# A COMPREHENSIVE STUDY OF ELM AND ITS DERIVATIVE NETWORKS ON MULTIPLE DATASET

**A REPORT**

**PRESENTED TO**

**THE FACULTY OF THE GRADUATE SCHOOL**

**AT THE UNIVERSITY OF MISSOURI**

**IN PARTIAL FULFILLMENT**

**OF THE REQUIREMENTS FOR THE DEGREE**

**MASTER OF SCIENCE**

**BY**

**WANG, XINGYU (MU-STUDENT)**

**DR. JAMES KELLER, PROJECT SUPERVISOR**

**DECEMBER 2017**

## • Table of Contents

# INTRODUCTION

## Overview

In 2006, Guang-Bin Huang proposed an Extreme Learning Machine [1] where a single layer feedforward neural network is used for classification or regression. The weights connecting the input layer and hidden layer are randomly assigned and never updated, the weights between the hidden layer and the outputs are learned in a single step. ELM training can be hundreds of thousands times faster than gradient descent-based learning methods. Also, it reaches the smallest norm solution among all the least squares.

In this study, ELM and its derivative networks will be tested and discussed. This will include:

1.Testing the performance of ELM when different random distributions are used to generate weights and biases.

2. Testing the performance of two derivative ELM networks, Self-Adaptive Evolutionary Extreme Learning Machine (which applies differential evolutionary algorithm on ELM) and Voting Based Extreme Learning Machine (which applies a voting method on ELM).

3. Testing the performance of ELM by real world collecting data.

## BACKGROUND

Gradient descent-based learning methods have been widely used in many fields due to their good performance. They calculate the loss function with respect to the weights, then iteratively change the weights to approximate ideal functions. But they are generally very slow due to gradient disappearance issues or converge to local minima. To address these issues, Huang proposed a single hidden layer neural network(SLFN) where the hidden layer need not to be changed [1]. The network was called Extreme Learning Machine(ELM). Before introducing the structure of the ELM, the structure of the SLFN should be mentioned first:

### Signal hidden layer neural network

For K arbitrary district sample $(x_i, t_i)$, where $x_i \in R^n$ and $o_i \in R^m$, standard SLFNs with L hidden nodes and activation function g(x) is like:

$$f_l(x) = \sum_{i=1}^{L} \beta_i \, g\left(w_i x_j + b_i\right) = o_j \qquad (1)$$

Where $\mathrm{j} = 1 \ldots, \mathrm{K}$

$w_i \in R^n$ is the weight connecting the ith hidden node and the input nodes. $\beta_i \in R^m$ is the weight vector connecting the ith hidden nodes and output nodes, $b_i$ is a bias of the ith hidden node. The above equation can be written as:

$$\mathrm{H}\beta = \mathrm{O} \tag{2}$$

Where $\mathrm{H}(w_1 \ldots w_L, b_1 \ldots b_L, x_1 \ldots x_k)$

$$= \begin{bmatrix} g(w_1, b_1, x_1) & \cdot \quad \cdot \quad \cdot & g(w_L, b_L, x_1) \\ \vdots & \cdot \quad \cdot \quad \cdot & \vdots \\ g(w_1, b_1, x_K) & \cdot \quad \cdot \quad \cdot & g(w_L, b_L, x_K) \end{bmatrix}_{(K\times L)}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix} \ \beta \in R^{L\times m} \quad \text{and} \quad \mathrm{O} = \begin{bmatrix} O_1^T \\ \vdots \\ O_L^T \end{bmatrix} \ \mathrm{O} \in R^{K\times m} \tag{3}$$

## Extreme learning machine

If change equation (2) to:

$$\mathrm{H}\beta = \mathrm{T} \tag{4}$$

Where weights and biases in $\mathrm{H}(w_1 \ldots w_L, b_1 \ldots b_L, x_1 \ldots x_K)$ are randomly generated and never chance, $\mathrm{T} \in R^{K\times m}$ is the label of the data set which we already know.

Solutions exist, and The Moors-Penrose pseudoinverse can be adopted in here to approximate a best fit (least square) solution ($\beta$).

$$\beta = H^{+}T \tag{5}$$

Where $H^{+}$ is the Moore-Penrose generalized inverse of matrix H

Huang proposed ELM based on the below two theories [1].

Theory 1. Given a standard SLFN with N hidden nodes and activation function g: which is infinitely differentiable in any interval, for N arbitrary distinct samples $(x_i, t_i)$, where $x_i \in R^n$ and $t_i \in R^m$, for any $w_i$ and $b_i$ randomly chosen from any intervals of $R^n$ and R, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix H of the SLFN is invertible and $\|H\beta - T\| = 0$

Theory 2. Given any small positive value $\varepsilon > 0$ and activation function g : which is infinitely differentiable in any interval, there exists $\tilde{N} \leq N$ such that for N arbitrary distinct samples $(x_i, t_i)$, where $x_i \in R^n$ and $t_i \in R^m$, for any $w_i$ and $b_i$ randomly chosen from any intervals of $R^n$ and R, respectively, according to any continuous probability distribution, then with probability one, $\|H_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - T_{N\times m}\| = 0$

Proofs of the two theories [1] show that if the hidden nodes number is high enough, SLFN can approximate any continuous function.

The ELM training algorithm can be summarized as follows:

Step 1 Randomly assign the hidden node parameters, e.g., the input weights $w_i$ and biases $b_i$ for additive hidden nodes, i = 1,…,L.

Step 2 Calculate the hidden layer output matrix **H**

Step 3 Obtain the least square solution by equation (5).

## Self-Adaptive Evolutionary Extreme Learning Machine(SaE-ELM)

In 2012, Jiuwen Cao, Zhiping Lin and Guang-Bin Huang proposed Self-Adaptive Evolutionary Extreme Learning Machine [2]. It's the combination of the Evolutionary Extreme Learning machine(E-ELM) and the Differential Evolution Extreme Learning Machine(DE-LM). SaE-ELM applies the self-adaptive differential evolutional algorithm to optimize the network input weights and hidden node biases. It overcomes the limitations of manually choosing trial vector generation strategies. The general procedures of SaE-ELM are demonstrated as follows.

Given a set of training data and L hidden nodes with an activation function g(·), we have:

Step 1. Initialization

A set of NP vectors, where all the network hidden node parameters are initialized as the populations of the first generation.

$$\boldsymbol{\theta}_{k,G} = [a^T_{1,(k,G)}, \quad \dots \quad a^T_{L,(k,G)}, \quad b^T_{1,(k,G)}, \quad \dots \quad b^T_{L,(k,G)},] \qquad (6)$$

Where $a_j$ and $b_j$ (j = 1,...,L) are randomly generated and G is the generation and k = 1,2...,NP.

Step 2. Calculations of output weights and RMSE

Calculate the network output weight matrix and root mean square error (RMSE) with respect to each population vector with the following equations, respectively

$$\beta_{k,G} = \mathbf{H}^{\dagger}_{k,G}\mathrm{T} \qquad (7)$$

$$RMSE_{k,G} = \sqrt{\frac{\sum_{i=1}^{N} || \sum_{j=1}^{L} \beta_j g(a_{j,(k,G)}, b_{j,(k,G)}, x_i) - t_i ||}{L \times N}} \qquad (8)$$

In the first generation, the population vector with the best RMSE is stored as $\theta_{\text{best},1}$ and $\text{RMSE}_{\text{best},1}$.

Step 3. Mutation and Crossover

The trial vector generation strategy is chosen from a candidate pool constructed by four strategies.

St 1: DE/rand/1

$$v_{i,G} = \boldsymbol{\theta}_{r_1^i,G} + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G})$$

St 2: DE/rand-to-best/2

$$v_{i,G} = \boldsymbol{\theta}_{r_1^i,G} + \mathrm{F} \cdot \left(\boldsymbol{\theta}_{best,G} - \boldsymbol{\theta}_{r_1^i,G}\right) + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}) + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_4^i,G} - \boldsymbol{\theta}_{r_5^i,G})$$

St 3: DE/rand/2

$$v_{i,G} = \boldsymbol{\theta}_{r_1^i,G} + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}) + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_4^i,G} - \boldsymbol{\theta}_{r_5^i,G})$$

St 4: DE/current-to-rand/1

$$v_{i,G} = \boldsymbol{\theta}_{i,G} + \mathrm{K} \cdot (\boldsymbol{\theta}_{r_1^i,G} - \boldsymbol{\theta}_{i,G}) + \mathrm{F} \cdot (\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G})$$

Where $\theta_{\mathrm{i,G}}$ is a vector at current generation, the indices $r_1^i\, r_2^i\, r_3^i\, r_4^i\, r_5^i$ are mutually exclusive integers randomly generated within the range $[1,2,\cdots,\mathrm{NP}]$, F is used to control the scaling of the difference vectors and is usually selected within the range from 0 to 2. The control parameter K is randomly generated within the region $0 \leq \mathrm{K} \leq 1$.

Remark 1

Different vector generation strategies have their own advantages. “DE/rand/1” is suitable for solving multimodal problems, but this strategy usually demonstrates slow convergence speed. “DE/rand-to-best/2” performs well when dealing with unimodal problems. However, when solving multimodal problems, this strategy is more likely to get stuck at a local optimum and lead to a premature convergence. “DE/rand-to-best/2”and “DE/rand/2” could lead to a better perturbation than one-difference-vector-based strategies., but they also require a high computational cost. “DE/rand/1” is a rotation-invariant strategy, that is efficient in solving multi-objective optimization problems [2].

Opportunities to select each one of the strategies is $p_{l,G}$ where $p_{l,G}$ (l = 1,2,3,4) represents the probability that strategy l should be chosen at the Gth generation and the probability $p_{l,G}$ is updated in the following ways.

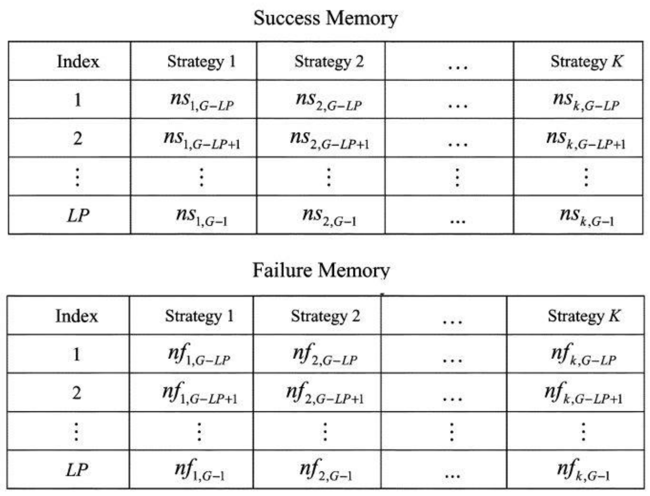
1. When G ≤LP, each strategy has the equal probability to be chosen
2. When G > LP, $p_{l,G} = \frac{s_{l,G}}{\sum_{l=1}^{4} s_{l,G}} \quad with\ s_{l,G} = \frac{\sum_{g=G-LP}^{G-1} ns_{l,g}}{\sum_{g=G-LP}^{G-1} ns_{l,g} + \sum_{g=G-LP}^{G-1} nf_{l,g}} + \varepsilon$ ,

   ((l =1,2,3,4)

where $ns_{l,g}$ denotes the number of trial vectors generated by the lth strategy at gth generations (iterations) that can successfully enter the next generation while $nf_{l,g}$ is the number of trial vectors generated by the lth strategy at gth generations (iterations) that are discarded in the next generation and ε is a small positive constant value to avoid the possible null success rate. LP is the learning period in paper [3]. Detail of LP will be denoted in remark 2.

Remark 2

If the probability of applying the kth strategy in the candidate pool to a target vector in the current population is $p_k$, k = 1,2...K, where K is the total number of strategies, the probabilities with respect to each strategy are initialized as 1/K. At the generation G, after evaluating all the generated trial vectors, the number of trial vectors generated by the kth strategy that can successfully enter the next generation is recorded as n$S_{k,G}$ while the number of trial vectors generated by the kth strategy that are discarded in the next generation is recorded as $nf_{k,G}$. LP (learning period) is the success and failure of memories to store these numbers within a fixed number of previous generations [3].



Success Memory

| Index | Strategy 1 | Strategy 2 | … | Strategy $K$ |
|---|---|---|---|---|
| 1 | $ns_{1,G-LP}$ | $ns_{2,G-LP}$ | … | $ns_{k,G-LP}$ |
| 2 | $ns_{1,G-LP+1}$ | $ns_{2,G-LP+1}$ | … | $ns_{k,G-LP+1}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $LP$ | $ns_{1,G-1}$ | $ns_{2,G-1}$ | … | $ns_{k,G-1}$ |

Failure Memory

| Index | Strategy 1 | Strategy 2 | … | Strategy $K$ |
|---|---|---|---|---|
| 1 | $nf_{1,G-LP}$ | $nf_{2,G-LP}$ | … | $nf_{k,G-LP}$ |
| 2 | $nf_{1,G-LP+1}$ | $nf_{2,G-LP+1}$ | … | $nf_{k,G-LP+1}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $LP$ | $nf_{1,G-1}$ | $nf_{2,G-1}$ | … | $nf_{k,G-1}$ |

Fig. 1 Success and Failure Memories

As illustrated in Fig. 1, at generation G, the number of trial vectors generated by different strategies that can enter or fail to enter the next generation over the previous LP generations are stored in different columns of the success and failure memories. Once the memories overflow after LP generations, the earliest records stored in the memories i.e. $\mathrm{n}s_{G-LP}$ or $\mathrm{n}f_{G-LP}$ will be removed so that those numbers calculated in the current generation can be stored in the memories.

A set of control parameters F and crossover rate CR are randomly generated for each target vector according to the normal distributions, respectively. For a given problem, proper values of CR usually fall into a small range, the mean value of CR is gradually adjusted according to previous CR values that have generated trial vectors successfully entering the next generation.

Step 4. Evaluation

All the trial vectors $\mathbf{u}_{k,G+1}$ generated at the (G+1)th generation are evaluated using the below equation:

1.$\boldsymbol{\theta}_{k,G+1} = \mathbf{u}_{k,G+1}$ if $RMSE_{\boldsymbol{\theta}_{k,G}} - RMSE_{\mathbf{u}_{k,G+1}} > \epsilon \cdot RMSE_{\boldsymbol{\theta}_{k,G}}$

2. $\boldsymbol{\theta}_{k,G+1} = \mathbf{u}_{k,G+1}$ if $|RMSE_{\boldsymbol{\theta}_{k,G}} - RMSE_{\mathbf{u}_{k,G+1}}| < \epsilon \cdot RMSE_{\boldsymbol{\theta}_{k,G}}$ and $||\beta_{k,G+1}|| < ||\beta_{k,G}||$

3. $\boldsymbol{\theta}_{k,G+1} = \boldsymbol{\theta}_{k,G}$ else

Where $\epsilon$ is the preset small positive tolerance rate.

Steps 3 and 4 are repeated until the preset goal is reached or the maximum learning iterations are completed. A validation set has no overlap with the training set used in the initialization step and is adopted in the evolutionary phase to avoid overfitting.

## Voting based extreme learning machine

Traditional ELM has its own drawback. Since the randomized hidden nodes are used and they remain unchanged during the training phase, some samples may be misclassified in certain realizations, especially for samples that are near the classification boundary. To fix the issue mentioned above and improve the classification performance of ELM. Voting based Extreme Learning Machine(V-ELM) was proposed by Jiuwen Cao, Zhiping Lin and Guangbin Huang in 2012 [4]. The proposed method incorporates the voting method into ELM in classification applications. The general procedures of V-ELM are as follows.

Step 1. Initiation

Several individual ELMs with the same number of hidden nodes and the same activation function in each hidden node are trained with the same dataset and the learning parameters of each ELM are randomly initialized independently.

Step 2. Voting

The final class label is then determined by majority voting on all the results obtained by these independent ELMs.

**Algorithm** V-ELM

```
Given:
  a training set $\aleph = \left\{(\boldsymbol{x}_n, \boldsymbol{t}_n) \middle| \boldsymbol{x}_n \in \mathbf{R}^d, \boldsymbol{t}_n \in \mathbf{R}^m\right\}_{n=1}^{N}$
  the hidden node output function $G(\boldsymbol{a}, b, \boldsymbol{x})$
  hidden node number $L$
  independent training number $K$
  zero valued vector $S_K \in \mathbf{R}^C$, $C$ is the number of classes
Training phase:
(1) Set $k = 1$
(2) while ($k \leq K$) do
        Randomly assign the learning parameters $\left(\boldsymbol{a}_j^k, b_j^k\right)$ $(j = 1, 2, \ldots, L)$
        of the $k$th ELM
(3)     Calculate the hidden layer output matrix $\mathbf{H}^k$
(4)     Calculate the output weight $\boldsymbol{\beta}^k$: $\boldsymbol{\beta}^k = (\mathbf{H}^k)^{\dagger}\mathbf{T}$, where $\mathbf{T}$ is the
        target output matrix
(5)     $k = k + 1$
(6) end while
Testing phase:
(1) for any testing sample $\boldsymbol{x}^{\text{test}}$,
(2) Set $k = 1$
(3)     while ($k \leq K$) do
            Using the $k$th trained basic ELM with learning parameters
            $\left(\boldsymbol{a}_j^k, b_j^k, \boldsymbol{\beta}_j^k\right)$ to predict the label of the testing sample $\boldsymbol{x}^{\text{test}}$, say,
            as $i$ where $i \in [1, 2, \cdots, C]$
(4)         Then $S_{K,\boldsymbol{x}^{\text{test}}}(i) = S_{K,\boldsymbol{x}^{\text{test}}}(i) + 1$
(5)         $k = k + 1$
(6)     end while
(7)     The final class label of the testing sample $\boldsymbol{x}^{\text{test}}$ is

            $$c^{\text{test}} = \arg \max_{i \in [1, \cdots, C]} \{S_{K,\boldsymbol{x}^{\text{test}}}(i)\}$$

(8) end for
```

# EXPERIMENT

## Experiment 1: Influence of random distribution

This experiment is divided into three sections. First, ELM performance was tested on a small dimensional dataset with weights and biases of ELM generated by different random distributions. Second, the same ELM was tested on a high dimensional dataset. Third, ELM was tested on a high dimensional dataset with weights and biases which were generated by different random distributions and concentrated at around 0.

### Section 1

Experiment Description: ELM was run on three sets of data (data 1, 2, 3). The data sets consist of 200 points of two-dimensional data and can be group into two clusters. Gaussian distribution function was applied to generate data. The mean of three sets are: [-3-3], [3 3] (dataset 1), [-1.8 -1.8] [1.8 1.8] (dataset 2) and [-0.6 -0.6] [0.6 0.6] (dataset 3). The variance of the data set is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

There were 400 data generated by gaussian distribution function. There were 200 were used to train ELM and another 200 used to test performance. The datasets were nominalized before being sent to a train or test machine.

Datasets were run with four ELMs multiple times, and each ELM's weights and biases were initialized with different distribution functions. They are: Normal distribution, where $\mu$=0 and $\sigma$=*1*. Uniform distribution, where maximal value is 1 and minimal value is -1. Binomial distribution, where number of trials=*1000* and probability of success for each trial=0.5(it will be normalized and extent to [-1 1]) and Student's t-distribution where degrees of freedom =3.

Choosing those parameters limit most of the sampling values in a range of [-1 1].

ELMs were run a total of 100 times to obtain 100 results. The mean and variance were calculated, the hidden layer neurons number was changed from 25 to 200 and the sigmoid activation function was used in network.

Ten percent of training and testing data was randomly exchange in each execution

## Data

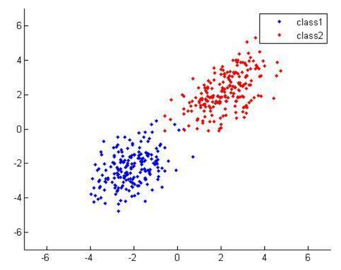
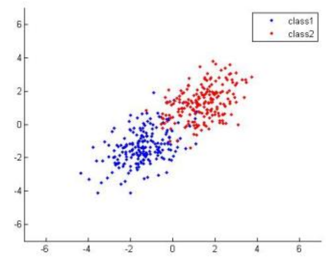
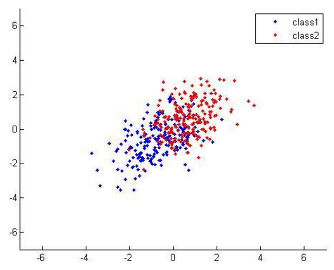
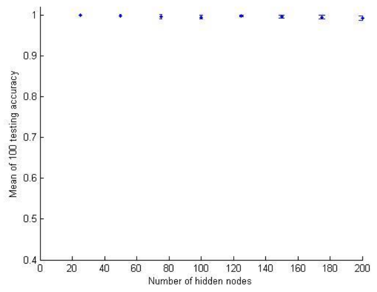
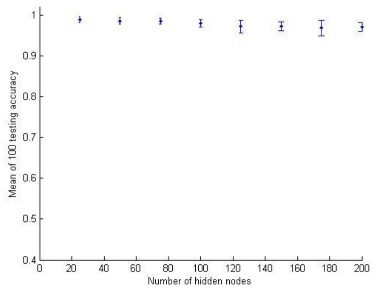
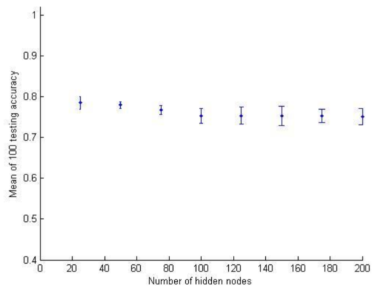
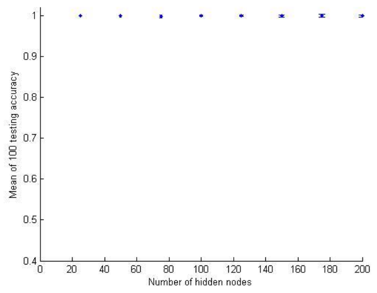
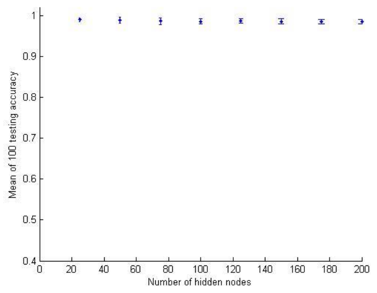
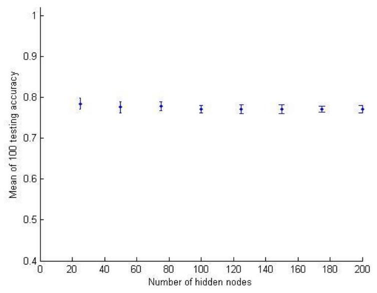
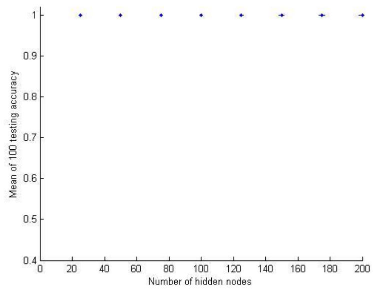
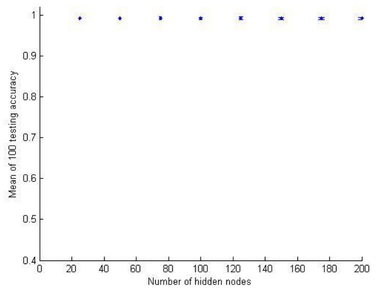
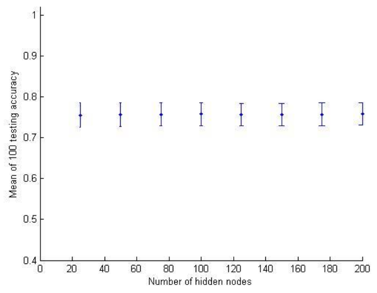
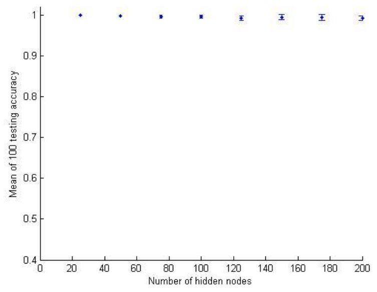
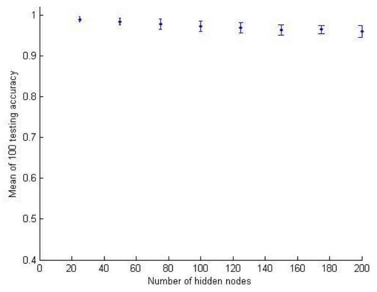
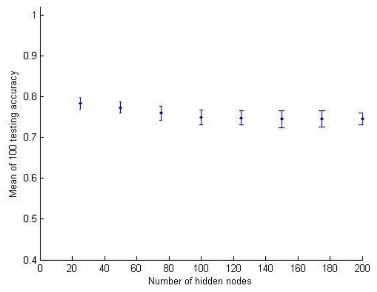
| Data1 | Data2 | Data3 |
|---|---|---|
|  |  |  |

Fig. 3 Training dataset

## The Result

| Distribution | Data1 | Data2 | Data3 |
|---|---|---|---|
| Normal distribution |  |  |  |
| *Uniform distribution* |  |  |  |
| Binomial distribution |  |  |  |
| Student's t-distribution |  |  |  |

- **Accuary is the mean of the 100 results**
- **Distance from point to upper bar or lower bar is the stand derivation of the 100 results**

Table. 1 ELM Result (small dimensional dataset)

The same data set was run by gradient descent algorithm [8] (gradient descent with momentum backpropagation was applied here) to judge performance. The network is a one hidden layer network. The number of hidden nodes are 50. Learning rate is 0.001, Momentum constant is 0.9, Maximum number of epochs to train is 100000 and Minimum performance gradient is 1e-4

| **Accuracy (%)** | | |
|---|---|---|
| Data 1 | Data 2 | Data 3 |
| 100 | 98.25 | 80.57 |

Table. 2 BP Result (small dimensional dataset)

## Discussion

The results of experiments with low dimensional datasets indicates that the binomial distribution has a relatively stable performance. Overfitting occurs when a model is excessively complex.

## Section 2

<u>Experiment Description</u>: ELM was run on three sets of data (data 1, 2, 3) that are almost the same as pervious data, but the dimension of the data set is changed to **1000**. The way to generate weights and biases is same as section 1. variance is a 1000*1000 matrix, where values of the diagonal line are 1 and values in another place are 0.

## The Result

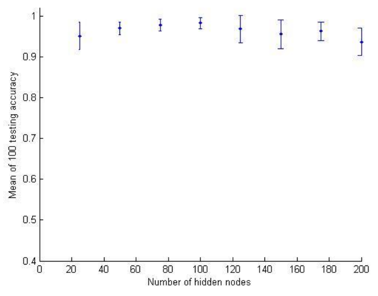
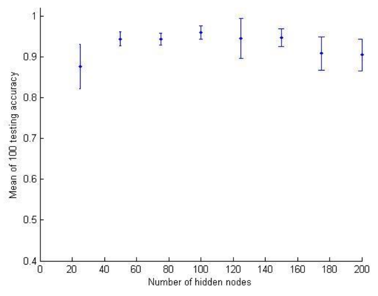
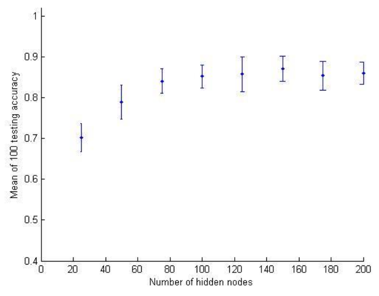
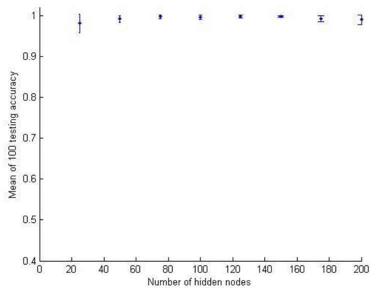
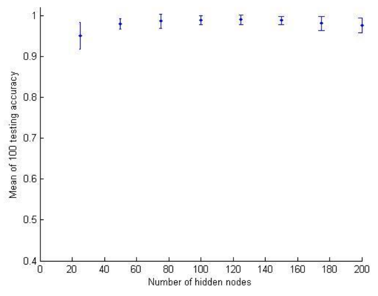
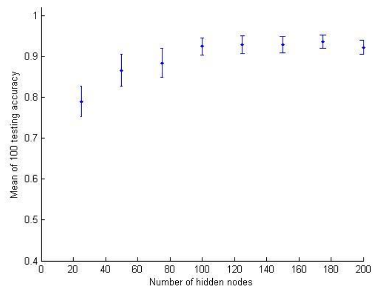
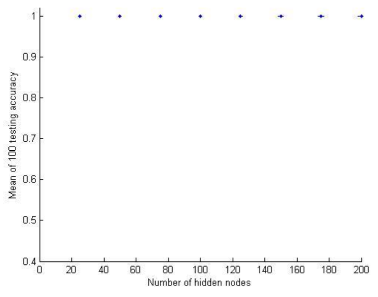
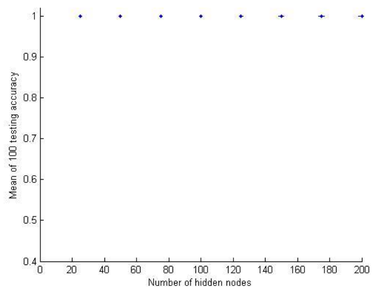
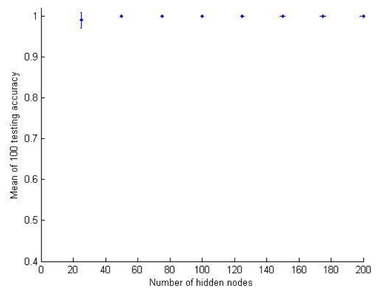
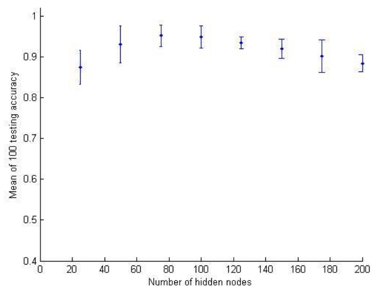
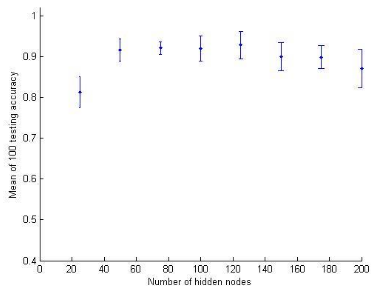
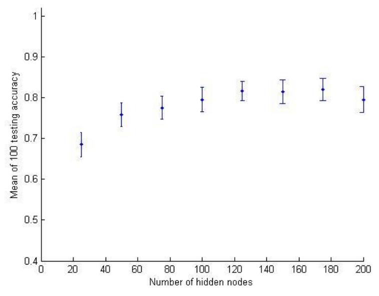
| Distribution | Data1 | Data2 | Data3 |
|---|---|---|---|
| **Normal distribution** |  |  |  |
| ***Uniform distribution*** |  |  |  |
| **Binomial distribution** |  |  |  |
| **Student's t-distribution** |  |  |  |
| • **Accuary is the mean of the 100 results**<br>• **Distance from point to upper bar or lower bar is the stand derivation of the 100 results** | | | |

Table. 3 ELM Result (large dimensional dataset)

ELMs were run 100 times to obtain 100 results. The mean and standard deviation were calculated, the hidden layer neurons number was changed from 25 to 200 and the sigmoid activation function was used in network.

Same as the section 1, the performance was judged by gradient descent algorithm. Parameters of the network maintains original value.

| **Accuracy (%)** | | |
|---|---|---|
| Data 1 | Data 2 | Data 3 |
| 100 | 100 | 98.95 |

Table. 4 BP Result (large dimensional dataset)

## Discussion

The results of experiments with high dimensional datasets indicates that the uniform distribution and normal distribution have almost the same performance. Binomial distribution has a stable performance and higher testing accuracy. T-distribution is unsteadiness and accuracy is relatively low.

## Section 3

Experiment Description: ELM was run on three sets of data (data 1, 2, 3). The three data sets are almost the same as pervious data. The dimension of the data set is **1000**. The way to generate weights and biases remains unchanged. But parameters of distribution function are changed. To the normal distribution, the $\mu = 0$ and $\sigma = 0.01$. To the uniform distribution, the range of the sampling values is limited in [-0.05 0.05]. So, weights values are highly concentrated around 0.

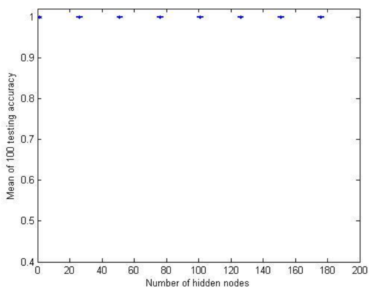
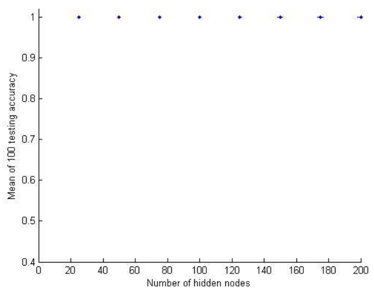
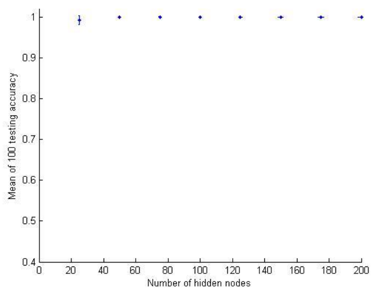
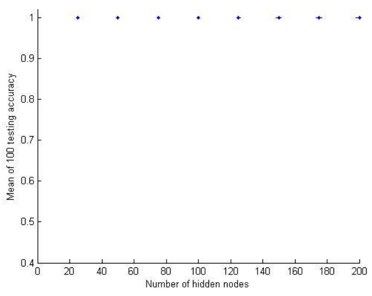
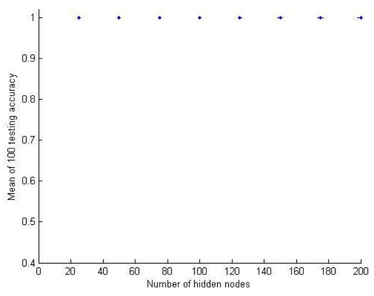
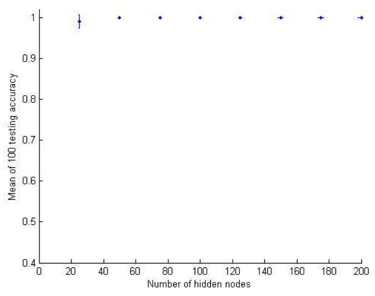
| Distribution | Data1 | Data2 | Data3 |
|---|---|---|---|
| Normal distribution |  |  |  |
| *Uniform distribution* |  |  |  |

Table. 5 ELM Result (large dimensional dataset, weights concentrated around 0)

## Analysis

This part contains two sections. Section 1 discusses the reasons why some distributions perform well. Section 2 discusses why overfitting will happen when a model is excessively complex.

### Section 1

This discussion returns to the original principles of ELM. In general, ELM is a network which uses the random projection in first layer and uses the least squared method in second layer. JL lemma plays an important role in random projection, it should be introduced in here:

Johnson and Lindenstrauss (JL) lemma [5]:

Given $0 < \varepsilon < 1$, a set X = $\{x_1, \cdots\cdots x_K\} \subset R^N,$ For any integer L > $\frac{4\ln(K)}{\frac{\varepsilon^2}{2} - \frac{\varepsilon^2}{3}}$, there is a linear map f: $R^N \rightarrow R^L$, y = Tx which can be found in randomize polynomial time, so that with probability (1 - $1/K^2$), $\forall (x_i, x_j) \in X \times X,$

$$(1 - \varepsilon) ||x_i - x_j||^2 \le ||y_i - y_j||^2 \le (1+\varepsilon) ||x_i + x_j||^2$$

The norm in equation is Euclidean, the integer L is the target dimension of the projection. The probability (1 - $1/K^2$) is close to 1 that it can be overlooked [6].

This theorem shows that, when a dataset is mapped from N space to L space by a random projection matrix, as long as L is big enough, **probabilistic speaking**, all or most of features of the dataset are retained.

Considering the ELM.

For K arbitrary district sample $(x_i, t_i)$, where $x_i \in R^N$ and $o_i \in R^m$, the number of the hidden layer neurons is L. The output of the first layer (H) is like:

$$\begin{bmatrix} g(w_1, b_1, x_1) & & g(w_L, b_L, x_1) \\ g(w_1, b_1, x_2) & \cdots & g(w_L, b_L, x_2) \\ g(w_1, b_1, x_3) & & g(w_L, b_L, x_3) \\ g(w_1, b_1, x_4) & & g(w_L, b_L, x_4) \\ \vdots & \cdots & \vdots \\ g(w_1, b_1, x_K) & \cdots & g(w_L, b_L, x_K) \end{bmatrix}_{(K \times L)}$$

Matrix H

Each of $g(w_j, b_j, x_i)$ is $g([w_{j1} * x_{i1} + w_{j2} * x_{i2} \ldots + w_{jN} * x_{iN}] + b_j)$ Where i = 1…K, j=1… L.

The original matrix is the random projection of the total data, where weights and biases map the K arbitrary district sample from N space to L space.

$$\begin{bmatrix} (w_1, b_1, x_1) & & (w_L, b_L, x_1) \\ (w_1, b_1, x_2) & \cdots & (w_L, b_L, x_2) \\ (w_1, b_1, x_3) & & (w_L, b_L, x_3) \\ (w_1, b_1, x_4) & & (w_L, b_L, x_4) \\ \vdots & \cdots & \vdots \\ (w_1, b_1, x_K) & \cdots & (w_L, b_L, x_K) \end{bmatrix}_{(K \times L)}$$

Original matrix

The matrix is inputted into the activation function g (e.g. sigmoid) to generate matrix H. The value of each element in each vector maybe increased or decreased during the process but features of the dataset are retained since the activation function is a continuous function (ELM requires that an activation function must be a continuous function).

After that, Moore-Penrose pseudo inverse is applied to classify matrix H. (The least squares method)

The principle of the ELM can be concluded as: random projection plus least squares method.

Now, considering each neuron's situation in ELM. To make it clear, a single hidden neuron ELM is applied to classify the dataset, under this situation, the size of matrix H will be $\mathrm{K} \times 1$. （one neuron output to all data).

Randomly picking up two values in the matrix H, if they are generated by same class dataset, their distance is designed as $\mathrm{d}_{sa}$. If they are generated by different class dataset, their distance is designated as $\mathrm{d}_{di}$. **Statistically speaking**, the probability of $\mathrm{d}_{sa} < \mathrm{d}_{di}$ is bigger than probability of $\mathrm{d}_{di} < \mathrm{d}_{sa}$. Thus, outputs of the hidden neurons tend to cluster and spread over an uncertain area.

Sometimes, some of neurons are unable to classify or even misclassify the dataset, luckily, L stochastic equations (L randomly features) are applied in ELM, the influence of those neurons will be offset by other neurons.

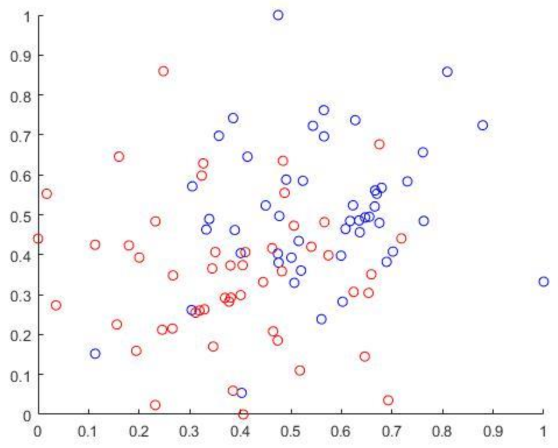
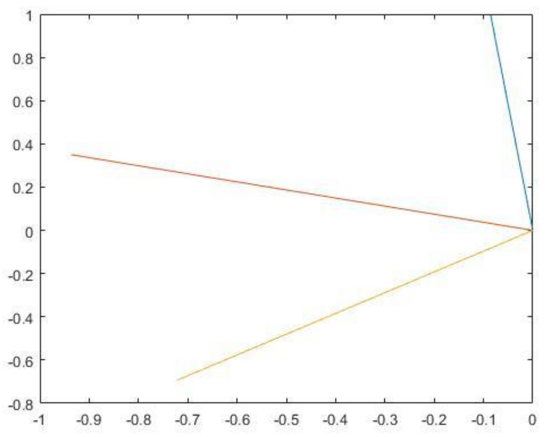
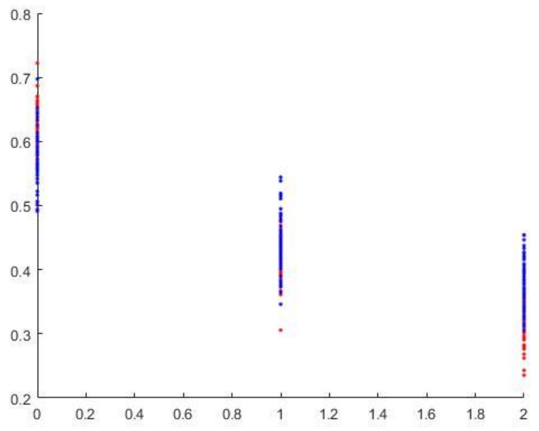
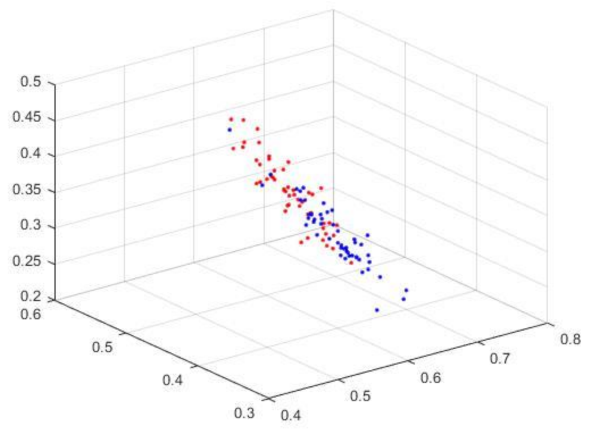
| **Red data from dataset one, blue data from dataset two** | |
|---|---|
| Dataset | Random map vectors (hidden nodes) |
|  |  |
| Values of output of the hidden neurons $\boldsymbol{g}(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{x})$ (sigmoid fuction) | The H matrix |
|  |  |

Table. 6 (figure.1 dataset, figure 2 random vectors (hidden nodes), figure.3 output of the hidden neurons( $g(w, b, x)$), figure.4 The H matrix)

(e.g. ELM with three hidden nodes was used to classify the gaussian distribution dataset. Outputs of the hidden neurons were showed in figure.3. Node 2 is useless since two classes data's outputs are too dispersed and means of them are too close. But node 1 and node 3 are useful on some degree. Figure 4 shows the H matrix. The influence of the node 2 was offset by node 1 and node3. Clearly, dataset can be classified in this space)

The situation that parameters of the equation $[w_{j1} * x_{i1} + w_{j2} * x_{i2} \dots + w_{jN} * x_{iN}] + b_j$ are simple (when the relative variability of each data is relatively small) is discussed below.

Assuming we have a one neuron ELM and the vector w in ELM follows the binomial distribution, where n =1 and p = 0.5. We want to use it to classify data [1 0] and [0 1].

Since vector w follows the binomial distribution, weights only have four possible results: [1 0] [0 1] [0 0] [1 1]. When we input data [1 0] and [0 1] into the linear equation, if w is [0 0] or [1 1], results have no difference. The network can't separate data. Under this condition, new neurons should be added until new weight is randomly assigned to [1 0] or [0 1]. But if uniform distribution is adopted to generate weights, the weights have very little chance of being assigned to a vector in which every element in the vector is the same. To this issue, each combination without this situation separates data well.

From another perspective, limiting parameter's variability broken the principle of the random projection.

The below shows the performance of binomial distribution on data set 3 (1000-dimensional data set) with different n and p.

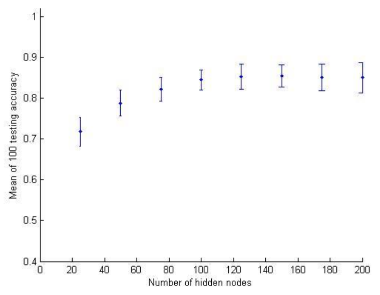
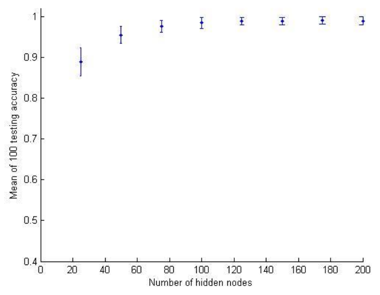
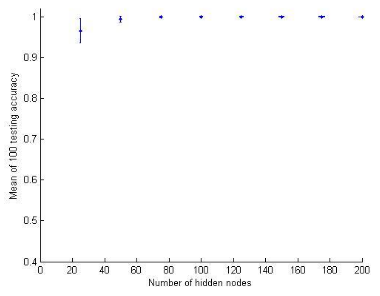
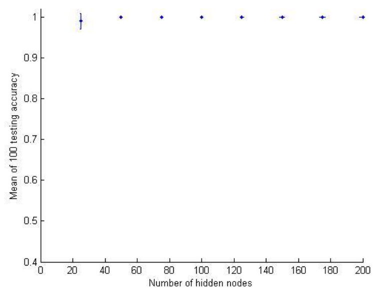
| ELM with Binomial distribution and same dataset | |
|---|---|
| **n =1 and p = 0.5** | **n =10 and p = 0.5** |
|  |  |
| **n =100 and p = 0.5** | **n =1000 and p = 0.5** |
|  |  |

Table. 7 ELM with Binomial distribution and identical datasets

Obviously, the performance of ELM becomes better when the values of weights are more and more complex

## Section 2

When a model is excessively complex, in other words, ELM has too many hidden neurons, vector $[g(w_j, b_j, x_1), \ldots, g(w_j, b_j, x_L)]$ will be too big, which means the system tends to “remember” each data, so overfitting occurs and decrease the testing accuracy.

## Experiment 2: Comparison between SaDE-ELM and V-ELM

Experiment Description: In this section, the performance of the V-ELM was compared with the SaDE-ELM. Testing was conducted on many real-world classification datasets.

For V-ELM, the number of hidden nodes was gradually increased, and the optimal number of nodes was then selected. Nine independent ELMs were used for voting.

For SaE-ELM, the number of hidden nodes is same as V-ELM. F and CR are generated by normal distribution respectively. F follows N (0.5,0.3) and CR follows N (0.5,0.1). The number of evolutionary times is 500. The size of evolutionary pool is 10.

Data was nominalized before send to a network.

Average results of 10 trials of simulations for each fixed size of SLFN were obtained and reported in the below table. In each training, 10 percent of the total dataset was circle shifted. Overall datasets come from the UCI repository [22].

## Testing data set information and testing result

| Datasets | Training | Testing | Attributes | Classes |
|---|---|---|---|---|
| diabetes | 512 | 256 | 8 | 2 |
| spambase | 3067 | 1534 | 57 | 2 |
| balancescale | 417 | 208 | 4 | 3 |
| Iris | 120 | 30 | 4 | 3 |
| Pen-Based Digits | 7328 | 3664 | 16 | 10 |
| waveform | 3750 | 1250 | 21 | 3 |
| Image Segmentation | 210 | 2100 | 19 | 7 |
| letter-recognition | 16000 | 4000 | 16 | 26 |
| Wine | 134 | 44 | 13 | 3 |
| bupa | 304 | 41 | 7 | 2 |
| ecoli | 269 | 67 | 8 | 8 |

Table. 8 UCI dataset

## Performance comparison.

| Datasets | V-ELM (K = 9) | | | SaD-ELM | | | Nodes |
|---|---|---|---|---|---|---|---|
| | Sigmoid Testing (%) | | | Sigmoid Testing (%) | | | |
| | Accuracy(%) | Var(%) | Running time(s) | Accuracy(%) | Var(%) | Running time(s) | |
| **diabetes** | 74.21 | **0.038** | 0.30 | **76.25** | 0.076 | 25 | 30 |
| **spambase** | 90.55 | **0.013** | 0.25 | 90.68 | 0.076 | 281 | 95 |
| **balancescale** | 87.74 | **0.12** | 0.27 | **90.67** | 0.224 | 35 | 30 |
| **iris** | 100 | 0.12 | 0.20 | 98.6 | 0.079 | 16 | 15 |
| **Pen-Based Digits** | 96.7 | 0.001 | 0.65 | 97.42 | 0.0027 | 62 | 80 |
| **waveform** | 82.59 | 0.007 | 0.52 | 82.56 | 0.0124 | 98 | 100 |
| **Image Segmentation data** | **88.10** | **0.024** | 0.33 | 86.29 | 0.075 | 58 | 30 |
| **letter-recognition** | **87.57** | 0.28 | 0.88 | 84.66 | 8.91 | 1120 | 800 |
| **wine** | 96.13 | 0.149 | 0.20 | 97.72 | 0.19 | 13.62 | 15 |
| **bupa** | 70.58 | **0.48** | 0.25 | 70.58 | 0.84 | 17.72 | 25 |
| **ecoli** | 84.28 | **0.08** | 0.20 | 84.54 | 0.62 | 22 | 34 |

- **Accuary is the tesing Accuary, it is the mean of the 10 results**
- **Variance is the mean of the 10 results**

Table.9 Testing results

The same dataset was run by gradient descent algorithm (gradient descent with momentum backpropagation was applied here) to judge performance of those two algorithms. The network is a one hidden layer network. The number of the hidden neurons is same as the V-ELM hidden neuron numbers referred to earlier. Learning rate is 0.001, momentum constant is 0.9, maximum number of epochs to train is 100000 and minimum performance gradient is 1e-4

| BP algorithm | | |
|---|---|---|
| **Data set** | **Accuracy (%)** | **Running time(s)** |
| diabetes | 75.94 | 500 |
| spambase | 91.65 | 623 |
| balancescale | 90.38 | 450 |
| iris | 96.23 | 270 |
| Pen-Based Digits | 92.56 | 4725 |
| waveform | 82.30 | 7000 |
| Image Segmentation data | 91.00 | 367 |
| letter-recognition | 85.40 | 10083 |
| wine | 90.91 | 286 |
| bupa | 76.47 | 2358 |
| ecoli | 80.6 | 480 |
| ● **Algorithm is easily sink into local minimal. Thus, each data set was run 5 times. The best result was recorded and showed in table** | | |

Table.10 BP results

In terms of stability, the performance of V-ELM is better than SaD-ELM. In terms of testing accuracy, each method has its own advantages on different datasets. In terms of speed, V-ELM is tens of thousands of times faster than SaD-ELM.

## Experiment 3: ELM performance on real world collecting data (Bed Sensor)

Experiment Description: In this section, the performance of the ELM was tested with a real-world data from a study called Recognition of Sleep Stages Using a Bed Sensor [8]. In this paper, the author attempted to identify human sleep stages by using a bed sensor.

In the study, heart beat rate and respiratory rate were collected, then, features extracted (dataset) from those signals. The SVM classifier with a Radial Basis Function (RBF) was used to classify the labeled sleep stage. The trained classifier was used to classify unlabeled data. In this experiment, ELM will replace SVM to classify the Sleep Stages. The 10-fold cross-validation strategy was adopted to validate the classifier. Results are shown below.

## Testing results

| **Two stage classification with bed sensor data with grid search shuffle data (10 times)** | | | |
|---|---|---|---|
| ELM (1200 hidden neurons) | | SVM | |
| Average accuracy | standard deviation | Average accuracy | standard deviation |
| 90.25 | 0.2471 | 93.31 | 1.25 |

Table.11 testing results on shuffle data dataset

| **Two stage classification with bed sensor data with grid search (leave-one-night-out) (10 times)** | | | |
|---|---|---|---|
| ELM (150 hidden neurons) (%) | | SVM (%) | |
| Average accuracy | standard deviation | Average accuracy | standard deviation |
| 79.95 | 1.47 | 69.85 | 14.7 |

Table.12 testing results on leave-one-night-out

The result of experiment shows ELM has good performance than SVM on leave on night out dataset.

# CONCLUSION

My experiments with ELM and its derivative networks have shown a comprehensive principle of ELM. a random projection plus least squares method. The training time of the extreme learning machine is very fast since it doesn't adopt the gradient decent based algorithm. It performs well on small dataset.

When ELM is adopted for some tasks, the weights and biases should be randomly selected. If they are limited, the performance of the extreme learning machine will be influenced.

Compared to a gradient descent based network, ELM needs more hidden nodes which means it need more time during a testing process. Also, ELM is relatively hard to design for a complex structure (e.g. multilayer structure).

Gradient descent based networks extract features through a back-propaganda process. ELM creates features (weights and biases) by randomly generation. It makes some neurons of ELM not working. For this reason, I doubt ELM's performance on big and complex datasets. Also, the regularization of ELM will be influenced by this situation.

SaDE-ELM provides a way to fix the weights and biases. But it's a time-consuming algorithm.

V-ELM makes a system stable, but it contributes little to enhanced accuracy.

All in all, ELM is a shallow neuron network model which performance well on small - and medium - sized datasets (it shows good results on MNIST datasets), but need improvement for larger ones.

## REFERENCE

[1] Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1-3):489–501

[2] Self-Adaptive Evolutionary Extreme Learning Machine Jiuwen Cao · Zhiping Lin · Guang-Bin Huang Published online: 18 July 2012 © Springer Science+Business Media, LLC. 2012

[3] Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization A. K. Qin, V. L. Huang, and P. N. Suganthan IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 13, NO. 2, APRIL 2009

[4] Cao JW, Lin ZP, Huang G-B (2012) Voting based extreme learning machine. Inf Sci 185(1):66–77

[5] S. Dasgupta and A. Gupta, “An elementary proof of the Johnson Lindenstrauss lemma,” in Proc. ACM SIGMOD-SIGACT-SIGART, 2002, pp.274–281.

[6] James Bezdek1 , Xiuyi Ye1 , Mihail Popescu2 , James Keller1 , Alina Zare1 Random Projection below the JL Limit University of Missouri , 1 ECE Dept. / 2 HMI Dept. Columbia, MO, USA

[7] Recognition of Sleep Stages Using a Bed Sensor Jialei Yang, James M. Keller, Life Fellow, IEEE, Mihail Popescu, Senior Member, IEEE and Marjorie Skubic, Senior Member, IEEE

[8] Random projections fuzzy k-nearest neighbor(RPFKNN) for big data classification, Mihail Popescu, James M. Keller Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference.

[9] Learning representations by back-propagating errors David E Rumelhart, Geoffrey E

Hinton, Ronald J Williams

[10] A. Tuson and P. Ross, “Adapting operator settings in genetic algorithms,” Evolut. Comput., vol. 6, no. 2, pp. 161–184, 1998.

[11] J. Gomez, D. Dasgupta, and F. Gonzalez, “Using adaptive operators in genetic search,” in Proc. Genetic Evolut. Comput. Conf., Chicago, IL, Jul. 2003, pp. 1580–1581.

[12] C. C. Chang, C. J. Lin," LIBSVM: A library for support vector machines," ACM Transactions on Intelligent Systems and Technology (TIST), vol. 2, no. 3, pp. 27, 2011

[13] J. L. Yang, "Recognition of sleep stages from sensor data", M.S. thesis, Dept. Elect. Comput. Eng., Univ. of Missouri, Columbia, MO,2015.

[14] P.L. Bartlett, the sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, IEEE Transactions Information Theory 44 (2) (1998) 525–536.

[15] A. E. Eiben, R. Hinterding, and Z. Michalewicz, “Parameter control in evolutionary algorithms,” IEEE Trans. Evolut. Comput., vol. 3, no. 2, pp. 124–141, Jul. 1999.

[16] R. Storn and K. V. Price, “Differential evolution-A simple and efficient heuristic for global optimization over continuous Spaces,” J. Global Optim., vol. 11, pp. 341–359, 1997.

[17] K. Price, R. Storn, and J. Lampinen, Differential Evolution—A Practical Approach to Global Optimization. Berlin, Germany: SpringerVerlag, 2005.

[18] V. Feoktistov, Differential Evolution: In Search of Solutions. Berlin, Germany: Springer-Verlag, 2006.

[19] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, “Differential evolution training algorithm for feed-forward neural networks,” Neural Process. Lett., vol. 7, no. 1, pp. 93–

105, 2003.

[20] R. Storn, “On the usage of differential evolution for function optimization,” in Proc. Biennial Conf. North Amer. Fuzzy Inf. Process. Soc., Berkeley, CA, 1996, pp. 519–523.

[21] R. Storn, “Differential evolution design of an IIR-filter,” in Proc. IEEE Int. Conf. Evolut. Comput., Nagoya, Japan, May 1996.

[22] https://archive.ics.uci.edu/ml/datasets.html

[23] http://www.ntu.edu.sg/home/egbhuang/elm_codes.html