Q2.2 | RBF with two block methods | N = 30 | ρ = 0.0001 | σ = 0.5 | 0.0023 | 0.0036 | 2.4698 |
| Q3 | MLP with  Decomposition method | N = 30 | ρ = 0.0001 | σ = 1 | 0.0149 | 0.0109 | 0.8939 |

FIGURE 1.1 MLP: x and y observed

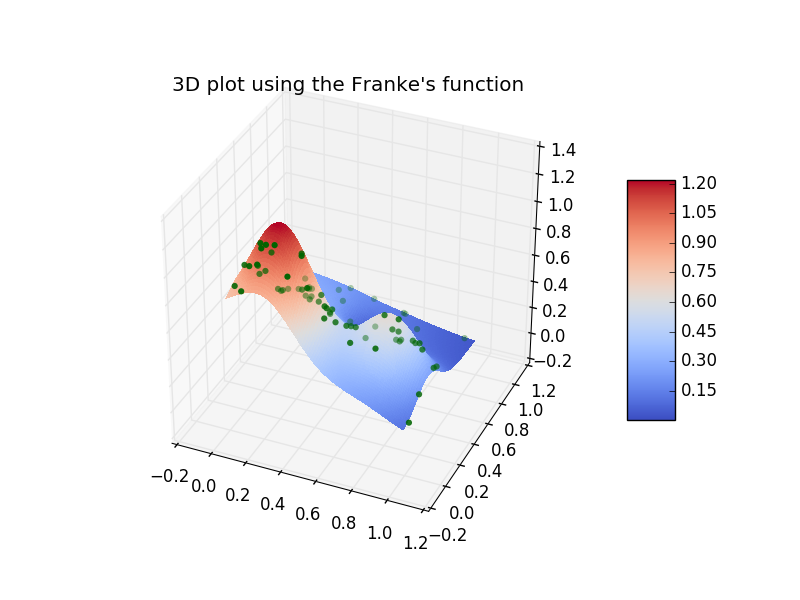


FIGURE 1.1 MLP: x observed, and y predicted

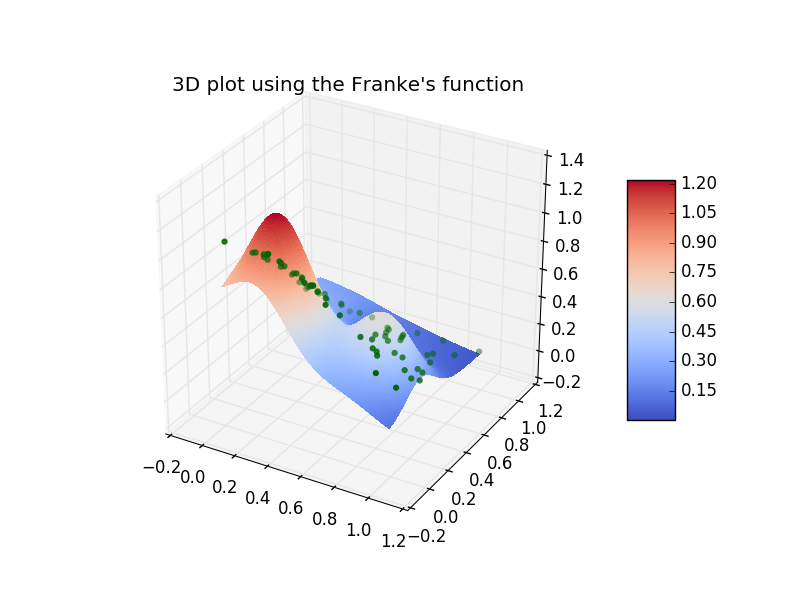


FIGURE 1.2.1 RBF: x and y observed

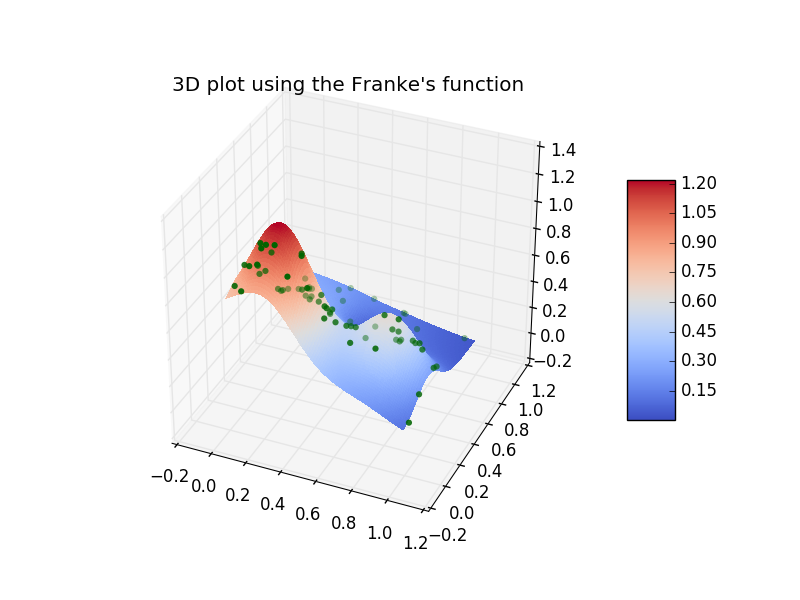


FIGURE 1.2.2 RBF: x observed, and y predicted

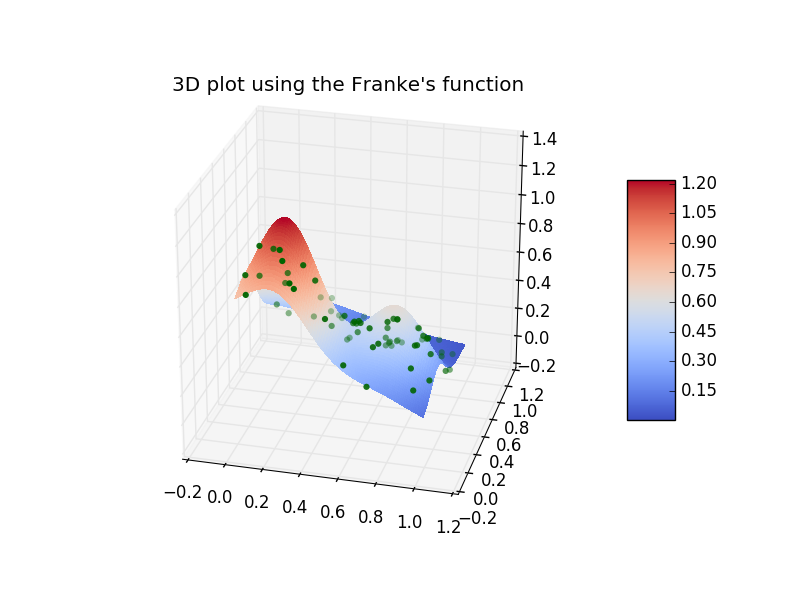


FIGURE 2.1.1 MLP with two block methods: x and y observed

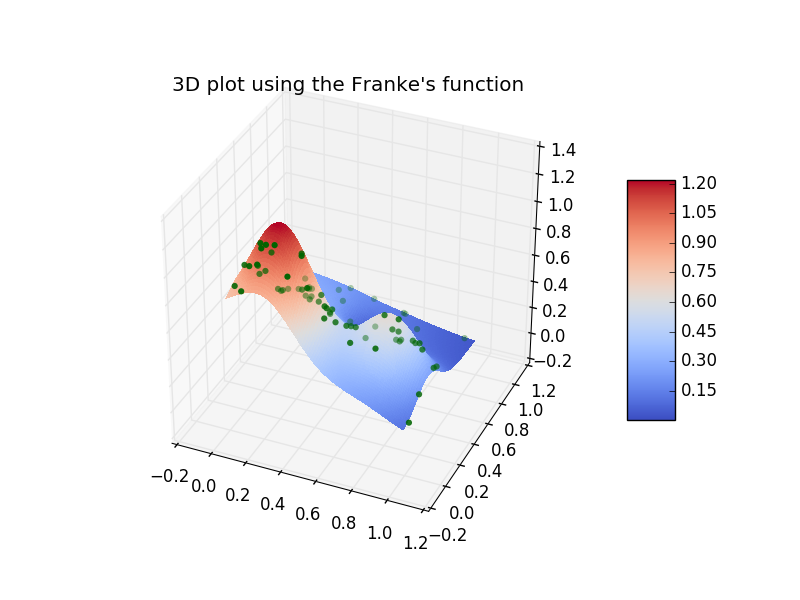


FIGURE 2.1.2 MLP with two block methods: x observed, and y predicted

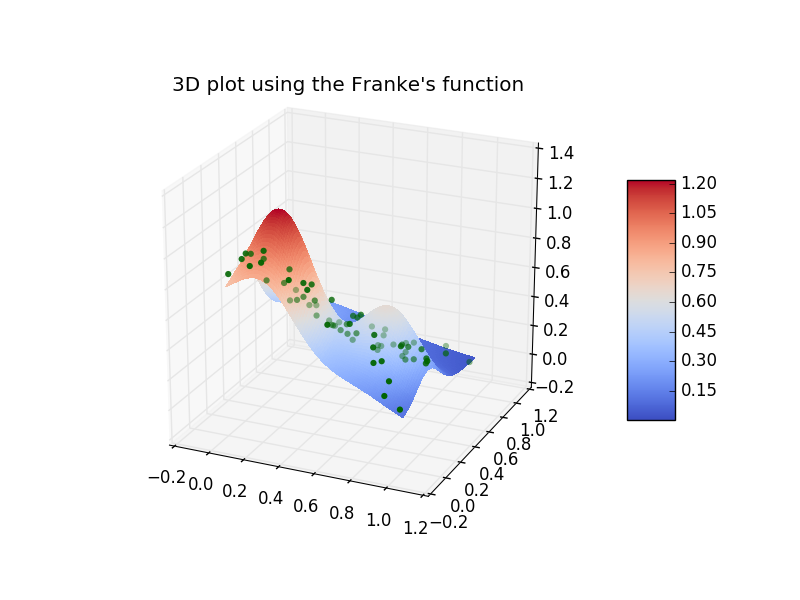


FIGURE 2.2.1 RBF with two block methods: x and y observed

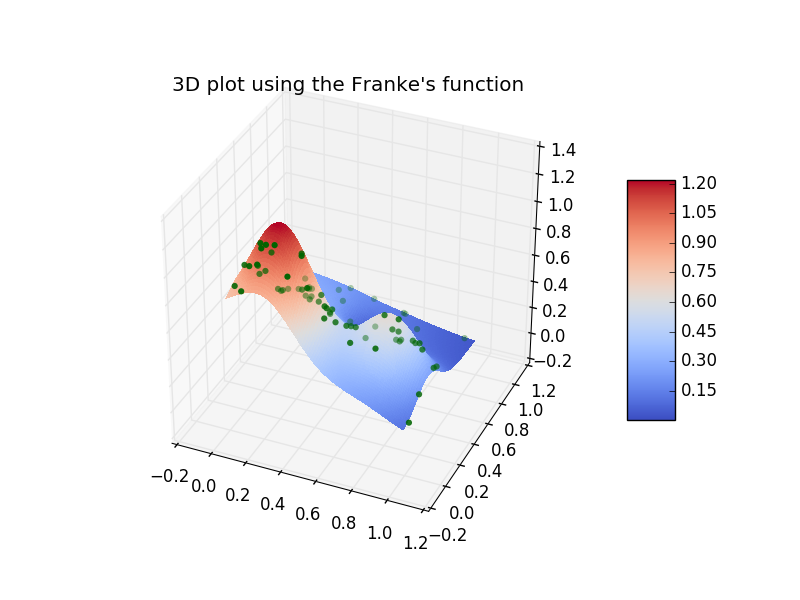
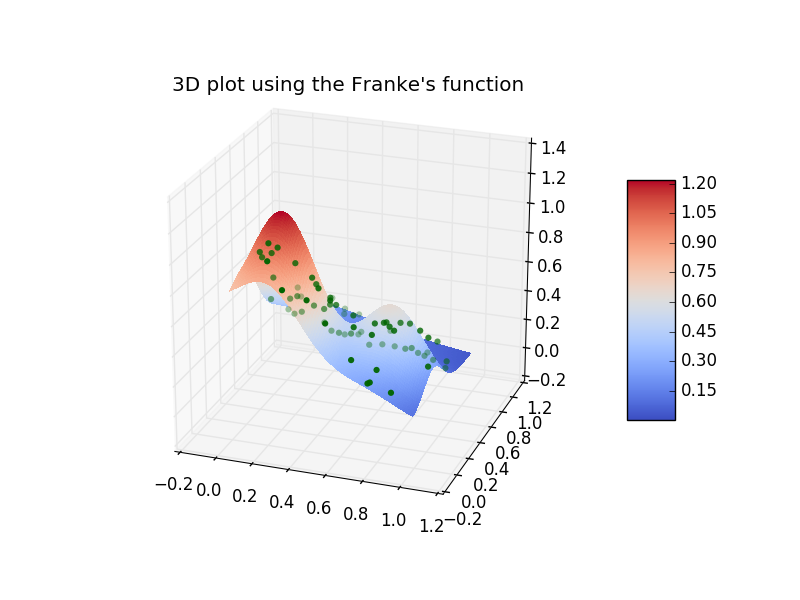


FIGURE 2.2.2 RBF with two block methods: x observed, and y predicted



1. FIGURE 3.1 MLP decomposition method: x and y observed

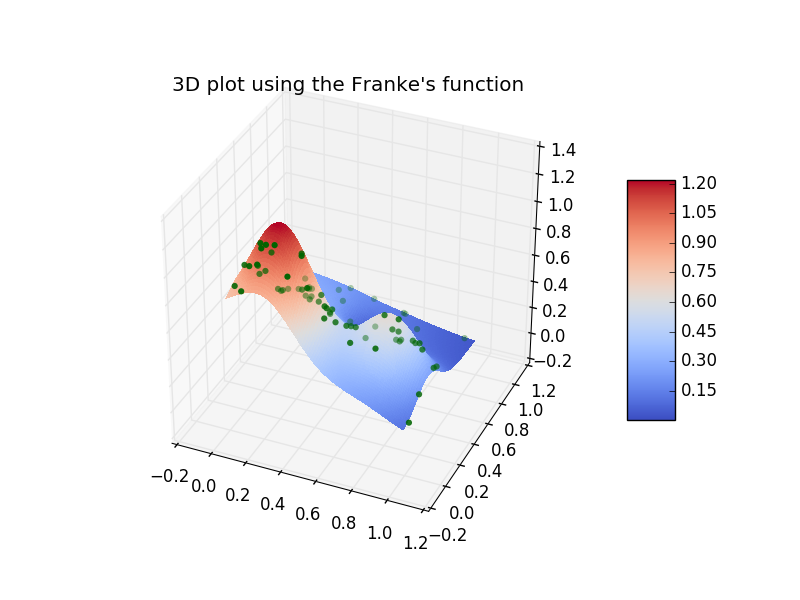


FIGURE 3.2 MLP decomposition method: x observed, and y predicted

