Extreme Learning Machines with Structured Matrices

Ruslan Mammadov

Bachelor’s thesis

Extreme Learning Machines with Structured Matrices

Ruslan Mammadov

02. July 2020

Chair of Data Processing
Technische Universität München

Ruslan Mammadov. *Extreme Learning Machines with Structured Matrices.* Bachelor's thesis, Technische Universität München, Munich, Germany, 2020.

Supervised by Prof. Dr.-Ing. Klaus Diepold and Matthias Kissel; submitted on 02. July 2020 to the Department of Electrical and Computer Engineering of the Technische Universität München.

Chair of Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Abstract

The use of structured matrices in Artificial Neural Networks has several advantages: they require significantly less memory, enable faster propagation, and reduce the computational complexity of matrix-vector and matrix-matrix multiplications. The reduction of computational complexity leads to reduced energy consumption, which is especially important for mobile devices.

In this thesis, the use of structured matrices as input weight matrices in Extreme Learning Machines (ELMs) was investigated. The classical ELM is a single-hidden layer feed-forward neural network, in which the input matrix is randomly generated and is not trained, and the output matrix is calculated using linear regression.

An important finding of this thesis is that the input weight matrices can be replaced by structured matrices in ELM without decreasing the accuracy. However, several aspects must be considered for that, such as the distribution of free parameters, choice of a suitable structured matrix, and degree of dissimilarity in the hidden layer's outputs.

Another result of this thesis is that the accuracy of ELMs with structured matrices as input weight matrices does not depend in general on the number of free parameters that were used to construct the structured matrices, but depends on how these parameters were used.

Contents

Abstract **3**

1 Introduction **7**

1.1 Motivation . . . 7

1.2 Goals . . . 7

1.3 Approach . . . 7

2 Background **9**

2.1 Artificial Neural Network . . . 9

2.1.1 Feed-forward Networks . . . 10

2.1.2 Training using Backpropagation . . . 11

2.2 Extreme Learning Machine . . . 12

2.2.1 Training using Linear Regression . . . 13

2.2.2 Advantages and Disadvantages . . . 13

2.3 Structured Matrices . . . 14

2.3.1 Notations . . . 15

2.3.2 Circulant Matrix . . . 15

2.3.3 Toeplitz Matrix . . . 16

2.3.4 Toeplitz-like Matrix . . . 17

2.3.5 Fastfood Transformation Matrix . . . 19

2.3.6 Vandermonde Matrix . . . 20

2.3.7 Overview . . . 20

3 Design of Experiments **21**

3.1 Replacement of Rectangular Input Weight Matrices . . . 21

3.2 Datasets . . . 22

3.3 Preprocessing . . . 23

3.4 Hyperparameters . . . 24

3.4.1 Activation Function . . . 24

3.4.2 Distributions of Weights and Biases . . . 25

3.4.3 Regularizations Values . . . 27

3.4.4 High-Performance Extreme Learning Machines . . . 27

4 Results and Discussion **29**

4.1 Results . . . 29

4.1.1 California Housing Dataset . . . 29

4.1.2 Pima Indian Diabetes Dataset 31
4.1.3 Forest Cover Type Dataset 32
4.1.4 MNIST Dataset 34
4.2 Number of Required Free Parameters for Input Weight Matrices 38
4.3 Discussion 39

5 Conclusion **43**

1 Introduction

1.1 Motivation

The use of structured matrices in Deep Neural Networks has been examined in many previous studies [18][22][4]. In these studies, the use of structured matrices significantly reduced the memory requirements without notably decreasing the prediction accuracy. In the first two studies mentioned, the use of structured matrices resulted in lower propagation time. This may have many benefits, especially for mobile devices, where the memory requirements and propagation speed are extremely important. Furthermore, this may also facilitate the development of Artificial Neural Networks because of the less strict system requirements for network training.

This motivates researchers to examine the possible use of structured matrices in other machine learning algorithms. In this thesis, the use of various structured matrices in Extreme Learning Machine (ELM) was studied.

1.2 Goals

The goal of this thesis was to investigate the impact of structured matrices as input weight matrices on the accuracy, computation time, and memory requirement of ELM. Particularly, the three following questions were answered by this thesis:

- What impact does replacing input weight matrices with structured matrices have on the accuracy performance of ELM in general?
- How does this impact depend on the number of free parameters? Here, the term 'free parameters' means randomly and independently initalized parameters that are required to construct the structured matrices.
- How does this impact depend on the class of structured matrices? Which classes of structured matrices have better or worse impact on the prediction accuracy?

1.3 Approach

In this thesis, six classes of structured matrices were investigated: Circulant, Toeplitz, Toeplitz-like, Fastfood Transformation, Vandermonde, and Low-Rank matrices. Firstly, the classical

ELM and ELMs with the aforementioned structured matrices as input weight matrices were implemented. In the implementation step, the distribution of the structured matrices' entries was addressed, because this characteristic may have an impact on accuracy performance. For example, in the case of Toeplitz-like matrices, the ELM was programmed to rescale the entries of Toeplitz-like matrix after initialization, so that the entries would have the desired variance.

In order to test the implemented ELMs, four datasets were chosen. The classical ELM was tested with different parameters on all datasets to find the appropriate hyperparameters. These hyperparameters were used for all implemented ELMs.

Then, implemented ELMs were tested on these four datasets with different neurones numbers, and the results were reported in the 'Results and Discussion' chapter. After the first results were obtained, further experiments were suggested and conducted to better understand these results.

The impact of structured matrices on ELMs in terms of computational complexity, propagation speed, and memory requirements was examined theoretically in the 'Background' and 'Results and Discussion' chapters. Finally, the results were analysed and discussed.

2 Background

2.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are non-biological ”networks of simple processing elements (called ’neurons’)” [20]. ANNs were inspired by the biological brain and are used for pattern classification, clustering, function approximation, predictions, and many other problems [12]. A typical neuron has N inputs and one output that can be calculated using the following formula [20]:

$$output = f(w \cdot x + b) \tag{2.1}$$

- x is an input vector $(x_1, \cdots, x_N)^T$ in which every component x_i is an individual input.
- w is a weight vector $(w_1, \cdots, w_N)^T$ in which every component w_i is a weight that is associated with the corresponding individual input x_i.
- b is a bias.
- $w \cdot x$ is the vector multiplication of the weight vector and input vector.
- $f(\cdot)$ is an activation function.

Some of the most popular activation functions are ReLU (Rectified Linear Unit) (2.2) and sigmoid (2.3).

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{2.2}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

According to the definition, the ANNs consist of interconnected artificial neurons. Two neurons can be called connected if the output of one neuron is used as an input for the other neuron. In other words, after the output of the first neuron is calculated, it is used as input for the next neuron for calculating its output. In this example, one can also say that the output of the first neuron is connected to the input of the second neuron if the order of neurons is important [12].

Using this definition of connections, one can describe ANN’s structure through a directed graph, in which the artificial neurons are nodes, and the edges are connections between

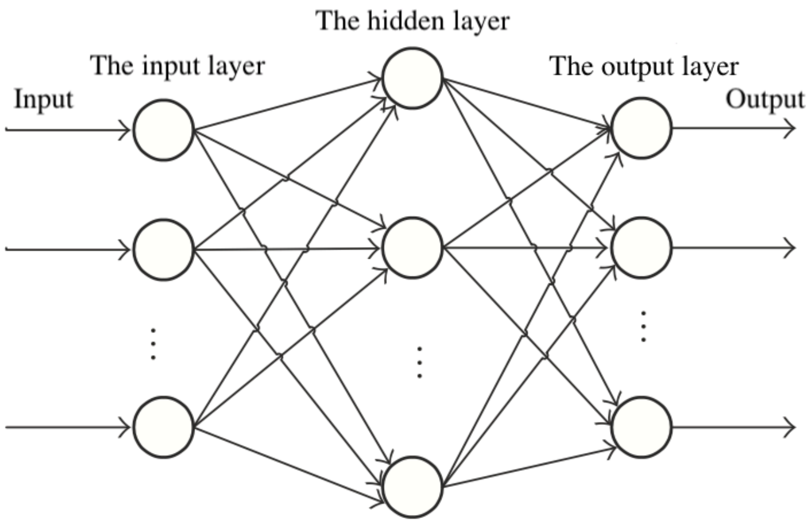


Figure 2.1: Example of a single-hidden layer feed-forward neural network (SLFN). Source: [21].

the neurons, or, more precisely, between inputs and outputs of neurons. In this context, the edges are directed from the outputs to the inputs [12]. An example of such graph is shown in Figure 2.1.

Based on the graph notation and the cycle definition in the graph theory, the ANN can be divided into two main groups: feed-forward networks that do not have any cycles and recurrent networks that have at least one cycle [12] In this thesis, I have used only feed-forward networks.

2.1.1 Feed-forward Networks

The neurons in the feed-forward networks are ordered in different layers [12]. The first and the last layers are input and output layers respectively. The layers between input and output layers are hidden layers. In a hidden layer, each neuron's input is connected with the outputs the of previous layers' neurons, while the output is connected with the inputs of the next layers' neurons. An example of a single-hidden layer feed-forward network (SLFN) architecture with one hidden layer is shown in Figure 2.1.

Although the neurons' outputs can be calculated using equation 2.1 and the graph notation, it is sometimes easier to use vector-matrix multiplication notation. Thus, the outputs of a hidden or output layer in a feed-forward neural network can be calculated using the weight matrix W, in which every row i is the weight vector of the ith neuron, and the bias vector b, in which every component i is the bias of the ith neuron. The final equations are (2.4) for one sample or (2.5) for many samples:

$$output = f(W \cdot x + b) \tag{2.4}$$

$$output = f(\begin{bmatrix} b & W \end{bmatrix} \cdot \begin{bmatrix} 1_{1,N} \\ X \end{bmatrix}) \tag{2.5}$$

- x is an input vector. X is an input matrix in which every column x_i is an input vector of the ith sample.
- N is a number of data samples. $1_{1,N}$ is $1 \times N$ matrix of ones.
- $f(\cdot)$ is an activation function that is applied element-wise to its inputs.
- $output$ is a matrix in which every column i is an output of the ith sample.

The weights W of the first hidden and output layers can be referred to as input and output weights respectively, as it was done in [10][16][2]. Thus, the output of SLFN (Fig. 2.1) can be calculated using equation (2.6) for one sample:

$$output = f_{output}(W_{output} \cdot f_{hidden}(W_{input} \cdot x + b_{hidden}) + b_{output}) \tag{2.6}$$

2.1.2 Training using Backpropagation

Before the neural network can be used to solve problems, it must be trained, i.e., the network's weights and biases have to be adjusted. In supervised training, the datasets with known outputs are used. Since ANN's final goal is to predict the results of unknown data that is not included in the datasets, it is common to divide the datasets into the training and testing sets and to use only the training set to train the network. In this way, the network's accuracy can be checked using the testing set that was never presented to the network before.

One of the most widely used supervised training algorithms is backpropagation. The backpropagation algorithm uses the partial derivative of the error function, also called a loss function, with respect to every weight or bias to adjust them [20]. One of the examples of error functions is mean-square-error (MSE) (2.9).

$$w_{ij}^{k+1} = w_{ij}^{k} - \alpha \frac{\partial E}{\partial w_{ij}}^{k} \tag{2.7}$$

$$b_{i}^{k+1} = b_{i}^{k} - \alpha \frac{\partial E}{\partial b_{i}}^{k} \tag{2.8}$$

$$MSE = \frac{1}{n} \Sigma_{t=1}^{n} \Big(output_{true,t} - output_{network,t} \Big)^2 \tag{2.9}$$

- w_{ij} in the weight of the jth input of the ith neuron. b_i is the bias of the ith neuron.

- k is a step number.
- E is an error function, e.g. MSE (2.9).
- n is the number of samples. t is an index of sample.

2.2 Extreme Learning Machine

Extreme Learning Machine (ELM) was proposed 2004 as a single-hidden layer feed-forward neural network, in which the input matrix and biases are chosen randomly, and the output matrix is calculated analytically, for example, using linear regression [10]. However, after 2004, different versions of ELM with more than one hidden layer [21][5] or with an input matrix calculated using training data [16][2] were proposed. Nevertheless, in all of these cases, the input matrix was not tuned by iterative methods, and the output matrix was calculated using linear regression. Because the linear regression was used to calculate the outputs, no output activation function was used. Furthermore, in all mentioned research studies, no output bias was used. According to [9], the output bias is not required and can even lead to suboptimal solutions.

In this thesis, I have discussed only the ELM with one hidden layer and with a randomly initialized input matrix, as it was proposed in the original paper [10]. To be consistent with other ELM studies [10][17][21], I have used the notation with transposed output matrix $\beta = W_{output}^T$ instead of output matrix W_{output} and targets $T = outputs^T$ instead of outputs in respect to the equation (2.6). Therefore, considering the other details mentioned above, the following final equation for ELM is obtained:

$$T = f_{hidden}(W_{input} \cdot x + b_{hidden})^T \cdot \beta \tag{2.10}$$

- x is a n long input vector that represents one sample.
- W_{input} is a $h \times n$ matrix, and b_{hidden} is a h long vector, where h is a number of hidden neurons.
- β is a $h \times t$ matrix, and b_{output} a t long vector, where t is a number of output neurons.
- T is a $1 \times t$ matrix. If there are $N > 1$ samples, the size of T would be $N \times t$.
- In the case of multiple $N > 1$ samples, the equation must be adjusted so that b_{hidden} and b_{output} have N same columns. In this case, the size of $input$ and T would be $n \times N$ and $N \times t$ accordingly.

2.2.1 Training using Linear Regression

Given the training data X consisting of N samples, where every column is a distinct sample, and targets T, where every row is a true output of the distinct sample, the typical ELM training proceeds as follows[10]:

1. Initialize the input matrix W_{input} and hidden layer biases b_{hidden} randomly.
2. Calculate the output of the hidden layer H for all samples. It can be done, for example, using equation (2.5):
$$H^T = f_{hidden}(\begin{bmatrix} 1_{h,1} & W_{input} \end{bmatrix} \cdot \begin{bmatrix} b^T_{hidden} \\ X \end{bmatrix}) \tag{2.11}$$
3. According to (2.10), the output for all samples can be calculated using $T = H \cdot \beta$. Therefore, if true outputs T are known, the output weights β can be calculated using linear regression, e.g., the ridge regression (2.12).
$$\beta = H^T \left(\alpha I + HH^T\right)^{-1} \cdot T \tag{2.12}$$
Here, α represents the penalty value for output weights. In the case of $\alpha = 0$, the left multiplicand becomes $H^T \left(HH^T\right)^{-1}$, which is an ordinary Moore-Penrose inverse H^+.

2.2.2 Advantages and Disadvantages

ELM has several advantages against standard SLFN in which all neurons are trained using backpropagation [10]:

- Since backpropagation uses many iterations to adjust neurons, while the weights are calculated directly in the case of linear regression, training of ELM takes significantly less time. This is supported by several experiments [10][8][11].
- It may be easier to achieve better generalization using ELM for the same network architecture. One of the reasons is that the norm of the weights calculated using Moores-Penrose is the smallest between all possible least-square solutions, and smaller weights in general lead to better generalization [10][11].
- Backpropagation tends to reach local minima in opposite to ELM [10].
- Backpropagation requires differentiable activation functions in opposite to ELM [10]

Despite the ELMs with many layers were proposed [21][5], one of the disadvantages of ELM against traditional neural networks is that it seems to be difficult to achieve the same variety of layers and the same depth for deep ELM as in deep neural networks. For example, in the study [7], residual networks with up to 152 layers were proposed, which seems to be impossible for ELM. Therefore, it is questionable if ELM can achieve the same accuracy in such domains as image classification, especially on complex databases, e.g., ImageNet [3].

2.3 Structured Matrices

Structured matrix is a matrix with a size $m \times n$ that can be constructed using much fewer parameters than mn[18]. I referred to these few parameters as free parameters. An example of structured matrices is a Circulant matrix (2.13) because the $n \times n$ Circulant matrix can be described with only n independent parameters.

$$C(v) = \begin{bmatrix} v_1 & v_n & \cdots & v_2 \\ v_2 & v_1 & \cdots & v_3 \\ \vdots & \vdots & \ddots & \vdots \\ v_n & v_{n-1} & \cdots & v_1 \end{bmatrix} \tag{2.13}$$

Here, $v \in \mathbb{R}^n$ is a vector consisting of free parameters.

Another well-known example of structured matrices are m-by-n Low-Rank matrices that have rank $r << max(m, n)$. These matrices can be described as follows [4]:

$$M = GH^T \tag{2.14}$$

Here, $G \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{n \times r}$ can be interpreted as matrices consisting of free parameters.

The use of structured matrices can reduce the memory requirements because one can save only the matrix's free parameters instead of every element of the matrix. Another advantage of structured matrices is that for some of these matrices, the matrix-vector multiplication can be performed faster than for general matrices because of their special structure. For example, the multiplication of $n \times n$ Circulant matrix with a vector can be computed in $O(n\, log\, n)$ time [18].

One of the subgroups of the structured matrices is Low Displacement Rank (LDR) matrices that are defined by a displacement operator $L : \mathbb{R}^{m \times n} \longrightarrow \mathbb{R}^{m \times n}$ that transform these matrices into matrices with rank $r << min(m, n)$ [18]:

$$L[M] = GH^T \tag{2.15}$$

Here, M is a structured matrix, G and H are $m \times r$ and $n \times r$ matrices. The rank r is called displacement rank of M under operator L. G and H are low-displacement generators. Thus, one of the advantages of the displacement operator is that if an inverse of displacement operator $L^{-1}[\cdot]$ exists for a specific class, one can create structured matrices M of this class using $M = L^{-1}[GH^T]$[18].

One of the most widely used displacement operators is Sylvester displacement operator (2.16), denoted by $\nabla_{A,B} : \mathbb{R}^{n \times n} \longrightarrow \mathbb{R}^{n \times n}$ [18].

$$\nabla_{A,B}[M] = AM - MB \tag{2.16}$$

- M is a $n \times n$ structured matrix.
- A and B are $n \times n$ fixed matrices that should be chosen depending on the class of structured matrices.

In the next sections, all structured matrices that are used in this thesis are described. However, firstly, the notations that are used frequently in the next subsections are explained.

2.3.1 Notations

1. $diag(x)$ denotes a diagonal matrix where $diag(x)_{ii} = x_{ii}$ and $\{diag(x)_{ij} = 0\}_{i \neq j}$.
2. I_n is a $n \times n$ identity matrix.
3. 0_n is a n long zero vector. $0_{n \times m}$ is a n-by-m zero matrix.
4. F_n denotes the n-by-n Discrete Fourier Transformation (DFT) matrix. The size of the DFT matrix is specified only if it is not obvious.
5. fft(x) denotes Fast Fourier Transformation (FFT) of the vector x. Since FFT is an algorithm that calculates the DFT transformation, which can be computed as multiplication of DFT matrix with x vector, fft(x) is equal to Fx. This equation is used frequently in the next sections.
6. ifft(x) denotes Inverse Fast Fourier Transformation (IFFT). Similarly to FFT, equation ifft$(x) = F^{-1}x$ is true.
7. $x \odot y$ denotes element-wise (Hadamard) multiplication between $x, y \in \mathbb{R}^n$ so that $(x \odot y)_i = x_i y_i$ and $(x \odot y) \in \mathbb{R}^n$.
8. $e_1, \cdots, e_n$ are canonical basis vectors in $\mathbb{R}^n$.

2.3.2 Circulant Matrix

Circulant matrices are square matrices with the structure described in the equation (2.13). A specific feature of circulant matrices is that their eigenvectors can be chosen or rather scaled to be equal to the columns of Discrete Fourier Transformation (DFT) matrix. Therefore, the n-by-n circulant matrix can be decomposed as [6, p. 32]:

$$C(v) = F^{-1}\Lambda F \text{ where } \Lambda = diag(Fv) \tag{2.17}$$

where $v \in \mathbb{R}^n$. Therefore, a vector $x \in \mathbb{R}^n$ can be multiplied with a Circulant Matrix using FFT [18]:

$$C(v)x = \text{ifft}(\text{fft}(v) \odot \text{fft}(x)) \tag{2.18}$$

Since both FFT and Inverse FFT element-wise multiplication require only $O(n\, logn)$ time, the matrix-vector multiplication with Circulant matrix (2.18) can also be performed in $O(n\, log\, n)$ time. n-by-n Circulant matrices can be described by n free parameters.

2.3.3 Toeplitz Matrix

Toeplitz matrices are square matrices that can be described by the following formula:

$$T(v) = \begin{bmatrix} v_n & v_{n-1} & \cdots & v_1 \\ v_{n+1} & v_n & \cdots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{2n-1} & v_{2n-2} & \cdots & v_n \end{bmatrix} \tag{2.19}$$

where $v \in \mathbb{R}^{2n-1}$ is a vector with free parameters.

As one can see in previous equation, there are some similarities between Toeplitz and Circulant matrices. One can exploit these similarities and expand the n-by-n Toeplitz matrix into the $(2n-1)$-by-$(2n-1)$ Circulant matrix[6, p. 44], for example :

$$T_{expanded}(v) = \left[\begin{array}{ccccc|cccc} v_n & v_{n-1} & \cdots & \cdots & v_1 & v_{2n-1} & v_{2n-2} & \cdots & v_{n+1} \\ v_{n+1} & v_n & \cdots & \cdots & v_2 & v_1 & v_{2n-1} & \cdots & v_{n+2} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ v_{2n-2} & v_{2n-3} & \cdots & \cdots & v_{n-1} & v_{n-2} & \cdots & v_1 & v_{2n-1} \\ v_{2n-1} & v_{2n-2} & \cdots & \cdots & v_n & v_{n-1} & \cdots & \cdots & v_1 \\ \hline v_1 & v_{2n-1} & v_{2n-2} & \cdots & \cdots & \cdots & \cdots & \cdots & v_2 \\ v_2 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_1 & v_{2n-1} & v_{2n-2} & \cdots & \cdots & v_n \end{array}\right] \tag{2.20}$$

here $v \in \mathbb{R}^{2n-1}$.

For ease of calculations, a permutation matrix P that satisfies $Pv = [v_n, \cdots, v_{2n-1}, v_1, \cdots, v_{n-1}]^T$ where $v \in \mathbb{R}^{2n-1}$ was defined. This permutation matrix is given as:

$$P = \begin{bmatrix} 0_{n\times(n-1)} & I_n \\ I_{n-1} & 0_{(n-1)\times n} \end{bmatrix} \tag{2.21}$$

As suggested by [6, p. 45], using this permutation matrix and equation (2.17), one can decompose the Toeplitz matrix:

$$\begin{aligned} T(v) &= \begin{bmatrix} I_n & 0_{(n-1)\times n} \end{bmatrix} C(Pv) \begin{bmatrix} I_n \\ 0_{n\times(n-1)} \end{bmatrix} \\ &= \begin{bmatrix} I_n & 0_{(n-1)\times n} \end{bmatrix} F_{2n-1}^{-1} diag(F_{2n-1}Pv) F_{2n-1} \begin{bmatrix} I_n \\ 0_{n\times(n-1)} \end{bmatrix} \end{aligned} \tag{2.22}$$

$$T(v) = \{(F_{2n-1}^{-1} \Lambda F_{2n-1})_{ij}\}_{i,j\in[1,n]} \text{ where } \Lambda = diag(F_{2n-1}Pv) \tag{2.23}$$

Similar to Circulant matrices, the matrix-vector multiplication with the Toeplitz matrix can be performed in $O(n\,log\,n)$ time. n-by-n Circulant matrices can be described by $2n-1$ free parameters.

2.3.4 Toeplitz-like Matrix

Before discussing the Toeplitz-like Matrices, the f-unit-circulant matrices Z_f (2.24) and f-Circulant matrices $Z_f(v)$ (2.25) should be defined [18]:

$$Z_f = [e_2, e_3, \cdots, e_n, fe_1] = \begin{bmatrix} 0 & 0 & \cdots & f \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \tag{2.24}$$

$$Z_f(v) = \begin{bmatrix} v_1 & fv_n & \cdots & fv_3 & fv_2 \\ v_2 & v_1 & \cdots & fv_4 & fv_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_1 & fv_n \\ v_n & v_{n-1} & \cdots & v_2 & v_1 \end{bmatrix} \tag{2.25}$$

Here, both matrices are square $n \times n$ matrices. The special cases of f-Circulant matrices are $Z_{-1}(v)$, which is a skew-Circulant matrix [18], and $Z_1(v)$, which is an ordinary Circulant matrix $C(v)$ [18]. Below, two examples of these matrices are illustrated:

$$Z_{-1}(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T) = \begin{bmatrix} 1 & -3 & -2 \\ 2 & 1 & -3 \\ 3 & 2 & 1 \end{bmatrix} \tag{2.26}$$

$$Z_1(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T) = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix} \tag{2.27}$$

Toeplitz-like Matrices are square LDR matrices that can be described by Sylvester displacement operator $\nabla_{A,B}$ where $A = Z_1$ and $B = Z_{-1}$ [18]:

$$\nabla_{Z_1, Z_{-1}}[M] = GH^T \tag{2.28}$$

- M is a n-by-n Toeplitz-like matrix
- G and H are n-by-r matrices where r is a displacement rank of M.

Toeplitz-like structures with displacement rank r can be constructed from $n \times r$ low-displacement generators $G = [g_1, \cdots, g_r]$ and $H = [h_1, \cdots, h_r]$ that satisfy the equation (2.28) using the following formula [18]:

$$M = \frac{1}{2} \sum_{i=1}^{r} Z_1(g_i) Z_{-1}(J_n h_i) \tag{2.29}$$

where $J_n = [e_n, e_{n-1}, \cdots, e_1]$ is the anti-identity reflection matrix [18].

Therefore, one can generate the Toeplitz-like matrices with displacement rank r from any $G = [g_1, \cdots, g_r]$ $H = [h_1, \cdots, h_r]$ using the generalized form of the equation (2.29) [18]:

$$M = \sum_{i=1}^{r} Z_1(g_i) Z_{-1}(h_i) \tag{2.30}$$

The advantage of this representation of Toeplitz-like Matrices is that one can use the FFT algorithm to perform matrix-vector multiplication faster. The fast matrix-vector multiplication with Circulant matrices was already described in the previous chapter (2.18). The fast multiplication with skew-Circulant matrices was shown in [18]:

$$Z_{-1}(v)x = \overline{\eta} \odot \text{ifft}(\text{fft}(\eta \odot v) \odot \text{fft}(\eta \odot x)) \tag{2.31}$$

$$Z_1(v)x = \text{ifft}(\text{fft}(v) \odot \text{fft}(x)) \tag{2.32}$$

- x and v are n long vectors.
- $\eta = [1, z, z^2, \cdots, z^{n-1}]^T$ where $z = \sqrt[n]{-1} = exp(\frac{i\pi}{n})$.

Thus, after combining the equations (2.30), (2.32), and (2.31), the product between a vector x and the Toeplitz-like Matrix M can be computed using the following formula [18]:

$$Mx = \sum_{i=1}^{r} Z_1(g_i) Z_{-1}(h_i)x = \text{ifft}\left[\sum_{i=1}^{r} \text{fft}(g_i) \odot \text{fft}(\overline{\eta} \odot \text{ifft}(\text{fft}(\eta \odot h_i) \odot \text{fft}(\eta \odot x)))\right] \tag{2.33}$$

A critical property of Toeplitz-Like matrices built using the equation (2.30) is that if the entries of G and H matrices are statistically independent with zero mean and standard deviation equal to σ, the entries of resulting Toeplitz-like matrix will have deviation equal to σ^2 and zero mean. To prove it, one can rewrite the equation (2.30):

$$M_{ij} = \sum_{m=1}^{r} \sum_{k=1}^{n} [Z_1(g_m)]_{ik} \cdot [Z_{-1}(h_m)]_{kj} \tag{2.34}$$

$$var(M_{RV}) = rn\, var(G_{RV} \cdot H_{RV}) \tag{2.35}$$

Here, M_{RV}, G_{RV}, and H_{RV} are random variables from the distributions which M, G and H matrices entries belong to. I assumed here that the distribution of G_{RV} and H_{RV} are symmetrical with zero mean so that the changing of sign does not affect the distribution. Another conclusion from the equation (2.34) is that for $rn \longrightarrow \infty$ the weights' distribution will be Gaussian.

Similarly to Circulant matrices, the matrix-vector multiplication with Toeplitz-like matrices can be performed in $O(rn\, log\, n)$ time. However, the number of FFT calculations required for Toeplitz-like matrices is higher as by Circulant matrices. n-by-n Toeplitz-like matrices can be described by $2rn$ free parameters.

2.3.5 Fastfood Transformation Matrix

Fastfood Transformation was proposed in 2013 by [14] as a tool for the approximation of kernel expansions. Fastfood Transformation was based on the Random Kitchen Sinks algorithm, in which the input is multiplied with a Gaussian random matrix, and a non-linear function is applied to the outputs. Fastfood Transformation works similarly but requires only $O(n\,log\,d)$ time in opposite to $O(nd)$ in Random Kitchen Sinks. Here, n and d are input and output dimensions respectively [14].

The main step of Fastfood Transformation is the multiplication of inputs with the Fastfood Transformation matrix, which I have discussed in this subsection. Fastfood Transformation matrices are square 2^n-by-2^n matrices that can be described by the following equation [14]:

$$V = SHG\Pi HB \tag{2.36}$$

- $d = 2^n$ is input and output dimensions of the single matrix.
- Binary scaling matrix B is a diagonal matrix with $B_{ii} \in \{-1, 1\}$. One of the impacts of B matrix is that the components of rows in $HG\Pi HB$ have zero correlation [14].
- H is the Wash-Hadamard matrix. Wash-Hadamard matrices always have the 2^l-by-2^l shape, where $l \in \mathbb{N}$. Therefore, the whole Fastfood Transformation matrix can only be a square 2^l-by-2^l matrix.
- $\Pi \in 0, 1^{d\times d}$ is a permutation matrix.
- Gaussian scaling matrix G is a diagonal matrix which entries are generated by standard normal distribution $\mathcal{N}(0,\ 1)$
- As a result, $HG\Pi HB$ entries are from Gaussian Distribution $\mathcal{N}(0,\ d)$ [14].
- Scaling matrix S is a diagonal matrix with random entries that are used to control the length distribution of rows in V. Without S, every row in $HG\Pi HB$ has the same length $l^2 := [(HG\Pi HB)(HG\Pi HB)^T]_{jj} = \sum_i G_{ii}^2 d$ [14].

Rescaling of the lengths is necessary for kernel approximation where different distributions of S correspond with different kernels. However, if the Fastfood Transformation matrix is used in ELM as an input matrix, there is no particular need for having rows with different lengths. Therefore, in this thesis, I have not used a Scaling matrix S. In this case, if it is desired that the standard deviation of the entries is $\sigma \in \mathbb{R}$, one can scale $HG\Pi HB$ with $\frac{\sigma}{\sqrt{d}}$.

Matrix-vector multiplication with the Walsh-Hadamard matrix H can be calculated using Fast Hadamard Transform in $O(n\,log\,n)$ time. n-by-n Binary scaling matrix B, permutation matrix Π, and Gaussian scaling matrix G each have only n elements. Therefore, vector-matrix multiplication with these matrices can be performed in $O(n)$ time. Thus, the Fastfood Transformation requires only $O(n\,log\,n)$ time. Additionally, the Fastfood Transformation matrix can be built using $4n$ free parameters, where n parameters are used for permutation matrix, and other n parameters are integer numbers from the set $\{-1, 1\}$.

2.3.6 Vandermonde Matrix

Vandermonde matrix has the following structure:

$$V(v) = \begin{bmatrix} 1 & v_1 & v_1^2 & \cdots & v_1^{n-1} \\ 1 & v_2 & v_2^2 & \cdots & v_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_n & v_n^2 & \cdots & v_n^{n-1} \end{bmatrix} \tag{2.37}$$

where $v \in \mathbb{R}^n$ is a vector with free parameters.

Matrix-vector multiplication with the Vandermonde matrix can be performed in $O(n\, log^2\, n)$ time [18]. n-by-n Vandermonde contains n free parameters.

2.3.7 Overview

In the following table, the information of this section is summarized. A non-structured matrix was added as a reference. The following notation was used:

1. P_{RV} is a random variable of free parameters, G_{RV} and H_{RV} are random variables of low-displacement generators' entries from the equation (2.30). I made an assumption here that G_{RV} and H_{RV} distributions are symmetrical with zero mean. $D(X)$ denotes the distribution of some random variable X.

2. r_d is a displacement rank. n is an input dimension. In the case of the Fastfood Transformation matrix, $n = 2^l$ where $l \in \mathbb{N}$. m is an output dimension of non-structured and Low-Rank matrices.

3. 'NFR' is a number of free parameters. 'Mul. Time' is matrix-vector multiplication time. 'Ref.' denotes references, 'Eq.' means equation. Entries distribution was given with respect to the equation in 'Eq.' column.

Name	Size	NFR	Mul. Time	Eq.	Entries Distribution	Ref.
Circulant	n-by-n	n	$O(n\, log\, n)$	(2.13)	$D(P_{RV})$	[6]
Toeplitz	n-by-n	$2n-1$	$O(n\, log\, n)$	(2.19)	$D(P_{RV})$	[6]
Toeplitz-like	n-by-n	$2r_d n$	$O(n\, log\, n)$	(2.30)	$D(\sum_{i=0}^{r_d n} G_{RV\,i} H_{RV\,i})$	[18]
Fastfood	2^l-by-2^l	$4n$	$O(n\, log\, n)$	(2.36)	Gaussian	[14]
Vandermonde	n-by-n	n	$O(n\, log^2\, n)$	(2.37)	-	-
Low-Rank	n-by-m	$nr + rm$	$O(nr + rm)$	(2.14)	$D(\sum_{i=0}^{r} P_{RV\,i} P_{RV\,i+r})$	[4]
Non-structured Matrix	n-by-m	nm	$O(nm)$	-	$D(P_{RV})$	-

3 Design of Experiments

The main goal of the experiments was to investigate the performance of the Extreme Learning Machine (ELM) with structured matrices as input weight matrices in terms of accuracy. In this thesis, only the standard versions of ELM proposed by [10] with one hidden layer and randomly initialized input weight matrix was tested. In this context, a randomly initialized matrix means that all entries were randomly initialized in the case of a non-structured matrix, and free parameters were randomly initialized in the case of a structured matrix. The general procedure of my experiments was as follows:

Firstly, ELMs with different weight input matrices were implemented: classical ELM with non-structured matrices, Circulant ELM with Circulant matrices, Toeplitz ELM with Toeplitz matrices, Toeplitz-like ELM with Toeplitz-like matrices, Fastfood ELM with Fastfood Transformation matrices, Vandermonde ELM with Vandermonde matrices, and Low-Rank ELM with low-rank matrices. For implementation, *Tensorflow* and *Keras* were used to calculate the hidden layer's outputs. For linear regression, the least-squares algorithm provided by *NumPy* was used for function approximation, and *sklearn*'s implementation of Ridge Regression Classifier was used for classification problems.

Since the experiments aimed to investigate only the prediction performance, fast matrix-vector multiplication algorithms of structured matrices were not used. Instead, the input matrices were just built from free parameters and multiplied with input vectors as a standard matrix.

Then, the classical ELM was tested with different parameters to choose the appropriate hyperparameters. Finally, implemented ELMs were tested with different numbers of neurons on four datasets. These steps are described more extensively in the next sections.

3.1 Replacement of Rectangular Input Weight Matrices

As shown in Table 2.3.7, almost all structured matrices are square matrices. Relatively, the replacement of rectangular input weight matrix with structured matrix is not trivial. In this section, the replacement process is explained.

- It is assumed that d is number of inputs, and n is number of hidden layer neurons. $\lceil \cdot \rceil$ denotes the ceil function.

- For Fastfood Transformation matrix (which can only have dimensions that are numbers of the form 2^l where l is an integer), if d is not in the form 2^l, zeros were attached to the input vectors till the dimension of the input vector became 2^l. The resulting input is referred to as $d_{expanded} = 2^{\lceil log_2(d) \rceil}$. After that, $\lceil n/d_{expanded} \rceil$ Fastfood matrices with the shape $d_{expanded} \times d_{expanded}$ were stacked. Then, if the output dimension $\lceil n/d_{expanded} \rceil \cdot d_{expanded}$ was greater than the desired output dimension n, the products between inputs and input weight matrix were pruned.

- For Toeplitz-like, Toeplitz, Circulant, and Vandermonde matrices, the implementation can be interpreted as follows: firstly, a square structured matrix with the dimension $dim_{max} = max(n, d)$ was created. Then, this matrix was pruned to achieve the desired size $n \times d$. The parameters that do not have any impact on the the pruned final matrix and are unnecessary to describe this matrix are not counted as free parameters in this thesis. For example, the pruned Toeplitz matrix

$$T_{pruned} = \begin{bmatrix} 3 & 2 \\ 4 & 3 \\ 5 & 4 \end{bmatrix} \tag{3.1}$$

  requires only four free parameters.

The resulting numbers of free parameters that are required to build the structured input weight matrices in the way explained in this section are the following:

- $4 \cdot \lceil n/d_{expanded} \rceil \cdot d_{expanded}$ for Fastfood matrix.
- $2r \cdot max(n, d)$ for Toeplitz-like matrices with low displacement rank r.
- $n + d - 1$ for Toeplitz matrix.
- $max(n, d)$ for Circulant matrix.
- n for Vandermonde matrix.

3.2 Datasets

I used four datasets in my experiments: California Housing, Pima Indians Diabetes, Forest Cover Type, and MNIST. California Housing dataset was obtained from *StabLib* repository[1] using the tools provided by *sklearn* framework. This dataset originates from [13]. According to the documentation of *sklearn*, this dataset describes the averaged housing prices in different districts of California. It contains 20,650 samples with eight numeric features and one target, which is the averaged house price.

[1] http://lib.stat.cmu.edu/datasets/

Pima Indians Diabetes Database (later Diabetes dataset) was downloaded from the *Kaggle* website[2]. According to the website documentation, the dataset originates from National Institute of Diabetes and Digestive and Kidney Diseases, and [19] were among the first researchers who used this dataset in Machine Learning and published their results. The dataset contains 768 instances that consist of eight numeric features and one target, which is either a positive or a negative outcome of diabetes diagnosis. 500 of 768 samples ($\approx 65,1\%$) belong to the class with negative outcome.

Forest Cover Type Dataset (later Forest dataset) was also downloaded from the *Kaggle* website[3]. However, the dataset's original source is Machine Learning Repository[4] of Center for Machine Learning and Intelligent System in the University of California. According to the website, the dataset contains observations of trees within a 30×30 m cell in Roosevelt National Forest in Colorado. The dataset includes 518,012 samples with 54 numerous features and one target, the cover type, an integer number ranging from 1 to 10. In the experiments, I tried to predict wether the sample belongs to the second cover type, which 283,031 samples ($\approx 54\%$) belong to.

MNIST[5] is a frequently used dataset in Machine Learning that originates from [15]. The database contains 70,000 black-white pictures of handwritten digits with 28×28 pixels. Consequently, each instance consists of 784 features in the range between 0 and 255 and one output target, which is the resulting digit. The MNIST dataset is divided into the 60,000 training and 10,000 test samples. In my experiments, to simplify the classification problem, I tried to predict only two classes: first class is number nine, the second is other numbers. Approximately 9,95% of all samples are pictures of nine.

3.3 Preprocessing

Firstly, the data had to be divided into training and test sets. As was mentioned in the previous section, MNIST database was already divided. In other datasets, the test-training used by [10] were used:

- California Housing: 8,000 training and 12,640 test samples.
- Diabetes dataset: 576 training and 193 test samples.
- Forest dataset: 100,000 training and 481,012 test samples.

The samples were shuffled before dividing.

[2]https://www.kaggle.com/uciml/pima-indians-diabetes-database
[3]https://www.kaggle.com/uciml/forest-cover-type-dataset
[4]https://archive.ics.uci.edu/ml/datasets/Covertype
[5]http://yann.lecun.com/exdb/mnist/

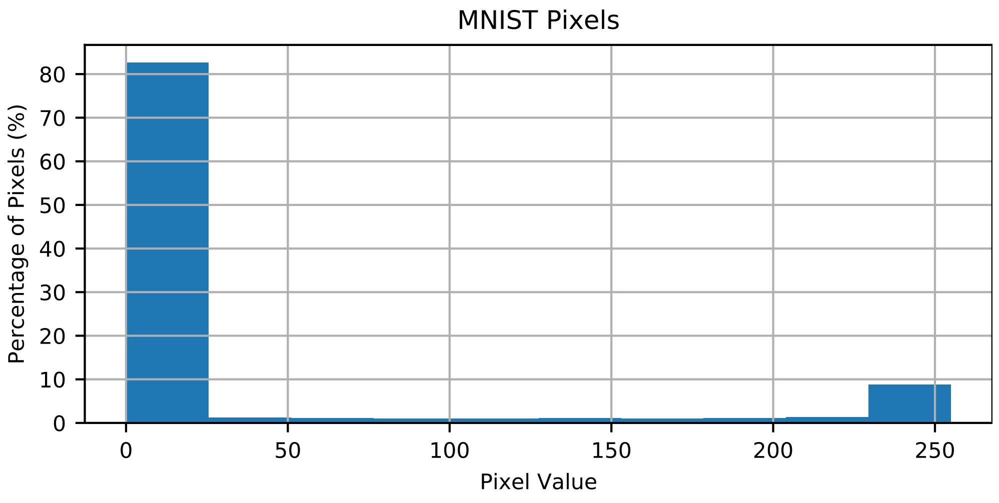


Figure 3.1: Histogram with pixel values.

Before the experiments were conducted, input data was normalized to zero mean and unit variance for all datasets except MNIST. In MNIST, the input was normalized to the range between 0 and 1. The reason for that is the unusual distribution of input features: as illustrated in figure 3.1, the distribution is very skewed to the left. Additionally, $\approx 80,86\%$ of all pixels have the exact value of zero, and $\approx 7,3\%$ of pixels have values between 250-255. Therefore, an ordinary scaling with $\frac{1}{255}$ seems to be a decent idea.

In classifications problems, i.e., in all datasets except California Housing, -1 and 1 were assigned to different output labels. In California Housing, the output was normalized to zero mean and unit variance.

3.4 Hyperparameters

Three hyperparameters were to be chosen: activation function, distributions of weights and biases, and regularizations values of linear regression.

3.4.1 Activation Function

Sigmoid activation function was used in all experiments, because it is one of the most common activation functions in ELM [1]. Moreover, this function was used in the experiments of [10], which were taken as reference for design of my experiments on the California, Diabetes and Forest datasets. Furthermore, HP-ELM - ELM implementation created by [1], which was used in my experiments as reference ELM - was optimized for sigmoid activation function [1]. The sigmoid function was plotted in figure (3.2) and the equation given in the caption.

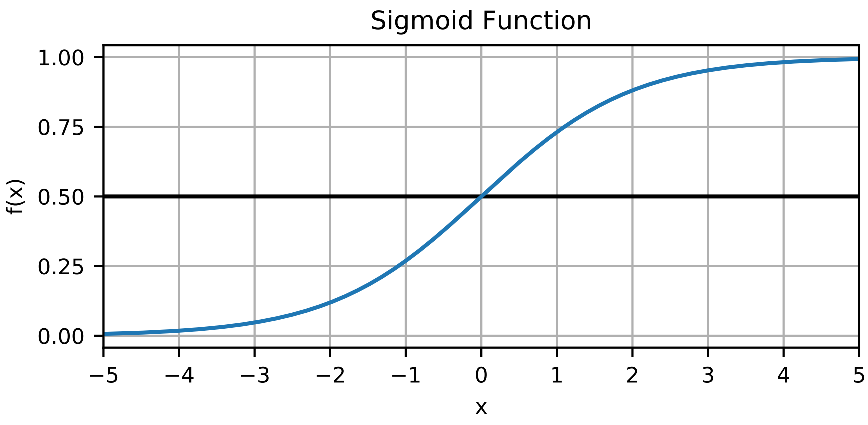


Figure 3.2: Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$.

3.4.2 Distributions of Weights and Biases

The weights of non-structured matrices and the free parameters of structured matrices were drawn from $\mathcal{N}(0, \frac{1}{\sqrt{d}})$ where d is a number of inputs. Biases were drawn from $\mathcal{N}(0, 1)$. MNIST database was an exception: here, weights and free parameters were drawn from $\mathcal{N}(0, \frac{6}{\sqrt{d}})$, which is rationalized among other decisions in this subsection. Additionally, after the weights were initalized, they were rescaled with $\frac{\sigma_{desired}}{\sigma_{calculated}}$ where σ denotes the standard deviations, so that the final deviation is exactly equal to the desired deviation. The desired variation is the variation used for initialization. In the following, it is explained why these distributions were chosen and why the weights were rescaled after initialization.

Since the input weights and hidden layers' biases are not adjusted in ELM, the choice of the right distribution can be crucial for the results. As explained by [1], if the variance of the weights is too high, then the activation function's inputs will also have a high variance; therefore, the output of the sigmoid function would be mostly 0 or 1. From another point of view, if the weights' variance is too low, the input of the activation functions will change within an extremely small range. Thus, the sigmoid function would work similarly to the linear function.

Therefore, it is important to ensure that the input of the sigmoid activation function is mostly in the appropriate interval. This input can be calculated using the formula $input = \sum_{i=0}^{d} x_i w_i + b$ where d is a number of inputs, w and x are weight and input vectors respectively, and b is a bias. To avoid misunderstandings, I referred to the variance of the $input$ in the previous equation as 'pre-nonlinearity activation variance'. If one assumes that the components $x_i w_i$ of this summation term are statistically independent of each other, x was normalized to zero mean and unit variance, and w has zero mean, one can calculate the

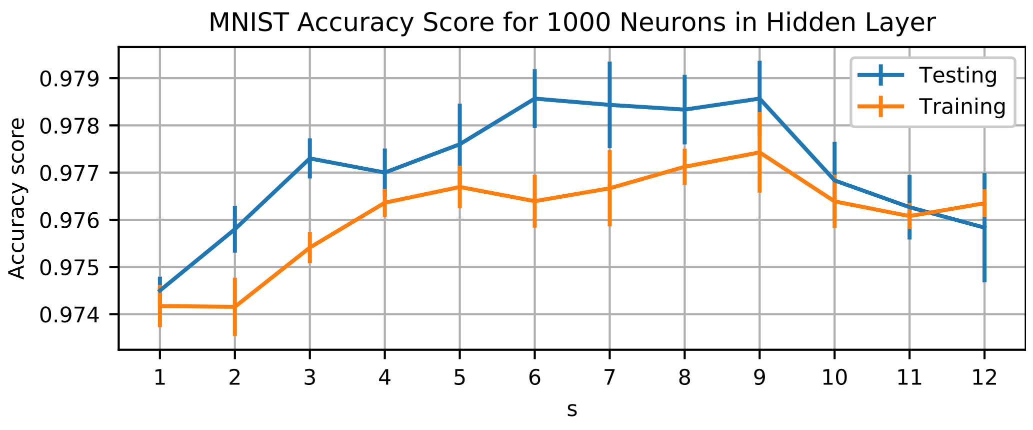


Figure 3.3: Comparison of different s on the MNIST dataset. The weights were drawn from $\mathcal{N}(0, \frac{s}{\sqrt{d}})$ where d is an input dimension. The averaged accuracy score and standard deviation after 50 trials are shown.

pre-nonlinearity activation variance as [1]:

$$var(\sum_{i=0}^{d} w_i x_i + b) = \sum_{i=0}^{d} var(w_i x_i + b) = \sum_{i=0}^{d} w_i^2 = d\, var(w) \tag{3.2}$$

Hence, [1] recommends to use the following parameters for initialization of weights and biases: zero mean for both weights and biases, weights' variation equal to $\frac{s}{\sqrt{(d)}}$ where s is near 1, and biases' variation equal to 1. That was also shown in the experiments done by [1]. However, in MNIST, the input data was not centered and normalized to unit variance. Because of this, different s parameters were tested, and based on the result $s = 6$ was chosen (figure 3.3).

Another question could be why Gaussian distribution was chosen, and not uniform distribution. There are three main reasons for that: firstly, normal distribution seems to be common for ELMs and was used by [1], whose recommendations I followed. Secondly, normal distribution showed slightly better results in my experiments, see figure 3.4. Finally, for Fastfood Transformation and Toeplitz-like matrices, only normally distributed weights were possible; thus, in order to make the comparison fair, all matrices were initialized using normal distribution.

However, while for almost all structured matrices, initialization of free parameters using normal distribution leads to weights being normally distributed, it is not the case for Toeplitz-like matrices. According to the equation (2.34), in this case, the weights of the Toeplitz-like matrices belong to the distribution of the sum of products of two centered independent normal random variables. These distributions were plotted in figure 3.5. However, the distribution already becomes very close to the Gaussian distribution in the case of only nine inputs.

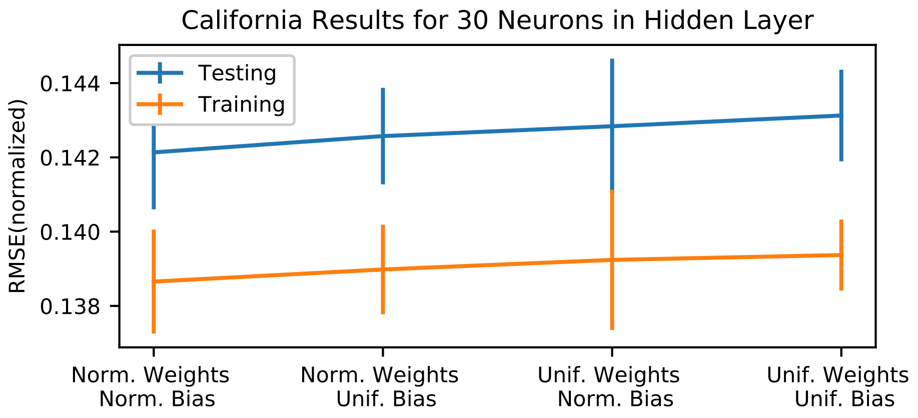


Figure 3.4: Comparison between initialization using normal ('Norm.') and uniform ('Unif.') distributions on the California Dataset. The averaged normalized RMSE and standard deviation after 50 trials are shown. In all four cases, the same variance and mean were used.

Therefore, free parameters of Toeplitz-like matrices were also initialized using the Gaussian distribution.

The last question is why the weights were rescaled after the initialization. The primary cause is that the weights' variance of Toeplitz-like matrices increases quadratically with a linear increase of free parameters' variation because of the equation (2.35). Therefore, the weights were rescaled after initialization, which led to better results than in other ELMs. Hence, to make the comparison fair, weights were rescaled in all ELMs.

3.4.3 Regularizations Values

To find the proper regularization values, I ran several tests with different regularization values on classical ELM. In result, the following values were chosen:

- California Housing dataset: $rcond = 10^{-6}$ for least square method provided by *NumPy*. According to the *NumPy* documentation, $rcond$ is "cutoff for small singular values." That means that the singular values that are smaller than the maximum value times 10^{-6} were removed after singular value decomposition.
- Diabetes and Forest datasets: Ridge parameter was set to 10^{-9} (α in equation (2.12)).
- MNIST: Ridge parameter was set to 10^{-11}.

3.4.4 High-Performance Extreme Learning Machines

In order to have some reference, another implementation of ELM was included in the experiments. The name of this implementation is 'High-Performance Extreme Learning Machines'

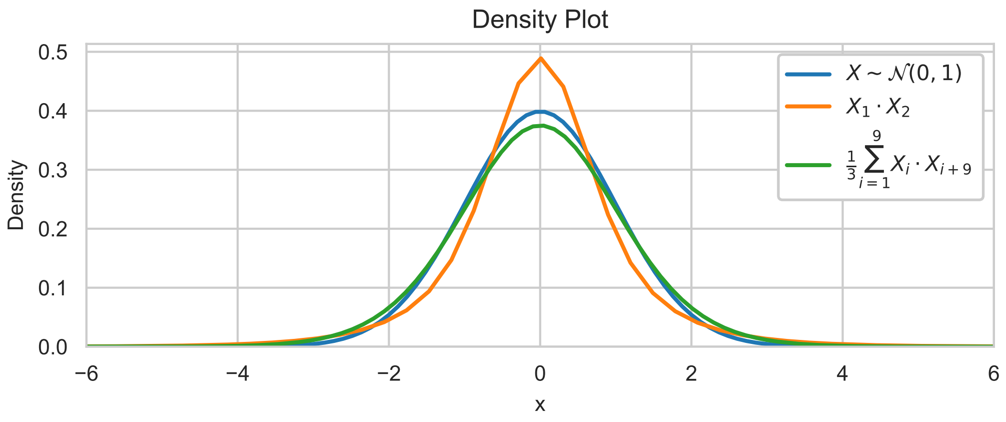


Figure 3.5: Density plots of standard normal variable, product of two standard normal independent random variables, and sum of 9 such products divided by 3. Indexes were used to emphasize that variables are independent.

(HP-ELM). HP-ELM was developed by [1]. I used the same hyperparameters for HP-ELM as for other ELMs except for regularization values and weights and biases distributions, because both aspects are regulated internally by HP-ELM and are not supposed to be changed by the user, i.e., there is no interface for these hyperparameters. The regularization of HP-ELM is comprehensively described in [1], and the weights and biases distributions are $\mathcal{N}(0, \frac{2}{\sqrt{d}})$ and $\mathcal{N}(0, 1)$ respectively.

4 Results and Discussion

In this section, firstly, the accuracy performance of implemented ELMs on every dataset is presented. Then, the number of required free parameters for different input weight matrices are shown, and finally, the results are discussed.

4.1 Results

In this section, results on four datasets are presented: California Housing dataset, Pima Indian Diabetes dataset, Forest Cover Type dataset, and MNIST dataset.

4.1.1 California Housing Dataset

The results of the experiments on the California Housing dataset are shown in Figure 4.1. The Low-Rank and Vandermonde ELMs had relatively high normalized RMSE (later just RMSE) values and therefore are not included in this Figure. Instead, the results of these ELMs are illustrated in Table 4.1. The ranks of two Low-Rank ELMs were chosen to be 7 and 4. These numbers are equal to $\frac{7}{8}$ and $\frac{1}{2}$ of the inputs number respectively.

As one can see in Figure 4.1, all implemented ELMs, except for Low Rank and Vandermonde ELMs, have similar accuracy values. Only the RMSE values of Low-Rank and Vandermonde ELMs are higher and have greater standard deviation, i.e., these ELMs are less stable. Another interesting observation is that Low-Rank matrices have an atypically high training RMSE deviation, and the Vandermonde ELM has a very low generalization property, i.e., the difference between training and testing results is significant.

Name	RMSE Mean		RMSE Dev.	
	Training	Testing	Training	Testing
Classical	0.1296	0.1340	0.0010	0.0015
Circulant	0.1297	0.1341	0.0010	0.0014
Low-Rank with $Rank = 7$	0.1369	0.1435	0.0069	0.0070
Low-Rank with $Rank = 4$	0.1653	0.1748	0.0160	0.0192
Vandermonde	0.1450	0.2207	0.0016	0.0933

Table 4.1: Results on the California Housing dataset for 75 neurons in hidden layer. The averaged normalized RMSE and standard deviation (RMSE Dev.) after 50 trials is shown.

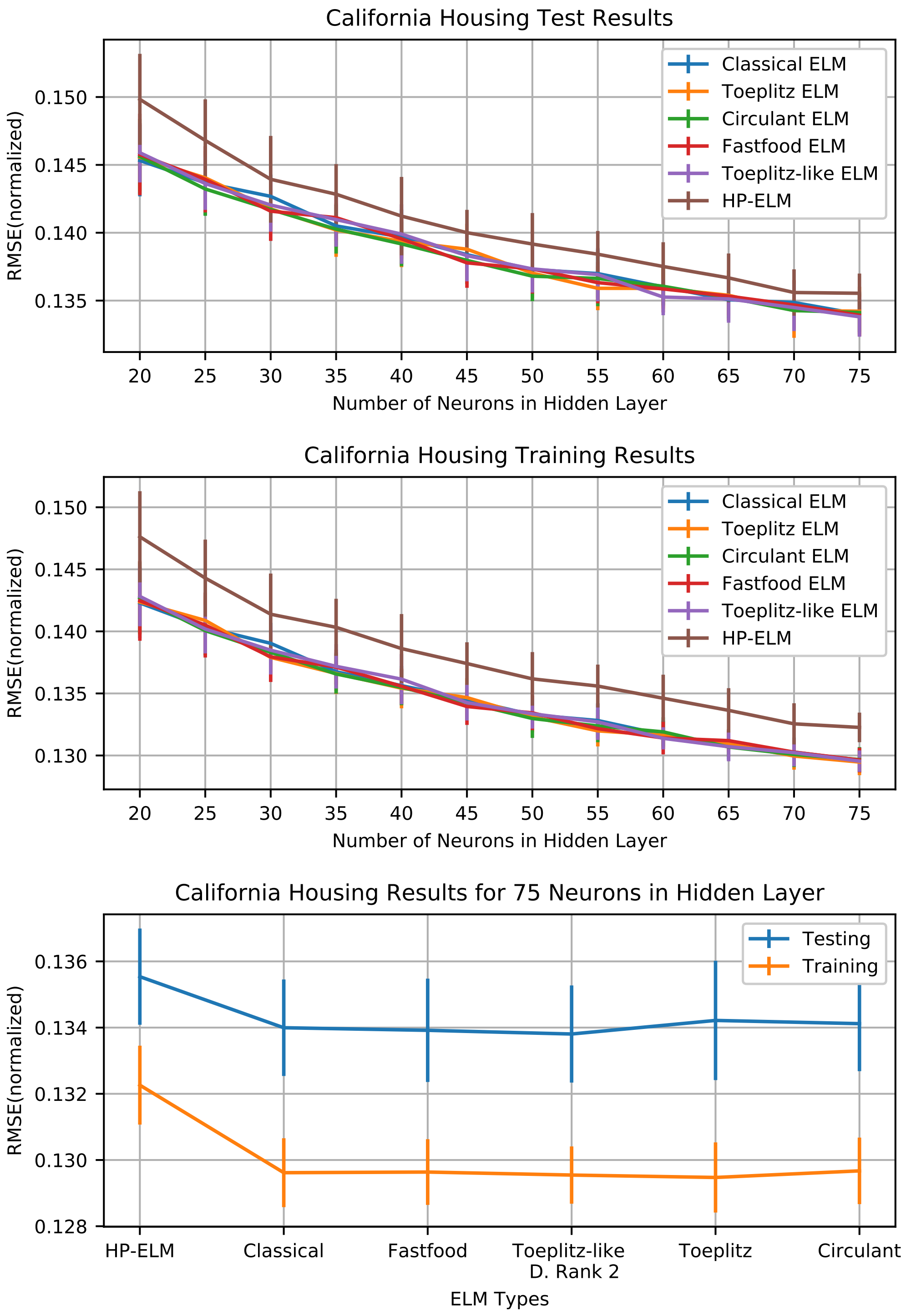


Figure 4.1: Test and training scores of different ELMs on the California Housing dataset. The averaged normalized RMSE and standard deviation of 50 trials is shown. The Toeplitz-like matrix has a displacement rank ('D. Rank') equal 2.

4.1.2 Pima Indian Diabetes Dataset

The results of the experiments on the Pima Indian Diabetes dataset are shown in Figures 4.2 and 4.3. Because the Vandermonde and Low-Rank matrices had relatively low prediction accuracies, they are not included in the first two subfigures. Instead, their results are illustrated in the last subfigure. The ranks of two Low-Rank ELMs were chosen to be 7 and 4. As in the previous section, these numbers are equal to $\frac{7}{8}$ and $\frac{1}{2}$ of the inputs number respectively.

The results in Figure 4.2 and in the first subfigure of 4.3, especially the training results, can be divided into two areas or cases: 4-8 neurons and 10-28 neurons. In the second area, all included ELMs show similar accuracies. In the first area, one can divide included ELMs into three groups: Classical and Fastfood ELMs with the best accuracies, the Circulant ELM with the worst accuracy, and other ELMs, which lay between two other groups and are very close to the first group. However, one should take into account that the number of neurons in the first area is lower than the number of inputs (which is 8). Therefore, it can be considered as an extreme case.

In the second subfigure of 4.3, one can see again that Vandermonde and Low-Rank ELMs show very low accuracy values. Moreover, Low-Rank ELMs also have high standard deviation, which means that they are unstable.

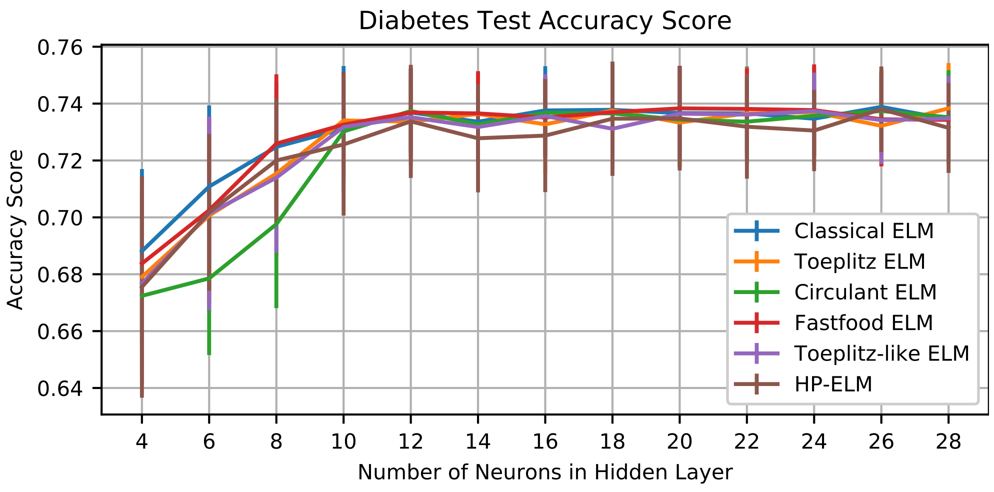
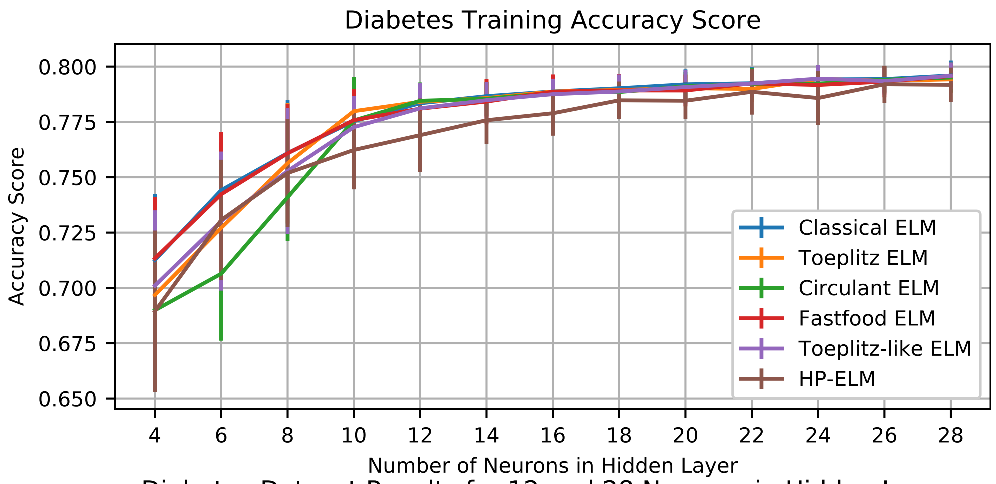


Figure 4.2: Test and training accuracy scores of different ELMs on the Diabetes dataset. The mean and standard deviation of 50 trials is shown. The Toeplitz-like matrix has a displacement rank equal 2.



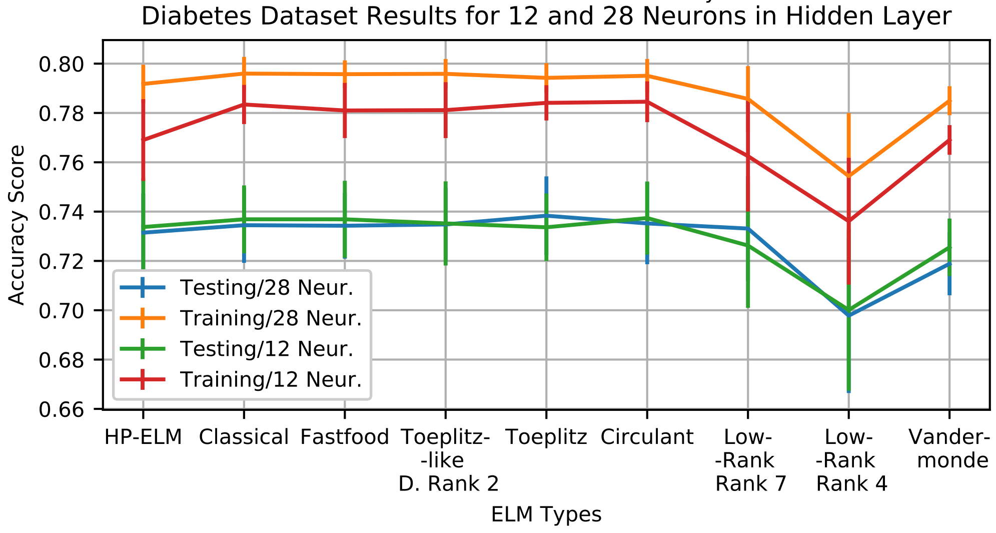


Figure 4.3: Test and training accuracy scores of different ELMs on the Diabetes dataset. The mean and standard deviation of 50 trials is shown. The Toeplitz-like matrix has a displacement rank ('D. Rank') equal 2.

4.1.3 Forest Cover Type Dataset

The results of the experiments on the Forest Cover Type dataset are shown in Figures 4.4 and 4.5. As in the previous sections, Vandermonde and Low-Rank ELMs are not included in Figure 4.4. The results of Low-Rank ELMs are illustrated in Figure 4.5. The ranks of Low-Rank ELMs were chosen to be 45 and 27. These numbers are equal to $\frac{5}{6}$ and $\frac{1}{2}$ of the inputs number (54) respectively. The Vandermonde ELM's results are shown in Table 4.2.

All ELMs showed similar accuracies except for Low-Rank and Vandermonde ELMs. The

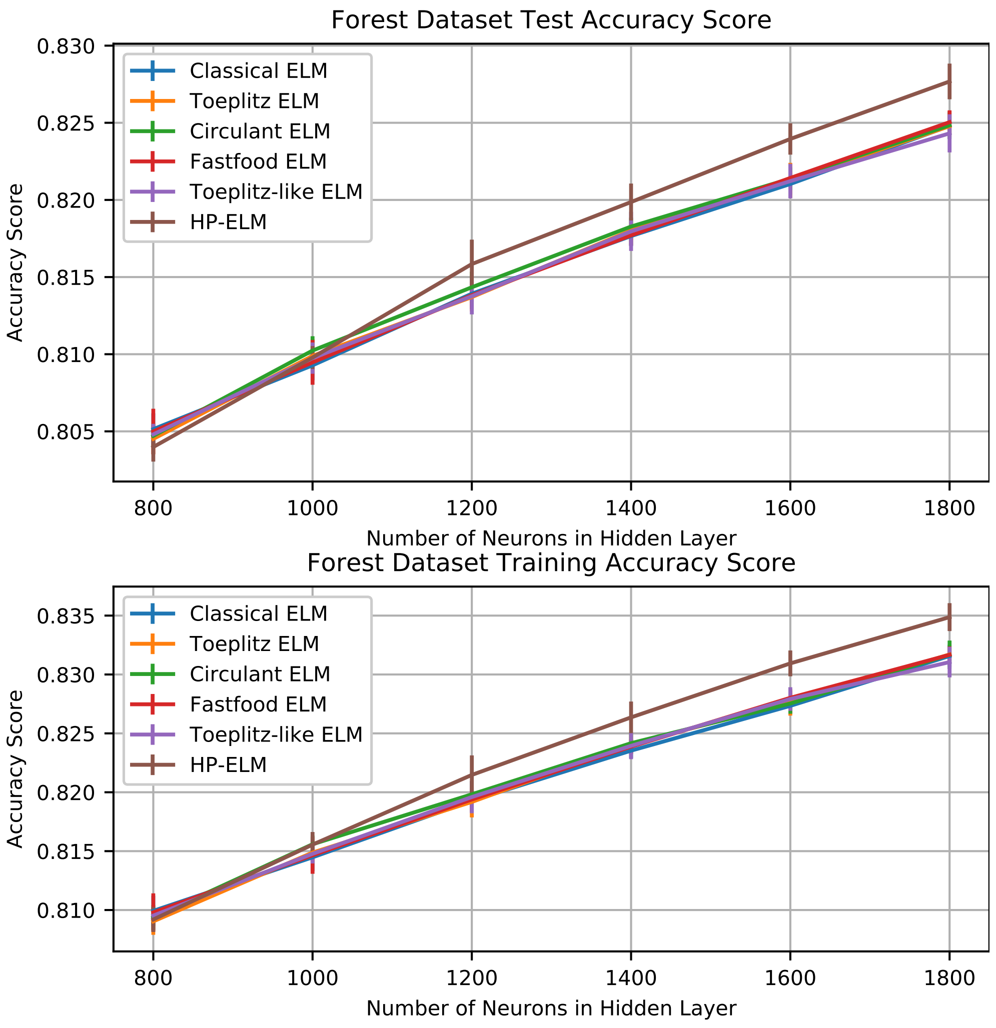


Figure 4.4: Test and training accuracy scores of different ELMs on the Diabetes dataset. The mean and standard deviation of 50 trials is shown. The Toeplitz-like matrix has a displacement rank equal 2.

results of the Vandermonde ELM were significantly lower than those of other ELMs. The Low-Rank ELMs showed slightly worse results than other ELMs. Even the Low-Rank ELM with the rank 45, which is equal to $\frac{5}{6}$ of the inputs number (54), was among the ELMs with the worst results. The Low-Rank ELM with the rank equal to 27 ($\frac{1}{2}$ of the inputs number) had the second worst results after the Vandermonde ELM. One should also take into account that such high ranks as 27 and 54 do not reduce the memory requirements significantly.

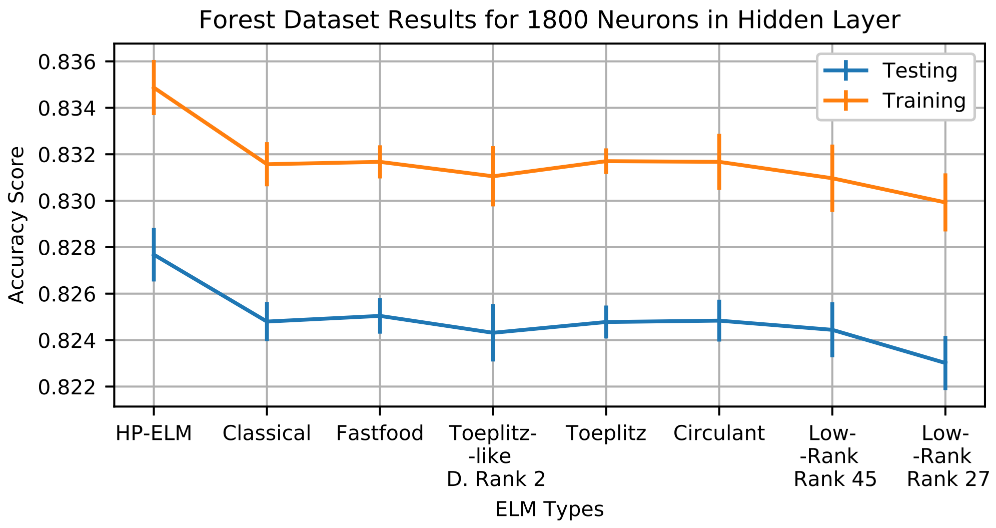


Figure 4.5: Test and training accuracy scores of different ELMs on the Forest dataset. The mean and standard deviation of 12 trials is shown. The Toeplitz-like matrix has a displacement rank ('D. Rank') equal 2.

Name	Acc. Mean		Acc. Dev.	
	Training	Testing	Training	Testing
Classical	0.8316	0.8248	0.00095	0.00084
Toeplitz-like	0.8311	0.8243	0.00130	0.00124
Low-Rank with $Rank = 45$	0.8309	0.8244	0.00145	0.00118
Low-Rank with $Rank = 27$	0.8299	0.8230	0.00125	0.00116
Vandermonde	0.7426	0.7390	0.00946	0.00955

Table 4.2: Results on the Forest dataset for 1800 neurons in hidden layer. The averaged accuracy scores ('Acc. Mean') and standard deviations ('Acc. Dev.') after 12 trials are shown. The Toeplitz-like matrix has a displacement rank equal 2.

4.1.4 MNIST Dataset

The results of the experiments on the MNIST dataset are shown in Figure 4.6. Two experiments were conducted: in the first experiment, pixels were not shuffled. In the second experiment, the pixels of all samples were shuffled using the same permutation matrix. The second experiment was conducted to prove the assumption that the locality or order of features can have an impact on the accuracy of the Circulant, Toeplitz and Toeplitz-like ELMs. This assumption is explained with an example of Circulant matrix in the 'Discussion' section.

As in the previous sections, Vandermonde and Low-Rank ELMs are not included in Figure 4.6 and in the fist two subfigures of Figure 4.7. The ranks of Low-Rank ELMs were chosen to be 686 and 392. These numbers are equal to $\frac{7}{8}$ and $\frac{1}{2}$ of the inputs number (784) re-

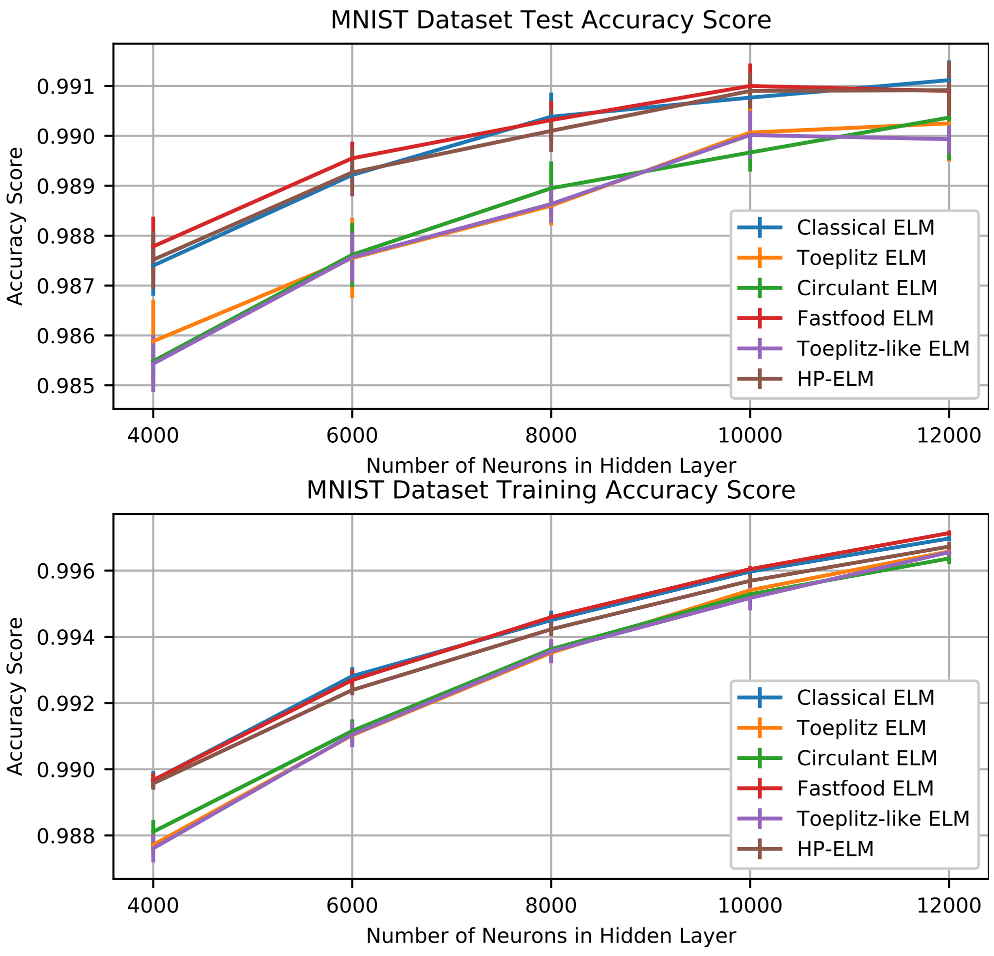


Figure 4.6: Test and training accuracy scores of different ELMs on the MNIST dataset. The means and standard deviations of 6 trials are shown. The Toeplitz-like matrix has a displacement rank equal 2.

spectively. The Low-Rank ELMs' results are illustrated in the last subfigure of Figure 4.6, the Vandermonde ELM's results in Table 4.3.

For the MNIST Dataset, in the first experiment (Figure 4.6), one can divide the included ELMs into two groups: Classical and Fastfood ELMs with the best results and other ELMs, which have slightly worse results. However, the difference between test accuracy values of these groups is only $\approx 0.2\%$ for 4000 neurons and $\approx 0.1\%$ for 12000 neurons. Furthermore, after the pixels were shuffled (Figure 4.7), all ELMs - except for Vandermonde and Low-Rank ELMs - showed similar results. As with the previous datasets, the Low-Rank and Vandermonde ELMs showed lower accuracies than other ELMs.

In order to examine the impact of the free parameters number on the accuracy performance, the Toeplitz-like ELMs with different low displacement ranks were tested, and the results are illustrated in Figure 4.8. According to this figure, it seems that low displacement rank and therefore the number of free parameters has a negligible impact on the accuracy score.

Name	Acc. Mean		Acc. Dev.	
	Training	Testing	Training	Testing
Classical	0.9898	0.9880	2,83 × 10^{-4}	7,23 × 10^{-4}
Toeplitz-like	0.9895	0.9876	2,83 × 10^{-4}	4.86 × 10^{-4}
Low-Rank with $Rank = 686$	0.9896	0.9877	1.80 × 10^{-4}	4.91 × 10^{-4}
Low-Rank with $Rank = 392$	0.9893	0.9875	2.87 × 10^{-4}	2.58 × 10^{-4}
Vandermonde	0.9007	0.8988	2.14 × 10^{-4}	2.79 × 10^{-4}

Table 4.3: Results on the MNIST dataset for 4000 neurons in the hidden layer. The pixels of images were shuffled for all samples using the same permutation matrix. The averaged accuracy scores ('Acc.') and standard deviations ('Acc. Dev.') after 12 trials are shown. The Toeplitz-like matrix has a displacement rank equal 2.

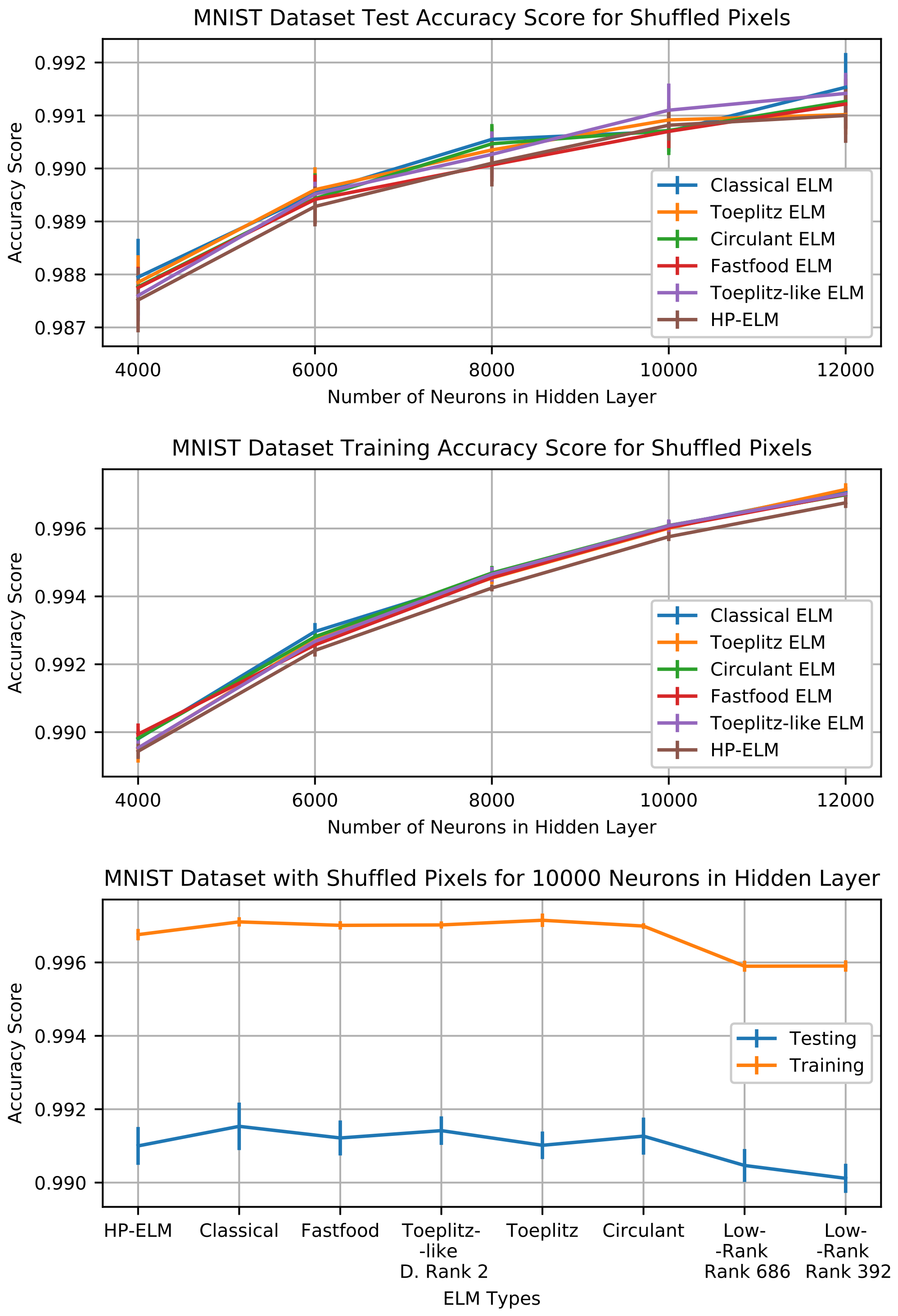


Figure 4.7: Test and training accuracy scores of different ELMs on the MNIST dataset. The pixels of images were shuffled for all samples using the same permutation matrix. The means and standard deviations of 6 trials are shown. The Toeplitz-like matrix has a displacement rank equal 2.

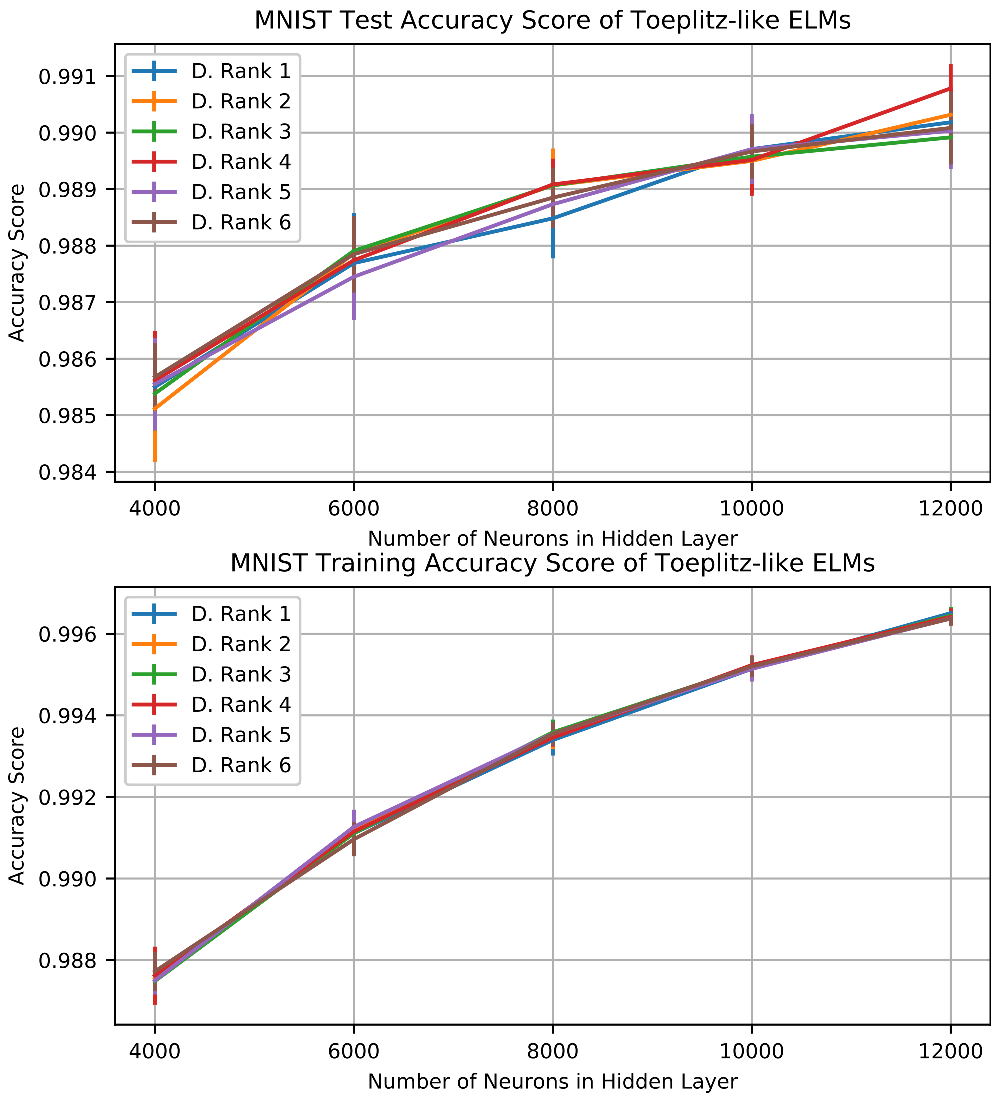


Figure 4.8: Test and training accuracy scores of the Toeplitz-like ELM with different low displacement ranks ('D. Rank') on the MNIST dataset. The means and standard deviations of 12 trials are shown.

4.2 Number of Required Free Parameters for Input Weight Matrices

In Table 4.4, the number of free parameters required for constructing the input weight matrices of the implemented ELMs are shown. These number were obtained through applying the formulas in section 3.1 and in Table 2.3.7. $\lceil \cdot \rceil$ denotes the ceil function.

Dataset	Classical	Low Rank	Fastfood	Toeplitz--like	Toeplitz	Circulant & Vandermonde
Cal.	$8n$	$8r + rn$	$4 \cdot \lceil \frac{n}{8} \rceil \cdot 8$	$4n$	$n + 7$	n
Diab.	$8n$	$8r + rn$	$4 \cdot \lceil \frac{n}{8} \rceil \cdot 8$	$4n$	$n + 7$	n
Forest	$54n$	$54r + rn$	$4 \cdot \lceil \frac{n}{64} \rceil \cdot 64$	$4n$	$n + 53$	n
MNIST	$784n$	$784r + rn$	$4 \cdot \lceil \frac{n}{1024} \rceil \cdot 1024$	$4n$	$n + 783$	n

Table 4.4: Number of free parameters required for constructing the input weight matrices depending on neurons number $n > 8$ in hidden layer. r denotes the rank of Low-Rank matrix in Low-Rank ELMs. Toeplitz-like matrix has a displacement rank equal 2. 'Cal.' means California Housing Dataset, 'Diab.' means Diabetes Dataset.

4.3 Discussion

Basing on the results, one can divide the ELMs with structured matrices as input weight matrices (later ELMs with structured matrices) into three groups. Firstly, the Fastfood ELM shows the same accuracy performance as classical ELM in all experiments.

The second group is Circulant, Toeplitz, and Toeplitz-like ELMs, which show the same results as Classical ELM on all datasets except for Diabetes dataset for number of neurons less than 10 and MNIST dataset if the pixels are not shuffled. However, in the case of MNIST database, the difference is extremely small, $0.1 - 0.2\%$, and it can be avoided by shuffling the features, which is rationalized later. This can be easily achieved by pre-multiplying the input features with a permutation matrix. The vector-matrix multiplication with the permutation matrix takes $O(n)$ time, and the permutation matrix can be built from n parameters, where n is the number of features. In the case of Diabetes dataset, the result is different, only if the number of neurons is less than number of inputs, which seems to be unusual for ELMs.

The reasons for these differences are not absolutely clear. One possible explanation is based on the observation that multiplication between many permutation matrices and Hadamard matrices leads, in general, to the matrices with entries that are highly independent [14]. Hence, Fastfood matrices may have more independent entries than other aforementioned structured matrices due to usage of permutation and Hadamard matrices. One could also describe it as better 'mixing' of free parameters. If the entries of weight matrix are more independent, this could lead to the outputs of hidden layer neurons being more independent and therefore more different. Thus, it may be easier for ELM to extract information from these outputs.

Distinctly, in Circulant matrices, for example, the difference between two subsequent rows is only that all entries in the first row, except for the last one, are moved by one place to the right in the next row (equation (2.13)). At the same time, in the case of image recognition, the pixels that are next to each other have often similar values. Therefore, the hidden layer neurons that are associated with the subsequent rows may also have similar output values. Hence, it may be more difficult for Circulant ELM to extract information from these outputs. This explanation is supported by the fact that after the pixels were shuffled, Circulant ELM showed the same accuracy as Classical ELM. This may be a good example of how dependencies between the entries of input weights matrices can negatively impact the prediction performance.

The third group is Vandermonde and Low-Rank ELMs, which showed significantly worse results than other ELMs. Hence, the corresponding matrices may be considered as examples of structured matrices that are not very suitable as input weight matrices in ELM. In Vandermonde matrix, for free parameters that are smaller than zero, the corresponding weights become extremely small for high power values. For example, in the case of MNIST dataset with 784 features, the last entries of the rows carry the power of 783. If the magnitude of any free parameter is greater than one, the last weights of the corresponding row will have extremely high values.

Regarding the Low-Rank ELMs, low prediction values of these ELMs may be explained as follows: the n-by-m Low-Rank matrix with rank $r < max(n,m)$ can be decomposed as $M = GH^T$ where M is a Low-Rank matrix, G and H^T are n-by-r and r-by-m full rank matrices respectively. Considering the multiplication of Low-Rank matrix with a vector x as $Mx = GH^T x$ where $x \in \mathbb{R}^n$, in the first step, which is multiplication $H^T x$, the product has only r components in opposite to n components in the vector x. Since $r < n$, one can find different vectors that will have the same result after multiplication with H^T. Therefore, some information, which was contained in the original vector x, may be lost after this step.

As an example, one may consider a 100-by-3 Low-Rank matrix M with rank $r = 2$ that can be decomposed as

$$M = GH^T \text{ with } G \in \mathbb{R}^{100\times 2} \text{ and } H^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{4.1}$$

Then, the product of Low-Rank matrix M and vectors $x_1 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$ and $\begin{bmatrix} 0 & 3 & 0 \end{bmatrix}^T$ will be the same

$$Mx_1 = Mx_2 = G \begin{bmatrix} 6 \\ 15 \end{bmatrix}$$

Hence, in this case, the Low-Rank ELM will be not able to differentiate between these two vectors . Thus, one can say that the information is lost. In real case scenarios, after multiplication with Low-Rank matrix, only fewer samples may have exactly the same results but many samples may have similar results. The differences within these similar results may be not distinguishable from noise.

Furthermore, Low-Rank matrices, when used as input weight matrices, can lead to more collinearities in the outputs of the hidden layer, since various samples may have similar results after multiplication with the input matrix. This is also supported by the observation that Low-Rank ELMs showed high deviations in accuracy levels for California and Diabetes datasets. However, for the Forest and MNIST Datasets, which contain more features and require more neurons, the deviations in accuracy levels of Low-Rank ELMs were similar to that of other ELMs.

Another observation can be seen from Table 4.4 illustrating that the number of free parameters required for structured matrices is significantly less than the ones required for Classical ELM. The results of all conducted experiments imply that the accuracy performance mostly does not depend on the number of free parameters. However, it depends on how these free parameters were used.

5 Conclusion

The idea of replacing input weight matrices with structured matrices was proposed in [18]. In this thesis, I explored the possibility of replacing, without weakening the output accuracy, input weight matrices with structured matrices in ELM.

After applying this idea to ELM, it was found that the following enlisted points, if not considered while replacing input weight matrices with structured matrices, can affect the accuracy:

- *Distributions of weights and free parameters* – in previous works, it was concluded that the distribution of entries in weight matrix have an impact on accuracy of ELM [1]. In this paper, I have shown that the distribution of entries of structured matrices are not necessarily equal to the distribution of free parameters, as shown in table 2.3.7. Therefore, the same must be considered while replacing input weight matrices with structured matrices. For example, the entries of Fastfood transformation matrix belong always to Gaussian distribution, and the entries of Toeplitz-like matrix have a variance that is quadratic to the variance of free parameters.
- *Choice of suitable structured matrix* – not all structured matrices are suitable for replacement in input weight matrices. For example - as shown in the experiments done under this thesis - ELMs with Low-Rank or Vandermonde matrices as input weight matrix have shown low accuracy values. In case of Vandermonde matrix, if it has high dimensions, it will result in extremely high or low magnitudes; and in case of Low-rank matrix, some of the information contained in the data samples may be lost after its multiplication with Low-Rank matrix, as is was shown in chapter 4.
- *Dissimilarity in outputs of hidden layer* - it may be preferable that the outputs of the hidden layer are as dissimilar as possible, which can be achieved if the entries of the weight input matrix are highly independent, as in the case of Fastfood Transformation matrix. In fact, ELM with Fastfood matrices showed similar accuracy as conventional ELM have shown in all experiments. However, ELMs with structured matrices having less independent entries than in Fastfood matrix, such as ELMs with Circulant, Toeplitz and Toeplitz-like matrices, also achieved the same accuracy as conventional ELMs in all the experiments except for following cases:
  - Experiments on Diabetes Dataset for number of neurons in the hidden layer less than the input numbers.
  - Experiments on MNIST dataset if the pixels are not shuffled.

> However, the first case is highly rare in occurrence, and in the second case, same accuracy as that of conventional ELMs was achieved by shuffling the features of input data using permutation matrix. In general, one can always consider combining Circulant, Toeplitz and Toeplitz-like matrices with permutation matrix in cases when input features show high correlations, for example, in MNIST dataset, as explained in chapter 4.

Another important conclusion may be that the accuracy of ELMs with structured matrices depends on how free parameters are used and are not dependent on their number, which can be correlated with the third point.

The replacement of input weights with structured matrices will eventually have the following benefits for ELM while achieving the same accuracy levels:

- Reduction in the processing time of the ELM algorithm (Table 2.3.7).
- Significant reduction in memory requirements (tables 2.3.7 and 4.4).

Therefore, the replacement of the input weight matrices with structured matrices in ELM seems to provide a win-win situation for reducing propagation time and reducing memory requirements without affecting its accuracy.

Bibliography

[1] A. Akusok, K.-M. Björk, Y. Miche, and A. Lendasse. “High Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications”. In: *Access, IEEE* PP (2015).

[2] D. Das, D. Nayak, R. Dash, and B. Majhi. *Backward-Forward Algorithm: An Improvement towards Extreme Learning Machine*. 2019.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[4] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. Freitas. “Predicting Parameters in Deep Learning”. In: *NIPS* (June 2013).

[5] S. Ding, N. Zhang, X. Xu, L. Guo, and J. Zhang. “Deep Extreme Learning Machine and Its Application in EEG Classification”. In: *Mathematical Problems in Engineering* 2015 (2015), pp. 1–11.

[6] R. M. Gray. “Toeplitz and Circulant Matrices: A Review”. In: *Foundations and Trends® in Communications and Information Theory* 2(3) (2006), pp. 155–239.

[7] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[8] G. Huang, G.-B. Huang, S. Song, and K. You. “Trends in extreme learning machines: A review”. In: *Neural Networks* 61 (2015), pp. 32–48.

[9] G.-B. Huang. “An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels”. In: *Cognitive Computation* 6 (2014), pp. 376–390.

[10] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. “Extreme learning machine: A new learning scheme of feedforward neural networks”. In: vol. 2. 2004, pp. 985–990.

[11] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. “Extreme learning machine: Theory and applications”. In: *Neurocomputing* 70(1) (2006), pp. 489–501.

[12] A. K. Jain, J. Mao, and K. M. Mohiuddin. “Artificial neural networks: A tutorial”. In: *Computer* 29(3) (1996), pp. 31–44.

[13] R. Kelley Pace and R. Barry. “Sparse spatial autoregressions”. In: *Statistics & Probability Letters* 33(3) (1997), pp. 291–297.

[14] Q. Le, T. Sarlos, and A. Smola. “Fastfood: Approximate Kernel Expansions in Loglinear Time”. In: *30th International Conference on Machine Learning, ICML 2013* (2014).

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86(11) (1998), pp. 2278–2324.

[16] M. Mcdonnell, M. Tissera, T. Vladusich, A. van Schaik, and J. Tapson. “Fast, Simple and Accurate Handwritten Digit Classification by Training Shallow Neural Network Classifiers with the ’Extreme Learning Machine’ Algorithm”. In: *PloS one* 10 (2015), e0134254.

[17] B. Qu, B. Lang, J. Liang, A. Qin, and O. Crisalle. “Two-hidden-layer extreme learning machine for regression and classification”. In: *Neurocomputing* 175 (2016), pp. 826–834.

[18] V. Sindhwani, T. Sainath, and S. Kumar. “Structured Transforms for Small-Footprint Deep Learning”. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 3088–3096.

[19] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. “Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus”. In: 1988.

[20] D. Svozil, V. Kvasnicka, and J. Pospichal. “Introduction to multi-layer feed-forward neural networks”. In: *Chemometrics and Intelligent Laboratory Systems* 39(1) (1997), pp. 43–62.

[21] D. Xiao, B. Li, and Y. Mao. “A Multiple Hidden Layers Extreme Learning Machine Method and Its Application”. In: *Mathematical Problems in Engineering* 2017 (2017), pp. 1–10.

[22] Z. Yang, M. Moczulski, M. Denil, N. De Freitas, L. Song, and Z. Wang. “Deep Fried Convnets”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1476–1483.