UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

UNDERGRADUATE SCHOOL

INTERSHIP AND DEVELOPMENT

BACHELOR THESIS

By

Dang Vu Lam

Information and Communication Technology

Title

Extreme Learning Machine

Supervisors

Dr. SEBASTIÁN BASTERRECH

Czech Technology University

Hanoi, July 2017

Contents

Acknowledgement

I would like to express my gratitude to all who had helped me on the journey.

Thank you, Dr. Sebastián Basterrech, my supervisor, who taught me everything I have conducted in this thesis. Thank you, for your patient, determinition and your knowledge. It is my privilege and honor to work under your supervision. I sincerely hope we will be able to continue this collaboration for many years to come.

Thank you, Dr. Doan Nhat Quang for being my internal supervisor. Thank you for your support during this internship. Your support is most appreciated.

I would like to express my appreciation to ICTLab and USTH. My 3 years endeavour at the University is unforgettable and will ever be endeared. My brief time working at ICTLab have teach me valuable lessions and give me the determination for the pursuit of an academic career.

My appreciation for my friends and family, without your support I could never be able to persisted. Thank you for supporting me during difficult times, to the very end.

Abstract

In this work we explore the simplicity of Extreme Learning Machine, its characteristic and the ability to use a swarm optimization method to archive exceptional performance.
Keywords: Extreme Learning Machine, Particle Swarm Optimization, Neural Network.

List of Abbreviations

- BP: Backpropagation
- ELM: Extreme Learning Machine
- MNIST: Mixed National Institute of Standards and Technology
- PSO: Particle Swarm Optimization
- IPSO: Improved Swarm Optimization
- SLFN: Single Layer Feedforward Network
- USTH: University of Science and Technology of Hanoi

List of Figures

List of Tables

Mathematical notations

In order to add clarity to the paper, we decided to introduce the following formalism:

- $F(X)$: The result of applying activation function f(x) to all element of the matrix X
- H : The output matrix of the hidden layer (before apply the output weight)
- $H^{\dagger}$: The Moore-Penrose psuedoinverse of the matrix H
- X : The input matrix, which include all datapoint in consideration
- y : The expected output matrix
- $\hat{y}$: The approximated output matrix
- w : The weights matrixes, including
  - w_i : Input weights matrix
  - w_o : Output weights matrix
- η : The predefined learning rate
- ε : Random error

1 Introduction

In the recent years, with the recover of machine learning field from what know as an "AI Winter", neural netwrok have taken the central stage of a cutting edge field which drive many advancement in computer science. Different networks architectures, varied in width, depth and activation function powered many systems, from the massively complex Watson used to researchs and diagnoses cancer, to trival case of authenticate access to our smartphone.

The awesome power of neural networks lay in its ability to adjust itself to fit the data presented. This is thank to weights - small bit of adjustable information connecting neurons in different layers in a neural networks. They reflect the relationship between the layers. Thus the main concern of the data scientist is to discover the best possile weights that fit the data presented to the system - this process is known as Machine Learning - an analogy to the similar process in human.

Neural networks learning by adjust this weights based on the root mean square error (RMSE). Traditionally these adjustment was made using gradient descent method - an iterative method that go against the derivative of the activation function at the given location in the weight space. Using chain rule to send this derivation back to the previous layer, this method is called Backpropagation [1]. This method have been proved to be a robust, reliable algorithm to find good fit model for any neural network regardless of shape or activation function - as long as the activation function is differentiable. However being an iterative method, BP have many major drawback,namely its speed and the possibility to be trapped in a local minima.

Extreme Learning Machine [2] is a novel method to address problem of weight adjustment. An analytical method, it overcome many problem posed by Backpropagation, especially the fact that least square method can never be trapped in a local minima. Furthermore, because all the weights are discovered at once, the speed of ELM is suggested to be hundred time faster than Backpropagation [2].

Particle Swarm Optimization is a metaheuristic optimization method [3]. Inspired by the behavior of a swarm of animals, PSO is one of the first and most successful swarm optimization method. In [4], Hutchison et al propose that PSO can be used to tune the input parameters of ELM. Compare to Backpropagation, PSO have several advantage. Using many instead of one particle to search in the weight space, it is both faster and present an oppotunity for parallelization of the algorithm. Also a global optimization algorithm, it is unlikely for PSO to be trapped in a local minima.

2 Background

2.1 Linear Regression

2.1.1 Assumption of the model

The Linear Model assume the linear relationship between the predictor X and respond Y:

$$Y = F(X) + \varepsilon = \beta_0 + \beta_1 * X_1 + ... + \beta_n X_n + \varepsilon \tag{1}$$

where Y is the respond, X is the predictor [5]. The vector $[\beta_0, .., \beta_n]$ is called a weights vector, and must be tuned in order for the model to yield accurate result.

2.1.2 Gradient Descent

The concept of Gradient Descent is to "go down" the gradient "well" at a predefined rate. After each "step" the algorithm re-evaluate for new weight vector. These steps are repeated until a pre-defined number of iterations or until the global minima have been reach. [6]
Another view on the algorithm is to consider it as an optimization process [1]. In this view, Gradient Descent is considered as a constrained minimization problem of the model's error function.
Bishop [6, p240] provide a simple formular to update weights vector base on Gradient information

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \tag{2}$$

whereas η is the pre defined learning rate and $E(w^{(\tau)})$ is the error function, which is given as

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} ||y(x_n, w) - t_n||^2 \tag{3}$$

From $E(w)$ the derivative $\nabla E(w)$ is given by

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{4}$$

An important aspect of Gradient Descent is its Learning Rate η. One of the major drawback of the method is the learning rate must be carefully considered. If η is too small, the algorithm might take

very long time to finish, might not converge or will not reach the minima before the exit condition is met. Meanwhile a η too large will risk overshooting the desired result and unable to converge.

2.1.3 Least Square

In contrast to the heuristic and iterative Gradient Descent method, we can calculate the coefficient directly using least square based method. The method that concerned in the context is called Moore Penrose psuedoinverse method.

2.2 Backpropagation

Backpropagation is one of the most popular method used in Data Mining and Artificial Intelligent field for obtaining a model for Artificial Neural Network. Discovered and re-discovered many time in the 20^{th} century, the method is popularized and refined by multiple independent party, one of the most prominion are Werbos and Le Cun [1].

2.2.1 Algorithm

Backpropagation algorithm can be considered layered version of Gradient Descent [6]. It use indefinitely differentiable activation function for its neurons. We denote these function as F and their derivative as F'. The algorithm then went through 2 steps: [7, 161]

- Forward Propagation

  The input X_i is feed into the network. The functions $F(X_i)$ are calculated and propagate forward into next layers. The derivative functions $F'(X_i)$ are stored.

- Back Propagation

  The constant 1 is assigned to the output unit and feed into the network in reverse. The incomming values to each node is added and multiply with value previously stored in those nodes. The result is then transfered upward to the input layer as derivative of the network function with respect to X_i

2.3 Extreme Learning Machine

Extreme Learning Machine (ELM) [2] is a simple yet powerful learning algorithm for single layer feedforward neural network. ELM omit the need to readjust the parameters of the network after initialization - all the training example are learned at once.

2.3.1 Mathematical Model

As a SLFN, ELM model are extremely simple:

$$\hat{Y} = w_o H = W_o F(w_i A + \beta) \tag{5}$$

$$Y = \hat{Y} + \varepsilon \tag{6}$$

2.3.2 Training Model

In order to train the SLFN, ELM is divided into 2 stages [2]:

- Non linear weights initialization

  During this stage, input weights matrix w_i are initialized and assigned randomly in range of $[-1, 1]$, typically follow a Gaussian distribution

- Linear weights learning

  After initialization of input weights, all the training data is feeded through the network and hidden layer output are captured as H. Output weights matrix can then be created using analytical method as following:

  $$w_i = Y * H^{\dagger} \tag{7}$$

  where $H^{\dagger}$ is the Moore - Penrose psuedoinverse product of H

2.4 Particle Swarm Optimization

Particle Swarm Optimization is a swarm optimization technique model after the movement of a swarm of animals such as a school of fish or a flock of bird.[3]. Originally designed to model social behavior, it is later realized to be a powerful tool for optimization problem.

In PSO, a swarm of particle is created by spawning uniformally distributed random particle in search space. Each iteration of the algorithm, the position of each particle is recalcuated using its current position and a velocity. This velocity draw the particle to the best global and local position:

$$V_{t+1} = V_t + c1 * (global\_best - X_t) + c2 * (local\_best - X_t) \tag{8}$$

$$X_{t+1} = X_t + V_{t+1} \tag{9}$$

where X_t is the current position, V_t is the current velocity. $c1$ and $c2$ are the weights. On some Implementation, when updating velocity, the current velocity is multiplied with a inertial factor called w. This algorithm is known Improved Partical Swarm Optimization (IPSO) [8]

PSO have been suggested as a good solution for weights optimization problem in neural networks[4] [9]. Being a metaheuristic method, PSO exceptionally suitable for application where derivative of the activation is unavailable or propagation of error signal is undesirable.

3 Implementation

3.1 Dataset Description: Salary Dataset

The salary dataset present a classical regression problem. Given a professor's details, we should be able to predict their expected salary.

Gender	Rank	YOE	Degree	YOR	Salary
male	full	25	doctorate	35	36350
male	full	13	doctorate	22	35350
male	full	7	doctorate	13	27959
female	full	8	doctorate	24	38045
female	assistant	1	doctorate	1	16686
female	assistant	1	doctorate	1	15000
male	full	10	doctorate	23	28200

Table 1: Sample from Salary Dataset

As standard in AI researches, the dataset is divided randomly with 70% of the dataset is used for training and 30% used for validation. Because ELM is not a gradient based nor having any feedback, there is no need to further divide the training dataset.

3.2 Extreme learning machine applied to regression problem

Applying ELM to single output regression problem like salary prediction is straight forward.
First the text values must be translated into numerical values. In Python this was done easily using a dictionary. The converted data is organized into numpy's matrix. Training and testing data is preemptively divided into 2 different files.
The experiment was designed to demonstrate ELM's bias-variance curve. Input data is dotted with randomly generated input weights before applied the activation functions. Output weights can then be learned using numpy built in least square function [10]. The input-output weights pair are recorded and tested using the testing dataset. The network was designed with parameterization in mind. All aspect of the network can be adjusted, from its size, activation function, etc. This allow us to test the prediction performance of ELM and plot the characteristic of the network in respect to its size and repeatability. Each iteration of the test is repeated 100 times, with 100 neuron increase in size after each

test. The average of the tests are show in blue. Orange line show the standard deviation. Grey horizontal line is reference result from simple linear regression using the same least square function in numpy [10].

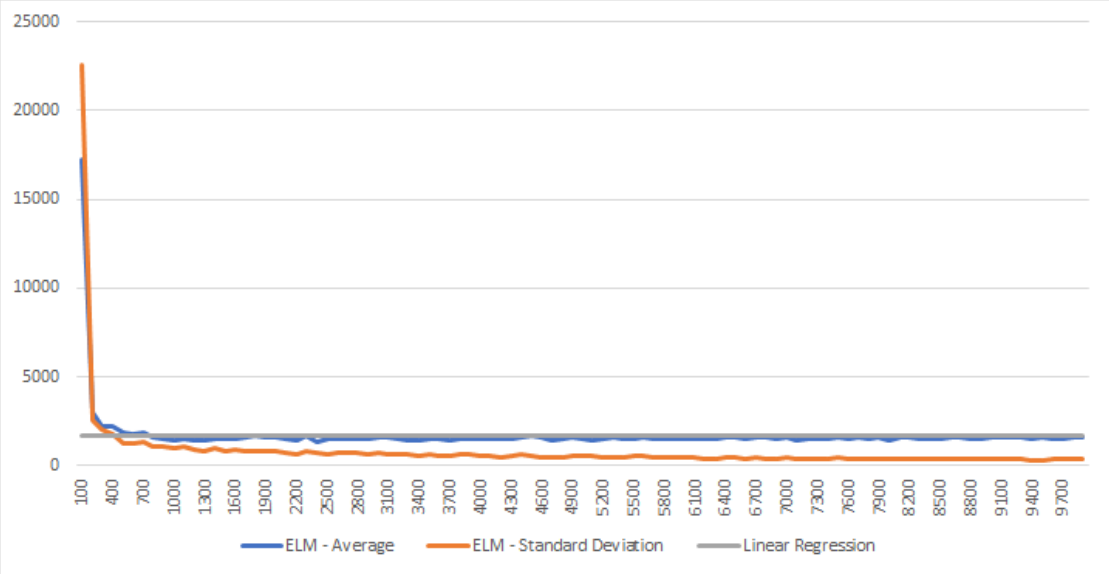

Figure 1: Result of ELM experiments on Salary Dataset

The result shown that not only lower neurons count networks yeild lower accuracy, they are also less predictable. The improvement in prediction performance stopped increasing and stable at 1000 neurons for this particular dataset. However with more neurons, the repeatability of the network increased. This imply that while on average, the average performance is stable above 1000 neurons count, at a level comparable with Linear Regression, the higher neurons count actually decrease best case performance of the network. As discussion in the next section, this derive an interesting point when designing ELM based networks for input weight optimization using PSO.

3.3 Particle Swarm Optimized Extreme Learning Machine

As suggested in [4], the prediction performance of ELM is directly linked to the quality of the input weights. While it is possible to yield result from any randomly generated input matrix, a correctly tuned input weights will result in exceptional performance. This is the drive behind classical Neural network learning algorithm such as Backpropagation. However the gradient descent nature of Backpropagation

lead to complication in calculation, namely the need to tune learning rate manually, the possibility to be trapped in local minima, as well as the parameters are inconvergable. Thus it is desirable to design an scheme which combine the analytical approach of ELM with the ability to search for the most suitable input weights.

In [4], the authors proved that PSO is a suitable candidate to solve the problem of search for global minima in the weights space. Our result from a repeated test on Salary dataset show that with 1/10 of the amount of the network width, the result is consistently better than only ELM. Namely, with 100 neurons wide network, the RMSE is only in range of 200 to 300, in comparation with the plot above.

Furthermore, the plot in section 1 shown that because of high derivation, lower neurons count network can outperform higher count in best case scenario. This remark is exciting because not only we are able to archive faster computation time using lower neurons count network, it is actually better for the accuracy if we use less neurons when combining ELM with PSO.

4 Discussion

4.1 Performance of ELM

Evidently, ELM systems are lack the prediction performance of traditional algorithm such as Back-propagation for similar number of neurons. Result from MNIST [11] compared to state of the art networks listed on MNIST website are vastly underperformed. However there are works have been done to improve such accuracy issue. Thanks to the speed that ELM able to learn, multiple algorithms have been proposed to solve the optimization of ELM input weights. As show in this work, PSO is a prominion candidate for this problem.

Another problem with ELM is the base algorithm can only be applied to SLFN. While SLFN is quite helpful for certain class of problem such as shown in this work, higher reasoning, such as XOR problem often require deeply layered architecture [12]. Several attemps was made to address this problem concerning ELM which we will elaborate in following section.

Finally, ELM is inheritently a batch algorithm. The requirement for all data present at the moment of initialization, while allow the incredible speed of ELM, deman equally impressive computing power in order to compute such large and complex linear equations. In [13], Liang et al have proposed a two steps strategy to solve this problem. Named Online Sequential ELM (OS-ELM), [13] have proved that the strategy deliver both speed of ELM and the benefit of an online sequential learning algorithm.

4.2 Deep ELM

[12] proved there are boundary where a single layer perceptron can not effectively approximate the original functions. While ELM was proved to be an universal approximator [2], it is still desirable to produce a layered version for the algorithm.

The problem arise with how to effectively train the weights connecting hidden layers. Because there are no feedback, no error signal propagated back through the network, there is no efficient method to adjust these weights which trapped between hidden layers. Thus they needed to be learn or analytically calculated individually in a layerwise fashion.

In order to train each layer individually, a robust architecture was designed in [14] based on sparse autoencoders - thus the algorithm was named AE-ELM. AE-ELM is then used to develop a feature space

where another algorithm called Hierarchical ELM or H-ELM [15]. By applying multiple layer of ELM based autoencoders, [15] prove that H-ELM can outperform many state of the art Deep Neural Network by bypassing the training phase of the process.

From the above reasoning, we project that it is possible to tune each layer of the network individually using the method covered in this thesis. Such algorithm will have the extra benefit of being able to tune just part of the network rather than the whole network, thanks to the fact that H-ELM is autoencoders stack on top of each other [15]. Base on RMSE, it is arguably easy to discover which of the layers needed to be tune, and we can focus our effort to those layers only.

Furthermore, since autoencoders transform the input into itself, it is reasonable to assume that each layer output is essentially similar to the original input. Thus we also propose to train each layer separately, independent from each other. The reason for this proposal is to ultilize the capabilities of high performance computing to train all the layer at once, and recombine them according to [14]. With the layers trained and optimized in parallel, we can archive much better speed than traditional neural networks.

4.3 Application of ELM

As a relatively recent discovery, ELM shown potential for great application in many fields where a good model needed to be discover quickly without regarding future need to update [2].

We believe one of such area is using neural network as a seach query. Using ELM it is now possible to produce good model for our intended outcome. Such system using Backpropagation is undesirable because of the iterative method make the time required to prepare such query higher than desirable. By fixing all the parameters of the network at the time of initialization, we now can produce one time use model that can return a match for not only extract fit, but also pattern match and close miss. This method can have great application in the field of Natural Language Processing as well as a replacement for database query and Regular Expression.

The system declared above would also find application in the field of Bioinformatics. Using ELM, it is easy to produce pattern finding network used to identify genomes in a sequence, and matching different combination of traits, gene and proteins together. Currently it is harder to use a Neural Network system on said application due to lead time required to train a good model. ELM can potentially eliminate this

lead time and open the gate for new exciting applications of Neural Network.

References

[1] Yann Lecun. A theoretical framework for back-propagation. In *Artificial neural networks*. IEEE Computer Society Press.

[2] Guang-Bin Huang, Qin-Yu Zhu, and Chee-kheong Siew. Extreme learning machine: Theory and applications.

[3] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. pages 39–43. IEEE.

[4] You Xu and Yang Shu. Evolutionary extreme learning machine – based on particle swarm optimization. In Jun Wang, Zhang Yi, Jacek M. Zurada, Bao-Liang Lu, and Hujun Yin, editors, *Advances in Neural Networks - ISNN 2006*, volume 3971, pages 644–652. Springer Berlin Heidelberg. DOI: 10.1007/11759966_95.

[5] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York.

[6] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer.

[7] Raúl Rojas. *Neural Networks*. Springer Berlin Heidelberg.

[8] Yong-gang Li, Wei-hua Gui, Chun-hua Yang, and Jie Li. Improved PSO algorithm and its application. 12(1):222–226.

[9] Qin-Yu Zhu, A.K. Qin, P.N. Suganthan, and Guang-Bin Huang. Evolutionary extreme learning machine. 38(10):1759–1763.

[10] Stéfan van der Walt, S Chris Colbert, and Gaël Varoquaux. The NumPy array: A structure for efficient numerical computation. 13(2):22–30.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 86(11):2278–2324.

[12] A. Newell. Perceptrons. an introduction to computational geometry. marvin minsky and seymour papert. m.i.t. press, cambridge, mass., 1969. vi + 258 pp., illus. cloth, $12; paper, $4.95. 165(3895):780–782.

[13] Nan-Ying Liang, Guang-Bin Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. 17(6):1411–1423.

[14] Le-le Cao, Wen-bing Huang, and Fu-chun Sun. Building feature space of extreme learning machine with sparse denoising stacked-autoencoder. 174:60–71.

[15] Jiexiong Tang, Chenwei Deng, and Guang-Bin Huang. Extreme learning machine for multilayer perceptron. 27(4):809–821.

A Python Implementation: Extreme Learning Machine

```
import numpy as np
import re

#dictionary
genders = {'male':1, 'female':-1}
ranks = {"full":1, "associate":0, "assistant":-1}
degrees = {"masters":1, "doctorate":-1}

#data input from ../salary.dat
#convert text input to numeric according to dictionary above
def inp(file):
    with  open(file, 'r') as input:
        length = 0
        A = np.array([[0, 0, 0, 0, 0, 0]]) #input layer size of 6
        y = np.array([0])
        for line in input.readlines():
            wordList = re.sub("[^\w]", " ", line).split()
            vect = np.array([0, float(genders[wordList[0]]), ranks[wordList[1]], float(wordList[2
            A = np.append(A, [vect], axis=0)
            y = np.append(y, [float(wordList[5])], axis=0)
            length += 1
    return np.delete(A, 0, axis = 0), np.delete(y, 0), length

#feed forward into the network
#sigmoid activation network
def feed_forward(A, syn, activ = "sigmoid"):
    if activ == "sigmoid":
        l1 = sigmoid(np.dot(A, syn))
```

```
    else:
        l1 = softplus(np.dot(A, syn))
    return l1


#sigmoid function
def sigmoid(x):
    return 1/(1+np.exp(-x))

#softplus function
def softplus(x):
    return np.log(1+np.exp(x))

def elm(seed = None, activ = "sigmoid", width = 1000):
    np.random.seed(seed)
    #func = np.random.rand(width, 1) #randomly assign activation function to the nodes of the net

    #read the training dataset
    [A, y, length] = inp('salary.dat')
    #randomized input weights
    syn0 = np.random.normal(size = (6, width))
    h = feed_forward(A, syn0, activ)
    w = np.linalg.lstsq(h, y)[0] #least square learning on the output weight of random layer

    #read the test dataset
    [A, y, length] = inp("salary_test.dat")
    #feed test data into network
    h = feed_forward(A, syn0, activ)

    #calculate error
```

```
err = np.abs(np.average(np.dot(h, w) - y))
#print('',err)
return err
```