BACKWARD EXTREME LEARNING MACHINE: IN APPLICATIONS OF DEEP NEURAL NETWORK PRETRAINING

A THESIS REPORT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTING IN ENGINEERING SYSTEMS
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2018
KMITL-2018-IC-M-011-02

BACKWARD EXTREME LEARNING MACHINE: IN APPLICATIONS OF DEEP NEURAL NETWORK PRETRAINING

PAVIT NOINONGYAO

A THESIS REPORT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTING IN ENGINEERING SYSTEMS
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2018
KMITL-2018-IC-M-011-02

THESIS TITLE	Backward Extreme Learning Machine: In Applications of Deep Neural Network Pretraining
STUDENT NAME	Mr. Pavit Noinongyao
STUDENT ID	60610022
DEGREE	Master of Engineering
PROGRAME	Computing in Engineering Systems (International Program)
ADVISOR	Dr. Ukrit Watchareeruetai

ABSTRACT

Deep learning breakthrough has started in 2006 with the proposal of a learning approach: greedy layer-wise unsupervised pretraining followed by supervised fine-tuning. The approach can be seen as initializing weights in a proper location before supervised learning is performed as opposed to the traditional way of pure random weight initialization. It allows deep neural networks to be trained effectively. However, unsupervised pretraining is commonly done using gradient-based iterative learning which adds substantial training time and hyperparameters to the learning process.

As opposed to gradient-based iterative learning, extreme learning machine (ELM) is an analytical learning scheme which is extremely fast and yields good generalization performance. Furthermore, it only requires a few parameters in the training procedure. However, it is still inferior to traditional gradient-based learning since it only applies for neural networks with single hidden layer. Although recently, a variant of ELM named multilayer-ELM (ML-ELM) has been proposed for applying ELM to learn for a deep neural network with the use of autoencoders, unsupervised neural network models which learn to encode features, it is not effective because of its incorrect use of transposed decoders' weights instead of encoders' weights to encode features and its no fine-tuning approach.

In this research, we seek to apply ELM for unsupervised pretraining to eliminate the problems of slow training and abundant hyperparameters in traditional unsuperivsed pretraining with gradient-based learning. We try to use autoencoders to build deep neural networks as ML-ELM but with more correct feature learning process (using encoders' weights instead of transposed decoders' weights as in ML-ELM) and also with the use of fine-tuning. To do this, we modify the learning process of ELM by exploiting it backwards and propose it as backward ELM (BELM). We then

extend BELM to learn for the case of autoencoders and name it as BELM-autoencoder (BELM-AE). Finally, we propose a way to use BELM-AE as an unsupervised neural network model to be stacked on top of each other and form stacked-BELM-AEs which is used as an unsupervised pretraining method.

We compare the proposed unsupervised pretraining method, stacked-BELM-AEs, with other comparable unsupervised pretraining methods namely stacked autoencoders and stacked ELM-autoencoders (stacked ELM-AEs), which is the feature learning part of ML-ELM, and also with random initialization and Xavier initialization methods. Experimental results show that, on average, pretrained networks or networks with weight initialization scheme outperform networks with random weight initialization. Furthermore, stacked-BELM-AEs achieves higher classification accuracy than the other methods on various benchmark datasets. It also requires significantly less number of training iterations (when fine-tuning) to achieve a desired accuracy when compared with the other methods. Moreover, the results also suggest that performing fine-tuning on a stacked ELM-AEs can significantly improve the performance which is better than ML-ELM which requires no fine-tuning.

ACKNOWLEDGEMENTS

Foremost, as a muslim, I would like to start this section by expressing my deepest gratefulness to Allah (may He be glorified and exalted) for nothing would happen without His permission.

I would like to acknowledge Dr. Ukrit Watchareeruetai, my advisor, for strengthening my skills, providing me a place, equipment and also continuous support in both technical and literary sides. His sincere devotion has considerably helped with all the problems I encountered during the study. Any flaw which may arise in the thesis should only be due to the author's mistakes.

I also would like to express my gratitude towards Asst. Prof. Dr. Chaiwat Nuthong for criticizing and suggesting a way to improve the research works and my presentation. Ultimately, I would like to thank my family, members of Computer Vision Laboratory and Applied Machine Learning Laboratory and other colleagues of the International College, King Mongkut's Institute of Technology Ladkrabang for every kindness of cooperation and suggestion.

Lastly, I am particularly thankful for Diyana Ngophaiwan who is always be there for me when all the hard times come, has been keeping me motivated, heartening me and giving me the strength to overcome any difficulties arisen during the development of the thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation and problem description

Deep learning is referred to as a set of learning procedures involving many levels of nonlinear transformation of representations to progressively create more meaningful representation in each level starting from raw inputs [25]. Normally and in this context, it is specifically referred to as deep neural networks (having many hidden layers). With its proven superiority over shallow learning architectures in terms of generalization performance, deep learning has gained massive popularity among machine learning researchers and practitioners. Modern days machine learning systems tend to favor deep learning over shallow ones. Numerous applications of deep learning have seen places in many fields such as in image recognition [24, 36, 37, 14], natural language processing [8, 1] and game playing [35, 29] etc.

Before 2006, learning of deep neural networks was extremely challenging due to the nature of the objective function with possibly many local optima each providing a different generalization error [10]. Traditional learning algorithm with random weight initialization is highly probable to leave the parameters or weights in a position which yields poor generalization performance. It was believed that deep neural networks were not possible to be trained effectively.

Deep learning has regained its popularity in the field and started a breakthrough after 2006 when Hinton and his colleagues proposed a learning scheme for a deep neural network called deep belief network (DBN) [16]. The method is based on greedily training a network layer-by-layer in an unsupervised fashion followed by a supervised fine-tuning. From this point, the greedy layer-

wise unsupervised pretraining part of this learning scheme would be referred to as just unsupervised pretraining or pretraining for simplicity. DBN can be seen as a stacking of unsupervised neural network models called restricted Boltzmann machines (RBMs) [17]. Following in 2007, another work by Bengio et al. [3] based on the same concept applied with another type of unsupervised neural network model, an autoencoder, had been proposed as stacked autoencoders. Excitement returned again in 2012 when Krizhevsky et al. won ImageNet competition [9] with a deep convolutional neural network [24], achieving almost half of the errors from other competitors. Dramatically, starting from this, people have seriously started paying attention to deep learning and increasingly, researches and products employing deep learning techniques have been produced.

Despite these success, however, the core learning algorithm which is a gradient-based iterative optimization such as stochastic gradient-descent [7], RMSprop [38] or ADAM [22] has caused a downside. Training a neural networks with back-propagation [4] and a gradient-based iterative learning method is a very slow process as its training time exponentially increases with respect to the number of layers. This is more problematic in the case where unsupervised pretraining is employed which would slow down the training process even more since it also uses gradient-based learning as well. Furthermore, unsupervised pretraining with gradient-based learning also means that an extra set (or more) of hyperparameters is required which would add more complication to the training procedure.

Moreover, regularization techniques such as dropout, the use of effective activation function such as ReLU [30] and many other techniques have depreciated the use of unsupervised pretraining. As a result, unsupervised pretraining, nowadays, is not as widely used as it was back then. Nevertheless, weight initialization (unsupervised pretraining can also be seen as a weight initialization method) is still very important for learning performance of a deep network. This has led to the proposal of weight initialization methods such as those proposed by Glorot and Bengio [11] and He et al. [13] which seek to initialize weights in order to maintain the variances of the activated units and of the gradients in each layer and suggest that the values of the randomly initialized weights should be bounded to a range taking into account the number of nodes in the connected layers.

On the other side in 2004, to solve the slow training problem of iterative learning algorithm, Huang et al. has proposed a new learning scheme called extreme learning machine (ELM) [20]

which is a learning algorithm for a single-hidden-layer feed-forward neural network (SLFN). The learning method is very simple and time efficient. It is based on randomly initializing weights from the input to the hidden layer and using the Moore-Penrose inverse [32] to analytically solve for optimal weights on the other side. Huang et al. reported that the method is , on average, more than 100 times faster comparing to the same networks trained with gradient-based learning on various benchmark datasets [20]. Many variants of ELM have been developed; for example, Hierarchical ELM (HELM) [15], online sequential ELM (OS-ELM) [27], self-adaptive differential evolutionary ELM (SaDE-ELM) [6], and ELM-based convolutional neural network (CNN-ELM) [42].

However, regardless of its fast learning, ELM could not reach the performance of deep learning since the method was proposed for an SLFN which is considered a shallow model. In consequence, there have been works done to apply ELM with deep neural networks. In 2013, Huang et al. [21] proposed a deep neural network model for ELM namely multilayer-ELM (ML-ELM). ML-ELM is based on stacking of ELM-based autoencoders called ELM-autoencoders (ELM-AEs). The process could be considered as greedy layer-wise unsupervised training which is similar to stacked autoencoders. Once the unsupervised training is done, a classification layer is also added to the unsupervisedy trained network but the weights (connecting to the classification layer) are calculated using least squares method . Unlike the original unsupervised pretraining approach, ML-ELM does not need any kind of fine-tuning.

Despite Huang et al. reported performance of ML-ELM, the use of autoencoder-based model in the unsupervised learning part is incorrect. More specifically, the first approach of stacking autoencoders to create deep networks is a stacked autoencoders. A stacked autoencoders uses encoders' weights to transform features in each layer. This can be interpreted as progressively creating better features in each layer by letting an autoencoder learns how to encode the current features (input to the current layer) in order to create more meaningful ones. ML-ELM, on the other hand, uses transposed decoders' weights to transform features which totally does not follow the autoencoder's concept and is not sound. Additionally, the concept of creating better features using only unsupervised learning and classifying without weight fine-tuning should not be optimal. This is because in the objective in the unsupervised training of autoencoders or ELM-AEs is to find a representation which best reconstructs the input, not to discriminate them.

The focus of this research is on unsupervised pretraining of deep neural networks where we

seek to create a combined approach aiming to achieve both generalization power of traditional deep learning and the fast and simple procedure of ELM based learning. To achieve that, we propose a new ELM-based learning scheme called backward extreme learning machine (B-ELM) which could be extended for an autoencoder case as an ELM-based unsupervised learning model called backward ELM-autoencoder (BELM-AE). Furthermore, we propose stacked BELM-AEs as a representation learning method and suggest its use as an unsupervised pretraining method for deep neural networks. We also apply the concept of unsupervised pretraining and fine-tuning to a stack of ELM-AEs to treat it as a pretraining method and show that it yields better performance than ML-ELM which does not use fine-tuning.

1.2 Objectives

The objectives of the this research are 1) to develop a stacked ELM-AEs with more correct use of autoencoders; 2) to study the potential use of ELM as an unsupervised pretrianing method; 3) to compare the performance of different unsupervised pretraining methods; 4) to study whether or not an ML-ELM could produce optimal solution without fine-tuning.

1.3 Structure of the thesis

The thesis is organized into six chapters:

- Chapter 2 provides background knowledge required to fully understand the thesis. This includes feed-forward neural networks, variants of its training procedure and models.
- Chapter 3 thoroughly investigates related literature and studies. It covers techniques for training deep neural networks for both gradient based learning and ELM approaches.
- Chapter 4 covers the proposed methods in details. It covers the proposed learning algorithm for an SLFN and how to extend it for deep neural networks.
- Chapter 5 explains the conducted experiments and discusses the results. Two experiments are covered: one evaluates classification performance and the other measures convergence rates.

- Chapter 6 concludes the thesis and discusses possible future works.

CHAPTER 2

BACKGROUND KNOWLEDGE

In order to fully understand the explanation of the next chapters we first provide the required background knowledge. This chapter is separated into four sections. Section 2.1 explains about feed-forward neural networks in general, Section 2.2 and Section 2.3 provide details on two different training methods of neural networks and Section 2.4 gives information on a specific model of a neural network called an autoencoder.

2.1 Feed-forward neural network

A neural network, also known as a multilayer perceptron (MLP), is referred to as an efficient and popular machine learning model for both classification and regression problems. A feed-forward neural networks, in particular, is a neural network architecture where the calculation only involves one direction of information flow from the input to the output. Stemming out from biology, neural networks try to mimic the information processing system of a brain and represent them as mathematical models [4].

An MLP composes of layers each has computing units called neurons or nodes. Computation is done starting from the input layer, through intermediate layers, also known as the hidden layers, to the output layer. Nodes are organized in layers, and each node in a layer is connected to all nodes in the next layer (fully connected) as shown in Fig. 2.1. Computation in a neural network is done by flowing values in one layer to the next layer via connected edges. Each node in the next layer combines all the values it receives. Mathematically, a node in a layer computes a weighted sum of

the values in the previous layer as in Eqs. 2.1 and 2.2. Given an input sample $\mathbf{x} = [x_1\ x_2\ \ldots\ x_D]^\top$, the output of the k^{th} node in the hidden layer h_k and in the output layer o_k of an SLFN with N hidden nodes and C output nodes are computed as

$$h_k = f(j_k^{(1)}), \tag{2.1}$$

$$j_k^{(1)} = \sum_{n=1}^{D} w_{i,n} x_n + b_i^{(1)}, \tag{2.2}$$

$$o_k = f(j_k^{(2)}), \tag{2.3}$$

$$j_k^{(2)} = \sum_{n=1}^{N} u_{i,n} h_n + b_i^{(2)}, \tag{2.4}$$

where $j_k^{(1)}$ and $j_k^{(2)}$ are pre-activated versions of h_k and o_k respectively, $b_j^{(l)}$ is a bias value associated with the k^{th} node of layer l (starting from the input layer which is layer 0), $f(\cdot)$ is a nonlinear activation function, $w_{i,k}$ is a weight connecting the i^{th} element of the hidden layer to the k^{th} element of the input layer and $u_{i,k}$ is a weight connecting the i^{th} element of the output layer to the k^{th} element of the hidden layer.

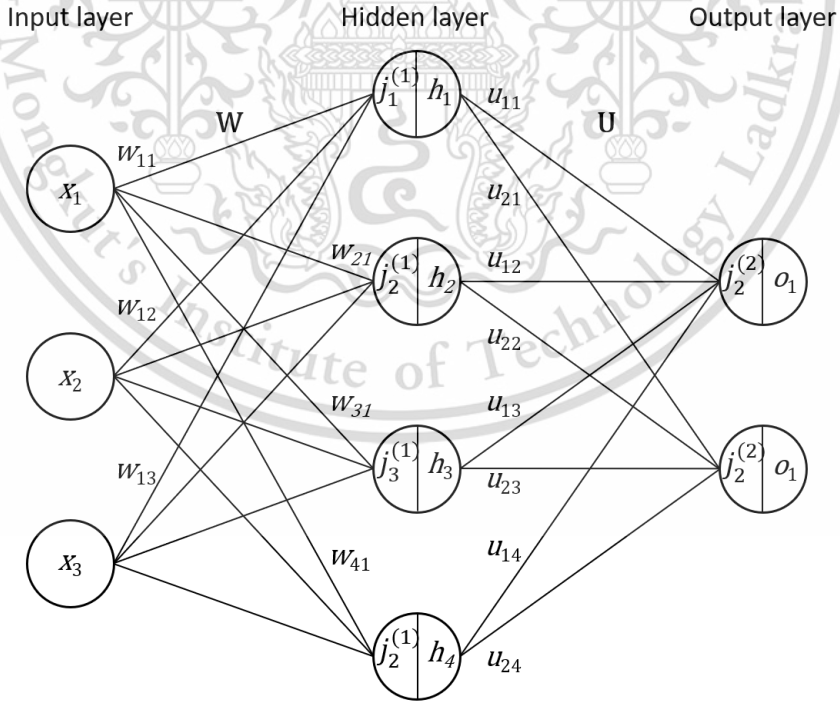


Figure 2.1: A single-hidden-layer feed-forward neural network.

2.2 Model training

Traditionally, a neural network is trained using variants of an iterative optimization algorithm called gradient descent [7]. Gradient descent requires gradients of loss function with respect to the parameters (weights) of the network to be trained. To calculate the gradient, an algorithm called back-propagation [34] was proposed by Rumelhart et al. It employs the chain rule in derivative to calculate the gradients. The parameters are then adjusted using gradient descent. In the following section, we give an example of how to calculate gradients using back-propagation. In the example, we assume the use of rectified linear unit (ReLU) defined as $f(x) = max(0, x)$ for an activation function.

2.2.1 Back-propagation

The algorithm begins after passing a batch of input samples to the network. It computes loss defined by a loss function E which, for classification tasks, is often chosen to be the cross-entropy loss function as:

$$E = -\sum_{i=1}^{S} \mathbf{t}_i \cdot log\left(\sigma(\mathbf{o}_i)\right), \tag{2.5}$$

where S is the number of total training samples, $\mathbf{t}_i = [t_{i,1}\ t_{i,2}\ \dots\ t_{i,C}]^\top$ is the expected output value of the i^{th} sample where $t_{i,k} \in \{0, 1\}$, $\mathbf{o}_i = [o_{i,1}\ o_{i,2}\ \dots\ o_{i,C}]^\top$ is the actual output value for the i^{th} sample, C is the number of output nodes (or the number of classes to be classified) and $\sigma(\cdot)$ is the softmax function defined as:

$$\sigma(\mathbf{o}_i) = \frac{e^{\mathbf{o}_i}}{\sum_{n=1}^{C} e^{o_{i,n}}}. \tag{2.6}$$

Next, we give an example of a back-propagation in an SLFN but the procedures generalize to the case with more than one hidden layer as well. The network is shown in Fig. 2.1. From the figure, the value in the left side of a node is a weighted sum of the nodes in the previous layer and the value in the right side is the output of passing it to ReLU. The objective here is to achieve a partial derivative of the error function with respect to each weight parameter in the network. Using the chain rule, the gradients can be computed as shown in Eqs. 2.7 and 2.8.

$$\begin{aligned}\frac{\partial E}{\partial u_{i,k}} &= \frac{\partial E}{\partial j_i^{(2)}}\frac{\partial j_i^{(2)}}{\partial u_{i,k}} \\ &= \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial j_i^{(2)}}\frac{\partial j_i^{(2)}}{\partial u_{i,k}},\end{aligned} \tag{2.7}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{i,k}} &= \frac{\partial E}{\partial j_i^{(1)}}\frac{\partial j_i^{(1)}}{\partial w_{i,k}} \\ &= \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial j_i^{(1)}}\frac{\partial j_i^{(1)}}{\partial w_{i,k}}.\end{aligned} \tag{2.8}$$

Each component of Eqs. 2.7 and 2.8 can be calculated as follows:

$$\frac{\partial E}{\partial o_i} = \frac{\partial \frac{1}{2}\sum_{n=1}^{C}(t_n - o_n)^2}{\partial o_i} = t_i - o_i \tag{2.9}$$

$$\frac{\partial o_i}{\partial j_i^{(2)}} = \frac{\partial max(0, j_i^{(2)})}{\partial j_i^{(2)}} = \begin{cases} 0, & \text{if } j_i^{(2)} \leq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{2.10}$$

$$\frac{\partial E}{\partial h_i} = \sum_{i=1}^{C} u_{i,k}\frac{\partial E}{\partial j_i^{(2)}} \tag{2.11}$$

$$\frac{\partial h_i}{\partial j_i^{(1)}} = \frac{\partial max(0, j_i^{(1)})}{\partial j_i^{(1)}} = \begin{cases} 0, & \text{if } j_i^{(1)} \leq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{2.12}$$

$$\frac{\partial j_i^{(2)}}{\partial u_{i,k}} = \frac{\partial \sum_{n=1}^{C} u_{i,n}h_n + b_i^{(2)}}{\partial u_{i,k}} = h_i, \tag{2.13}$$

$$\frac{\partial j_i^{(1)}}{\partial w_{i,k}} = \frac{\partial \sum_{n=1}^{N} w_{i,n}x_n + b_i^{(1)}}{\partial w_{i,k}} = x_i. \tag{2.14}$$

2.2.2 Gradient descent

Once the gradients of the error function with respect to each weight parameter is computed, the weights are adjusted using gradient descent. This is done by adding to each weight parameter a fraction of its negative error gradient. It can be seen that at each step, the weight parameters are moving to the direction which most reduces the error. The following formula is applied to all weight parameters for each iteration of the learning.

$$w_{i,k}^{l} \Leftarrow w_{i,k}^{l} - \alpha \frac{\partial E}{\partial w_{i,k}^{l}}, \tag{2.15}$$

where $w_{i,k}^{l}$ is a weight parameter connecting the i^{th} node of layer l to the k^{th} node of layer $l-1$ (in the case of an SLFN, $w_{i,k}^{1} = w_{i,k}$ and $w_{i,k}^{2} = u_{i,k}$), α is a predefined positive learning rate. The weight adjustment is done in an iterative fashion. After all the weights are adjusted for the current iteration, the gradients are then recalculated using back-propagation and gradient descent is then applied again until a predefined number of steps is reached or other termination criteria are satisfied.

2.3 Extreme learning machine

Extreme learning machine (ELM) , proposed by Huang et al. [20], is a fast learning procedure for an SLFN. Instead of using iterative algorithm such as gradient-based learning with back-propagation to learn the weights of the network, it uses an analytical method to compute them by formulating the network computation as matrix equations and solve for the least squares solution by using the Moore-Penrose matrix inverse [32].

In general, a feed-forward neural network with traditional learning procedure consumes a lot of time due to the gradient-based learning methods. ELM overcomes the difficulties in iterative learning methods by randomly generating the input weights $\mathbf{W}$ and the hidden layer biases $\mathbf{b}$ and analytically calculating the output weights using the Moore-Penrose matrix inverse. The overall process of ELM is visualized in Fig. 2.2.

As Section 2.1 describes the sample-by-sample calculation of an SLFN, for the general case with S samples, it can be described using matrix equations. More formally, let $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_S]$

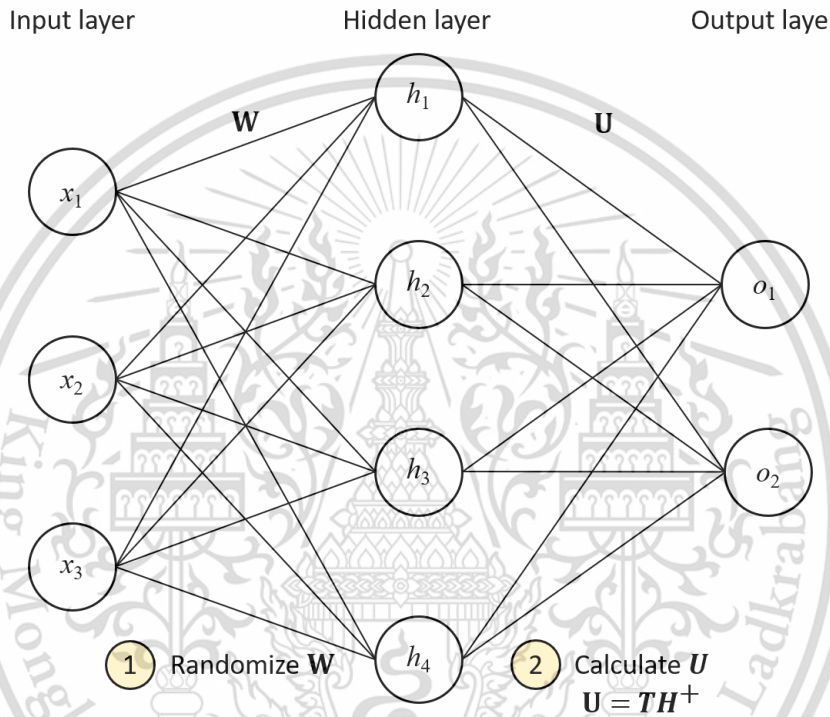


Figure 2.2: An SLFN trained with ELM. The weight **W** is randomized and the weight **U** is calculated using least squares method.

be a matrix of input samples and $\mathbf{T} = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots\ \mathbf{t}_S]$ be a matrix of corresponding expected outputs, an SLFN with N hidden nodes then could be mathematically described as the following equations:

$$\mathbf{O} = f(\mathbf{UH} + \mathbf{b}^{(2)}), \tag{2.16}$$

$$\mathbf{H} = f(\mathbf{WX} + \mathbf{b}^{(1)}), \tag{2.17}$$

where $\mathbf{W}$ is the weight matrix connecting the input nodes to the hidden nodes, $\mathbf{U}$ is the weight matrix connecting the hidden nodes to the output nodes $\mathbf{b}^{(1)} = [b_1^{(1)}\ b_2^{(1)}\ \ldots\ b_N^{(1)}]^\top$ and $\mathbf{b}^{(2)} = [b_1^{(2)}\ b_2^{(2)}\ \ldots\ b_N^{(2)}]^\top$ are the bias vectors of the hidden layer and the output layer respectively, and $\mathbf{O} = [\mathbf{o}_1\ \mathbf{o}_2\ \ldots\ \mathbf{o}_S]$ is the matrix containing the actual outputs of all samples. Calculating the least squares solution is equivalent to finding $\mathbf{W}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \mathbf{U}$ satisfying

$$\mathbf{T} = f(\mathbf{U}f(\mathbf{WX} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}). \tag{2.18}$$

However, in ELM, the bias vector and the activation function of the output layer are omitted leading to the simplified version of the above equation as:

$$\mathbf{T} = \mathbf{U}f(\mathbf{WX} + \mathbf{b}), \tag{2.19}$$

$$\mathbf{T} = \mathbf{UH}, \tag{2.20}$$

where $\mathbf{b}$ is the same as $\mathbf{b}^{(1)}$ with the number of layer omitted since there is only one bias vector.

ELM simply randomly initializes values for the matrix $\mathbf{W}$ so that the matrix $\mathbf{H}$ can be calculated. Huang et al. also suggests that constraining $\mathbf{W}$ to be an orthogonal matrix and $\mathbf{b}$ to be a unit vector could improve the performance. In other words, $\mathbf{W}$ and $\mathbf{b}$ should satisfy the following equations:

$$\mathbf{W}\mathbf{W}^\top = \mathbf{I}, \tag{2.21}$$

for the case of $N \leq D$ and

$$\mathbf{W}^\top\mathbf{W} = \mathbf{I}, \tag{2.22}$$

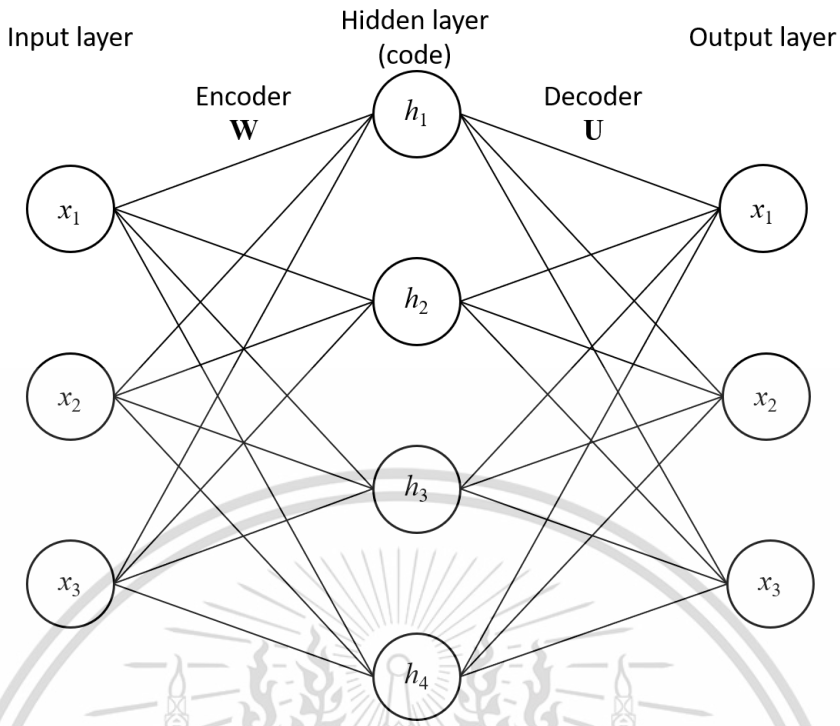


Figure 2.3: An autoencoder with the input of dimension 3 and the code of dimension 4.

for the case of $N \geq D$,

$$\mathbf{b}^\top \mathbf{b} = 1. \tag{2.23}$$

Once $\mathbf{H}$ is known, the learning procedure seeks to find matrix $\mathbf{U}$ by solving the equation

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{H}^+, \tag{2.24}$$

where $\mathbf{H}^+$ is the Moore-Penrose inverse of $\mathbf{H}$. Alternatively, we can calculate $\mathbf{U}$ using the least squares equation with a regularization parameter c as

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{H}^\top \left(\frac{\mathbf{I}}{c} + \mathbf{H}\mathbf{H}^\top \right)^{-1}, \tag{2.25}$$

where $\mathbf{I}$ is the identity matrix of size $N \times N$.

2.4 Autoencoder

An autoencoder is an unsupervised neural network model with its target output being the input itself as depicted in Fig. 2.3. It composes of two parts called an encoder $g_e(\cdot)$ and a decoder $g_d(\cdot)$ which can be mathematically defined as functions by:

$$\mathbf{h} = g_e(\mathbf{x}), \tag{2.26}$$

for an encoder and

$$\mathbf{r} = g_d(\mathbf{h}), \tag{2.27}$$

for a decoder where **x** is an input sample, **h** is a code and **r** is the reconstructed input. In autoencoder terminology, the model seeks to find a representation, or a code, which best represents the data. This code can be seen as a layer in a neural network and there can be any number of encoder and decoder layers. An autoencoder learns a representation by training the encoder and the decoder functions to minimize reconstruction error. Given a set of input samples $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_S]$, its loss function $E(\cdot)$ is defined as follows:

$$E(\mathbf{W}, \mathbf{U}) = \frac{1}{S}\sum_{i=1}^{S} \|\mathbf{x}_i - \mathbf{r}_i\|_2^2, \tag{2.28}$$

where S is the number of training samples. Normally, the weights parameters would be optimized using a gradient-based learning algorithm.

Still, an autoencoder might fail to learn useful information. For example, if an autoencoder learns the identity function to map from the input layer to the output layer then the learned representation is useless. One way to solve this problem is to constrain the number of hidden nodes (code size) to be smaller than the input dimension. This kind of autoencoder is referred to as an undercomplete autoencoder [12]. Another way is to use a code size which is larger than the input dimension, referred to as an overcomplete autoencoder, but with a sparsity constraint as in sparse autoencoders [5]. Other types of autoencoders exist such as a denoising autoencoder [39] and a contractive autoencoder [33]. However, the focus here is only on an undercomplete autoencoder since it is the one used in the experiments.

CHAPTER 3

LITERATURE REVIEW

In this chapter, related literature and studies are investigated in details. The chapter starts by explaining a technique which was a breakthrough of deep learning called greedy unsupervised pretraining in Section 3.1 followed by a learning method for deep neural networks based on ELM in Section 3.2 and lastly, modern weight initialization methods are covered in Section 3.3.

3.1 Greedy unsupervised pretraining of deep networks

In 2006, a breakthrough in deep learning has started with the proposal of a learning algorithm for a deep neural networks called deep belief net (DBN) by Hinton et al. [16]. The method uses an RBM which can be seen as a single-hidden-layer unsupervised model of a neural network to learn the distribution of input data at each layer and stack them on top of one another. This stack of RBMs is then supervisedly fine-tuned with gradient-based learning. This training strategy is referred to as greedy layer-wise unsupervised pretraining followed by supervised fine-tuning. The approach can be also applied to other unsupervised model of neural networks as Bengio et al. has found that using autoencoders as building blocks for the unsupervised learning produces comparable results [3].

The superiority of pretrained deep networks over those with pure random initialization could be explained from the nature of deep networks where the surface of the objective function is non-convex with many local minima each providing a different generalization error [10]. The effect of unsupervised pretraining then could be seen as an initialization of parameters to be in a region in

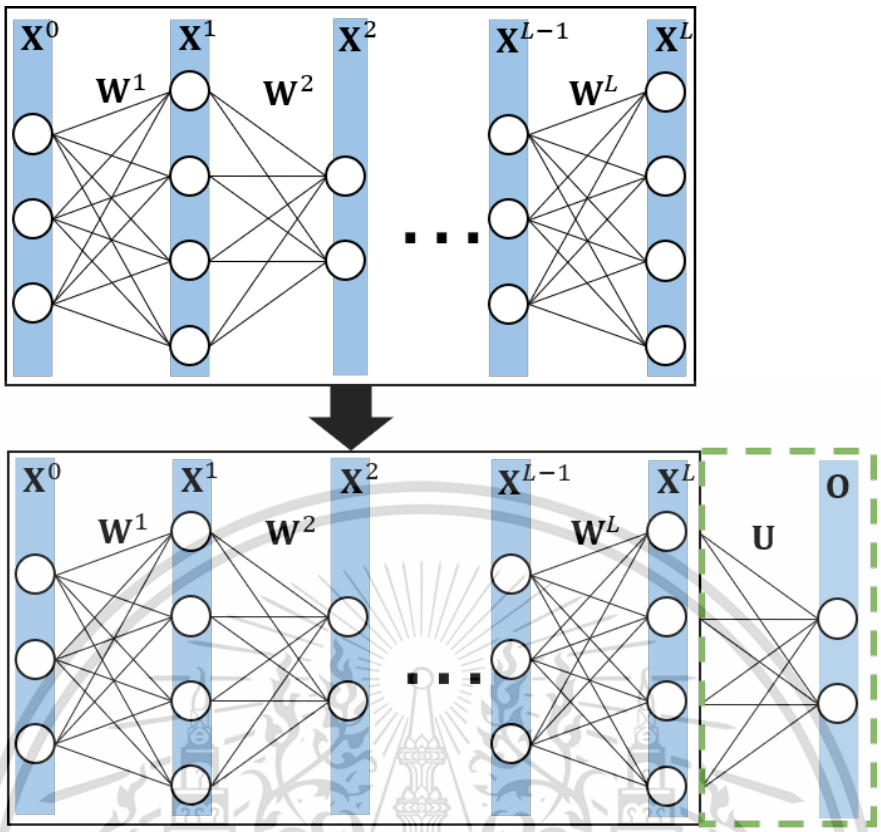


Figure 3.1: Unsupervised pretraining followed by supervised fine-tuning: (top) an unsupervised pretrained network; (bottom) the final network after adding a classification layer for supervised fine-tuning.

parameter space where the basin of attraction yields better generalization performance than the case of random initialization. It is also found that unsupervised pretraining acts as a kind of regularizer which is known to improve generalization performance.

To train a deep neural network following unsupervised pretraining strategy with autoencoders, the main procedures are as follows: 1) greedily train each layer individually in an unsupervised fashion; 2) add a classification layer and supervisedly fine-tune the whole network with gradient-based learning. In other words, there are two processes: 1) the unsupervised learning part for feature learning and 2) the supervised learning part for classification. The process is described visually for a stacked autoencoders with L hidden layers in Fig. 3.1 where the top one is a neural network initialized with unsupervised pretraining and the bottom one is the network after adding a classification layer to do supervised fine-tuning.

3.1.1 Training deep neural networks with stacked autoencoders

For greedy layer-wise unsupervised training with autoencoders, for each hidden layer l, an autoencoder is trained to reconstruct its input to the layer denoted as $\mathbf{X}^l$. The obtained encoder's weight $\mathbf{W}$ is then used as layer l weight, $\mathbf{W}^l$, in the deep network. The input to the next layer $\mathbf{X}^{l+1}$ is then obtained by transforming the current input $\mathbf{X}^l$ by $\mathbf{W}^{l+1}$ with the following equation:

$$\mathbf{X}^{l+1} = f(\mathbf{W}^{l+1}\mathbf{X}^l). \tag{3.1}$$

The whole pretraining process is described in Fig 3.2.

Once a pretrained network is obtained, supervised fine-tuning is performed by firstly adding a classification layer at the end of the network. Then the network is supervised trained with gradient-based learning.

3.2 Training deep neural networks with ELM

To be able to utilize both fast-learning of ELM and superiority of deep neural networks, a multi-hidden-layers architecture of neural networks based on ELM was proposed in 2013 [21] to apply ELM concept to deep neural networks. The method is based on learning representations layer-by-layer with ELM-based autoencoders (ELM-AEs) and using the final learned representation to be classified with least squares method.

3.2.1 ELM-autoencoder (ELM-AE)

ELM-AE [21] is a single-hidden-layer autoencoder where the encoder's weights $\mathbf{W}$ and biases $\mathbf{b}$ are randomly drawn from any continuous distribution and the decoder's weights are calculated using the following equation:

$$\mathbf{U} = \mathbf{X}\mathbf{H}^+, \tag{3.2}$$

which is the same process done in ELM with target matrix $\mathbf{T}$ substituted by $\mathbf{X}$ instead. The overall process is depicted in Fig. 3.3.

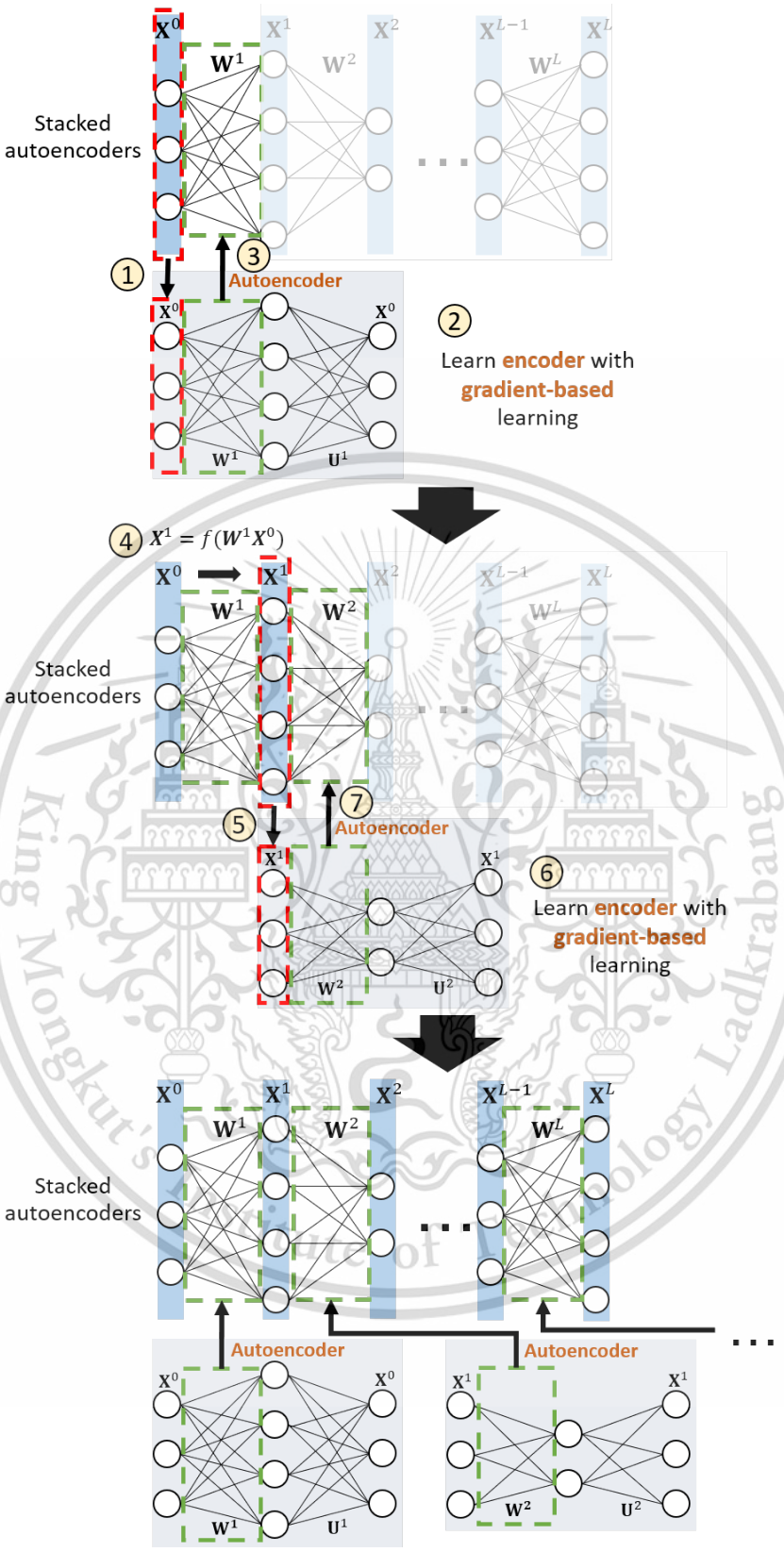


Figure 3.2: The whole training process of the unsupervised part of stacked autoencoders with L hidden layers (unsupervised pretraining process). The last row depicts the whole process statically.

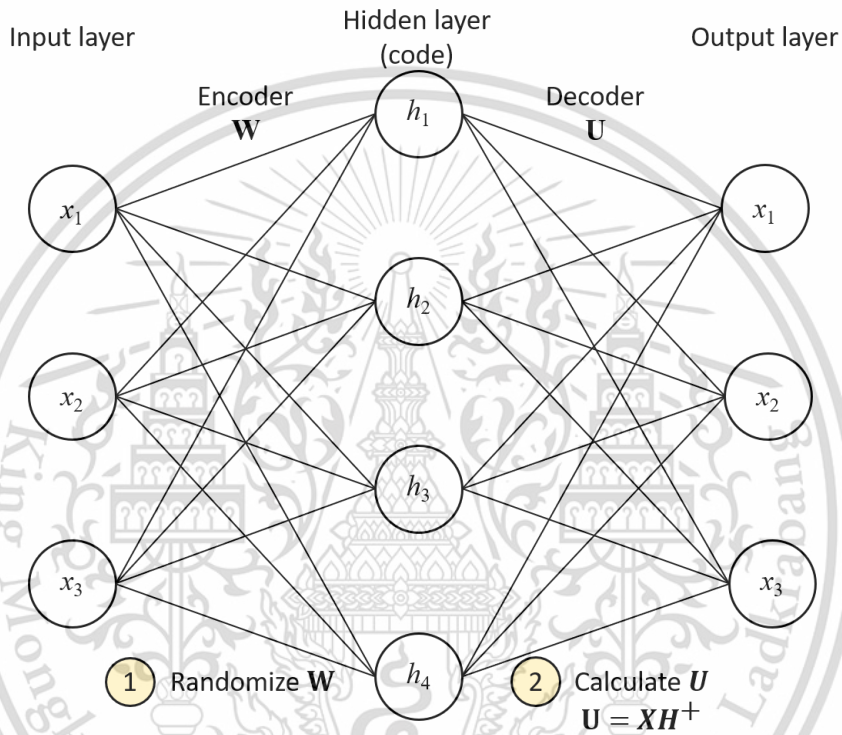


Figure 3.3: Overall process of ELM-AE. The difference from ELM is that the expected output **T** is the input **X** itself.

3.2.2 Multilayer-ELM (ML-ELM)

ML-ELM [21] is a multilayer neural network model for ELM. Being inspired by stacked autoencoders, it tries to adapt the approach to be used purely with ELM. As mentioned previously that greedy layer-wise unsupervised pretraining can be seen as having two parts; 1) the unsupervised one for learning features and 2) the supervised one for classification. Likewise, ML-ELM also consists of these two parts but with some modification.

The unsupervised or feature learning part is depicted in Fig. 3.4. It is built by training an ELM-AE layer-by-layer and stacking the transpose of the decoders' weights on top of each other to form a deep network. For each layer l, an input to the layer $\mathbf{X}^l$ is fed to an ELM-AE and the decoder's weights $\mathbf{U}^l$ are learned. Then the transpose of the decoder's weights ${\mathbf{U}^l}^\top$ are then used as the weights for layer l of the deep neural network. The difference from stacked autoencoders here is with the use of unsupervised learning model which in this case is an ELM-AE instead of an autoencoder. Also, the transpose of the decoders' weights are used to form deep networks as opposed to a stacked autoencoders which uses encoders' weights.

The supervised part is quite different from a stacked autoencoders since its authors claim that ML-ELM does not require fine-tuning process (but we would like to show that it does require). Rather, after the classification layer is added as in stacked autoencoders, the weights $\mathbf{U}$ are then calculated using least squares method. In other words, $\mathbf{U}$ is calculated as:

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{X}^{L^+} \tag{3.3}$$

where $\mathbf{X}^{L^+}$ is the Moore-Penrose inverse of the output from the last hidden layer $\mathbf{X}^L$.

3.3 Weight initialization methods

In a stochastic process like an optimization of deep neural network parameters, especially with non-convex objective function, the performance of the training highly depends on the starting point of the parameters or weights. In order to learn effectively with gradient-based learning, there needs to be a good initialization strategy. Unsupervised pretraining could also be considered as weight initialization methods since the methods produce start points (pretrained weights) for the optimiza-

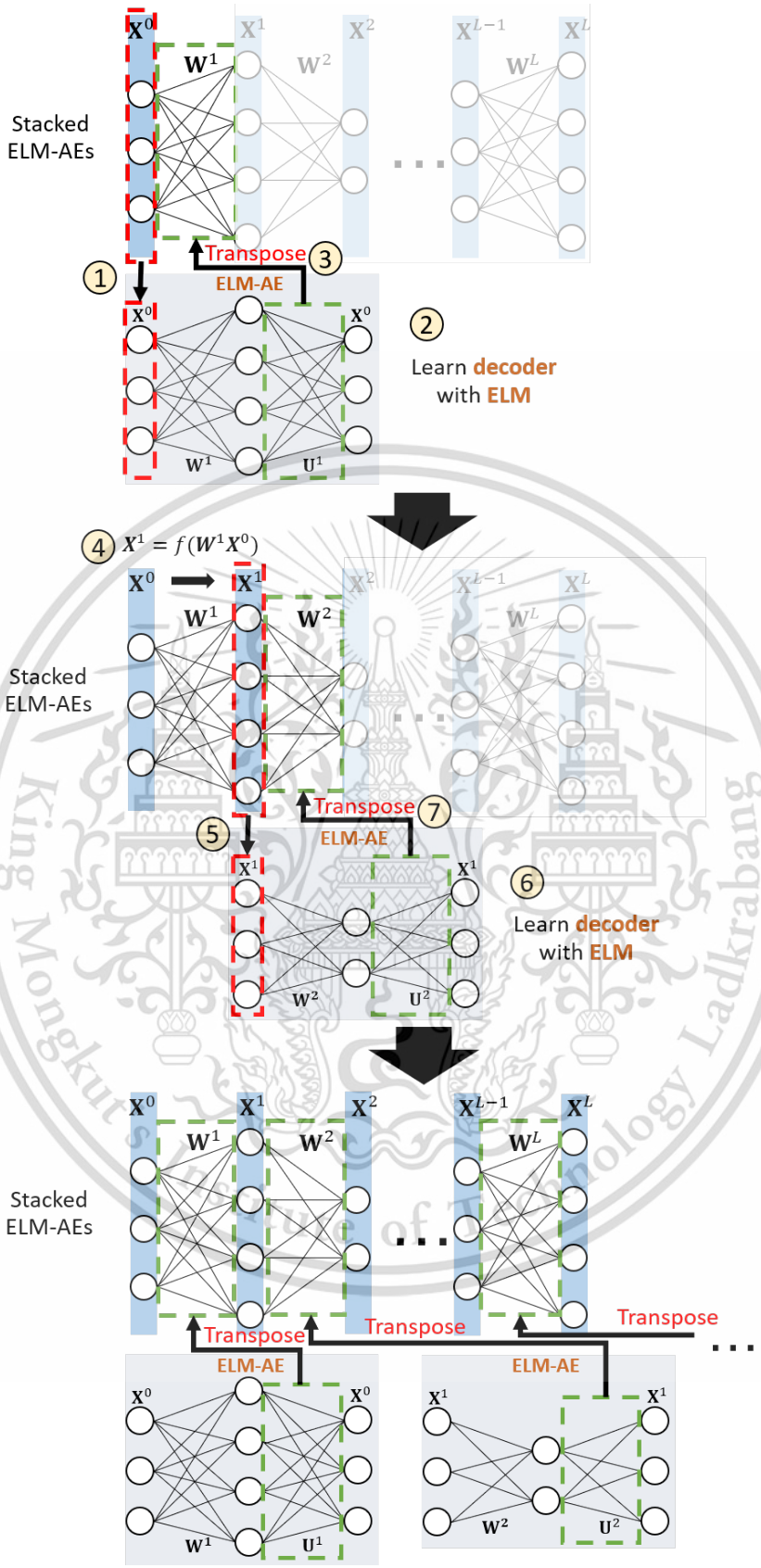


Figure 3.4: The whole learning process of the unsupervised part (stacked-ELM-AEs) of an ML-ELM with L hidden layers. The last row depicts the whole process statically.

tion of the networks to be trained. Glorot and Bengio [11] studied the behavior of activated units and error gradients and proposed an initialization scheme called a normalized initialization which is now recognized as Xavier or Glorot initialization. The effectiveness of the initialization scheme relies on an idea that the variances of the activation units should be the same when going forward through the network and the variances of the back-propagated gradient should also be the same when propagating back through the network. As a result, Glorot initialization suggests that the weights should be drawn from a uniform distribution taking into consideration the number of nodes in the connected layers. Mathematically, a weight matrix **W** should be defined as:

$$U \sim \left[-\frac{\sqrt{6}}{\sqrt{N^l + N^{l+1}}}, \frac{\sqrt{6}}{\sqrt{N^l + N^{l+1}}} \right] \tag{3.4}$$

where N^l is the number of nodes in layer l. He et al. [13] proposed another weight initialization scheme based on the same idea with an emphasis on a family of ReLU activation functions.

CHAPTER 4

PROPOSED METHODS

In this chapter, the proposed methods are explained in detail. The chapter is decomposed into sections describing thoroughly each of the the proposed methods and their mechanism. Section 4.1 introduces backward ELM (B-ELM) as a standalone training method for an SLFN. Section 4.2 proposes a representation learning scheme based on B-ELM. Finally, Section 4.3 gives details on how to use the proposed method as a pretraining scheme to train deep neural networks. Parts of this chapter were published as a research paper in 2018 [31].

4.1 Backward ELM (B-ELM)

In this section, we propose a learning algorithm based on ELM for an SLFN namely, backward ELM (B-ELM). The method is motivated by the incorrect process in ML-ELM where new representations are obtained by transforming the inputs with the transpose of decoders' weights. In this way, despite its authors' claim of the expressive power the learned representation has, it does not follow the concept of autoencoder and should be called an "autodecoder" instead. However, with traditional ELM, this problem could not be solved because weights between the input layer and the hidden layer (which corresponds to the encoder's weight in autoencoder terminology) is always randomized. This issue, therefore, leads to the idea of executing ELM backwards.

The main difference between B-ELM and ELM is that the process of ELM is executed backwards as shown in Fig. 4.1. In other words, B-ELM randomly initializes the weight matrix $\mathbf{U}$ and analytically calculates the weight matrix $\mathbf{W}$. Also, the backward flow requires additional steps in

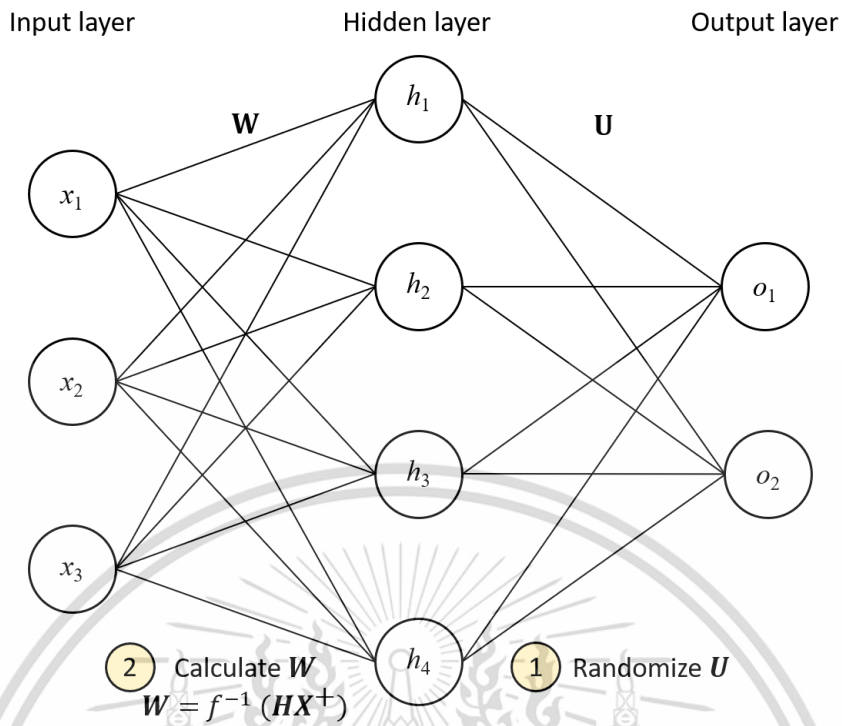


Figure 4.1: The overall process of B-ELM. It randomizes $\mathbf{U}$ (or $\mathbf{U}^{+}$) and calculate $\mathbf{W}$

calculating $\mathbf{W}$.

Overall, the process starts by randomly generating the output weight matrix $\mathbf{U}$, and then calculating its Moore-Penrose inverse $\mathbf{U}^{+}$. Alternatively, we can directly randomize the matrix $\mathbf{U}^{+}$ to speed up the process since the matrix $\mathbf{U}$ would be discarded right away after obtaining $\mathbf{U}^{+}$. Here, we would like to find a matrix $\mathbf{H}$ which can be transformed into the target matrix $\mathbf{T}$ via $\mathbf{U}$. To do this, $\mathbf{H}$ is calculated using $\mathbf{U}^{+}$ as follows:

$$\hat{\mathbf{H}} = \mathbf{U}^{+}\mathbf{T}. \tag{4.1}$$

However, in a forward computation of a neural network, $\mathbf{H}$ is calculated as

$$\mathbf{H} = f(\mathbf{W}\mathbf{X} + \mathbf{b}). \tag{4.2}$$

This rises a constraint that $\mathbf{H}$ should be bounded in to the range of the chosen activation function $f(\cdot)$. It is possible that $\hat{\mathbf{H}}$ contains values outside the range of $f(\cdot)$. Therefore, $\hat{\mathbf{H}}$ needs to be bounded to the activation function range; for example, (0, 1) for the sigmoid function or (-1, 1) for

the hyperbolic tangent function. This is done by scaling and shifting $\hat{\mathbf{H}}$ into the range. Therefore, each element $\tilde{h}_{ij}$ of a constrained hidden matrix $\tilde{\mathbf{H}}$ is then calculated as

$$\tilde{h}_{ij} = \frac{h_{ij} - min(\hat{\mathbf{H}})}{max(\hat{\mathbf{H}}) - min(\hat{\mathbf{H}})}(ub - lb) + lb, \tag{4.3}$$

where h_{ij} is the element (i, j) of $\hat{\mathbf{H}}$, ub and lb are the upper-bound and lower-bound of the activation function range, respectively.

Once the constrained hidden matrix $\tilde{\mathbf{H}}$ is obtained, the next step is to calculate the pre-activated hidden matrix $\mathbf{J}$ which is simply done by passing $\tilde{\mathbf{H}}$ into the inverse activation function $f^{-1}(\cdot)$ as

$$\mathbf{J} = f^{-1}(\tilde{\mathbf{H}}). \tag{4.4}$$

We can simplify Eq. 4.2 by using an augmented input matrix $\tilde{\mathbf{X}}$ defined as:

$$\tilde{\mathbf{X}} = \left(\mathbf{X}^\top \;\middle|\; \mathbf{1}\right)^\top, \tag{4.5}$$

and an augmented version of $\mathbf{W}$ denoted as $\tilde{\mathbf{W}}$ which is defined by:

$$\tilde{\mathbf{W}} = \left(\mathbf{W} \;\middle|\; \mathbf{b}\right), \tag{4.6}$$

to omit the bias term and rewrite the equation as:

$$\mathbf{H} = f(\tilde{\mathbf{W}}\tilde{\mathbf{X}}). \tag{4.7}$$

Then the least squares solution of the input weight matrix $\tilde{\mathbf{W}}$, is calculated using the Moore-Penrose inverse as

$$\hat{\mathbf{W}} = \mathbf{J}\mathbf{X}^+, \tag{4.8}$$

or is calculated using the regularized version of least squares method as

$$\hat{\mathbf{W}} = \mathbf{J}\tilde{\mathbf{X}}^\top(\frac{\mathbf{I}}{c} + \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top)^{-1}, \tag{4.9}$$

where $\mathbf{X}^+$ is the Moore-Penrose inverse of $\tilde{\mathbf{X}}$ and c is a regularization parameter. Once $\hat{\mathbf{W}}$ is obtained, we can calculate a new $\mathbf{H}$ as:

$$\mathbf{H} = f(\hat{\mathbf{W}}\tilde{\mathbf{X}}), \tag{4.10}$$

and find the least squares solution to solve for a new $\tilde{\mathbf{U}}$ as in ELM. In other words, we would like to find $\tilde{\mathbf{U}}$ such that the following equation holds:

$$\mathbf{T} = \tilde{\mathbf{U}}\mathbf{H}. \tag{4.11}$$

Then we calculate $\tilde{\mathbf{U}}$ using least squares method as

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{H}^+. \tag{4.12}$$

The training procedure of B-ELM is described algorithmically in Algorithm 1.

Algorithm 1 Training procedure of B-ELM

Input

- $\tilde{\mathbf{X}}$ augmented input samples
- $\mathbf{T}$ target labels
- D dimension of $\mathbf{x}_i$ for each sample i
- N number of hidden nodes

Output

- $\tilde{\mathbf{W}}$ augmented weight matrix (the encoder's weight)

1) Randomize $\mathbf{U}^+$ of size $N \times D$;
2) Calculate $\hat{\mathbf{H}}$ with Eq. 4.1
3) Calculate $\tilde{\mathbf{H}}$ with Eq. 4.3;
4) Calculate $\mathbf{J}$ with Eq. 4.4;
5) Calculate $\tilde{\mathbf{W}}$ with Eq. 4.8 or 4.9;
6) Calculate $\tilde{\mathbf{U}}$ with Eq. 4.12;

The interesting point here is that we compute the pseudoinverse of the input matrix $\mathbf{X}$ instead of the hidden matrix $\mathbf{H}$. This means that the training time, which heavily depends on the pseudoinverse computation, does not increase much when the number of hidden nodes increases. Furthermore, we can calculate $\mathbf{X}^+$ once and retrain the network by re-initializing matrix $\mathbf{U}$ many times without much additinoal computational cost as opposed to retraining with traditional ELM.

B-ELM yields impressive results for small number of hidden nodes. However, after a certain

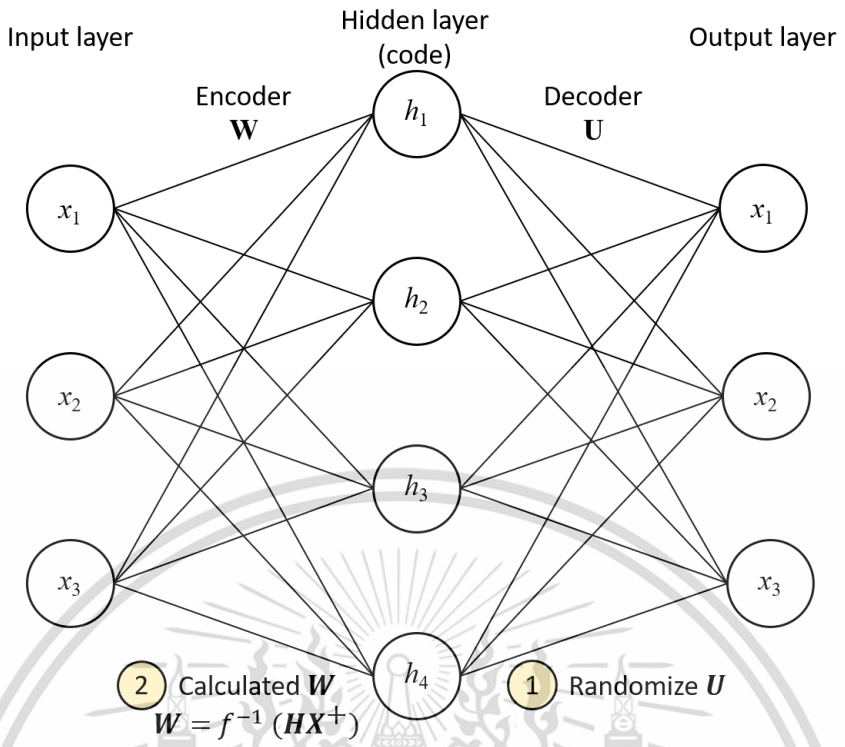


Figure 4.2: The overall process of BELM-AE. The input matrix **X** is used in the output layer instead of labels **T**.

number of nodes is reached, classification accuracy seems to be saturated and stop improving as the number of nodes increases. This problem remains to be studied. Nevertheless, the intention behind the proposal of the method is to resolve the unclear use of the transpose of decoders' weights to transform representations as done in ML-ELM. Regardless of the poor performance of B-ELM with many hidden nodes, using B-ELM for representation learning yields excellent performance as would be discussed in Section 4.2.

4.2 Representation learning with stacked-BELM-AEs

Although autoencoder is an effective unsupervised learning model [2], the traditional gradient-based learning is slow and also adds many hyperparameters to the training process which makes the process more difficult to be tuned.

On the other hand, as ELM is known for its fast training process and simple training proceudre, it would be desirable to combine the fast and simple process of ELM while preserving the original

concept and effectiveness of autoencoders. For this, we also propose a training method for an autoencoder based on B-ELM namely BELM-autoencoder (BELM-AE) [31]. It is an ELM-based learning algorithm for an autoencoder. As a learning algorithm for an autoencoder, the method seeks to find a representation which can best reconstruct the input itself.

The training process of BELM-AE is the same as B-ELM except that the target matrix $\mathbf{T}$ is replaced by the input matrix $\mathbf{X}$ as visualized in Fig. 4.2 and described as an algorithm in Algorithm 2. Therefore, the matrix $\mathbf{T}$ in Eq. 4.1 is substituted by $\mathbf{X}$ giving:

$$\hat{\mathbf{H}} = \mathbf{U}^{+}\mathbf{X}. \tag{4.13}$$

The next step is then to calculate the augmented matrix $\tilde{\mathbf{W}}$, which is the encoder's weight, as in Eq. 4.8 or 4.9.

Algorithm 2 Procedure of BELM-AE

Input
- $\tilde{\mathbf{X}}$ augmented input samples
- D dimension of $\mathbf{x}_i$ for each sample i
- N code size

Output
- $\tilde{\mathbf{W}}$ weights of the encoder

1) Randomize $\mathbf{U}^{+}$ of size $N \times D$;
2) Calculate $\mathbf{H}$ with Eq. 4.13
3) Calculate $\tilde{\mathbf{H}}$ with Eq. 4.3;
4) Calculate $\mathbf{J}$ with Eq. 4.4;
5) Calculate $\tilde{\mathbf{W}}$ with Eq. 4.8 or 4.9;

Normally, the weight matrix $\mathbf{U}$ (the decoder's weight) could be randomly drawn from any continuous distribution. However, constraining it to be an orthogonal matrix (satisfying Eq. 4.14) could eliminate redundant features which might appear in $\mathbf{H}$ during the backward flow and improve the learned representation. Note that this only applies for the case where $N < D$.

$$\mathbf{U}\mathbf{U}^{\top} = \mathbf{I}. \tag{4.14}$$

Like autoencoders, BELM-AEs can be stacked on top of one another to form a deep network as a stacked BELM-AEs. Being inspired from the flawed feature learning process of ML-ELM, the

proposed method aims to eliminate the unclear use of the transposed decoders' weights to form a deep network as proposed in the original work. To achieve the goal, stacked BELM-AEs employs a more principled way to construct a deep network using autoencoders learned with ELM-based method. The process of forming such a network is the same as in stacked autoencoders with the use of BELM-AEs instead of autoencoders.

Given $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_S]^\top$ as input samples, $\mathbf{H}^l$ as hidden matrix of layer l, a neural network with L hidden layers each with N^l hidden nodes can be learned by stacked BELM-AEs as in Algorithm 3. The input of each layer, starting from the input layer, is used to train a BELM-AE. Next, the learned encoder's weight is taken to be the weight of the current layer in the deep network. Once we obtained the weights of the current layer, we then proceed to the next layer by transforming the current input with the obtained weights to create the input of the next layer. These processes repeat for each layer and the whole process is depicted in Fig. 4.3.

Algorithm 3 Forming of a stacked-BELM-AEs

Initialize $\mathbf{X}^0$ as $\mathbf{X}$
for each hidden layer l **do**
  Train a BELM-AE with N^l hidden nodes to reconstruct $\mathbf{X}^{l-1}$ as Algorithm 2;
  Initialize $\tilde{\mathbf{W}}^l$ with $\tilde{\mathbf{W}}$ of the constructed BELM-AE;
  Calculate $\mathbf{X}^l$ as $\mathbf{X}^l = \tilde{\mathbf{W}}^l\mathbf{X}^{l-1}$
end for

Note that the use of encoders' weights to form a deep network in stacked BELM-AEs is different from stacked ELM-AEs which uses transposed decoders' weights. Recall from our objective to develop an effective pretraining method, stacked ELM-AEs could be seen as a fast pretraining method already except that the concept behind it is not sound. More specifically, an autoencoder seeks to learn a set of weights to encode the representation into a code that can be transformed back to the original form. Therefore, the important part in the learning of an autoencoder is the weights of the encoder. The concept of stacked autoencoders makes sense since it uses encoders' weights to form a deep network which could be interpreted as progressively creating better code as going deeper into the network. Hence, the use of transposed decoders' weights to form a deep network is erroneous and does not follow the original concept of autoencoders. This leads to the proposal of stacked BELM-AEs as a more proper way of applying ELM-based method to create deep networks.

The advantage of a BELM-AE over a traditional autoencoder is that it benefits from the fast

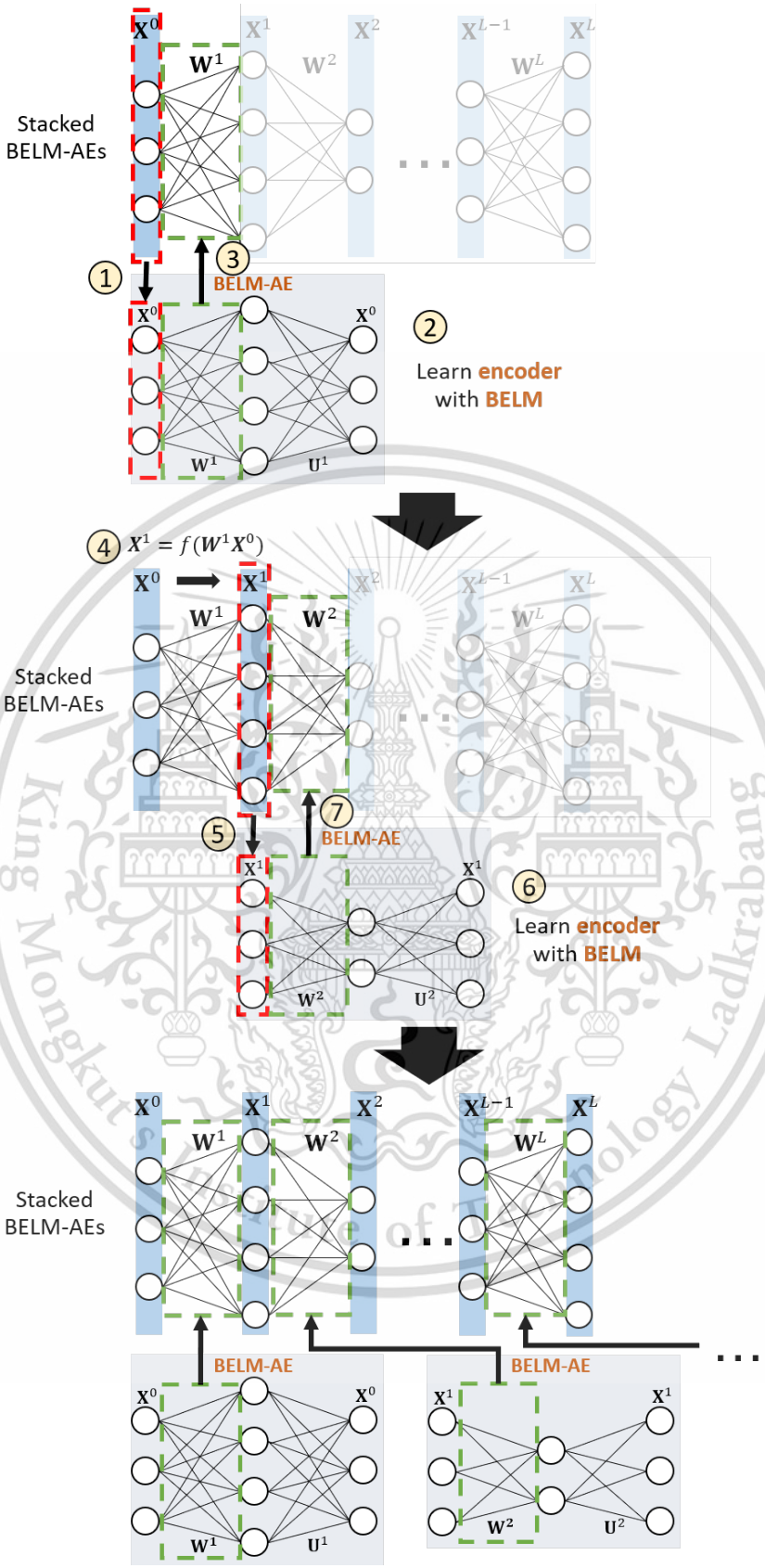


Figure 4.3: The whole training process of the unsupervised part of stacked BELM-AEs with L hidden layers (unsupervised pretraining process). The last row depicts the whole process statically.

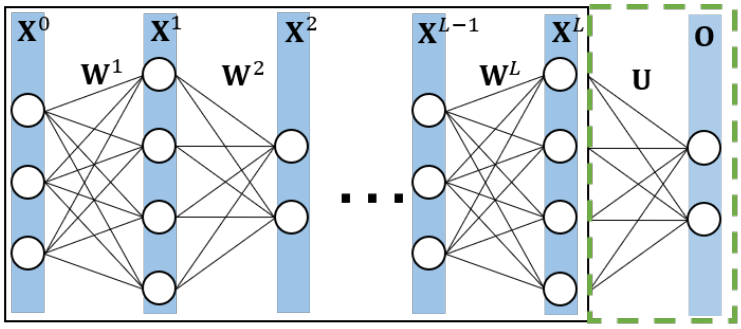


Figure 4.4: A stacked BELM-AEs before and after adding a classification layer. After a stacked BELM-AEs (the one in the black solid box) is constructed, a classification layer (the one in the green dotted box) is added to do supervised fine-tuning.

learning capability of ELM. Also, BELM-AE only requires one parameter, namely, the number of hidden nodes. Therefore, the hardwork of trying to find optimal parameters can be avoided unlike a traditional autoencoder using gradient-based learning which comes with a lot of training parameters.

4.3 Application of stacked-BELM-AEs as pretraining method

One of the popular applications of these neural network-based representation learning such as stacked RBMs (in the case of DBN) and stacked autoencoders (the proposed stacked BELM-AEs also falls into this category) is to use it as a weight initialization or pretraining method for a deep neural network. This means that the whole network would be then supervised fine-tuned.

Its authors claimed that fine-tuning is not needed as for ML-ELM. However, our experiments have shown that the performance of stacked ELM-AEs with fine-tuning is better. Therefore, also for the case of stacked BELM-AEs, only pretraining alone does not provide good classification results. We hypothesize that the cause of this is from the nature of the objective function in the learning of BELM-AE (this is also the case for other representation learning methods). The goal of BELM-AE is to find a code for reconstructing the inputs not for discriminating them which is the goal in classification tasks. Therefore, it is desirable to fine-tune the neural network with gradient-based learning and treat stacked-BELM-AEs as a pretraining method as stacked-autoencoders.

To treat stacked BELM-AEs as a pretraining method, once a pretrained network is obtained by Algorithm 3, a classification layer is added to the end of the network as depicted in Fig. 4.4. After that, the network is trained with gradient-based learning as commonly done with a traditional neural network.

Even though gradient-based learning is deployed here for fine-tuning, the pretrained networks tend to converge to optimal solutions with less training iterations than networks without pretraining or networks pretrained with other methods as would be shown in our experiment. Therefore, even if learning of stacked-BELM-AEs adds on training time, it is worth doing since the training iteration required to achieve the same accuracy when fine-tuning would be much less resulting in faster training.

CHAPTER 5

EXPERIMENTAL RESULTS AND DISCUSSIONS

Experiments have been conducted to evaluate the methods proposed in the previous chapter. This chapter describes in details the objectives and processes of each experiment and also shows and discusses the results.

5.1 Datasets

The experiments were conducted using well known medium size benchmark datasets described as follows:

- **MNIST** [26] – A dataset of 28×28 gray-scale hand-written digit images comprises 70,000 samples with 10 classes. Each of the 10 classes in the dataset represents images of a number from 0 to 9.
- **Fashion-MNIST** [41] – A dataset of 28×28 gray-scale fashion images comprises 70,000 samples with 10 classes. The 10 classes are images of T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags and ankle boots.
- **CIFAR-10** [23] – A dataset of 32×32 color images of objects comprises 60,000 samples with 10 classes. The 10 classes are images of airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships and trucks.

Figure 5.1: Examples of images from MNIST dataset [26].

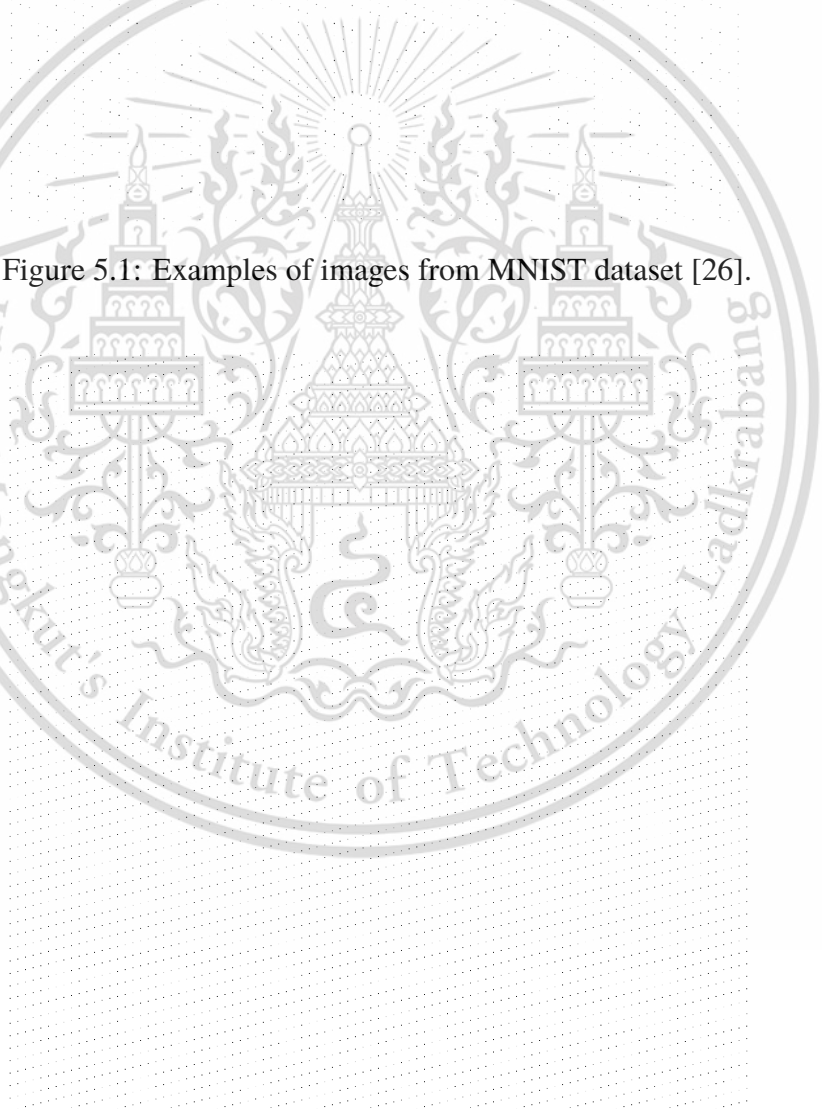


Figure 5.2: Examples of images from Fashion-MNIST dataset [41].

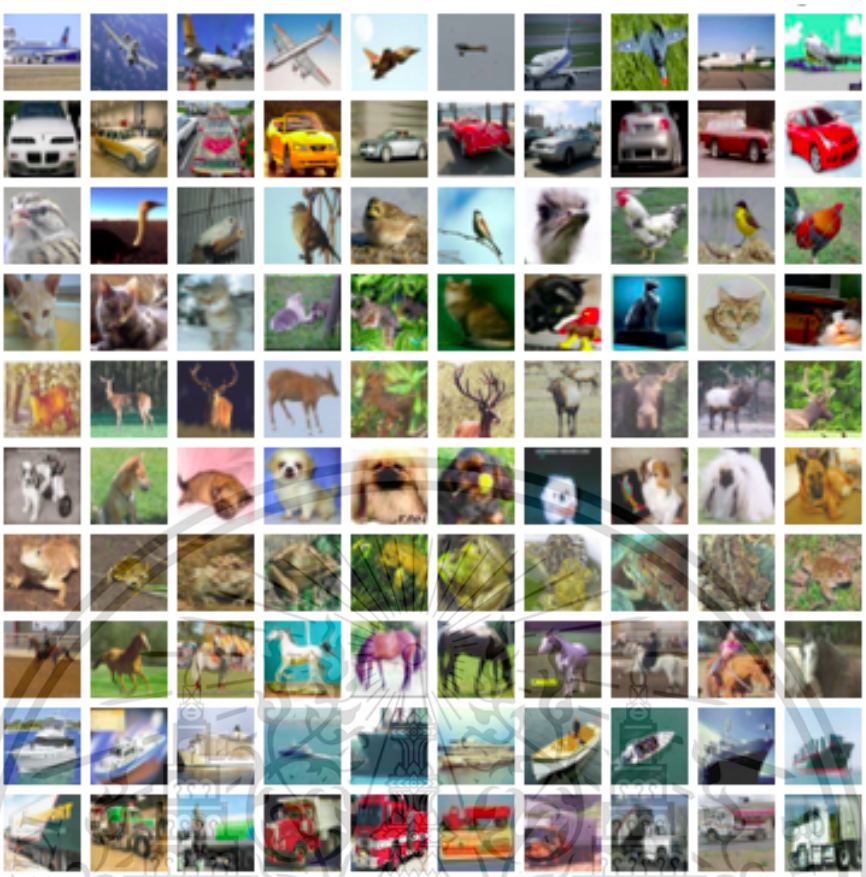


Figure 5.3: Examples of images from CIFAR-10 dataset [23].

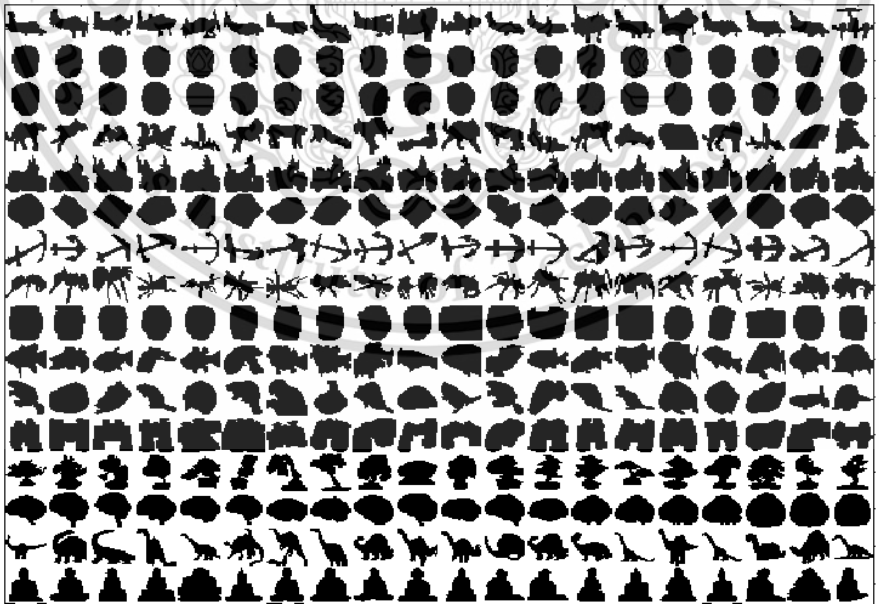


Figure 5.4: Examples of images from Caltech-Silhouettes dataset [28].

Table 5.1: Number of samples for training and testing sets of each dataset.

Dataset	Train samples	Test samples
MNIST	60,000	10,000
Fashion-MNIST	60,000	10,000
Caltech-Silhouettes	4,100	2,307
CIFAR-10	50,000	10,000

- **Caltech-Silhouettes** [28] – A dataset of 28×28 binary images of objects comprises 6,407 samples with 101 classes. The object classes are such as binary silhouette images of faces, watches, ants, elephants, fishes, pianos, lamps, cannons, etc.

Figures 5.1 - 5.4 show some examples of data in each dataset. The datasets were divided into training and testing sets as shown in Table 5.1.

5.2 Experiments

The hardware setup for the experiments is as follows:

- CPU: AMD Ryzen 1600
- GPU: NVIDIA GeForce GTX 1080ti 11GB
- RAM: DDR4 16 GB

The methods were implemented purely in Python with TensorFlow and NumPy libraries for scientific computation and parallel computing utilities. The implementation was executed mainly using GPU (all ELM based learnings were executed in CPU because of memory limitation) in all experiment configurations. Each configuration was executed 10 times.

5.2.1 Experiment 1: Classification performance

Objective

The experiment compared the classification performance of neural networks with unsupervised pretraining to those without and also to ML-ELM. Unsupervised pretrained networks in this experiment can be separated into networks pretrained with stacked BELM-AEs (the proposed pretraining

Table 5.2: Architectures of MLPs used in the experiments (all are fully connected layers).

Dataset	Number of nodes		
	Hidden layer 1	Hidden layer 2	Hidden layer 3
MNIST	500	500	500
Fashion-MNIST	500	500	500
Caltech Silhouettes	500	500	500
CIFAR-10	1,000	1,000	1,000

Table 5.3: Training epochs for each dataset of the first and second run.

Dataset	Number of training epochs	
	First run	Second run
MNIST	50	200
Fashion-MNIST	50	200
Caltech-Silhouettes	250	1,000
CIFAR-10	80	320

method), stacked ELM-AEs and stacked autoencoders. Networks without pretraining consist of those with random initialization and the other one with Xavier initialization.

Experiment setup

A set of neural networks were constructed. The architectures are fixed with in the same dataset and are shown in Table 5.2. Other hyperparameters governing the training process are shown in Table 5.4. For the configuration with no pretraining, a neural network was trained right away with gradient-based learning for the specified number of epochs. For the case with pretraining, a network was pretrained with the pretraining method first and then was fine-tuned using gradient-based learning for the specified number of epochs as well. We also varied the number of training epochs for each configuration as shown in Table 5.3 to see if the performance improves with more training epochs.

Result and discussion

Classification accuracies (mean and standard deviation) and total training time of each method were measured and reported in Tables 5.5 and 5.6. Some part of the results were adapted from [31]. The term BP in the tables refers to the use of gradient-based iterative learning with back-propagation.

Table 5.4: Training parameters for all executions.

Parameter	Value
Batch size	200
Learning rate	0.0001
L2 weight (α)	0.05
Keeping probability (dropout)	0.5
Optimization method	ADAM [22]

From the results, it can be seen that neural networks with unsupervised pretraining or with a weight initialization scheme achieve better performance on average and need lower number of training iterations to obtain the same accuracy as opposed to those with pure random weight initialization. Furthermore, stacked BELM-AEs significantly outperforms other pretraining methods (and also the random initialization setting) in all cases. For the case of networks initialized with Xavier initialization, the proposed method only yields slightly less accuracy on Caltech-Silhouettes and CIFAR-10 datasets while performing obviously better on MNIST and Fashion-MNIST datasets. Also, the results suggest that using stacked ELM-AEs as a pretraining method (with fine-tuning) is better than using it purely with ELM (for the case of ML-ELM). Although an ML-ELM is trained faster but a stacked ELM-AEs with fine-tuning could achieve significantly higher accuracy on most configurations. Furthermore, for a stacked ELM-AEs, we can improve the accuracy without affecting the training time by adjusting training parameters such as learning rate but this is not the case for an ML-ELM. Lastly, the pretraining times of the pretraining methods used in the experiment are shown in Table 5.7. Obviously, stacked BELM-AEs is slower than stacked ELM-AEs but is faster than traditional stacked autoencoders trained with gradient-based learning.

5.2.2 Experiment 2: Measuring convergence rate

Objective

The experiment measured and compared how fast (in terms of the number of training iterations) the training and testing accuracies of each training setting converges to a stable point (stop increasing).

Table 5.5: Performance comparison in terms of classification accuracy and total training time for the first run on MNIST, Fashion-MNIST Caltech-Silhouettes and CIFAR-10 datasets. The networks were trained with 50 epochs for both MNIST and Fashion-MNIST, 250 epochs for Caltech-Silhouettes and 80 epochs for CIFAR-10 datasets when fine-tuning with BP (no training epochs for ML-ELM).

Training method	MNIST			Fashion-MNIST			Caltech-Silhouettes			CIFAR-10		
	Acc (%)	SD	Time (s)	Acc (%)	SD	Time (s)	Acc. (%)	SD	Time (s)	Acc. (%)	SD	Time (s)
ML-ELM [21]	94.27	0.22	**37.09**	82.49	0.39	**36.25**	**63.85**	**0.62**	**3.75**	40.63	0.58	**95.98**
BP + random init.	81.63	0.32	43.09	81.16	0.43	44.37	23.68	1.49	17.50	33.08	0.62	121.39
BP + Xavier init.	96.89	**0.13**	43.57	74.54	0.27	44.90	63.32	0.78	16.71	**53.97**	**0.32**	134.49
BP + stacked autoencoders	97.11	0.43	195.65	79.70	0.41	108.48	30.25	4.21	28.63	20.71	1.50	384.68
BP + stacked ELM-AEs	96.69	0.21	80.98	81.06	2.26	70.54	47.79	3.48	20.24	30.79	3.04	194.74
BP + stacked BELM-AEs (proposed)	**98.10**	0.31	90.62	**88.52**	**0.20**	80.13	60.30	1.13	24.23	52.90	0.42	347.76

Table 5.6: Performance comparison in terms of classification accuracy and total training time for the second run on MNIST, Fashion-MNIST, Caltech-Silhouettes and CIFAR-10 datasets. The networks were trained with 200 epochs for both MNIST and Fashion-MNIST, 1,000 epochs for Caltech-Silhouettes and 320 epochs for CIFAR-10 datasets when fine-tuning with BP (no training epochs for ML-ELM).

Training method	MNIST			Fashion-MNIST			Caltech-Silhouettes			CIFAR-10		
	Acc (%)	SD	Time (s)	Acc (%)	SD	Time (s)	Acc. (%)	SD	Time (s)	Acc. (%)	SD	Time (s)
ML-ELM [21]	94.27	0.22	**37.09**	82.49	0.39	**36.25**	63.85	0.62	**3.75**	40.63	0.58	**95.98**
BP + random init.	95.43	0.13	168.93	84.35	**0.13**	177.37	42.39	0.91	71.87	45.36	**0.31**	559.51
BP + Xavier init.	98.31	0.07	169.47	81.63	**0.13**	172.11	**64.07**	**0.22**	61.59	**51.68**	1.04	522.84
BP + stacked autoencoders	98.20	0.26	331.82	84.94	0.11	235.31	48.67	1.71	87.37	32.78	1.19	825.14
BP + stacked ELM-AEs	98.31	0.07	238.69	85.58	1.68	192.91	58.70	2.22	57.66	38.90	4.82	636.77
BP + stacked BELM-AEs (proposed)	**98.45**	**0.06**	256.79	**88.34**	0.41	202.71	62.58	0.41	64.61	50.75	1.57	770.36

Table 5.7: Pretraining time of each unsupervised pretraining method for each dataset.

Pretraining method	Pretraining time (s)			
	MNIST	Fashion-MNIST	Caltech Silhouettes	CIFAR-10
Stacked autoencoders	133.35	109.66	11.25	262.79
Stacked ELM-AEs	28.48	29.80	2.64	74.77
Stacked BELM-AEs (proposed)	40.10	39.67	10.53	220.63

Experiment setup

The neural networks were constructed with the same parameters as in Section 5.2.1 except that in each training, each network was trained with 100,000 iterations to ensure that testing accuracies from all configurations are saturated before the training ends, and the training and testing accuracies were measured every 1,000 iterations to make the plotted graphs easy to read. In these executions the activation function used is hyperbolic tangent since it produces the best results in general for all experiment settings. Convergence rates are shown by plotting the number of training iterations against the training and testing accuracies.

Result and discussion

The results are shown in Figs. 5.5–5.12 for training and testing accuracies of MNIST, Fashion-MNIST, Caltech-Silhouettes and CIFAR-10 respectively. It can be seen from these graphs that the pretrained networks perform better than networks with random initialization (meaning the testing accuracies increase faster with respect to the number of training iterations). Furthermore, among the pretraining method, the ELM-based ones are superior. Moreover, the proposed pretraining method, i.e., stacked BELM-AEs significantly outperforms other methods on average. Only in the case of Caltech-Silhouettes and CIFAR-10 that the stacked BELM-AEs are the second best inferior to Xavier initialization. To figure out the reasons behind this, we also have to look at the graphs of training accuracies (as shown in Figs. 5.5, 5.7, 5.9, 5.11). It can be seen that for all datasets, training accuracies of stacked BELM-AEs going up the fastest. The gaps between training and testing accuracies for MNIST and FASHION-MNIST of stacked BELM-AEs are acceptable. However, those for Caltech-Silhouettes and CIFAR-10 are too large. These behavior express a kind of overfitting which could be solved by controlling the regularization parameters (can be done both in the unsupervised and the supervised phase).

Figure 5.5: Number of training iterations plotted against training accuracy on MNIST dataset.

Figure 5.6: Number of training iterations plotted against testing accuracy on MNIST dataset.

Figure 5.7: Number of training iterations plotted against training accuracy on Fashion-MNIST dataset.

Figure 5.8: Number of training iterations plotted against testing accuracy on Fashion-MNIST dataset.

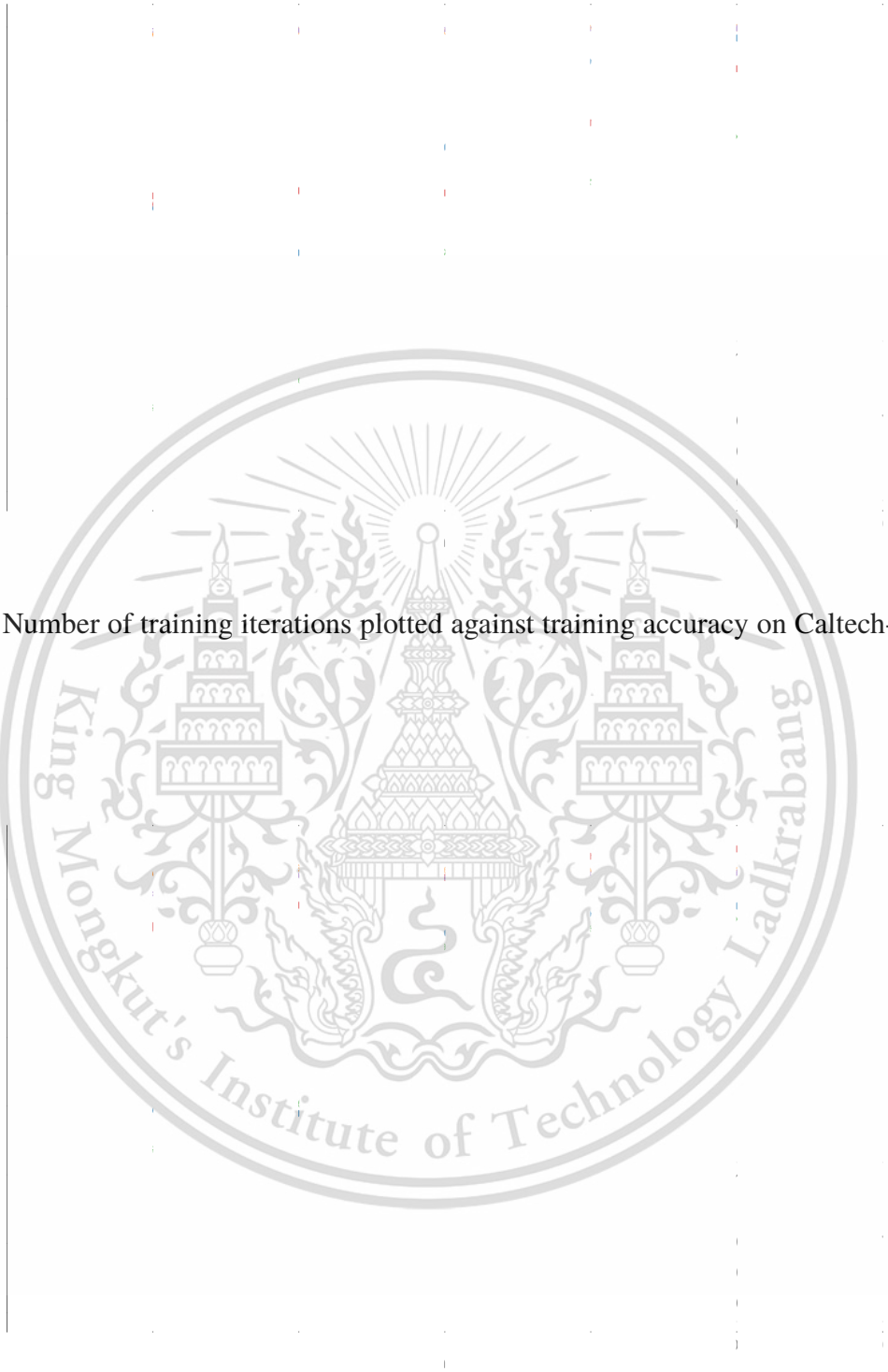


Figure 5.9: Number of training iterations plotted against training accuracy on Caltech-Silhouettes dataset.

Figure 5.10: Number of training iterations plotted against testing accuracy on Caltech-Silhouettes dataset.

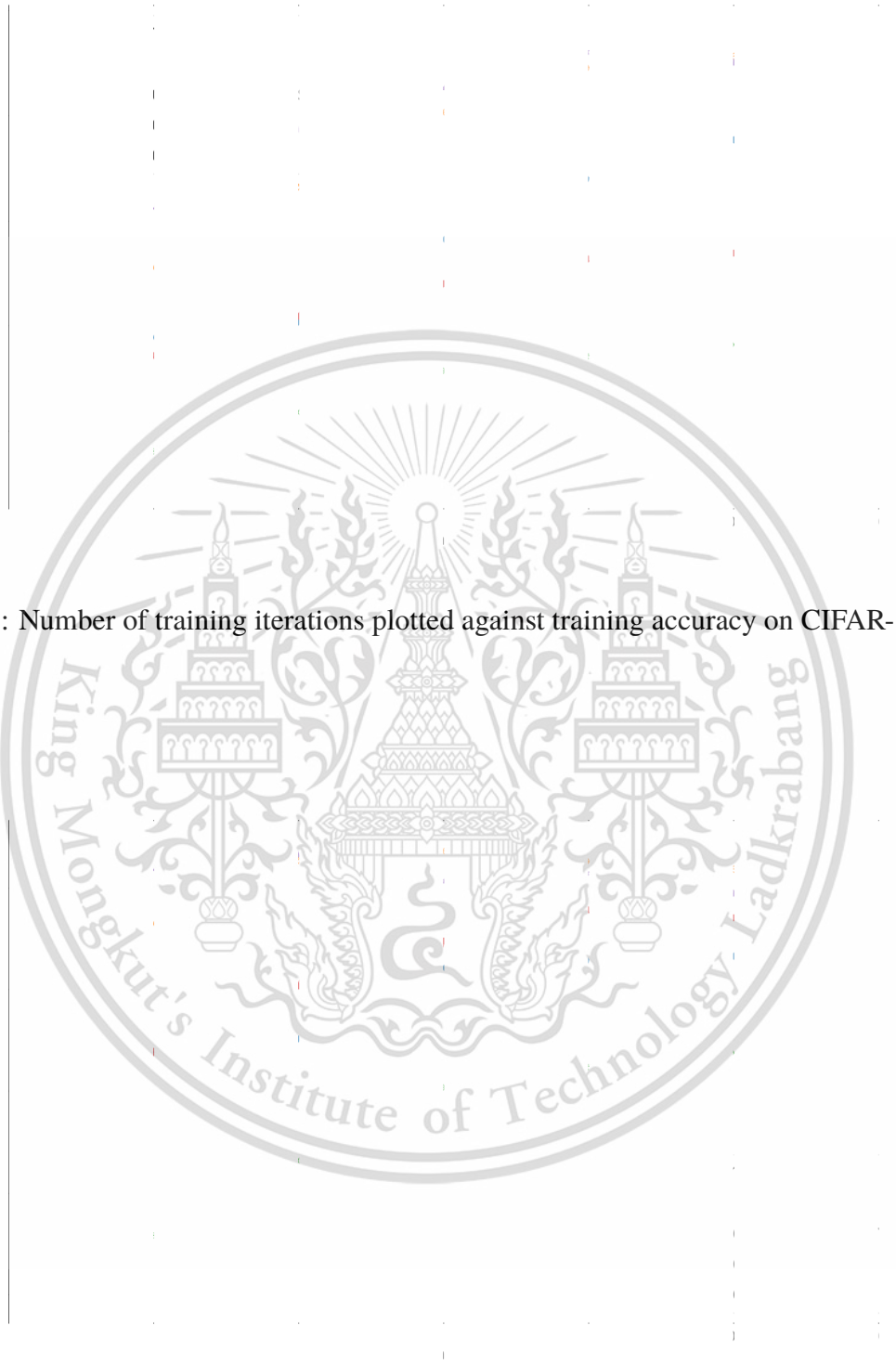


Figure 5.11: Number of training iterations plotted against training accuracy on CIFAR-10 dataset.

Figure 5.12: Number of training iterations plotted against testing accuracy on CIFAR-10 dataset.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In conclusion, we have proposed a learning scheme for an SLFN namely, B-ELM, and extended it for the case of autoencoder as BELM-AE. We also apply BELM-AE to learn for the case of deep neural networks by stacking BELM-AEs on top of one another (stacked BELM-AEs). Furthermore, we propose an application of stacked BELM-AEs as a pretraining method for deep neural networks. Lastly, we suggest the use of stacked ELM-AEs (unsupervised training part of ML-ELM) as a pretraining method as well. Conclusions about the proposed methods and the findings of this research could be drawn as the following points:

1) Unsupervised pretraining should be done as experimental results have shown that most of the cases, with unsupervised pretraining, better performance can be achieved with the same number of training epochs.

2) It also suggests that using ELM-based pretraining is superior to traditional pretraining with autoencoders (trained with gradient-based learning) as confirmed by the considerably faster training time and better accuracy achieved.

3) Using greedy-layer-wise unsupervised training alone to form a deep network as in ML-ELM is not enough to get good performance, fine-tuning is still needed. Therefore, stacked ELM-AEs approach should be used as a pretraining method which from the experimental results,

yields better performance than the case of ML-ELM. Nevertheless, with its unclear process of representation learning, the method, still, is inferior to the proposed stacked BELM-AEs.

4) Stacked ELM-AEs is not sound as it uses the transpose of the decoders' weights to transform representation which leads to the proposal of stacked BELM-AEs as a pretraining method.

5) Pretraining with stacked BELM-AEs, has significantly outperformed other pretraining methods namely, stacked ELM-AEs and stacked autoencoders in terms of classification performance while requiring comparable training time to stacked ELM-AEs which is considerably faster than stacked autoencoders learned with gradient-based method.

6) Stacked BELM-AEs also yields better performance than a well known weight initialization method, Xavier initialization, in most cases.

7) It has also been shown that the testing accuracy of a neural network pretrained with stacked BELM-AEs, on average, converges much faster and to a higher degree than other pretraining methods which as a result, amounts to faster training.

6.2 Future works

Future works include but not limited to researches on how to improve performance of B-ELM in the case of many hidden nodes. Furthermore, applications of stacked BELM-AEs on other types of networks such as convolutional and recurrent neural networks are interesting to be investigated. Moreover, it is still not very clear why stacked BELM-AEs yields such impressive results. Therefore, a study on the behavior of the network pretrained with stacked-BELM-AEs is needed to fully understand its potential and to further develop the method. Also, training a deep neural network purely with ELM-based learning scheme is another interesting area and could be further explored.

REFERENCES

[1] D. Bahdanau, K. Cho and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, 2014.

[2] Y. Bengio, A. Courville and P. Vincent, "Representation learning: A review and new perspectives," IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol.35, no.8, pp.1798–1828, 2013.

[3] Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, "Greedy layer-wise training of deep networks," Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS), pp.153–160, 2007.

[4] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[5] Y. Boureau and Y. LeCun, "Sparse feature learning for deep belief networks," Proceedings of the 25th Advances in Neural Information Processing Systems (NIPS), pp.1185–1192, 2008.

[6] J. Cao, Z. Lin and G.B. Huang, "Self-adaptive evolutionary extreme learning machine," Neural Processing Letters, vol.36, no.3, pp.285–305, 2012.

[7] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," Proceedings of the Academy of sciences, pp.536–538, 1847.

[8] K. Cho, B.V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv preprint arXiv:1406.1078, 2014.

[9] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR), pp.248–255, 2009.

[10] D. Erhan, Y. Bengio, A. Courville, P.A. Manzagol and P. Vincent, "Why does unsupervised pre-training help deep learning?," Journal of Machine Learning Research, vol.11, pp.625–660, 2010.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTAT), pp.249–256, 2010.

[12] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[13] K. He, X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp.1026–1034, 2015.

[14] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.

[15] Y.L. He, Z.Q. Geng, Y. Xu, and Q.X. Zhu, "A hierarchical structure of extreme learning machine (HELM) for high-dimensional datasets with noise," Neurocomputing, vol.128, pp.407–414, 2014.

[16] G.E. Hinton, S. Osindero and Y.W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol.18, no.7, pp.1527–1554, 2006.

[17] G.E. Hinton and R.R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol.313, no.5786, pp.504–507, 2006.

[18] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R.R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580, 2012.

[19] G.B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), vol.42, no.2, pp.513–529, 2012.

[20] G.B. Huang, Q.Y. Zhu, C.K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," Proceedings of 2004 International Joint Conference on Neural Networks (IJCNN), vol.2, pp.985–990, 2004.

[21] L.C.C. Kasun, H. Zhou, G.B. Huang and C.M. Vong, "Representational learning with ELMs for big data," IEEE Intelligent Systems, vol.28, no.6, pp.31–34, 2013.

[22] D.P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," Proceedings of the 3rd International Conference on Learning Representation (ICLR), 2015.

[23] A. Krizhevsky and G.E. Hinton, "Learning multiple layers of features from tiny images,", Technical report, University of Toronto, vol.1, no.4, p.7, 2009.

[24] A. Krizhevsky, I. Sutskever and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," Proceedings of the 25th Advances in Neural Information Processing Systems (NIPS), pp.1097–1105, 2012.

[25] Y. LeCun, Y. Bengio and G.E. Hinton, "Deep learning," Nature, pp.436–444, 2015.

[26] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol.86, no.11, pp.2278–2324, 1998.

[27] N.Y. Liang, G.B. Huang, P. Saratchandran, "A fast and accurate online sequential learning algorithm for feedforward networks," IEEE Transactions on Neural Networks, vol.17, no.6, pp.1411–1423, 2006.

[28] B.M. Marlin, K. Swersky, B. Chen and N. de Freitas, "Inductive principles for restricted Boltzmann machine learning," Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTAT), pp.509–516, 2010.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, et al., "Human-level control through deep reinforcement learning," Nature, vol.518, pp.529–533, 2015.

[30] V. Nair and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines," Proceedings of the 27th International Conference on Machine Learning (ICML), pp.807–814, 2010.

[31] P. Noinongyao, U. Watchareeruetai, "An Extreme Learning Machine Based Pretraining Method for Multi-Layer Neural Networks," Proceedings of the Joint 10th International Conference on Soft Computing and Intelligent Systems and the 19th International Symposium on Advanced Intelligent Systems, 2018.

[32] R. Penrose, "A generalized inverse for matrices," Mathematical Proceedings of the Cambridge Philosophical Society, vol.51, no.3, pp.406–413, 1955.

[33] S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction." Proceedings of the 28th International Conference on Machine Learning (ICML), pp.833–840, 2011.

[34] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning representations by back-propagating errors," Nature, vol.323, no.6088, pp.533-–536, 1986.

[35] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol.529, pp.484–489, 2016.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., "Going deeper with convolutions," Proceedings of IEEE conference on Computer Vision and Pattern Recognition (CVPR), pp.1–9, 2015.

[38] T. Tieleman and G.E. Hinton, Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning, Technical report, 2012.

[39] P. Vincent, H. Larochelle, Y. Bengio and P.A. Manzagol, "Extracting and composing robust features with denoising autoencoders," Proceedings of the 25th International Conference on Machine Learning (ICML), pp.1096–1103, 2008.

[40] P. Vincent, H. Larochelle, I. Lajoie, Y.Bengio and P.A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," Journal of Machine Learning Research, vol.11, pp.3371–3408, 2010.

[41] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," arXiv preprint arXiv:1708.07747, 2017.

[42] Y. Yoo and S.Y. Oh, "Fast training of convolutional neural network classifiers through extreme learning machines," Proceedings of 2016 International Joint Conference on Neural Networks (IJCNN), pp.1702–1708, 2016.

[43] M.D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," European Conference on Computer Vision (ECCV), Cham, pp.818–833, 2014.

SCIS&ISIS2018
with ISWS2018

December 5-8, 2018
Toyama International Conference Center, Toyama, Japan

10th International Conference on Soft Computing and Intelligent Systems
and 19th International Symposium on Advanced Intelligent Systems
in conjunction with Intelligent Systems Workshop 2018

SOFT

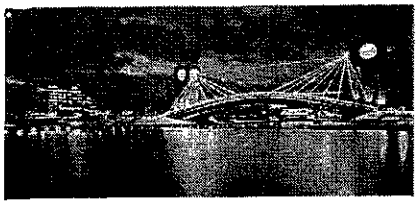
KIIS

IEEE SMC
Systems, Man, and Cybernetics Society

INTERNATIONAL FUZZY SYSTEMS ASSOCIATION

NAFIPS
North American Fuzzy Information Processing Society

TOYAMA
Prefectural
University



SCIS & ISIS 2018

10th International Conference on Soft Computing and Intelligent Systems
19th International Symposium on Advanced Intelligent Systems
5-8 December 2018 • Toyama, Japan

SCIS-ISIS 2018

Author Index

Search

Symposium on Advanced Intelligent Systems

http://scis2018.j-soft.org

Install Microsoft Media Player
Install Apple QuickTime Player
Install VLC Player
WinZip

Proceedings

SCIS-ISIS 2018
IEEE Catalog Number: CFP1864T-USB
ISBN: 978-1-5386-2633-7

17:20-17:40 **Integrating the Dynamic Password Authentication with Possession Factor and CAPTCHA**
Th3-4-4 Detchasit Pansa and Thawatchai Chomsiri
17:40-18:00 **IoT-based Smart Mat System for Remote Monitoring System of User**
Th3-4-5 Jung-Sook Kim
18:00-18:20 **Design of the Smart Query-Response Interface of Remote Emergency Medical Image Reading System in Mobile Environment**
Th3-4-6 Jung-Sook Kim and Taesub Chung

[Th4-4] OS: Affective Engineering and its Interactive Approach
Chair: Muneyuki Unehara (Nagaoka University of Technology, Japan)
16:20-16:40 **Utilizing Gamification to Improve User Participation in Online Judge**
Th4-4-1 Cynthia Sinly, Andre Rusli and P. M. Winarno
16:40-17:00 **Snack Food Texture Estimation by Neural Network**
Th4-4-2 Shigeru Kato, Ryuji Ito, Naoki Wada, Tomomichi Kagawa and Masayoshi Yamamoto
17:00-17:20 **Estimation of Human Movement State by Smartphone Sensor Information without GPS**
Th4-4-3 Shino Iwashita, Yuuki Shimanuki and Atsushi Hayashi
17:20-17:40 **Creation of Ideal User's Voice using User's own UTAU Voice and Interactive Genetic Algorithm**
Th4-4-4 Asami Inoue, Kota Nomura and Makoto Fukumoto
17:40-18:00 ***Multi-Objective Decision Support by Individual Affective Preferences Identification**
Th4-4-5 Muneyuki Unehara, Shin'Ya Yamazaki, Koichi Yamada and Izumi Suzuki
18:00-18:20 ***Selection of Impression Word for Generating Rhythm Pattern of Drum Set**
Th4-4-6 Mio Suzuki and Yuki Shimazu

[Th5-4] GS: Fuzzy Control, Fuzzy Sets and Logic, Fuzzy Systems
Chair: Tomoharu Nakashima (Osaka Prefecture University, Japan)
16:20-16:40 **Adaptive Fractional High Order Sliding Mode Fuzzy Control of Active Power Filter**
Th5-4-1 Siyang Li and Juntao Fei
16:40-17:00 **On α-Migrativity of Fuzzy Implications and the Generalised Laws of Importation**
Th5-4-2 Michal Baczynski, Balasubramaniam Jayaram and Radko Mesiar
17:00-17:20 **Some Closure Properties of Fuzzy Multiset Regular Languages**
Th5-4-3 Pavel Martinek
17:20-17:40 **An Adaptive Controller with Dynamic Gain Adjustment for Wheeled Mobile Robot**
Th5-4-4 Bing-Gang Jhong and Mei-Yung Chen
17:40-18:00 **A Note on Brake System of Train Set with Wheel Slide Protection System**
Th5-4-5 Yuya Kanazawa, Yasunori Endo, Shin-Ichi Nakazawa and Daisuke Hijikata
18:00-18:20 ***Soft Confidence-Weighted Learning for Fuzzy Classifiers and its Application to Dynamic Classification Problems**
Th5-4-6 Tomoharu Nakashima

[Th6-4] GS: Machine Learning (1)
Chair: Fusaomi Nagata (Sanyo-Onoda City University, Japan)
16:20-16:40 **An Extreme Learning Machine based Pretraining Method for Multi-Layer Neural Networks**

Th6-4-1 Pavit Noinongyao and Ukrit Watchareeruetai

16:40-17:00 ***Non-parametric Continuous Self-Organizing Map**

Th6-4-2 Ryuji Watanabe, Hideaki Ishibashi, Tohru Iwasaki and Tetsuo Furukawa

17:00-17:20 ***Design and Training Application for Deep Convolutional Neural Networks**

Th6-4-3 Fusaomi Nagata, Kenta Tokuno and Keigo Watanabe

17:20-17:40 ***A Discussion on Improve Response for Successive Input Patterns in SpikeProp**

Th6-4-4 Kengo Onoda, Haruhiko Takase, Hidehiko Kita and Hiroharu Kawanaka

17:40-18:00 **Consistency Assessment between Diploma Policy and Curriculum Policy using Character-level CNN**

Th6-4-5 Kazuteru Miyazaki and Msaaki Ida

18:00-18:20 ***Solar Irradiance Forecasting with Convolutional Long Short-Term Memory Network using Sky Images**

Th6-4-6 Tomohiro Nagatani, Koki Nishimura, Takeru Moriyama and Shunji Maeda

[Th7-4] OS: Recent Trends of Clustering Methodologies (1) + GS: Decision Making and Risk Management (1)

Chair: Katsuhiro Honda (Osaka Prefecture University, Japan)

16:20-16:40 **On Fuzzy Clustering for Incomplete Spherical Data and for Incomplete Multivariate Categorical Data**

Th7-4-1 Yuchi Kanzawa

16:40-17:00 **Performance Comparison of Collaborative Filtering using Fuzzy Clustering for Spherical Data**

Th7-4-2 Tadafumi Kondo and Yuchi Kanzawa

17:00-17:20 **A Study on Clustering Unweighted Network Data and its Evaluation using Cluster Validity Measure**

Th7-4-3 Daiki Kobayashi and Yukihiro Hamasuna

17:20-17:40 **Refining Fuzzy c-Means Membership Functions to Assimilate A Priori Knowledge of Cluster Sizes**

Th7-4-4 Jiarui Li, Yukio Horiguchi and Tetsuo Sawaragi

17:40-18:00 **Automatic Estimation of Cluster Number in Fuzzy Co-clustering based on Competition and Elimination of Clusters**

Th7-4-5 Seiki Ubukata, Kazuki Yanagisawa, Akira Notsu and Katsuhiro Honda

18:00-18:20 **A Fuzzy Approach to Weight Vector from a Group of Pairwise Comparison Matrices based on Inclusion Relation**

Th7-4-6 Tomoe Entani

2018 Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems

An Extreme Learning Machine Based Pretraining Method for Multi-Layer Neural Networks

Pavit Noinongyao and Ukrit Watchareeruetai*
International College
King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Road, Ladkrabang, Bangkok 10520 Thailand
Email: 60610022@kmitl.ac.th, *ukrit.wa@kmitl.ac.th

***Abstract*—One approach in training a deep neural network to perform effectively is to do unsupervised pretraining on each layer, followed by fine-tuning the whole network. A common way is to train an unsupervised model of neural network such as restricted Boltzmann machines or autoencoders and stack them on top of another. Although these unsupervised pretraining approaches yield good performance, relying on back-propagation, due to iterative learning process, they still suffer from a long pre-training time. Extreme learning machine (ELM) is an analytical training approach which is extremely fast and gives a solution with a good generalization performance. In this paper, we apply a new ELM based unsupervised learning, named backward ELM based autoencoder (BELM-AE), to pretrain each layer of a neural network before using a back-propagation based learning algorithm to fine-tune the whole network. Experimental results show that the new pretraining method requires significantly shorter training time and also yields better testing performance on various datasets.**

***Index Terms*—extreme learning machine; pretraining; autoencoder; backward extreme learning machine**

I. Introduction

Greedy layer-wise pretraining has been successfully adopted to help in training deep neural networks effectively [4], [16]. The rationale behind the pretraining is to find a set of initial weights that are better than random values, hopefully to help speed up the subsequent learning process. Commonly, restricted Boltzmann machines (RBMs) [4] or autoencoders [16] are trained in an unsupervised manner, layer-by-layer, to form a deeper network. The whole network is then fine-tuned using a learning algorithm based on gradient back-propagation (BP) [14]. Although the pretraining approach enables more effective learning of deep neural networks, it also considerably increases amount of time to neural network training due to iterative process of a BP-based algorithm.

Extreme learning machine (ELM) [6] is an extremely fast learning scheme originally proposed to learn a single-hidden-layer feedforward neural network (SLFN) (Fig.1). ELM uses an analytical approach to learn weights for a neural network. This is done by randomly initializing the input weight matrix W and calculating the output weight matrix U as the least squares solution using the Moore-Penrose inverse [11]. The learning scheme is more than 100 times faster than BP-based learning, tested on various benchmark datasets, while still gives an outstanding performance in terms of accuracy [6]. There are many variants of ELM proposed so far; for example, online sequential ELM (OS-ELM) [12], self-adaptive differential evolutionary ELM (SaDE-ELM) [1], and ELM-based convolutional neural network (CNN-ELM) [17], which all share the same core concept.

With the superiority of deep architectures over shallow ones, ELM has also been extended to construct a multi-layer perceptron (MLP) as done in multi-layer ELM (ML-ELM) [7] and hierarchical ELM (H-ELM) [3]. Specifically, ML-ELM trains an MLP by unsupervised training one layer at a time and stacks the trained layers on top of the previous one (greedy layer-wise pretraining), followed by training the last layer using the traditional ELM. The unsupervised training method underlying ML-ELM and H-ELM is a variant of ELM namely ELM-based autoencoder (ELM-AE) which has been proposed for training an autoencoder [7]. Despite being reported to have good performance, the methods rely purely on greedy layer-wise unsupervised learning (except the last layer). Without fine-tuning the whole network, the learned weights cannot be optimal for classification.

In this work, we apply an ELM-based unsupervised learning to pretrain a neural network and fine-tune it using BP-based algorithm to achieve better performance without increasing much training time. Furthermore, as the rationale behind ELM-AE that uses the transpose of learned decoder to transform inputs into a new feature space is unclear, a new method, named backward ELM based autoencoder (BELM-AE), is proposed to be more intuitive and closer to autoencoder but still preserve advantage of ELM's fast training procedure. Experimental results show that using the proposed method for pretraining can yield significantly better performance. The main contributions of this research are as follows: 1) propose the use of ELM to perform pretraining; 2) propose BELM-AE which is better than ELM-AE for training an autoencoder.

The remaining of this paper is organized as follows. Section II provides the background knowledge about the traditional ELM and ELM-AE. Section III explains the detail of the proposed BELM-AE as a pretraining method for a deep neural network. Section IV presents and discusses experimental results. Section V concludes the paper and gives some ideas for future research.

978-1-5386-2633-7/18/$31.00 ©2018 IEEE
DOI 10.1109/SCIS-ISIS.2018.00123

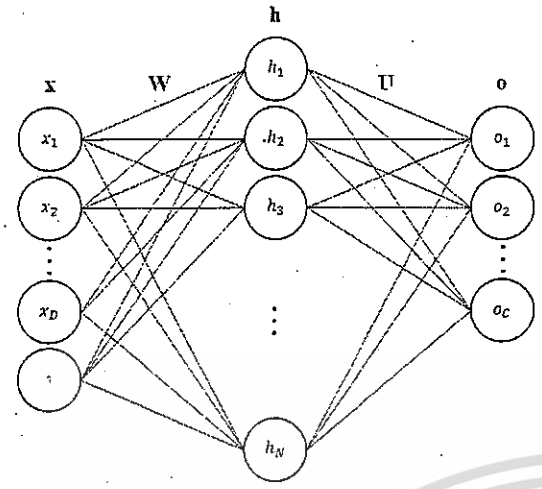


Fig. 1. Single-hidden-layer feedforward neural network with input dimension of size D, N hidden nodes and output dimension of size C.

II. Background Knowledge

In this section, the key idea and methodology of ELM and ELM-AE are provided.

A. Extreme Learning Machine

ELM was first proposed to be used with a SLFN (Fig. 1), which comprises two sets of weights: $\mathbf{W}$ connecting the input layer to a hidden layer (on the left side) and $\mathbf{U}$ connecting the hidden layer to the output layer (on the right side). Let D denotes the input dimension, N the number of hidden nodes, and C the output dimension. Given an input instance $\tilde{\mathbf{x}} = [x_1\ x_2\ \dots\ x_D]^\top$, its augmented input vector $\mathbf{x} = [x_1\ x_2\ \dots\ x_D\ 1]^\top$ is fed into the SLFN. The output of the hidden layer $\mathbf{h} = [h_1\ h_2\ \dots\ h_N]^\top$ can be calculated by

$$\mathbf{h} = f(\mathbf{W}\mathbf{x}), \tag{1}$$

where f is an activation function. The output of SLFN $\mathbf{o} = [o_1\ o_2\ \dots\ o_C]^\top$ is then calculated as:

$$\mathbf{o} = \mathbf{U}\mathbf{h}. \tag{2}$$

In ELM, the first set of weights $\mathbf{W}$ and the bias vector $\mathbf{b}$ are randomly initialized while the second set of weights $\mathbf{U}$ is analytically calculated to achieve the least squares error on a training set. In other words, it formulates this SLFN into a matrix equation and solves it using the Moore-Penrose inverse [11]. Firstly, given a random weight matrix $\mathbf{W}$, a random bias vector $\mathbf{b}$, and a training input matrix $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_S]$ of S instances, a random representation matrix $\mathbf{H} = [\mathbf{h}_1\ \mathbf{h}_2\ \cdots\ \mathbf{h}_S]$ is calculated as

$$\mathbf{H} = f(\mathbf{W}\mathbf{X}). \tag{3}$$

According to the following linear system:

$$\mathbf{T} = \mathbf{U}\mathbf{H}, \tag{4}$$

where $\mathbf{T} = [\mathbf{t}_1\ \mathbf{t}_2\ \cdots\ \mathbf{t}_S]$ is the target (label) matrix of the input matrix $\mathbf{X}$, ELM then calculates the least squares solution $\hat{\mathbf{U}}$ to the linear system as follows:

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{H}^\dagger, \tag{5}$$

where $\mathbf{H}^\dagger$ is the Moore-Penrose inverse [11] of $\mathbf{H}$. Another version of ELM with a better generalization performance [5] can be written as follows:

$$\hat{\mathbf{U}} = \mathbf{T}\mathbf{H}^\top\left(\frac{\mathbf{I}}{\gamma} + \mathbf{H}\mathbf{H}^\top\right)^{-1}, \tag{6}$$

where $\mathbf{I}$ is the identity matrix of the same size as $\mathbf{H}\mathbf{H}^\top$ and γ is a parameter controlling the effect of L2 regularization.

B. ELM-Autoencoder (ELM-AE)

An autoencoder [16] is an MLP model with targeted output being equal to the input. It can be divided into two main parts namely an encoder and a decoder. The encoder and decoder can be thought of as layers and there can be more than one layers in the encoder and the decoder. The encoder maps the input to another representation called a code and the decoder tries to map the code back into the original input, i.e., reconstruction. An autoencoder is trained to minimize the reconstruction error. Disregarding the decoder, the encoder of a trained autoencoder can be stacked on top of each other to form a deep neural network (stacked autoencoders [16]), and then the whole network is fine-tuned by a BP-based learning algorithm.

In [7], a method called ELM-based autoencoder (ELM-AE) was proposed to train an autoencoder with one hidden layer. Since ELM-AE aims to reconstruct the input matrix $\mathbf{X}$ itself from a random representation $\mathbf{H}$, the target matrix $\mathbf{T}$ in Eq. (5) is replaced by the input matrix $\mathbf{X}$. Therefore, the equation is changed to be

$$\hat{\mathbf{U}} = \mathbf{X}\mathbf{H}^\dagger. \tag{7}$$

In contrast to the ordinary autoencoders, after the decoder $\mathbf{U}$ of ELM-AE is learned, its transpose, i.e., $\mathbf{U}^\top$, is used to transform the input $\mathbf{X}$ to a new representation $\mathbf{X}'$ while the encoder generated randomly is discarded. Stacked autoencoders in this case can be done by training another autoencoder with $\mathbf{X}'$. Stacking ELM-AEs is the key process to construct a deep neural network in ML-ELM [7] and H-ELM [3]. In ML-ELM and H-ELM, the last layer, which takes the output from the stacked ELM-AEs as its input, is created using the traditional ELM (supervised learning).

III. Proposed Method

In this section, the detail of the proposed BELM-AE is given. Then we explain how to construct a stacked BELM-AEs and use it as an unsupervised pretraining method for an MLP.

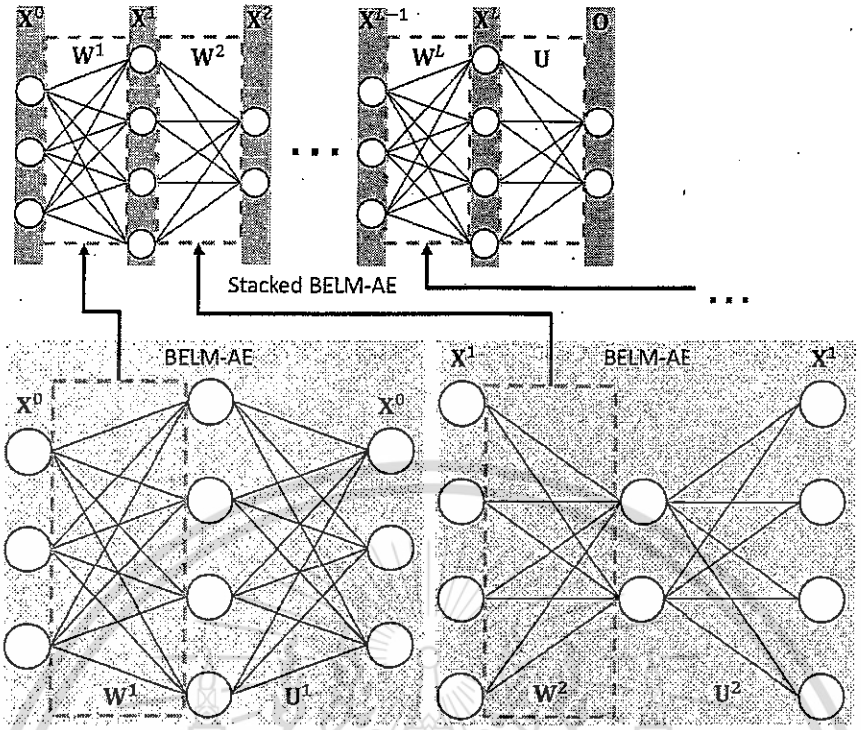


Fig. 2. Stacked BELM-AEs with L hidden layers.

A. BELM-AE

The proposed BELM-AE aims to construct a SLFN-based autoencoder that minimizes the reconstruction error as in ELM-AE; however, the key differences between these two approaches can be summarized as follows: 1) the core concept of ELM is exploited backwards, i.e., BELM-AE randomly generates the output weight matrix $\mathbf{U}$ and analytically solves for the input weight matrix $\mathbf{W}$, and 2) after the autoencoder was trained, the encoder part of BELM-AE is used instead of the transpose of the decoder as in ELM-AE.

Firstly, BELM-AE randomly generates the output weight matrix $\mathbf{U}$, and then calculates the Moore-Penrose inverse $\mathbf{U}^{\dagger}$. Actually, this process can be speeded-up by directly randomizing $\mathbf{U}^{\dagger}$ because the decoder $\mathbf{U}$ of BELM-AE is discarded after construction. The matrix $\mathbf{U}^{\dagger}$ is then used to calculate the least squares solution of $\mathbf{H}$ using

$$\hat{\mathbf{H}} = \mathbf{U}^{\dagger}\mathbf{T}. \tag{8}$$

However, since $\mathbf{H}$ is calculated without any constraints, it is possible that the value of an element in $\mathbf{H}$ is not bounded to the range of the activation function f; for example, (0, 1) for the sigmoid function or (-1,1) for the hyperbolic tangent function. To constrain $\mathbf{H}$ to be bounded, it is scaled into the range of a chosen activation function. Therefore, a constrained hidden matrix $\tilde{\mathbf{H}}$ is then calculated as

$$\tilde{H}_{ij} = \frac{H_{ij} - min(\mathbf{H})}{max(\mathbf{H}) - min(\mathbf{H})}(ub - lb) + lb, \tag{9}$$

where H_{ij} is the element (i, j) of $\mathbf{H}$, ub and lb are the upper-bound and lower-bound of the activation function's range, respectively.

Once the constrained hidden matrix $\tilde{\mathbf{H}}$ is obtained, the next step is to calculate an unactivated hidden matrix $\mathbf{J}$ which is simply done by passing $\tilde{\mathbf{H}}$ into the inverse activation function f^{-1} as

$$\mathbf{J} = f^{-1}(\tilde{\mathbf{H}}), \tag{10}$$

Lastly, the least squares solution of the input weight matrix $\mathbf{W}$, i.e., the encoder, is then calculated using the Moore-Penrose inverse as

$$\hat{\mathbf{W}} = \mathbf{J}\mathbf{X}^{\dagger}, \tag{11}$$

or is calculated using the regularized version of least squares method as

$$\hat{\mathbf{W}} = \mathbf{J}\mathbf{X}^{\top}(\frac{\mathbf{I}}{\gamma} + \mathbf{X}\mathbf{X}^{\top})^{-1}. \tag{12}$$

To summarize, the proposed BELM-AE can be described as follows:

1) Randomly generate the Moore-Penrose inverse $\mathbf{U}^{\dagger}$ of the output weight matrix $\mathbf{U}$
2) Calculate the unconstrained hidden matrix $\mathbf{H}$ from $\mathbf{U}$ using Eq.(8)
3) Calculate the constrained hidden matrix $\tilde{\mathbf{H}}$ using Eq.(9)
4) Calculate the unactivated hidden matrix $\mathbf{J} = \mathbf{W}\mathbf{X}$ using Eq.(10)

5) Solve for the input weight matrix $\mathbf{W}$ using Eq.(11) or Eq.(12)

B. Layer-wise Pretraining using Stacked BELM-AEs

An MLP with L hidden layers can be pretrained in an unsupervised, layer-wise manner using the proposed BELM-AEs. For a layer l in the MLP, the target matrix $\mathbf{T}^l$ is assigned as the input matrix $\mathbf{X}^l$. The training process continues as in the previous section. After training is done, the learned encoder's weight $\mathbf{W}^l$ is used to transform the layer's input matrix $\mathbf{X}^l$ to a new feature space $\mathbf{X}^{l+1}$ which will be used as the input matrix of the next layer $l+1$. The process is depicted in Fig. 2. The weight matrix $\mathbf{U}$ connecting to the output layer is not trained individually but rather trained together with the whole network when fine-tuning with a BP-based algorithm.

Aside from using BELM-AE as oppose to ELM-AE, the main difference between the proposed method and ELM-based approaches such as ML-ELM and H-ELM is the fine-tuning step. In ML-ELM and H-ELM, the authors claim that after doing layer-wise pretraining with ELM-AE, the network does not need to be fine tuned and the representation in the last layer can be used for classification right away. However, because each layer is constructed greedily and in an unsupervised manner (except the last layer), the representation in each layer is not optimized to the problem being solved. Moreover, as autoencoders are optimized to minimize the reconstruction error, not discrimination error (although they are still good representations, and much better than random). Fine-tuning process can adjust these representations into ones that are more suitable for classification.

IV. Experimental Results

This section presents the datasets used in our experiments and describes how the experiments were set-up. The results are then reported and discussed.

A. Dataset

Three image datasets are used to test the methods as follows:

- **MNIST** [10] – A dataset with 70,000 28×28 gray-scale images of hand written digits categorized into 10 classes (60,000 for training and 10,000 for testing).
- **Caltech Silhouettes** [13] – A binary and square version of Caltech101 dataset [2] consisting of 6,407 28×28 silhouette images of 101 classes (4,100 for training and 2,307 for testing).
- **CIFAR-10** [9] – A dataset with 60,000 32×32 color images of objects categorized into 10 classes (50,000 for training and 10,000 for testing).

B. Experiment Set-up

Experiments were carried out to compare performance of neural networks trained with and without layer-wise pretraining. There are three methods used to pretrain the networks which are 1) stacked autoencoders, 2) stacked ELM-AEs, and 3) stacked BELM-AEs.

In the experiments the architecture of the neural networks were fixed within the same dataset. The networks all have three hidden layers (excluding the input and the output layers) and the number of hidden nodes in each layer are shown in Table I (all are fully connected).

For pretraining with stacked ELM-AEs and stacked BELM-AEs, the only parameter is the activation function which was chosen to be the sigmoid function. For stacked autoencoders, each layer was trained individually and then the whole autoencoder network was fine-tuned as well. The number of pretraining epochs for all datasets were set to be the same. The activation function used for autoencoder was also chosen to be the sigmoid function. Note that the activation function used for these autoencoder based network might be changed before fine-tuning.

After pretraining, the networks were then fine-tuned by a BP-based algorithm with dropout [15] (with keeping probability 0.5). The network without pretraining was trained once with this BP-based algorithm from the beginning. We executed the experiments with 3 activation functions, namely, sigmoid, rectified linear unit (ReLU) and hyperbolic tangent functions. Only performance of the best setting is reported. Common training parameters for all experiment settings are as shown in Table II. The cost function of fine-tuning is mean square error with L2 regularization defined as

$$J(\mathbf{W}^1, \ldots, \mathbf{W}^L, \mathbf{U}) = \frac{1}{S}\sum_{i=1}^{S} \|\mathbf{t}^i - \mathbf{o}^i\|_2^2 + \alpha\left(\sum_{j=1}^{L} \|\mathbf{W}^j\|_2^2 + \|\mathbf{U}\|_2^2\right), \tag{13}$$

where $\mathbf{W}^i$ is a weight matrix connecting the layer $i-1$ to the layer i, L is the total number of hidden layers, S is the number of training instances, $\mathbf{t}$ is a target vector, $\mathbf{o}$ is an output vector of the last layer and α is L2 weight.

The hardware setup for the experiments is as follows:

- CPU: AMD Ryzen 1600
- GPU: NVIDIA GeForce GTX 1080ti 11GB
- RAM: DDR4 16 GB

The methods were implemented purely in Python with TensorFlow and NumPy libraries for scientific computation and parallel computing utilities. The implementation was executed mainly using GPU (all ELM based learnings were executed in CPU because of memory limitation) in all experiment configurations. The classification accuracy and training time of the proposed and compared methods were measured. Each configuration was executed 10 times.

C. Results and Discussion

Classification accuracies (mean and standard deviation) of each method were measured and reported in Table III. The number of training epochs of fine-tuning were set to be the same for all methods (except ML-ELM which has no training epoch) which are 50, 250, and 80 for MNIST, Caltech-Silhouettes and CIFAR-10 datasets, respectively. It is clear that the network pretrained with the proposed method achieved the best accuracy on all datasets. ML-ELM performs

TABLE I
ARCHITECTURES OF MLPS USED IN THE EXPERIMENTS (ALL ARE FULLY CONNECTED LAYERS).

Dataset	Number of nodes		
	Hidden layer 1	Hidden layer 2	Hidden layer 3
MNIST	500	500	500
Caltech Silhouettes	500	500	500
CIFAR-10	1,000	1,000	1,000

TABLE II
TRAINING PARAMETERS FOR ALL EXECUTIONS.

Parameter	Value
Batch size	200
Learning rate	0.01
L2 weight (α)	0.05
Keeping probability (dropout)	0.5
Optimization method	ADAM [8]

quite well with Caltech-Silhouettes and CIFAR-10 datasets but very poorly with MNIST. Furthermore, although ML-ELM is extremely fast, its accuracy is still much less than the proposed method in all configurations. Most of the cases, the networks with pretraining achieved better accuracies than the one without. However, on CIFAR-10, only BELM-AE is better. It seems that stacked autoencoders and ELM-AE overfit and produce bad representation which is worst than no pretraining approach in this case.

Table IV presents the results of the same experiment with the number of training epochs increased by a factor of 4. Therefore, in this setting, the networks were fine-tuned by 200, 1,000 and 320 epochs for MNIST, Caltech-Silhouettes and CIFAR-10 datasets, respectively. After the number of training epochs were considerably increased, the proposed method still significantly beats others in terms of accuracy. Considering the two tables, it can be seen that the proposed method fine-tuned with much less epochs still achieves much better accuracy than the case without pretraining with four times more training epochs.

Tables III and IV also show the total training time (pre-training + fine-tuning time). The results from the two tables suggest that it is worth doing pretraining with the proposed method. This is because training with BP-based algorithm alone from the beginning would require considerably more training epochs, hence, much more training time. The graphs in Fig. 3 and 4 confirm our hypothesis about the proposed method requiring less training epochs. From the graphs, testing accuracies were plotted against number of training iterations. It can be seen that testing accuracies of the network pretrained with the proposed method (solid blue line) go up the fastest on both graphs.

Pretraining time for each pretraining method is shown in Table V. Note that for ELM-based pretraining, they were executed using CPU while stacked autoencoders pretraining were executed using GPU. Clearly, the proposed method and stacked ELM-AEs are faster than stacked autoencoders since they do not use a BP-based iterative algorithm. However,

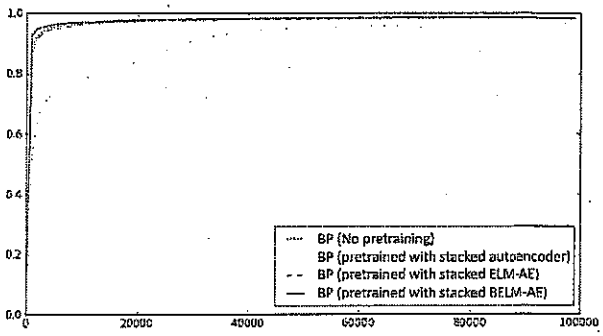


Fig. 3. Testing accuracies on MNIST dataset plotted against number of training iterations (batch size of 200)

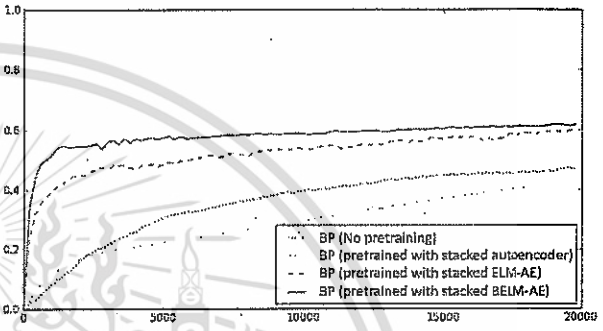


Fig. 4. Testing accuracies on Caltech Silhouettes dataset plotted against number of training iterations (batch size of 200)

stacked ELM-AEs is faster than the proposed method. This is only because of the chosen network architecture which employs compress representation. Since ELM-AE calculates the least squares solution using the hidden representation matrix $\mathbf{H}$ as oppose to BELM-AE which calculates using the input matrix $\mathbf{X}$, ELM-AE will be faster than BELM-AE if the dimension of $\mathbf{h}$ is smaller than that of $\mathbf{x}$. Regardless of this disadvantage from the chosen architecture, the proposed method yields significantly better accuracy than stacked ELM-AEs.

V. CONCLUSION AND FUTURE WORK

Using greedy-layer-wise unsupervised training alone to form a deep network as in ML-ELM is not enough to get good performance, fine-tuning is still needed. Therefore, stacked ELM-AEs approach can be used as a pretraining method for a neural network . Experimental results have shown that, most of the cases, with pretraining, better performance can be achieved with the same number of training epochs. Moreover, the proposed method, pretraining with stacked BELM-AEs, has significantly outperformed other pretraining methods namely, stacked ELM-AEs and stacked autoencoders while requiring comparable training time to ELM-AE which is considerably faster than BP-based autoencoders. It has also been shown that the testing accuracy of an MLP, when pretrained with stacked

TABLE III

PERFORMANCE COMPARISON IN TERMS OF CLASSIFICATION ACCURACY AND TOTAL TRAINING TIME. THE NETWORKS WERE TRAINED WITH 50, 250, AND 80 EPOCHS FOR MNIST, CALTECH-SILHOUETTES AND CIFAR-10 DATASETS RESPECTIVELY WHEN TRAINED WITH BP (NO TRAINING EPOCHS FOR ML-ELM).

Training method	MNIST			Caltech Silhouettes			CIFAR-10		
	Acc (%)	SD	Time (s)	Acc. (%)	SD	Time (s)	Acc. (%)	SD	Time (s)
ML-ELM [7]	89.51	0.78	**24**	53.89	0.85	**2**	40.06	0.36	**68**
BP (no pretraining)	81.63	0.32	52	23.68	1.49	17	33.08	0.62	121
BP (pretrained with stacked autoencoders)	97.11	0.43	196	30.25	4.21	29	20.71	1.50	335
BP (pretrained with stacked ELM-AEs)	96.69	0.21	78	47.79	3.48	20	30.79	3.04	182
BP (pretrained with stacked BELM-AEs)	**97.49**	**0.13**	68	**57.00**	**1.39**	28	**52.90**	**0.42**	327

TABLE IV

PERFORMANCE COMPARISON IN TERMS OF CLASSIFICATION ACCURACY AND TOTAL TRAINING TIME (INCREASED THE NUMBER OF EPOCHS BY A FACTOR OF FOUR). THE NETWORKS WERE TRAINED WITH 200, 1000, AND 320 EPOCHS FOR MNIST, CALTECH-SILHOUETTES AND CIFAR-10 DATASETS RESPECTIVELY WHEN TRAINED WITH BP (NO TRAINING EPOCHS FOR ML-ELM).

Training method	MNIST			Caltech Silhouettes			CIFAR-10		
	Acc. (%)	SD	Time (s)	Acc. (%)	SD	Time (s)	Acc. (%)	SD	Time (s)
ML-ELM [7]	89.51	0.78	24	53.89	0.85	2	40.06	0.36	68
BP (no pretraining)	95.43	0.13	217	42.39	0.91	72	45.36	0.31	560
BP (pretrained with stacked autoencoders)	98.20	0.26	332	48.67	1.71	87	32.78	1.19	792
BP (pretrained with stacked ELM-AEs)	98.31	0.07	235	57.76	1.71	79	38.90	4.82	625
BP (pretrained with stacked BELM-AEs)	**98.41**	0.08	239	**63.00**	1.06	80	**50.75**	1.57	772

TABLE V

PRETRAINING TIME OF EACH METHOD FOR EACH DATASET.

Pretraining method	Pretraining time (s)		
	MNIST	Caltech Silhouettes	CIFAR-10
Stacked autoencoders	141	11	213
Stacked ELM-AEs	25	3	62
Stacked BELM-AEs	37	10	207

BELM-AEs, converges much faster than other pretraining methods which means faster training. As a future work, we aim to explore the use of backward ELM to construct other types of neural networks.

REFERENCES

[1] J. Cao, Z. Lin, and G.B. Huang, "Self-adaptive evolutionary extreme learning machine," Neural Processing Letters, vol.36, no.3, pp.285–305, 2012.

[2] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," Computer Vision and Image Understanding, vol.106, no.1, pp.59–70, 2007.

[3] Y.L. He, Z.Q. Geng, Y. Xu, and Q.X. Zhu, "A hierarchical structure of extreme learning machine (HELM) for high-dimensional datasets with noise," Neurocomputing, vol.128, pp.407–414, 2014.

[4] G.E. Hinton, S. Osindero, and Y.W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol.18, no.7, pp.1527–1554, 2006.

[5] G.B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol.42, no.2, pp.513–529, 2012.

[6] G.B. Huang, Q.Y. Zhu, and C.K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," Proceedings of 2004 International Joint Conference on Neural Networks (IJCNN), vol.2, pp.985–990, 2004.

[7] L.C.C. Kasun, H. Zhou, G.B. Huang, and C.M. Vong, "Representational learning with ELMs for big data," IEEE Intelligent Systems, vol.28, pp.31–34, 2013.

[8] D.P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," Proceedings of 3rd International Conference on Learning Representation (ICLR), 2015.

[9] A. Krizhevsky and G. Hinton, Learning Multiple Layers of Features from Tiny Images, University of Toronto, 2009.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol.86, no.11, pp.2278–2324, 1998.

[11] S.J. Leon, Linear Algebra with Applications (9th Edition), Pearson, 2014.

[12] N.Y. Liang, G.B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," IEEE Transactions on Neural Networks, vol.17, no.6, pp.1411–1423, 2006.

[13] B.M. Marlin, K. Swersky, B. Chen, and N. de Freitas, "Inductive principles for restricted Boltzmann machine learning," Proceedings of 13th International Conference on Artificial Intelligence and Statistics, pp.509–516, 2010.

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors," Nature, vol.323, pp.533–536, 1986.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol.15, pp.1929–1958, 2014.

[16] P. Vincent, H. Larochelle, I. Lajoie, Y.Bengio, and P.A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," Journal of Machine Learning Research, vol.11, pp.3371–3408, 2010.

[17] Y. Yoo and S.Y. Oh, "Fast training of convolutional neural network classifiers through extreme learning machines," Proceedings of 2016 International Joint Conference on Neural Networks (IJCNN), pp.1702–1708, 2016.