Comparison of Various Algorithms

Ticao Zhang

April 25, 2018

Using the peaks functions please compare algorithms used in previous projects such as: NBN, SVM, ELM, DCT and ErrCor. Plot obtained surfaces and compare RMSE errors. Compare training time and network complexity.

1 Peaks function

First of all, we generate data file for peaks function placing data points on regular 30×30 grid. To generate data points we use the following equation

$$\begin{aligned} z = \quad & (0.3 - 1.8x + 2.7x^2)e^{-1-6y-9x^2-9y^2} \\ & - (0.6x - 27x^3 - 243y^5)e^{-9x^2-9y^2} - 1/30e^{-1-6x-9x^2-9y^2}; \end{aligned} \quad (1)$$

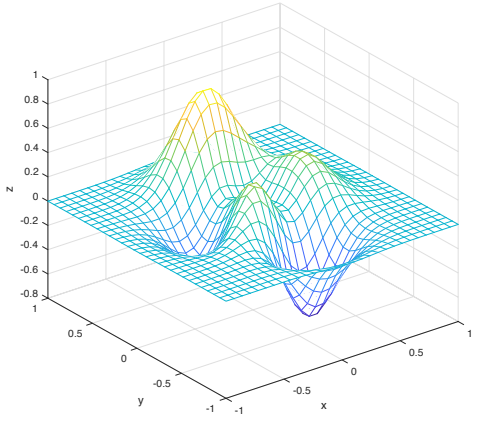

Figure 1: the original peaks function

2 Data preparation

2.1 Training data

Now we modify *prepare_data.m* script to generate $N_1 = 2000$ training points for the MATLAB peaks function. Note that the generated points are randomly located. The generated training peaks function is stored in $\mathbf{z} \in \mathbb{R}_1^N$ according to (1). Therefore, the average power for the transmitted signal is

$$p = \frac{\sum_{i=1}^{N_1} z_i^2}{N_1}. \tag{2}$$

We also add noise $\mathbf{n} \in \mathbb{R}_1^N$ to the generated training data, where $n_i \sim \mathcal{N}(0, \sigma^2)$. then the *signal-to-noise ratio* (SNR) can be defined as

$$\rho = 10 \log \frac{p}{\sigma^2} \tag{3}$$

and $\mathbf{z}' = \mathbf{z} + \mathbf{n}$.

In the training process, we set $\rho = 20$dB.

2.2 Testing data

Similarly, we also generate $N_2 = 2000$ testing data corrupted with noise. To plot the 2D surface based on the testing data, we use *scatteredInterpolant* function. Thus we get the desired testing surface.

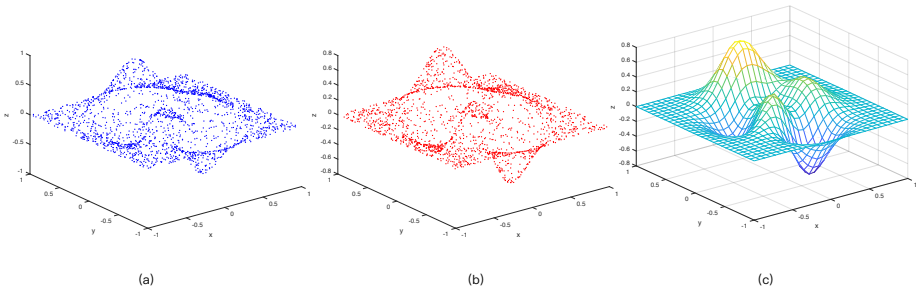

Figure 2: idea case. (a) training data (b) testing data (c) desired surface for testing (with noise)

In the following experiment, we will use the noise-corrupted training/testing data. The SNR is set as $\rho = 20$dB.

3 Algorithm description

In this section, we briefly give a short description of various algorithms. To compare the performance of different algorithms, we define the metric

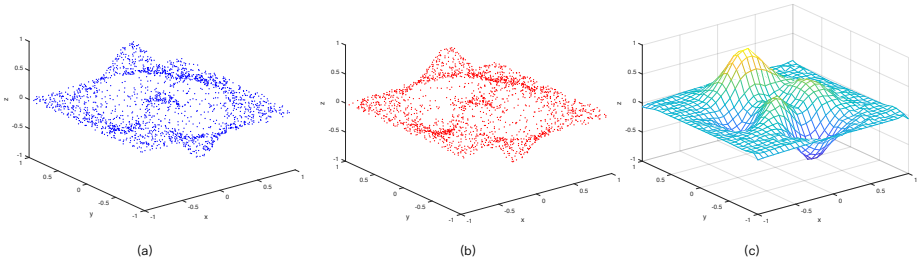

Figure 3: noisy case. (a) training data, $\rho = 20$dB (b) testing data, $\rho = 20$dB (c) generated testing surface

RMSE (root mean square error) as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}. \tag{4}$$

3.1 Support Vector Machine – SVM

SVM is a supervised learning model that analyzes data used for classification and regression analysis. In addition to performing linear classification, SVM can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

3.1.1 Parameter setting

We use the Radial basis function (RBF) kernel, where $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma||\mathbf{x} - \mathbf{x}'||^2\right)$. The parameter γ is the width of the RBF units and the parameter C is used to adjust a balance between accuracy and the number of support vectors. Using simple grid search method in Project 2, we find the optimal parameter that minimize the training error is $\gamma = 562$ and $C = 177$.

For the SVM method, we use function *svmtrain* to train the model and *svmpredict* for testing. The option parameters are set as *-s 3 -t 2 -h 0*, which means

- svm_type – epsilon-SVR
- kernel_type – radial basis function
- no use of the shrinking heuristics

3.1.2 Results

```
optimization finished, #iter = 1041
nu = 0.000115
obj = -6.452609, rho = -0.062118
nSV = 478, nBSV = 0
Mean squared error = 0.00530378 (regression)
Squared correlation coefficient = 0.92235 (regression)
Mean squared error = 0.00758613 (regression)
Squared correlation coefficient = 0.889584 (regression)
training time = 0.0803
training rms = 0.0728
testing rmst =  0.0871
C = 177.0000
gamma = 562.0000
```

It can be seen that the total iteration time is 893, the number of support vectors is 426, the number of class is 2.

3.2 Particle Swarm Optimization – PSO

particle swarm optimization (PSO) is a population based stochastic optimization technique inspired by the social behavior of bird flocking or fish schooling. The system is initialized with a population of random solutions and searches for optima by updating generations. Compared with genetic algorithm (GA), PSO is easy to implement and there are few parameters to adjust.

PSO is initialized with a group of random particles and then searches for optima by updating generations. In every iteration, each particle is updated by following two best values. The first one is the best solution (fitness) is has achieved so far (pbest). Another "best" value best value that is tracked bu the particle swarm optimizer is the best value, obtained so far by any particle in the population. Since it is a global best value, it is called gbest. The particle updates its velocity and positions with following two equations

$$\begin{aligned} v &= v + c_1 \cdot \text{rand}() \cdot (pbest - present) + c_2 \cdot \text{rand}() \cdot (gbest - present) \\ present &= present + v \end{aligned} \tag{5}$$

where v is the velocity, *present* is the current particle. rand() is a random number between $(0, 1)$. $c_1 = c_2 = 2$ are learning factors.

In artificial neural network (ANN), we can use PSO to replace back-propagation learning algorithm. It showed PSO is a promising method to train ANN. It is both faster and gets better results.

3.2.1 Parameter setting

PSO starts with a group of a randomly generated population, then has fitness values to evaluate the population, updates population and searches for the optimum with random techniques. However, they do not guarantee success.

The parameters include the number of particles, dimension of particles, range of particles, maximum change a particle can change, learning factor and the stop condition.

- maxite = 100;
- nd = 2; % dimensions
- nb = 30; % initial population (number of birds)
- nn = 5; % winners
- rad0 = 0.7; % initial radius

3.2.2 Results

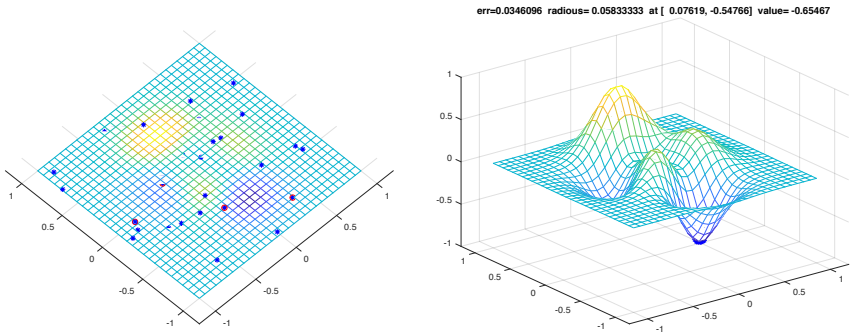

Figure 4: PSO is used to find the minimum of a peaks function

12 iteration is needed

3.3 Discrete Cosine Transformation – DCT

Discrete Cosine Transformation (DCT) can be used for the reduction of the system size. Training patterns are transformed by DCT algorithm to compute frequency coefficients. Then all frequency coefficients smaller than a threshold will be set as 0. After that, IDCT is applied to reduced set of coefficients. Therefore, only a small number of DCT coefficients is sufficient to reconstruct the surface.

A big problem of DCT is we need to convert randomly distributed patterns into a regular grid. One method is to use Matlab's *griddata* function. First of all, we normalize the input and output to [-1,1]. Then The *griddata* function uses several different methods for regular grid points. Then for each occurrence of not defined point, we use k nearest neighbors, where the value of the lacking point is computed as a weighted average from all nearest

neighbors. The weights are inversely proportional to the distance to the not defined point.

For ease of implementation, we simply use the matlab function *scattered-Interpolant* to generate grid points from the training data. Then we use this grid data to do DCT and get the corresponding coefficients. The coefficients are then used to reconstruct the surface. Finally, a comparison is used to get the training RMSE.

3.3.1 Parameter setting

We set the threshold for training RMSE as 0.2.

3.3.2 Results

The generated DCT coefficients are as follows

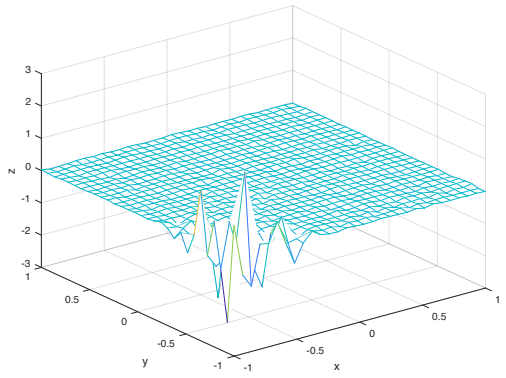

Figure 5: DCT coefficients for the gridded training data

```
training RMSE threshold = 0.2
training RMSE = 0.19539
number of non-zero coefficients = 44
compression ratio = 0.048889
testing RMSE = 0.20135
traing_time = 0.0077192
testing_time = 0.00028227
```

In DCT, the number of neurons is equal to the number of frequency coefficients retained after process of its reduction. Therefore, we only need 44 DCT coefficients.

3.4 Improved Second Order – ErrCor

This Improved Second Order (ISO) algorithm is derived from NBN algorithm and improved Levenberg marquardt algorithm is used for traditional

neural network training. In the proposed approach, all the parameters such as input weight, output weights, centers and widths are adjusted by second order update rule to train the RBF network.

The main difference between NBN and ErrCor is the type of activation function and the network architecture.

3.4.1 Parameter setting

The parameters are set as follows

- maximum_iteration = 30;
- maximum_error = 0.00001;
- maximum_D_error = 0.01;
- rad0=0.1;
- ww0=1;
- mu = 1;

3.4.2 Results

```
#RBFs =   1, iter =   3, RMSE =0.13116 RMSEv =0.13049
train_time = 0.03560 test_time = 0.00105
#RBFs =   2, iter =   7, RMSE =0.09289 RMSEv =0.09277
train_time = 0.09758 test_time = 0.000954
#RBFs =   3, iter =   5, RMSE =0.08121 RMSEv =0.08067
train_time = 0.08157 test_time = 0.0012374
#RBFs =   4, iter =   4, RMSE =0.05641 RMSEv =0.05720
train_time = 0.07066 test_time = 0.0017094
#RBFs =   5, iter =   5, RMSE =0.03637 RMSEv =0.03748
train_time = 0.10606 test_time = 0.0022986
#RBFs =   6, iter =   9, RMSE =0.03624 RMSEv =0.03740
train_time = 0.26796 test_time = 0.0023865
#RBFs =   7, iter =   8, RMSE =0.03800 RMSEv =0.03884
train_time = 0.25723 test_time = 0.0028698
#RBFs =   8, iter =  10, RMSE =0.02790 RMSEv =0.02818
train_time = 0.35866 test_time = 0.003162
#RBFs =   9, iter =  10, RMSE =0.02377 RMSEv =0.02405
train_time = 0.40535 test_time = 0.0036764
#RBFs =  10, iter =   8, RMSE =0.02154 RMSEv =0.02212
train_time = 0.33836 test_time = 0.0039647
#RBFs =  11, iter =  10, RMSE =0.02071 RMSEv =0.02077
train_time = 0.45389 test_time = 0.0042025
#RBFs =  12, iter =  16, RMSE =0.02028 RMSEv =0.01982
train_time = 0.83513 test_time = 0.0048028
#RBFs =  13, iter =   5, RMSE =0.02012 RMSEv =0.02042
train_time = 0.24414 test_time = 0.0057493
#RBFs =  14, iter =   8, RMSE =0.01988 RMSEv =0.02073
```

```
train_time = 0.45673  test_time = 0.0052449
#RBFs =  15, iter =   9, RMSE =0.02719 RMSEv =0.02768
 train_time = 0.59828  test_time = 0.0062451
```

It can be seen that only a small number of neurons, e.g. 10, can make this algorithm achieves a small RMSE error 0.02212. Besides that, the training time is very short.

3.5 Extreme Learning Machine – ELM

ELM us characterized by a large number of randomly generated neurons and training only at the output layer. A large number of neurons are required for this technique, but it saves drastically on computation time.

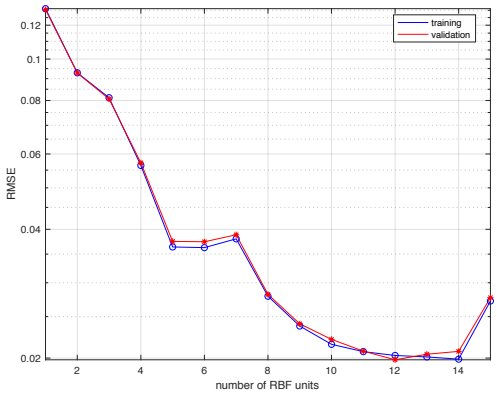

Figure 6: RMSE versus the number of neurons

3.5.1 Parameter setting

The parameters are set as follows

- H – number of hidden neurons
- N0 – Number of initial training data used in the initial phase of OSLEM
- B – size of block data learned by OSELM in each step

We set $H = 1000$, $N0 = 10000$ and $B = 1000$. Besides that, *elm_type* is set to 0 for regression and *activation function* is set to radial basis function (rbf).

3.5.2 Results

```
hidden neurons = 1000
Activation fun = 'rbf'
initial training=1000
Block size =   1000
TrainingTime = 1.57
TestingTime = 0.07
Training rms = 0.021229
Testing rmst = 0.022828
```

The number of neurons is 1000. ELM provides low testing error with little training required. However, the resulting network sizes must be very large to produce effective results. The trade-off between error and network size is difficult and dependent on the requirement for a specific scenario.

3.6 Neuron by Neuron – NBN

Traditional second order method (Newton) is fast but has two problems, firstly, it is difficult to find Hessian, secondly, in most cases (nonlinearity) Newton method will not work. Steepest descent method is more stable but not fast.

Levenberg-Marquardt (LM) algorithm combines the speed of Newton algorithm with the stability of the steepest descent method by introducing an approximation for Hessian matrix. It uses the following formula to calculate the weights in subsequent iterations

$$\mathbf{w}_{k+1} = \mathbf{w}_k - (\mathbf{J}_k^T \mathbf{J}_k + \mu \mathbf{I})^{-1} \mathbf{J}_k^T \mathbf{e}. \tag{6}$$

where $\mathbf{e}$ is the cumulative error vector on outputs, $\mathbf{I}$ is identity unit matrix, μ is a learning parameter and $\mathbf{J}$ is Jacobin of the output errors. However, LM algorithm only fits for multi-layer perceptron (MLP) network.

Neuron by Neuron (NBN) algorithm with forward-backward computation is developed based on LM algorithm, but it can handle arbitrarily connected neuron (ACN) networks. It uses quasi Hessian to approximate the gradient matrix in error back propagation (EBP) algorithm.

3.6.1 Parameter setting

In the training process, the maximum iteration time is set to 50, maximum required error is 0.001. Other parameters are set as default. We use a bipolar neuron.

In the testing process, threshold for success rate evaluation is set as 0.09. Other parameters are set as default.

We vary the number of neurons in the network architecture and find that when the number of neurons is 10, the best RMSE for testing is achieved.

3.6.2 Results

The number of neurons is 10. The best RMSE for testing that can be achieved is in trial 4. We print the results here.

```
10 neurons 'FCC'
neuron # 3 connected to: # 1, # 2,
neuron # 4 connected to: # 1, # 2, # 3,
neuron # 5 connected to: # 1, # 2, # 3, # 4,
neuron # 6 connected to: # 1, # 2, # 3, # 4, # 5,
neuron # 7 connected to: # 1, # 2, # 3, # 4, # 5, # 6,
neuron # 8 connected to: # 1, # 2, # 3, # 4, # 5, # 6, # 7,
neuron # 9 connected to: # 1, # 2, # 3, # 4, # 5, # 6, # 7, # 8,
neuron #10 connected to: # 1, # 2, # 3, # 4, # 5, # 6, # 7, # 8,
    # 9,
neuron #11 connected to: # 1, # 2, # 3, # 4, # 5, # 6, # 7, # 8,
    # 9, #10,
neuron #12 connected to: # 1, # 2, # 3, # 4, # 5, # 6, # 7, # 8,
    # 9, #10, #11,
ni=>  2 - number of inputs,
no=>  1 - number of outputs,
nn=> 10 - number of neurons
trial  1, training RMSE=0.0373, time=1.6357, testing RMSE
    =0.0380,  time=0.0051
trial  2, training RMSE=0.0319, time=1.6010, testing RMSE
    =0.0333,  time=0.0045
trial  3, training RMSE=0.0415, time=1.4803, testing RMSE
    =0.0428,  time=0.0045
trial  4, training RMSE=0.0282, time=1.4253, testing RMSE
    =0.0298,  time=0.0045
trial  5, training RMSE=0.1306, time=0.2894, testing RMSE
    =0.1281,  time=0.0046

10 neurons FCC,   1 trials
average training time=1.5033, RMSE=0.0339, std=0.0000
average testing time=0.0078, RMSEt=0.0339, std=0.0000
when threshold=0.0900, success rate=1.0000
```

4 Performance comparison

4.1 Obtained surface

We use $N_2 = 2000$ points for testing and compute the corresponding RMSE. However, to generate the corresponding surface, we use a trick here. With the generated N_2 points, *scatteredInterpolant* function helps generate a 30×30 grid surface. Different reconstructed surface is plotted here.

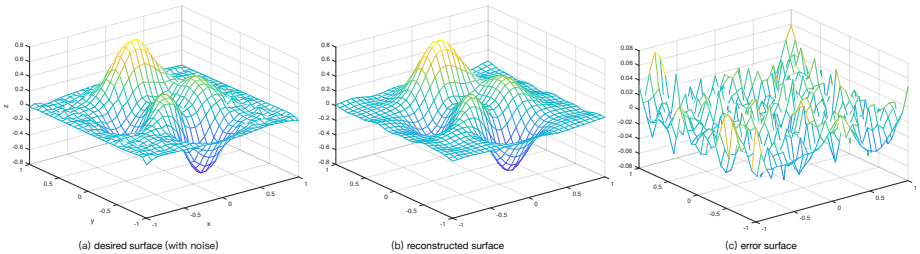

Figure 7: DCT

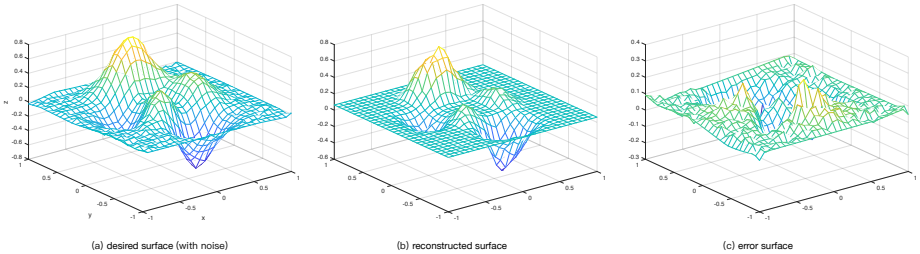

Figure 8: SVM

4.2 RMSE

Algorithm	Network complexity	training time (s)	RMSE for training	RMSE for testing
NBN	10 neurons	1.4253	0.0282	0.0298
ELM	1000 neurons	1.57	0.021229	0.022828
DCT	44 neurons	0.0077192	0.19539	0.20135
PSO	–	–	–	–
ErrCor	10 neurons	0.33836	0.02154	0.02212
SVM	426 neurons	0.078937	0.069199	0.094753

4.3 Remarks

For DCT, The number of stored parameters is equal to the number of nonzero coefficients.

For SVM, nodes are equal to the number ofRBF units. parameters needed to store are computed as $P = n_w + n_c + n_d$ where: $n_w = n_{\text{RBF}} + 1$ the number of weights, $n_c = 2n_{\text{RBF}}$ the number of center locations of RBFs and $n_d = 1$ the number of RBF diameters values.

For ELM and ErrCor, nodes are equal to the number ofRBF units. parameters needed to store are computed as $P = n_w + n_c + n_d$ where: $n_w = n_{\text{RBF}} + 1$ the number of weights, $n_c = 2n_{\text{RBF}}$ the number of center locations of RBFs and $n_d = n_{\text{RBF}}$ the number of RBF diameters values.

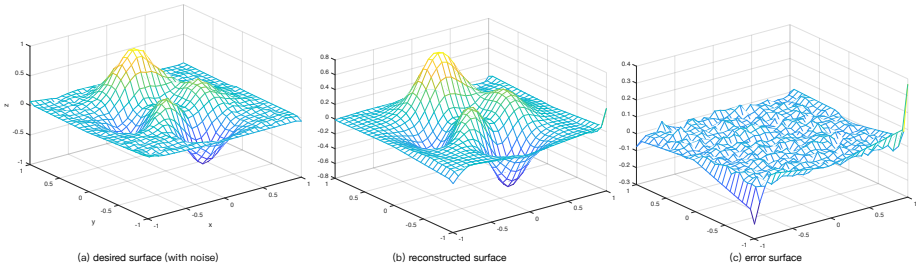

Figure 9: ELM

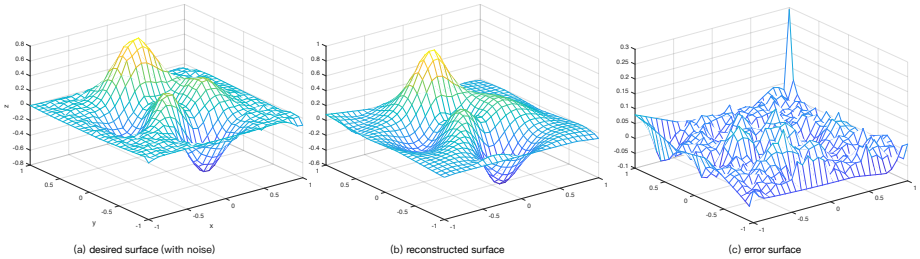

Figure 10: NBN

For NBN, different architectures have different storages. For example, in ANN-SLP, nodes are equal to number of neurons in hidden layer n_h. $P = n_w$, $n_w = (n_i + 1)n_h + n_h + 1$ with n_i is the number of inputs, $n_i = 2$ for this case. In ANN-FCC, nodes are equal to all neurons n_w and $P = n_w$.

- The training time for SVM should include the time to find the optimal parameter γ and C. It takes a long time to find the optimal parameter with the simple *grid search* method.

- The number of hidden neurons for ELM should be large enough, or ELM will not have a good performance. The larger B is, the less training time will be. Therefore, ELM may be more fitful to large networks.

- NBN only needs 10 neurons. However, this algorithms also has flaws. One problem is that Hessian matrix inversion needs to be calculated each time for weight updating. For large networks, this inversion conversion is going to be a disaster and the speed gained by second order approximation may be totally lost. Another problem is that Jacobian matrix has to be stored for computation. Sometimes the memory cost for Jacobian matrix storage may be too huge to be practical.

- ErrCor is an improved second order algorithm in which RBF networks can be designed very compactly; at the same time, the network per-

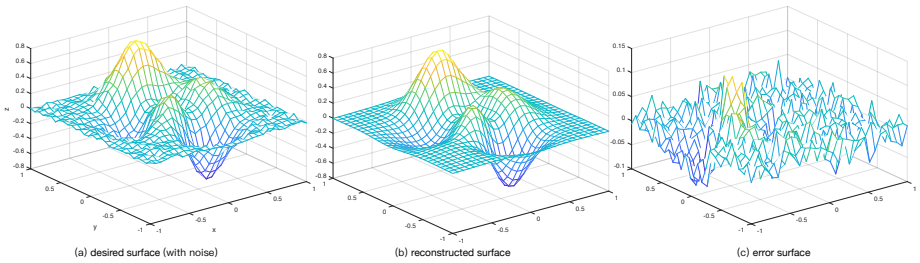

Figure 11: ErrCor

formance such as training speed and approximation accuracy is improved. The LM algorithm used in NBN is not suitable for problems with large number of patterns. This limitation of LM is eliminated by using a different matrix multiplication routine, which replaces storage and computation of Jacobian matrix with Jacobian vector.

- Note that the RMSE for DCT can be adjusted manually. A large number of DCT coefficients would incur a sufficient low RMSE. Thus, DCT needs to arrive a tradeoff between the number of neurons and the RMSE. Despite the DCT process is fast, a big problem is to find the optimal number of neurons that satisfy the RMSE threshold for training, the algorithms needs to adjust steps which takes a long time. Last, DCT only operates on gridded input. It lacks generality. Numerical methods can be applied to make the input data gridded, unfortunately, this also brings distortion, especially for non-smooth input data.

- PSO is used to find a minimal of an unknown function

5 Conclusion

As the comparison results shown in table 4.2, ErrCor can reach smaller training/testing errors than other algorithms, with very compact RBF network consisting of a very small number of RBF units. Compact RBF networks benefit the design in two aspects. First of all, compact architecture could be more efficient for hardware implementation. On the other hand, the less number of RBF units for design, the better generalization ability the trained network can obtain.

The compression nature of DCT can significantly reduces the system size with controlled lost of precision. The reduced data can be used for further manipulation like approximation or data conversion.