Radial Basis Function (RBF)

Amir.M Mousavi.H

Department of Computer Engineering
Shahid Rajee University

Tehran, Iran

AmirMahmood.Mousavi@yahoo.com

***Abstract* - Neural Networks are very powerful models for classification and regression tasks. Radial Basis Function Neural Network or RBFNN is one of the unusual but extremely fast, effective and intuitive Machine Learning algorithms. The 3-layered network can be used to solve both classification and regression problems. This paper presents a general introduction and discussion of how a Radial Basis Function Neural Network works. then we show the implementation result of a Radial Basis Function Neural Network on Artificial data for both classification and regression problem and then analyze the network's behavior in presence of different situations and compare our codes to ready functions in term of performance.**

Keywords— Radial Basis Function, Neural Network, Clustering, RBF, Classification, Regression, Artificial Data

I. INTRODUCTION

In the field of mathematical modeling, a radial basis function network is an artificial neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. Radial basis function networks have many uses, including function approximation, time series prediction, classification, and system control. They were first formulated in a 1988 paper by Broomhead and Lowe, both researchers at the Royal Signals and Radar Establishment.[1][2][3]

Overaly, Each RBF neuron stores a “prototype”, which is just one of the examples from the training set. When we want to classify a new input, each neuron computes the Euclidean distance between the input and its prototype. Roughly speaking, if the input more closely resembles the class A prototypes than the class B prototypes, it is classified as class A. in figure.1. a simple architecture of RBF is shown. The figure.1. illustration shows the typical architecture of an RBF Network. It consists of an input vector, a layer of RBF neurons, and an output layer with one node per category or class of data.

The input vector is the n-dimensional vector that you are trying to classify. The entire input vector is shown to each of the RBF neurons.

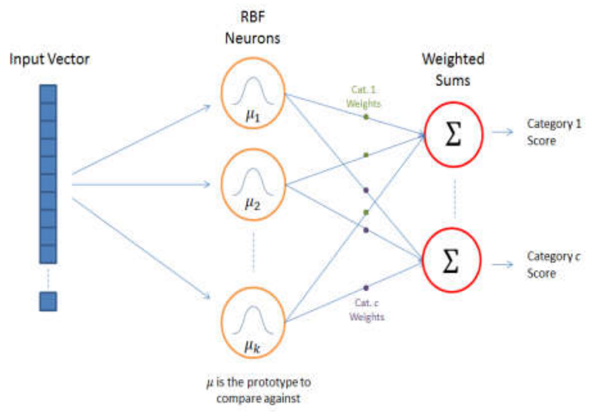


Figure.1. a simple architecture of RBF

RBF Neurons: Each RBF neuron stores a “prototype” vector which is just one of the vectors from the training set. Each RBF neuron compares the input vector to its prototype, and outputs a value between 0 and 1 which is a measure of similarity. If the input is equal to the prototype, then the output of that RBF neuron will be 1. As the distance between the input and prototype grows, the response falls off exponentially towards 0. The shape of the RBF neuron’s response is a bell curve, as illustrated in the network architecture diagram. The neuron’s response value is also called its “activation” value. The prototype vector is also often called the neuron’s “center”, since it’s the value at the center of the bell curve.

Output nodes: The output of the network consists of a set of nodes, one per category that we are trying to classify. Each output node computes a sort of score for the associated category. Typically, a classification decision is made by assigning the input to the category with the highest score. The score is computed by taking a weighted sum of the activation values from every RBF neuron. By weighted sum we mean that an output node associates a weight value with each of the RBF neurons, and multiplies the neuron’s activation by this weight before adding it to the total response.
Because each output node is computing the score for a different category, every output node has its own set of weights. The output node will typically give a positive weight to the RBF neurons that belong to its category, and a negative weight to the others.

RBF Neuron Activation Function: Each RBF neuron computes a measure of the similarity between the input and its prototype vector (taken from the training set). Input vectors which are more similar to the prototype return a result closer to 1. There are different possible choices of similarity functions, but the most popular is based on the Gaussian. Below is the equation for a Gaussian with a one-dimensional input.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where x is the input, mu is the mean, and sigma is the standard deviation. This produces the familiar bell curve shown below, which is centered at the mean, mu (in figure.2 the mean is 5 and sigma is 1).

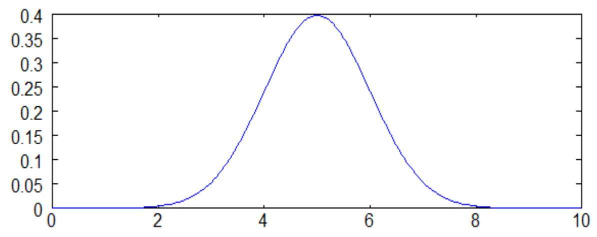


Figure.2. Gaussian function example

The RBF neuron activation function is slightly different, and is typically written as:

$$\varphi(x) = e^{-\beta\|x-\mu\|^2}$$

In the Gaussian distribution, mu refers to the mean of the distribution. Here, it is the prototype vector which is at the center of the bell curve. For the activation function, phi, we aren't directly interested in the value of the standard deviation, sigma, so we make a couple simplifying modifications.

The first change is that we've removed the outer coefficient, 1 / (sigma * sqrt(2 * pi)). This term normally controls the height of the Gaussian. Here, though, it is redundant with the weights applied by the output nodes. During training, the output nodes will learn the correct coefficient or "weight" to apply to the neuron's response.

The second change is that we've replaced the inner coefficient, 1 / (2 * sigma^2), with a single parameter 'beta'. This beta coefficient controls the width of the bell curve. Again, in this context, we don't care about the value of sigma, we just care that there's some coefficient which is controlling the width of the bell curve. So we simplify the equation by replacing the term with a single variable.

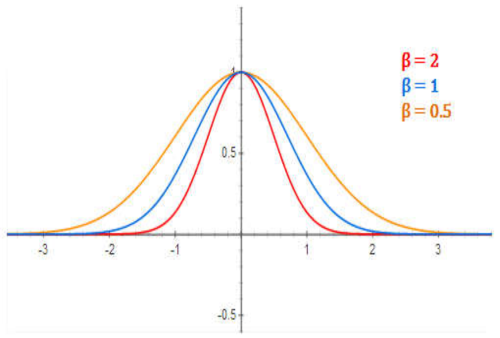


figure.3.RBF Neuron activation for different values of beta

There is also a slight change in notation here when we apply the equation to n-dimensional vectors. The double bar notation in the activation equation indicates that we are taking the Euclidean distance between x and mu, and squaring the result. For the 1-dimensional Gaussian, this simplifies to just (x - mu)^2. It's important to note that the underlying metric here for evaluating the similarity between an input vector and a prototype is the Euclidean distance between the two vectors.

Also, each RBF neuron will produce its largest response when the input is equal to the prototype vector. This allows to take it as a measure of similarity, and sum the results from all of the RBF neurons. As we move out from the prototype vector, the response falls off exponentially. Recall from the RBFN architecture illustration that the output node for each category takes the weighted sum of every RBF neuron in the network–in other words, every neuron in the network will have some influence over the classification decision. The exponential fall off of the activation function, however, means that the neurons whose prototypes are far from the input vector will actually contribute very little to the result.

II. METHODS & MATERIALS

In order to implement RBF, we used MATLAB 2014a for classification and python version 3.7.0 for regression.

On the other hand to train the network, we generated artificial data, a random 2 dimensional array coming from a mixture of gaussian distribution (for the first pick with mean = [2, -3] cov = [[1.2, 0], [0, 1.2]] and and for second pick with mean = [2,2] , cov = [[1, 0], [0, 1.5]] and for third pick with mean = [-3,3] , cov = [[0.5, 0], [0, 0.5]]). For each pick we generated 150 ,100, 50 samples, respectively. In figures.4- 5 classes are shown in different colors:

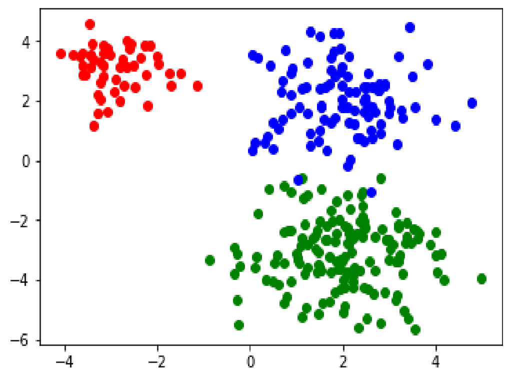


Figure.4 data distribution

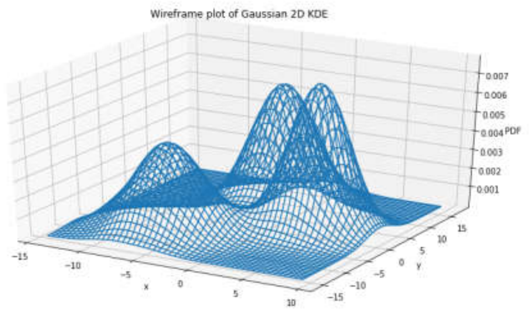


Figure.5 data distribution in 3D

For the hyperparameters we set:

- Centers- The prototype vectors stored in the RBF neurons.
- betas- The beta coefficient for each corresponding RBF neuron.
- Theta- The weights for the output layer. There is one row per neuron and one column per output node / category.
- Activation function = Gaussian function
- epoch numbers: 100
- default map size = 10*10
- clustering algorithm : K-means

Our network includes 2 input neurons, 3 neurons in output layer and 15 neurons for middle layer. The network's topology is shown in figure.6.

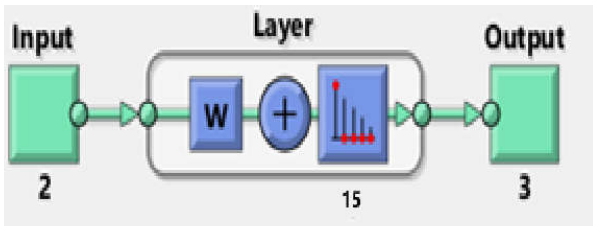
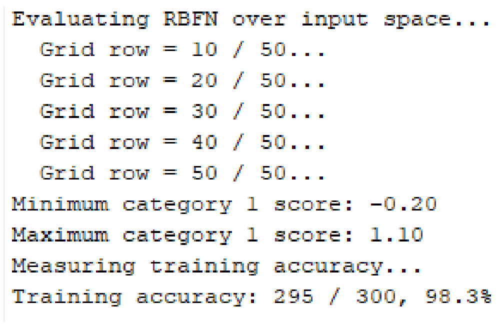


Figure.6. The network's topology for 3 classes problem

III. EXPERIMENTAL RESULTS

our results have shown that a radial basis function (RBF) with help of a clustering algorithm, can classify unknown data distribution with a high accuracy and make it visualize.
We started to test the model with 15 neurons for middle layer for 3 classes problem, our model achieved 98.3% for classification.



```
Evaluating RBFN over input space...
  Grid row = 10 / 50...
  Grid row = 20 / 50...
  Grid row = 30 / 50...
  Grid row = 40 / 50...
  Grid row = 50 / 50...
Minimum category 1 score: -0.20
Maximum category 1 score: 1.10
Measuring training accuracy...
Training accuracy: 295 / 300, 98.3%
```

Figure.7. performance of model

As we can observe, it indicates that RBF can classify the overlapped data as strong as a shallow MLP but its computational load is very high for the 300 samples it took about 1.5 second to calculate.

```
runnig time: 1.47300
```

For further experiment we plotted the centroids and decision boundary. (figure.8-9)

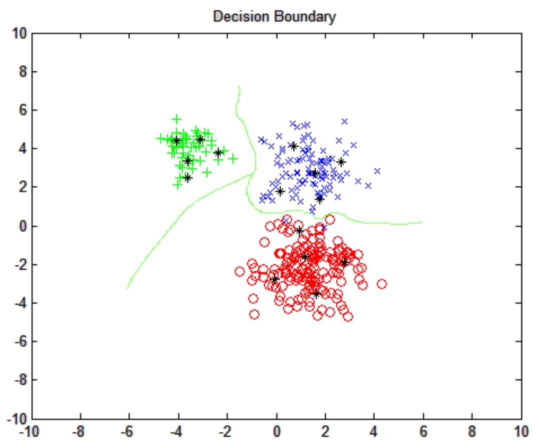
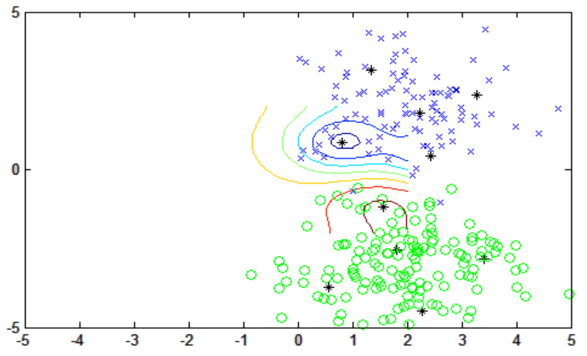
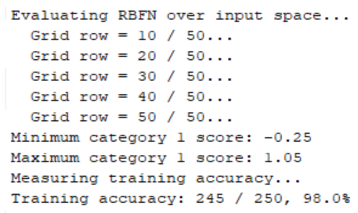


Figure.8. centroids and decision boundary of RBF

if we increase number of neurons we will have more percise decision boundary.

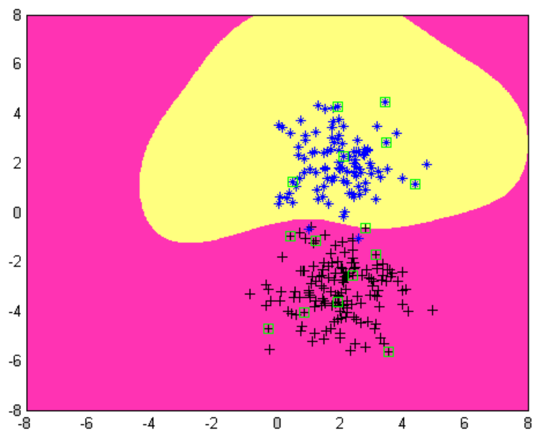
For further experiment we compare our code to MATLAB's library for RBF. (figures.8-9)

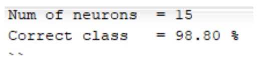




```
Evaluating RBFN over input space...
  Grid row = 10 / 50...
  Grid row = 20 / 50...
  Grid row = 30 / 50...
  Grid row = 40 / 50...
  Grid row = 50 / 50...
Minimum category 1 score: -0.25
Maximum category 1 score: 1.05
Measuring training accuracy...
Training accuracy: 245 / 250, 98.0%
```

Figure.8. our model in 2 classes problem





```
Num of neurons  = 15
Correct class   = 98.80 %
```

Figure.7. MATLAB's model in 2 classes problem

As it can be seen, it seems that our code is almost the same as MATLAB's library, MATLAB is a little better because it put centers in the places with more density or uses optimized hyperparameters, overally the performance of both models are similar.

After running main experiments, we started to investigate more and add some supplementary data.

First, we focused on number of neurons impact on performance, therefor we checked number of neurons from 1 to 150, the result is shown in figure.10:

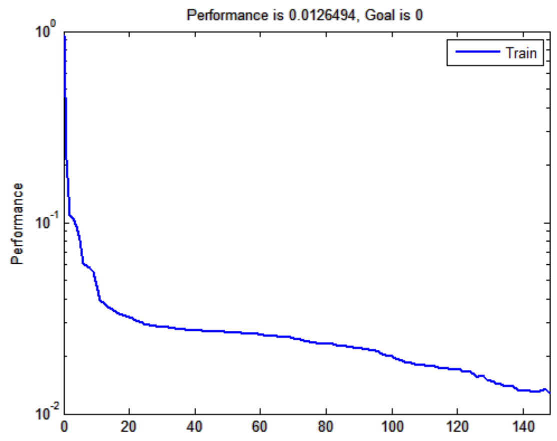


Figure.10. number of neurons vs performance

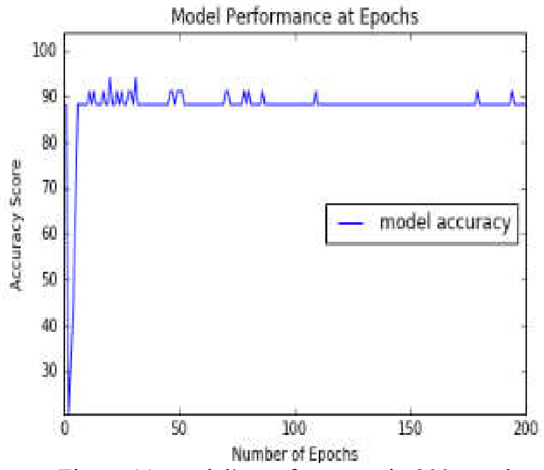


Figure.11. model's performance in 200 epochs

As it is shown in figure.10. if we increase number of neurons, performance will increase and in this case we achieved 100% performance for classification, but it might increase some problems, which including computational load, if we observe figure.10. we can see that after 15 neurons, our loss is decreasing with a low slope but we are spending a lot of computational resources, on the other hand it make the decision boundary localize.(figure.12)

As it's shown, we must consider the generalization of model and try to find the optimum number of neurons.

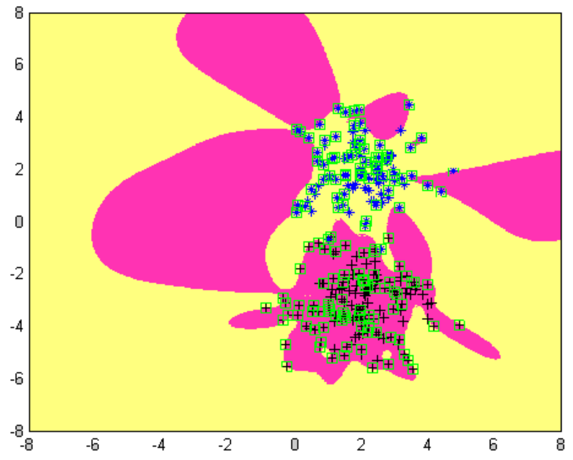
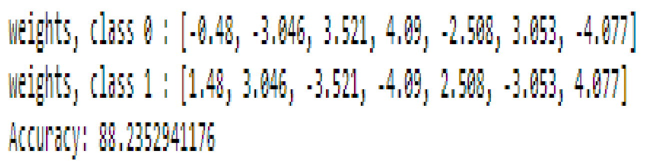


Figure.12. visualizing data with 150 neurons

Then we tested the model for 30 samples and we achieved 88.5% performance.



```
weights, class 0 : [-0.48, -3.046, 3.521, 4.09, -2.508, 3.053, -4.077]
weights, class 1 : [1.48, 3.046, -3.521, -4.09, 2.508, -3.053, 4.077]
Accuracy: 88.2352941176
```

As we expected RBF can be used for classification tasks and it works properly and with the right hyper parameters can achieve to MLP's performance in the classification.

Regression:
So far, we discussed classification tasks but RBF can be useful for regression as well.
Here we simulated and implemented a regression problem and we are going to present an RBF model to be fitted in the generated data.
First, we used a SIN(X) as our data, and then we tried to test different parameters of RBF (including number of neurons and sigma), the results are shown(figures.13-)

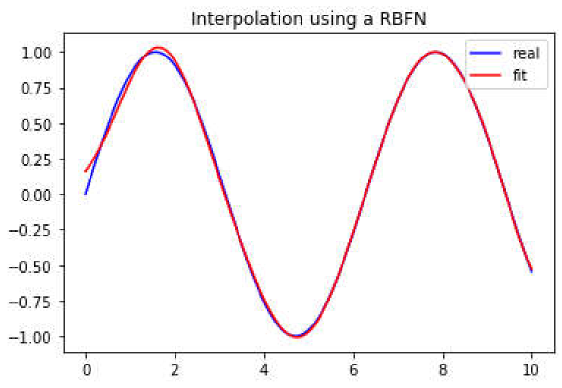


Figure.13. # of neurons: 10 & sigma: 0.5

Is it can be seen it fits the model perfectly, for further experiment, we extended the domain of data to see how model works with the same parameters:

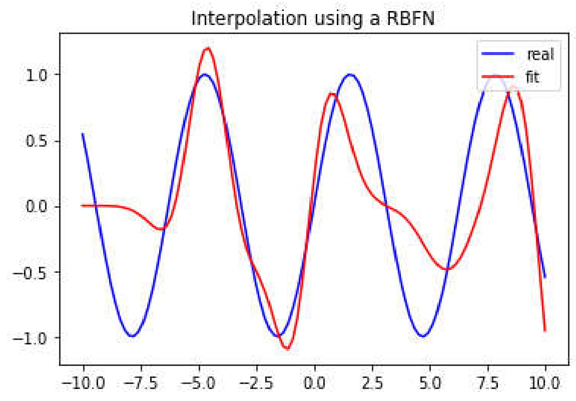


Figure.14. # of neurons: 10 & sigma: 0.5 (extended mode)

As a result, we can observe that model fails so we need to change the parameters So we changed number of neurons:

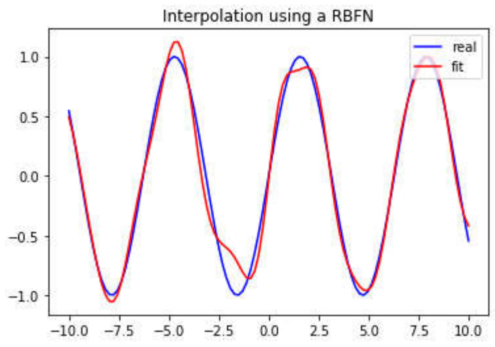


Figure.15. # of neurons: 20 & sigma: 0.5 (extended mode)

It is better than previous version but it has no fitted perfectly. We change the sigma to see the effect:

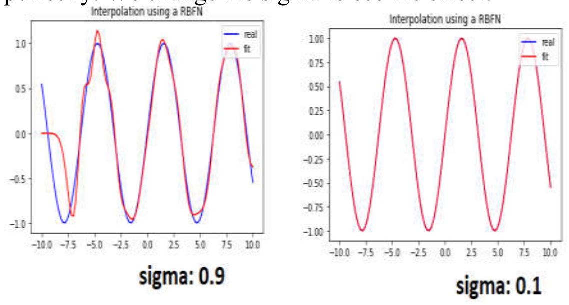


Figure.16. # of neurons: 20 & sigma: 0.9, 0.1

As we expected, if we decrease the sigma our fitting gets better.
Then we start to test model robustness in different data pattern, we change our model data to COS(X), EXP (X), TAN(X).

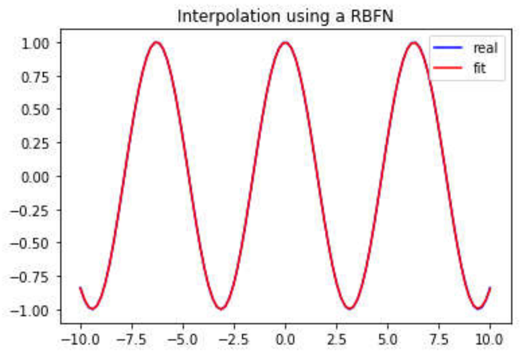


Figure.17. # of neurons: 20 & sigma: 0.2 (COS(X) mode)

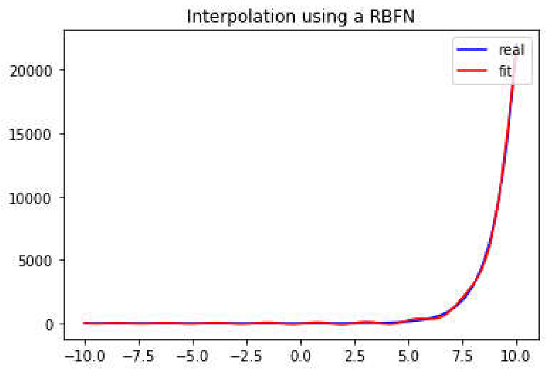


Figure.18. # of neurons: 20 & sigma: 0.2 (EXP(X) mode)

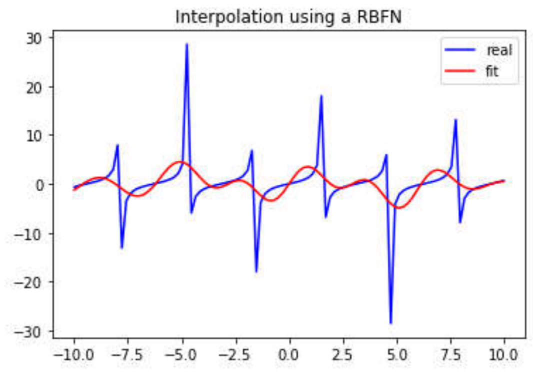


Figure.19. # of neurons: 20 & sigma: 0.2 (TAN(X) mode)

As it can bee seen, our model performs perfectly for COS(X) and EXP (X) but TAN(X) fails, we changed the parameters to see if we can improve the performance.

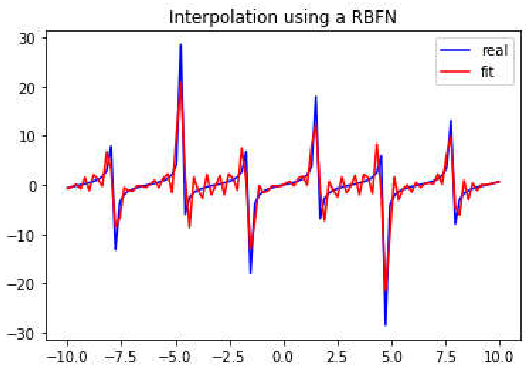


Figure.20. # of neurons: 200 & sigma: 1.0 (TAN(X) mode)

We observed that if we increase the number of neurons and sigma, we can improve the performance but our still could not be fitted perfectly.

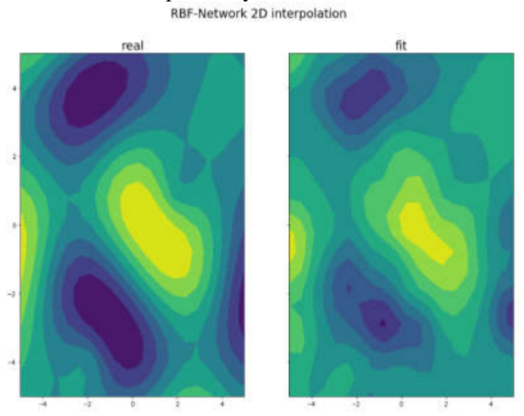


Figure.21. 2D representation (TAN(X) mode)

IV. CONCLUSION

In the discipline of machine learning, classification and regression are a quickly moving field. Its growth has been fueled by technological advances in digital imaging, computer processors and mass storage devices. In this paper an attempt is made to a famous neural network which is radial basis function network (RBFN). Here we tested how RBFN works and how the number of neurons will effect on performance. We number of neurons must be estimated based on problem because of its computational load we can't use as much as neurons that we want . Also it was tested that in classification RBF can perform as well as MLP. Then we compared our code with MATLAB's library, it seemed our model with a little difference performing worth in classification and MATLAB's library chooses more appropriate centers. And for final experiment, we implemented regression problem and fitted an RFB model and we tested how number of neurons and sigma are effecting the performance and our model despite its robustness, failed to solve TAN(X) perfectly. for future work, we are going to focus on this problem and improve the performance.

REFERENCES

[1] Broomhead, D. S.; Lowe, David (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks (Technical report). RSRE. 4148.

[2] Broomhead, D. S.; Lowe, David (1988). "Multivariable functional interpolation and adaptive networks" (PDF). Complex Systems. 2: 321–355.

[3] Schwenker, Friedhelm; Kestler, Hans A.; Palm, Günther (2001). "Three learning phases for radial-basis-function networks". Neural Networks. 14 (4–5): 439–458. CiteSeerX 10.1.1.109.312. doi:10.1016/s0893-6080(01)00027-2. PMID 11411631.

[4] T. Ahadli, Introduction to Regressions: Linear regression with Python (2018)

[5] T. Ahadli, A Friendly Introduction to K-Means Clustering algorithm (2020)

[6] Prof. G. Vachkov, Multistep Modeling for Approximation and Classification by Use of RBF Network Models (2016), Innovative Issues in Intelligent Systems

[7] T. Ahadli, C++/Python Codes for classification of MNIST Digits Data Set using RBFNN (2020)