PUBLISHED VERSION

Mark D. McDonnell, Migel D. Tissera, Tony Vladusich, André van Schaik, Jonathan Tapson
Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the 'extreme learning machine' algorithm

http://hdl.handle.net/2440/94666

Fast, Simple and Accurate Handwritten Digit Classification by Training Shallow Neural Network Classifiers with the 'Extreme Learning Machine' Algorithm

Mark D. McDonnell[1] *, **Migel D. Tissera**[1], **Tony Vladusich**[1], **André van Schaik**[2], **Jonathan Tapson**[2]

1 Computational and Theoretical Neuroscience Laboratory, Institute for Telecommunications Research, School of Information Technology and Mathematical Sciences, University of South Australia, Mawson Lakes, SA 5095, Australia, **2** Biomedical Engineering and Neuroscience Group, The MARCS Institute, The University of Western Sydney, Penrith, NSW 2751, Australia

* mark.mcdonnell@unisa.edu.au

Abstract

Recent advances in training deep (multi-layer) architectures have inspired a renaissance in neural network use. For example, deep convolutional networks are becoming the default option for difficult tasks on large datasets, such as image and speech recognition. However, here we show that error rates below 1% on the MNIST handwritten digit benchmark can be replicated with shallow non-convolutional neural networks. This is achieved by training such networks using the 'Extreme Learning Machine' (ELM) approach, which also enables a very rapid training time ($\sim$ 10 minutes). Adding distortions, as is common practise for MNIST, reduces error rates even further. Our methods are also shown to be capable of achieving less than 5.5% error rates on the NORB image database. To achieve these results, we introduce several enhancements to the standard ELM algorithm, which individually and in combination can significantly improve performance. The main innovation is to ensure each hidden-unit operates only on a randomly sized and positioned patch of each image. This form of random 'receptive field' sampling of the input ensures the input weight matrix is sparse, with about 90% of weights equal to zero. Furthermore, combining our methods with a small number of iterations of a single-batch backpropagation method can significantly reduce the number of hidden-units required to achieve a particular performance. Our close to state-of-the-art results for MNIST and NORB suggest that the ease of use and accuracy of the ELM algorithm for designing a single-hidden-layer neural network classifier should cause it to be given greater consideration either as a standalone method for simpler problems, or as the final classification stage in deep neural networks applied to more difficult problems.

Introduction

The current renaissance in the field of neural networks is a direct result of the success of various types of deep network in tackling difficult classification and regression problems on large datasets. It may be said to have been initiated by the development of Convolutional Neural Networks (CNN) by LeCun and colleagues in the late 1990s [1] and to have been given enormous impetus by the work of Hinton and colleagues on Deep Belief Networks (DBN) during the last decade [2]. It would be reasonable to say that deep networks are now considered to be a default option for machine learning on large datasets.

The initial excitement over CNN and DBN methods was triggered by their success on the MNIST handwritten digit recognition problem [1], which was for several years the standard benchmark problem for hard, large dataset machine learning. A high accuracy on MNIST is regarded as a basic requirement for credibility in a classification algorithm. Both CNN and DBN methods were notable, when first published, for posting the best results up to that respective time on the MNIST problem.

The standardised MNIST database consists of 70,000 images, each of size 28 by 28 greyscale pixels [3]. There is a standard set of 60,000 training images and a standard set of 10,000 test images, and numerous papers report results of new algorithms applied to these 10,000 test images, e.g. [1, 4–9].

In this report, we introduce variations of the Extreme Learning Machine algorithm [10] and report their performance on the MNIST test set. These results are equivalent or superior to the original results achieved by CNN and DBN on this problem, and are achieved with significantly lower network and training complexity. This poses the important question as to whether the ELM training algorithm should be a more popular choice for this type of problem, and a more commonplace algorithm as a first step in machine learning.

Table 1 summarises our results, and shows some comparison points with results obtained by other methods in the past (note that only previous results that do not use data augmentation methods are shown, and only one of our new results is for such a case). Our new results surpass results using earlier deep networks, but recent regularisation methods such as drop connect [6], stochastic pooling [7], dropout [8] and so-called 'deeply supervised networks' [9] have enabled deep convolutional networks to set new state-of-the-art performance for MNIST for the case where no data-augmentation is used. Nevertheless, our best result for a much simpler single-hidden-layer neural network classifier trained using the very fast ELM algorithm, and without using data augmentation, is within just 41 errors out of 10000 test images of the state-of-the-art.

The Extreme Learning Machine: Notation and Training

The Extreme Learning Machine (ELM) training algorithm [10] is relevant for a single hidden layer feedforward network (SLFN), similar to a standard neural network. However, there are three key departures from conventional SLFNs. These are (i) that the hidden layer is frequently very much larger than a neural network trained using backpropagation; (ii) the weights from the input to the hidden layer neurons are randomly initialised and are fixed thereafter (i.e., they are not trained); and (iii) the output neurons are linear rather than sigmoidal in response, allowing the output weights to be solved by least squares regression. These attributes have also been combined in learning systems several times previously [11–14].

The standard ELM algorithm can provide very good results in machine learning problems requiring classification or regression (function optimization); in this paper we demonstrate that it provides an accuracy on the MNIST problem superior to prior reported results for similarly-sized SLFN networks [1, 15].

Table 1. Comparison of our results on the MNIST data set with published results using other methods.
The percentages listed in brackets are the mean error percentage we obtained from 10 independent realisations of each method. The remaining percentage for the results obtained in this report are from the trained ELM out of the 10 repeats with the best results. Values for each trial shown in Figs 3 and 4 demonstrate small spreads either side of the mean value. The abbreviations CIW, C and RF are explained in later sections. Note: SLFN is "Single-hidden Layer Feedforward Network"; DLFN is "Dual-hidden Layer Feedforward Network." The result described as 'Distortions' was obtained by augmenting the training set using affine and elastic distortions, as describe in the main text. 'Deep. Conv. Net' is an abbreviation for 'Deep Convolutional Network'.

Grouping	Method	Error in testing	Reference
Selected Non-ELM	SLFN, 784–1000–10	4.5%	[1]
	Deep Belief Network	1.25%	[2]
	Deep Conv. Net LeNet-5	0.95%	[1]
	Deep Conv. Net (dropconnect)	0.57%	[6]
	Deep Conv. Net (stochastic pooling)	0.47%	[7]
	Deep Conv. Net (maxout units and dropout)	0.45%	[8]
	Deep Conv. Net (deeply-supervised)	0.39%	[9]
Past ELM	ELM, 784–1000–10	6.05%	[15]
	C-ELM	~ 5%	[21]
	CIW-ELM, 784–1000–10	3.55%	[15]
	ELM, 784–7840-10	2.75%	[27]
	ELM, 784-unknown-10	2.61%	[20]
	CIW-ELM, 784–7000-10	1.52%	[15]
	ELM+backpropagation, 784–2048-10	1.45%	[23]
	Deep ELM, 784–700-15000–10	0.97%	[20]
ELM & backprop	RF-(CIW & C)-ELM, 784—(2 × 6400)-20-500-10	0.91% (1.04%)	This report
SLFN ELM	RF-ELM,784–15000–10	1.36% (1.48%)	This report
	CIW-ELM,784–15000–10	1.28% (1.45%)	This report
	C-ELM, 784–15000–10	1.26% (1.33%)	This report
	RF-CIW-ELM, 784–15000–10	1.03%(1.10%)	This report
	RF-C-ELM, 784–15000–10	0.9% (0.99%)	This report
	RF-C-ELM, 784–25000-10, Distortions	0.57% (0.62%)	This report
DLFN ELM	RF-(CIW& C)-ELM, 784-(2 × 15000)-20-500-10	0.83% (0.87%)	This report

doi:10.1371/journal.pone.0134254.t001

We begin by introducing three parameters that define the dimensions of an ELM used as an N-category classifier: L is the dimension of input vectors, M is the number of hidden layer units, and N is the number of distinct labels for training samples. For the case of classifying P test vectors, it is convenient to define the following matrices:

- $\mathbf{X}_{\text{test}}$, of size $L \times P$, is formed by setting each column to equal a single test vector.

- $\mathbf{Y}_{\text{test}}$, of size $N \times P$, numerically represents the prediction vector of the classifier.

To map from input vectors to network outputs, two weights matrices are required:

- $\mathbf{W}_{\text{in}}$, of size $M \times L$, contains the input weight matrix that maps length-L input vectors to length M hidden-unit inputs.

- $\mathbf{W}_{\text{out}}$, of size $N \times M$, contains the output weights that project from the M hidden-unit activations to a length N class prediction vector.

We also introduce matrices to describe inputs and outputs to/from the hidden-units:

- $\mathbf{D}_\text{test} := \mathbf{W}_\text{in}\mathbf{X}_\text{test}$, of size $M \times P$, contains the linear projections of the input vectors that are inputs to each of the M hidden-units. A bias for each hidden unit can be added by expanding the size of the input dimension from L to $L+1$, and setting the additional input element to always be unity for all training and test data, with the bias values included as an additional column in $\mathbf{W}_\text{in}$.

- $\mathbf{A}_\text{test}$, of size $M \times P$, contains the hidden-unit activations that occur due to each training vector, and is given by

$$\mathbf{A}_\text{test} := f(\mathbf{D}_\text{test}), \tag{1}$$

where $f(\cdot)$ is shorthand notation for the fact that each element of $\mathbf{D}_\text{test}$ is nonlinearly converted term-by-term to the corresponding element of $\mathbf{A}_\text{test}$. For example, if the hidden unit response is given by the logistic sigmoid function, then

$$(\mathbf{A}_\text{test})_{i,j} = f((\mathbf{D}_\text{test})_{i,j}) = \frac{1}{1 + \exp(-(\mathbf{D}_\text{test})_{i,j})}. \tag{2}$$

Many nonlinear activation functions can be equally effective, such as the rectified linear unit (ReLU) function [16], the absolute value function or the quadratic function. As with standard artificial neural networks, the utility of the nonlinearity is that it introduces hidden-unit responses that represent correlations or 'interactions' between input elements, rather than simple linear combinations of them.

The overall conversion of test data to prediction vectors can be written as

$$\mathbf{Y}_\text{test} = \mathbf{W}_\text{out}\, f(\mathbf{W}_\text{in}\mathbf{X}_\text{test}). \tag{3}$$

We now describe the ELM training algorithm. We introduce K to denote the number of training vectors available. It is convenient to introduce the following matrices that are relevant for training an ELM: $\mathbf{X}_\text{train}$, of size $L \times K$, $\mathbf{A}_\text{train}$, of size $M \times K$, and $\mathbf{Y}_\text{train} = \mathbf{W}_\text{out}\mathbf{A}_\text{train}$, of size $N \times K$ are defined analogously to $\mathbf{X}_\text{test}$, $\mathbf{A}_\text{test}$ and $\mathbf{Y}_\text{test}$ above. We also introduce $\mathbf{Y}_\text{label}$, of size $N \times K$, which numerically represents the labels of each class of each training vector; it is convenient to define this mathematically such that each column has a 1 in a single row, and all other entries are zero. The only 1 entry in each column occurs in the row corresponding to the label class for each training vector.

Ideally we seek to find to find $\mathbf{W}_\text{out}$ that satisfies

$$\mathbf{Y}_\text{label} = \mathbf{W}_\text{out}\mathbf{A}_\text{train}. \tag{4}$$

However, the number of unknown variables in $\mathbf{W}_\text{out}$ is NM, and the number of equations is NK. Although an exact solution potentially exists if $M = K$, it is usually the case that $M < K$ (i.e., there are many more training samples than hidden units) so that the system is overcomplete. The usual approach then, is to seek the solution that minimises the mean square error between $\mathbf{Y}_\text{label}$ and $\mathbf{Y}_\text{train}$. This is a standard least squares regression problem for which the exact solution is $\mathbf{W}_\text{out} = \mathbf{Y}_\text{label}\mathbf{A}_\text{train}^\top(\mathbf{A}_\text{train}\mathbf{A}_\text{train}^\top)^{-1}$, assuming that the inverse exists (in practice it usually does).

It can also be useful to regularise the problem to reduce overfitting, by ensuring that the weights of $\mathbf{W}_\text{out}$ do not become large. The standard ridge-regression approach [17] produces the following closed form solution for the output weights:

$$\mathbf{W}_\text{out} = \mathbf{Y}_\text{label}\mathbf{A}_\text{train}^\top(\mathbf{A}_\text{train}\mathbf{A}_\text{train}^\top + c\mathbf{I})^{-1}, \tag{5}$$

where $\mathbf{I}$ is the $M \times M$ identity matrix, and c can be optimised using cross-validation techniques. As is discussed in more detail below, we have found QR decomposition [18] to be the most effective method for solving for $\mathbf{W}_{\text{out}}$.

Faster and more accurate performance by shaping the input weights non-randomly

In the conventional ELM algorithm, the input weights are randomly chosen, typically from a continuous uniform distribution on the interval $[-1, 1]$ [19], but we have found that other distributions such as bipolar binary values from $\{-1, 1\}$ are equally effective.

Beyond such simple randomisation of the input weights, small improvements can be made by ensuring the rows of $\mathbf{W}_{\text{in}}$ are as mutually orthogonal as possible [20]. This cannot be achieved exactly unless $M \leq L$, but simple random weights typically produce dot products of distinct rows of $\mathbf{W}_{\text{in}}$ that are close to zero, albeit not exactly zero, while a dot product of each row with itself is always much larger than zero. In addition, it can be beneficial to normalise the length of each row of $\mathbf{W}_{\text{in}}$, as occurs in the orthogonal case.

In contrast, we can also aim to find weights that, rather than being selected from a random distribution, are instead chosen to be well *matched* to the statistics of the data, with the hope that this will improve generalisation of the classifier. Ideally we do not want to have to learn these weights, but rather just form the weights as a simple function of the data.

Here we focus primarily on improving the performance of the ELM algorithm by biasing the selection of input layer weights in six different ways, several of which were recently introduced in the literature, and several of which are novel in this paper. These methods are as follows:

1. Select input layer weights that are random, but biased using the training data samples, so that the dot product between weights and training data samples is likely to be large. This is called Computed Input Weights ELM (CIW-ELM) [15].

2. Ensure input weights are constrained to a set of difference vectors of between-class samples in the training data. This is called Constrained ELM (C-ELM) [21].

3. Restrict the weights for each hidden layer neuron to be non-zero only for a small, random rectangular patch of the input visual field; we call this Receptive Field ELM (RF-ELM). Although we believe this method to be new to ELM approaches, it is inspired by other machine learning approaches that aim to mimic cortical neurons that have limited visual receptive fields, such as convolutional neural networks [22].

4. Combine RF-ELM with CIW-ELM, or RF-ELM with C-ELM; we show below that the combination is superior to any of the three methods individually.

5. Pass the results of a RF-CIW-ELM and a RF-C-ELM into a third standard ELM (thus producing a two-layer ELM system); we show below that this gives the best overall performance of all methods considered in this paper.

6. Application of the backpropagation method of [23]. This method highlights that the performance of an ELM can be enhanced by adjusting all input layer weights simultaneously, based on all training data. The output layer weights are maintained in their least-squares optimal state by recalculating them after input layer backpropagation updates. The process of backpropagation updating of the input weights followed by standard ELM recalculation of the output weights can be repeated iteratively until convergence.

RF-ELM and its combination with CIW-ELM and C-ELM, and the two-layer ELM are reported here for the first time. We demonstrate below that each of these methods

independently improves the performance of the basic ELM algorithm, and in combination they produce results equivalent to many deep networks on the MNIST problem [1, 2]. First, however, we now describe each method in detail.

Computed Input Weights for ELM

The CIW-ELM approach is motivated by considering the standard backpropagation algorithm [24]. A feature of weight-learning algorithms is that they operate by adding to the weights some proportion of the training samples, or a linear sum or difference of training samples. In other words, apart from a possible random initialization, the weights are constrained to take final values which are drawn from a space defined in terms of linear combinations of the input training data as basis vectors—see Fig 1. While it has been argued, not without reason, that it is a strength of ELM that it is not thusly constrained [25], the use of this basis as a constraint on input weights will bias the ELM network towards a conventional (backpropagation) solution.

The CIW-ELM algorithm is as follows [15]:

1. For use in the following steps only, normalize all training data by subtracting the mean over all training points and dimensions and then dividing by the standard deviation.

2. Divide the M hidden layer neurons into N blocks, one for each of N output classes; for data sets where the number of training data samples for each class are equal, the block size is $M_n = M/N$. We denote the number of training samples per class as K_n, $n = 1, \ldots, N$. If the training data sets for each class are not of equal size, the block size can be adjusted to be proportional to the data set size.

3. For each block, generate a random sign (± 1) matrix, $\mathbf{R}_n$ of size $M_n \times K_n$.

4. Multiply $\mathbf{R}_n$ by the transpose of the input training data set for that class, $X_{\mathrm{train},n}^\top$, to produce $M_n \times L$ summed inner products, which are the weights for that block of hidden units.

5. Concatenate these N blocks of weights for each class into the $M \times L$ input weight matrix $\mathbf{W}_{\mathrm{in}}$.

6. Normalize each row of the input weight matrix, $\mathbf{W}_{\mathrm{in}}$, to unity length.

7. Solve for the output weights of the ELM using standard ELM methods described above.

Constrained Weights for ELM

Recently, Zhu *et al.* [21] have published a method for constraining the input weights of ELM to the set of difference vectors of between-class samples. The difference vectors of between-class samples are the set of vectors connecting samples of one class with samples of a different class, in the sample space—see Fig 1. In addition, a methodology is proposed for eliminating from this set the vectors of potentially overlapping spaces (effectively, the shorter vectors) and for reducing the use of near-parallel vectors, in order to more uniformly sample the weight space.

The Constrained ELM (C-ELM) algorithm we used is as follows [21]:

1. Randomly select M distinct pairs of training data such that:

   a. each pair comes from two distinct classes;

   b. the vector length of the difference between the pairs is smaller than some constant, ϵ.

2. Set each row of the $M \times L$ input weight matrix $\mathbf{W}_{\mathrm{in}}$ to be equal to the difference between each pair of randomly selected training data.
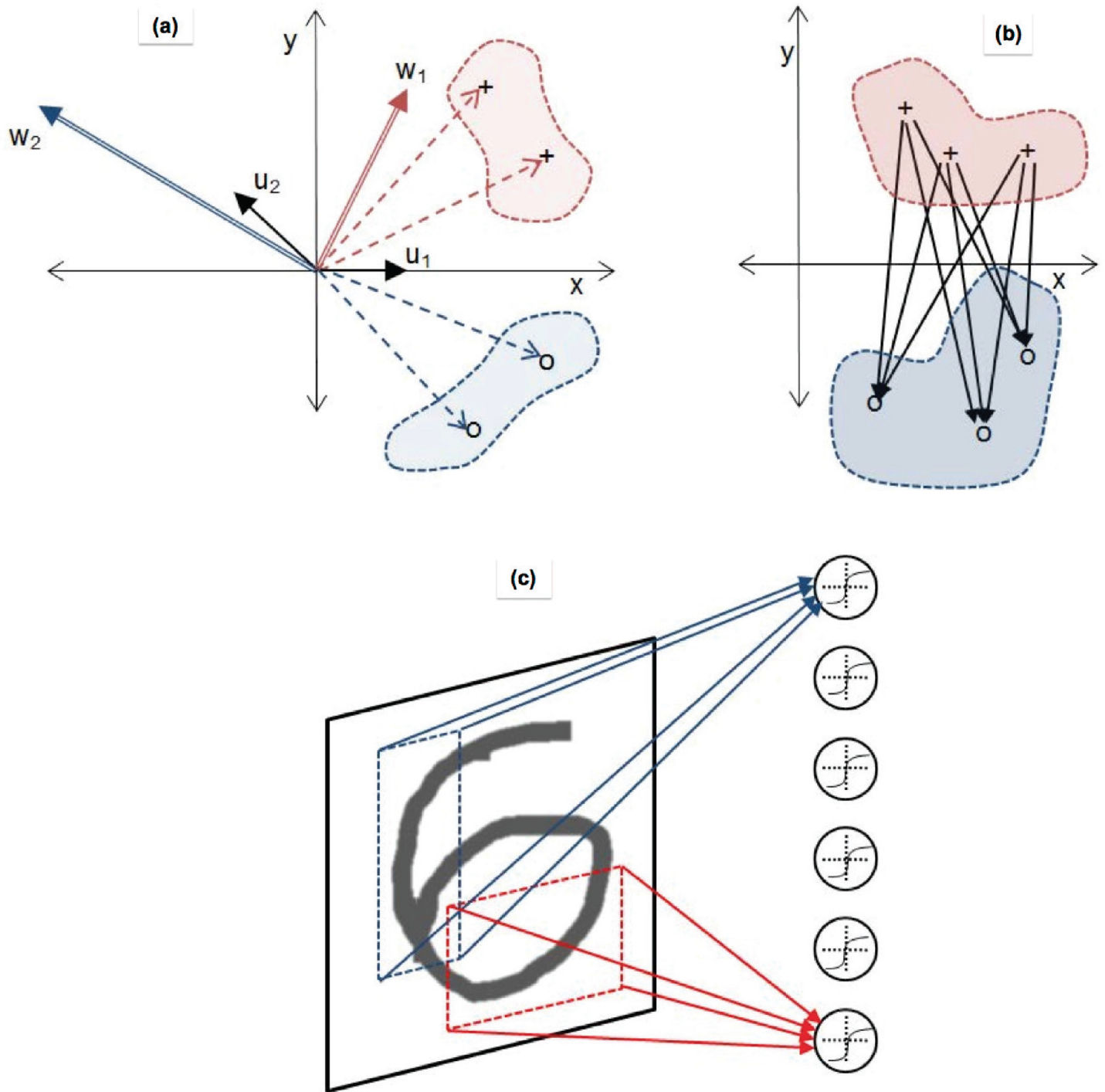
Fig 1. Illustration of the three core methods of shaping ELM input weights. In (a), which is a cartoon of the Computed Input Weights ELM (CIW-ELM) process [15], two classes of input data are indicated by '+' and 'o' symbols. The vectors to the '+' symbols are multiplied by random bipolar binary {−1, 1}) vectors u_1 and u_2 to produce a biased random weight vector w_1. Similarly the weights to the 'o' class are also multiplied by random vectors u_1 and u_2 to produce a biased random weight vector w_2. Note that in practice we would not use the same random binary vectors. In (b), we show the Constrained ELM (C-ELM) process [21]. The black arrows are weight vectors derived by computing the difference of two classes; in this case, the difference between the '+' elements and the 'o' elements. In (c), we illustrate the Receptive Field ELM (RF-ELM) method; weights for each hidden layer neuron are restriced to being non-zero for only a small random rectangular receptive field in the original image plane.

doi:10.1371/journal.pone.0134254.g001

3. Set the bias for each hidden unit equal to the scalar product of the sum of each pair of randomly selected training data and the difference of each pair of randomly selected training data.

4. Normalize each row of the input weight matrix, $\mathbf{W}_{\text{in}}$, and each bias value, by the vector of the difference of the corresponding pair of randomly selected training data.

5. Solve for the output weights of the ELM using standard ELM methods described above.

Receptive Fields for ELM

We have found that a data-blind (unsupervised) manipulation of the input weights improves generalization performance. The approach has the added bonus that the input weight matrix is sparse, with a very high percentage of zero entries, which could be advantageous for hardware implementations, or if sparse matrix storage methods are used in software.

The RF-ELM approach is inspired by neurobiology, and strongly resembles some other machine learning approaches [22]. Biological sensory neurons tend to be tuned with preferred receptive fields so that they receive input only from a subset of the overall input space. The region of responsiveness tends to be contiguous in some pertinent dimension, such as space for the visual and touch systems, and frequency for the auditory system. Interestingly, this contiguity aspect may be lost beyond the earliest neural layers, if features are combined randomly.

In order to loosely mimic this organisation of biological sensory systems, in this paper where we consider only image classification tasks, for each hidden unit we create randomly positioned and sized rectangular masks that are smaller than the overall image. These masks ensure only a small subset of the length-L input data vectors influence any given hidden unit—see Fig 1.

The algorithm for generating these 'receptive-field' masks is as follows:

1. Generate a random input weight matrix $\mathbf{W}$ (or instead start with a CIW-ELM or C-ELM input weight matrix).

2. For each of M hidden units, select two pairs of distinct random integers from $\{1, 2, \ldots L\}$ to form the coordinates of a rectangular mask.

3. If any mask has total area smaller than some value q then discard and repeat.

4. Set the entries of a $\sqrt{L} \times \sqrt{L}$ square matrix that are defined by the two pairs of integers to 1, and all other entries to zero.

5. Flatten each receptive field matrix into a length L vector where each entry corresponds to the same pixel as the entry in the data vectors $\mathbf{X}_{\text{test}}$ or $\mathbf{X}_{\text{train}}$.

6. Concatenate the resulting M vectors into a receptive field matrix $\mathbf{F}$ of size $M \times L$.

7. Generate the ELM input weight matrix by finding the Hadamard product (term by term multiplication) $\mathbf{W}_{\text{in}} = \mathbf{F} \circ \mathbf{W}$.

8. Normalize each row of the input weight matrix, $\mathbf{W}_{\text{in}}$, to unity length.

9. Solve for the output weights of the ELM using standard ELM methods described above.

We have additionally found it beneficial to exclude pixels from the mask if most or all training images have identical values for those regions. For the MNIST database, this typically means ensuring all masks exclude the first and last 3 rows and first and last 3 columns. For MNIST we have found that a reasonable value of the minimum mask size is $q = 10$, which enables masks of size 1×10 and 2×5 and larger, but not 3×3 or smaller.

Note that unlike the random receptive field patches described in [22] for use in a convolutional network, our receptive field masks are not of uniform size, and not square; we found it beneficial to ensure a large range of receptive field areas, and large diversity in ratios of lengths to widths.

Combining RF-ELM with CIW-ELM and C-ELM

All three approaches described so far provide weightings for pixels for each hidden layer unit. CIW-ELM and C-ELM weight the pixels to bias hidden-units towards a larger response for training data from a specific class. The sparse weightings provided by RF-ELM bias hidden-units to respond to pixels from specific parts of the image.

We have found that enhanced classification performance can be achieved by combining the shaped weights obtained by either CIW-ELM or C-ELM with the receptive field masks provided by RF-ELM. The algorithm for either RF-CIW-ELM or RF-C-ELM is as follows.

1. Follow the first 5 steps of the above CIW-ELM or the first 2 steps of the C-ELM algorithm, to obtain an un-normalized shaped input weight matrix, $\mathbf{W}_{\text{in, s}}$.

2. Follow the first 6 steps of the RF-ELM algorithm to obtain a receptive field matrix, $\mathbf{F}$.

3. Generate the ELM input weight matrix by finding the Hadamard product (term by term multiplication) $\mathbf{W}_{\text{in}} = \mathbf{F} \circ \mathbf{W}_{\text{in, s}}$.

4. Normalize each row of the input weight matrix, $\mathbf{W}_{\text{in}}$, to unity length.

5. If RF-C-ELM, produce the biases according to steps 3 and 4 of the C-ELM algorithm, but use the masked difference vectors rather than the unmasked ones.

6. Solve for the output weights of the ELM using standard ELM methods described above.

Combining RF-C-ELM with RF-CIW-ELM in a two-layer ELM: RF-CIW-C-ELM

We have found that results obtained with RF-C-ELM and RF-CIW-ELM are similar in terms of error percentage when applied to the MNIST benchmark, but the errors follow different patterns. As such, a combination of the two methods seemed to offer promise. We have combined the two methods using a multiple-layer ELM which consists of an RF-C-ELM network and a RF-CIW-ELM network in parallel, as the first two layers. The outputs of these two networks are then combined using a further ELM network, which can be thought of as an ELM-autoencoder, albeit that it has twenty input neurons and ten output neurons; the input neurons are effectively two sets of the same ten labels. The structure is shown in Fig 2. The two input networks are first trained to completion in the usual way, then the autoencoder layer is trained using the outputs of the input networks as its input, and the correct labels as the outputs. The result of this second-layer network, which is very quick to implement (as it uses a hidden layer of typically only 500–1000 neurons), is significantly better than the two input networks (see Table 1). Note that the middlemost layer shown in Fig 2 consists of linear neurons, and therefore it can be removed by combining its input and output weights into one connecting weight matrix. However, it is computationally disadvantageous to do so because the number of multiplications will increase.
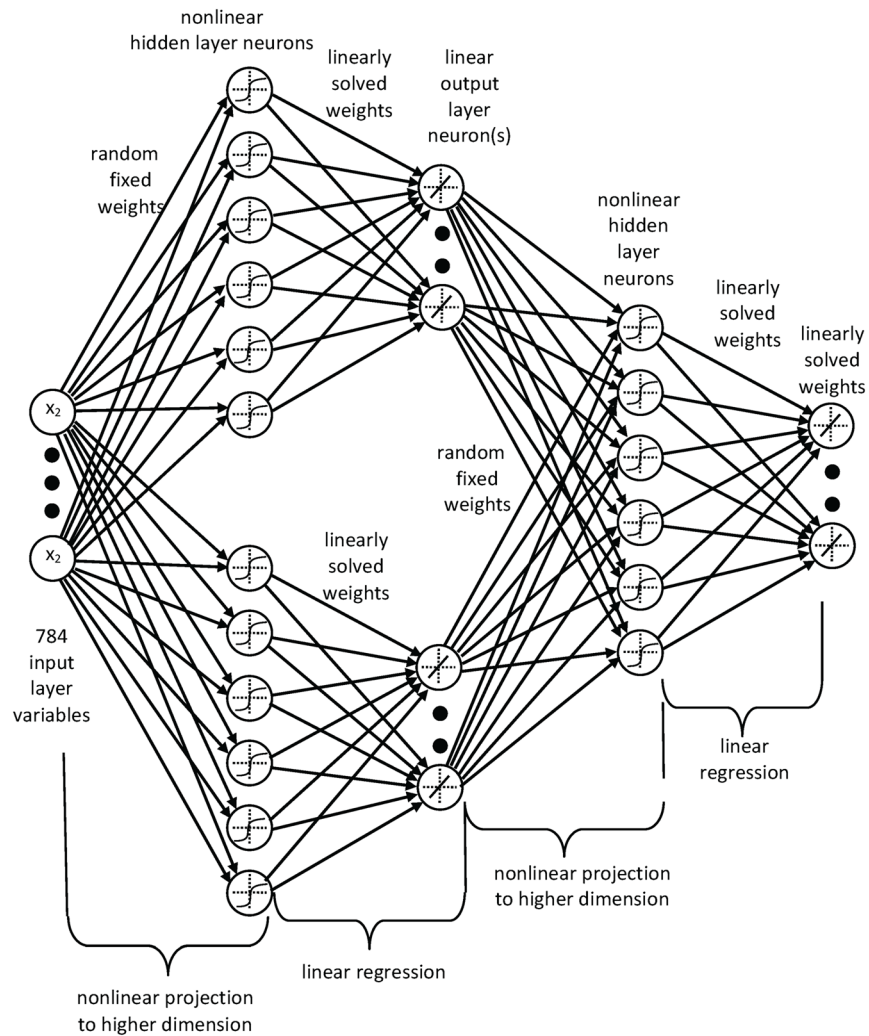
Fig 2. Combined two-layer RF-CIW-ELM and RF-C-ELM network. This figure depicts the structure of our multilayer ELM network that combines a CIW-RF-ELM network with a C-ELM network, using what is effectively an autoencoder output. Note that the middle linear layer of neurons can be removed by combining the output layer weights of the first network with the input layer weights of the second; we have not shown this here, in order to clarify the development of the structure.

Fine Tuning by Backpropagation

In all of the variations of ELM described in this report, the hidden layer weights are not updated or trained according to the output error, and the output layer weights are solved using least squares regression. This considerably reduces the trainability of the network, as the number of free parameters is restricted to the output layer weights, which are generally $\sim 10^4$ in number. It has been argued that any network in which the total number of weights in the output layer is less than the number of training points will likely be enhanced by using backpropagation to train weights in previous layers [26]. Hence, following the example of deep networks, and specific ELM versions of backpropagation [23], we have experimented by using

backpropagation to fine-tune the hidden layer weights. This does re-introduce the possibility of overfitting, but that is a well-understood problem in neural networks and the usual methods for avoiding it will apply here. For simplicity, a batch mode backpropagation was implemented, using the following algorithm. Note that as in Eq (2), we assume a logistic activation function in the hidden layer neurons, for which the derivative can be expressed as $f' = f(1 - f)$.

1. Construct the ELM and solve for the output layer weights $\mathbf{W}_{out}$ as described above.

2. Perform iterative backpropagation as follows:

   a. Compute the error for the whole training set: $\mathbf{E} = \mathbf{Y}_{label} - \mathbf{W}_{out}\mathbf{A}_{train}$.

   b. Calculate the weights update, as derived by [23]: $\Delta\mathbf{W}_{in} = \xi\left[(\mathbf{W}_{out}^\top\mathbf{E}) \circ (\mathbf{A}_{train} - \mathbf{A}_{train} \circ \mathbf{A}_{train})\right]\mathbf{X}_{train}^\top$ where ξ is the learning rate, and $\circ$ indicates the Hadamard product (elementwise matrix multiplication).

   c. Update the weights, $\mathbf{W}_{in} = \mathbf{W}_{in} - \Delta\mathbf{W}_{in}$.

   d. Re-calculate $\mathbf{A}_{train}$ with the new $\mathbf{W}_{in}$.

   e. Re-solve for $\mathbf{W}_{out}$ using least squares regression and continue.

   f. Repeat from step a) for a desired number of iterations or until convergence.

As illustrated in the Results section, this process has shown a robust improvement on all of the SLFN ELM solutions tested here, provided learning rates which maintained stability were used.

Results and discussion for the MNIST benchmark

SLFN with shaped input-weights

We trained ELMs using each of the six input-weight shaping methods described above, as well as a standard ELM with binary bipolar ({−1, 1}) random input weights plus row-normalisation. Following the normalisation of rows of the input weight matrix to unity, we multiplied the values of the entire input weight matrix by a factor of 2, for all seven methods, as this scaling was found to be close to optimal in most cases.

Our results are shown in Fig 3. To obtain an indication of variance resulting from the randomness inherent in each input-weight shaping method, we trained 10 ELMs using each method, and then plotted the ensemble mean as a function of hidden-layer size, M. We also plotted (see markers) the actual error rates for each trained network. It can be seen in Fig 3A that the error rate decreases approximately log-linearly with the log of M for small M, before slowing as M approaches about 10^4. Fig 3B shows the error rate when the actual training data is used. Since our best test results occur when the error rate on the training data is smaller than 0.2%, and we found no significant test error improvement for larger M (see Figs 3C and 3D) we conclude that increasing M further than shown here produces over fitting. This can be verified by cross-validation on the training data.

ELM with shaped input-weights and backpropagation

We also trained networks using each of the methods described in the previous section, plus 10 iterations of ELM-backpropagation using a learning rate of $\xi = 0.001$. As can be inferred from the fact that the training set still has relatively high error rates for most M (Fig 4B), this use of backpropagation is far from optimal, and does not give converged results. As shown in Fig 4A, in comparison with Fig 3A, these 10 iterations at a fixed learning rate still provide a significant
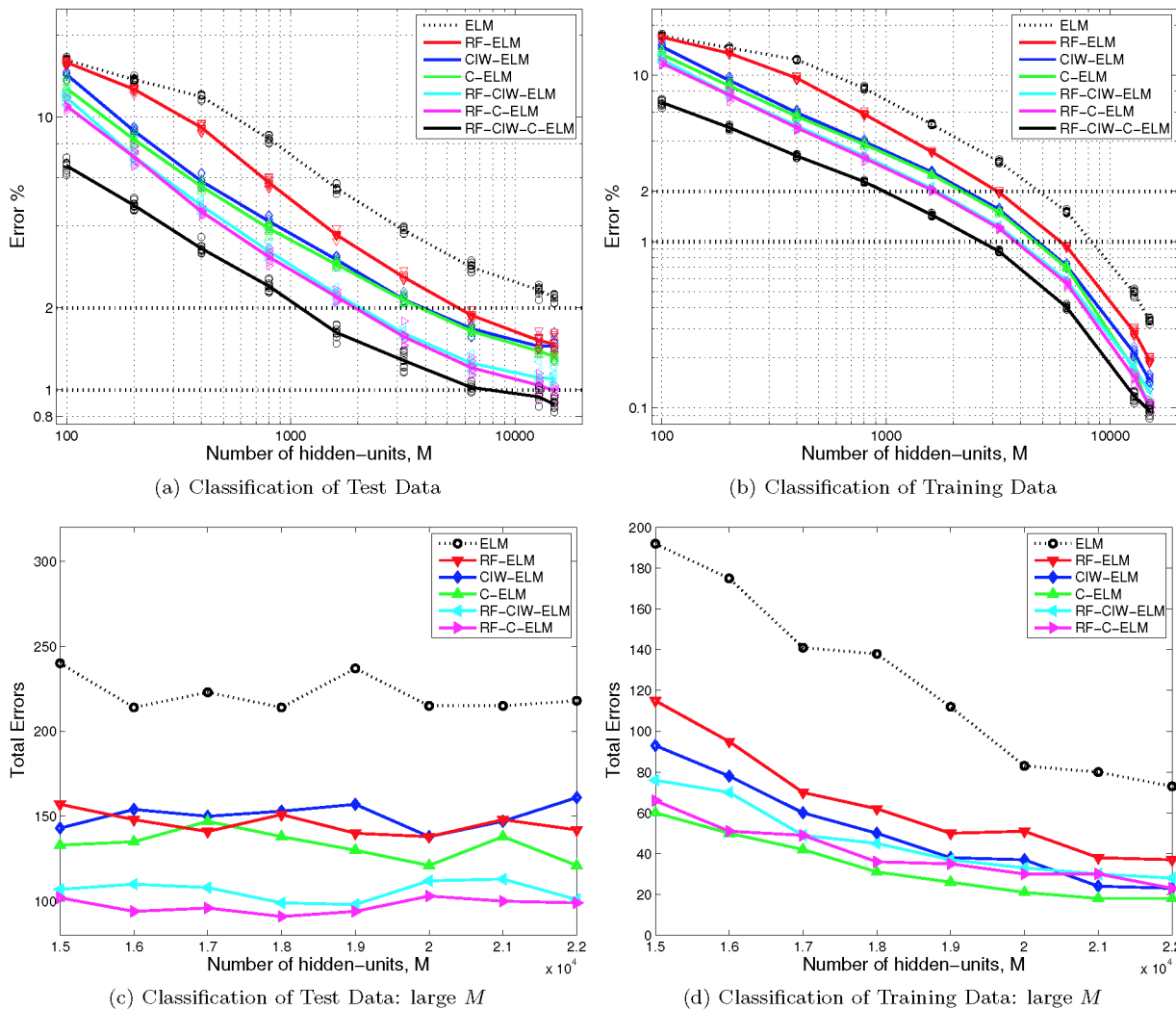
Fig 3. Error rates for MNIST images for various SLFN ELM methods with shaped input weights. The first row shows the mean error percentage from 10 different trained networks applied to classify (a) the 10000-point MNIST test data set, and (b) the 60000-point MNIST training data set used to train the networks, for various different sizes of hidden layer, M. Markers show the actual error percentage from each of the 10 networks. Note that the data for the combination RF-CIW-C-ELM method is plotted against M used in just one of the three parts of the overall network; the total number of hidden units used is actually $2M+500$. Therefore RF-CIW-C-ELM does not outperform the other methods for the same total number of hidden-units for small M. However it can be seen that for large M RF-CIW-C-ELM produces results below 1% error on the test data set and provide the best error rates overall. The second row illustrates that increasing the number of hidden-units above about $M = 15000$ leads to overfitting, since as shown in (c), the total number of errors plateaus, whilst the total number of errors on the training set continues to decrease (shown in (d)). Note that (c) and (d) show results from a single trained network only.

doi:10.1371/journal.pone.0134254.g003

improvement in the error rate for small M. On the other hand, the improvement for $M = 12800$ is minimal, which is not surprising given that the error rate on the training data without using backpropagation for this value of M is already well over 99% (Fig 3B).

It is likely that we can get further enhancements of our error rates by optimising the backpropagation learning rate and increasing the number of iterations used. Moreover, several methods for accelerating convergence when carrying out backpropagation have been described previously [23], and we have not used those methods here. However, the best error rate result reported previously for those methods applied to the MNIST benchmark was 1.45%, achieved
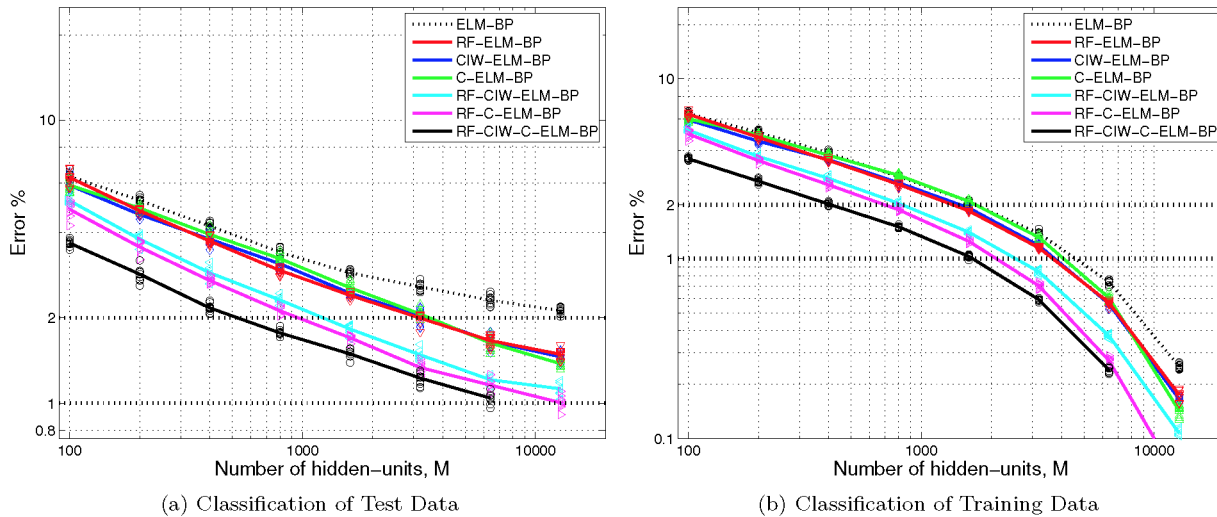
Fig 4. ELM-backpropagation error rates for MNIST for various SLFN ELM methods with shaped input weights. Each trace shows the mean error percentage from 10 different trained networks applied to classify (a) the 10000-point MNIST test data set, and (b) the 60000-point MNIST training data set used to train the networks, for various different sizes of hidden layer, M, when ten iterations of backpropagation were also used. Markers show the actual error percentage from each of the 10 networks. In comparison with Fig 3, it can be seen that backpropagation significantly improves the error rate for small M with all methods, but has little impact when $M = 12800$. The total number of hidden units used for RF-CIW-C-ELM is actually $2M+500$, but each parallel ELM has M hidden-units.

doi:10.1371/journal.pone.0134254.g004

with 2048 hidden units, and the best error rate for the backpropagation method we used is 3.73% [23]. Our results for $M \geq 3200$ hidden units and RF-C-ELM-BP or RF-CIW-ELM-CP match or surpass the former result, despite using the least advanced backpropagation method in prior work [23]. The latter value of 3.73% is easily surpassed by all 7 methods for just 800 hidden units. Moreover, the training time reported in prior work [23] is significantly slower than the time we required when using shaped input weights (see the following section). These outcomes indicate that the use of input-weight shaping prior to backpropagation outperforms backpropagation alone by a large margin, in terms of both error rate and training time.

Runtime efficiency

In many applications, the time required to train a network is not considered as important as the time required for the network to classify new data. However, there do exist applications in which the statistics of training data change rapidly, meaning retraining is required, or deployment of a trained classifier is required very rapidly after data gathering. For example, in financial, sports, or medical data analysis, deployment of a newly trained classifier can be required within minutes of acquiring data, or retraining may be required periodically, e.g. hourly. Hence, we emphasize in this paper the rapidity of training. The speed for testing is negligible in comparison.

The mean training runtime for each of our methods is shown in Fig 5. They were obtained using Matlab running on a Macbook pro with a 3 GHz Intel Core i7 (2 dual cores, for a total of 4 cores), running OS X 10.8.5 with 8 GB of RAM. The times plotted in Fig 5 are the total times for setup and training, excluding time required to load the MNIST data into memory from files. The version of Matlab we used by default exploits all four CPU cores for matrix multiplication and least squares regression. Note that the differences in run time for each method are negligible, which is expected, since the most time-consuming part is the formation of the
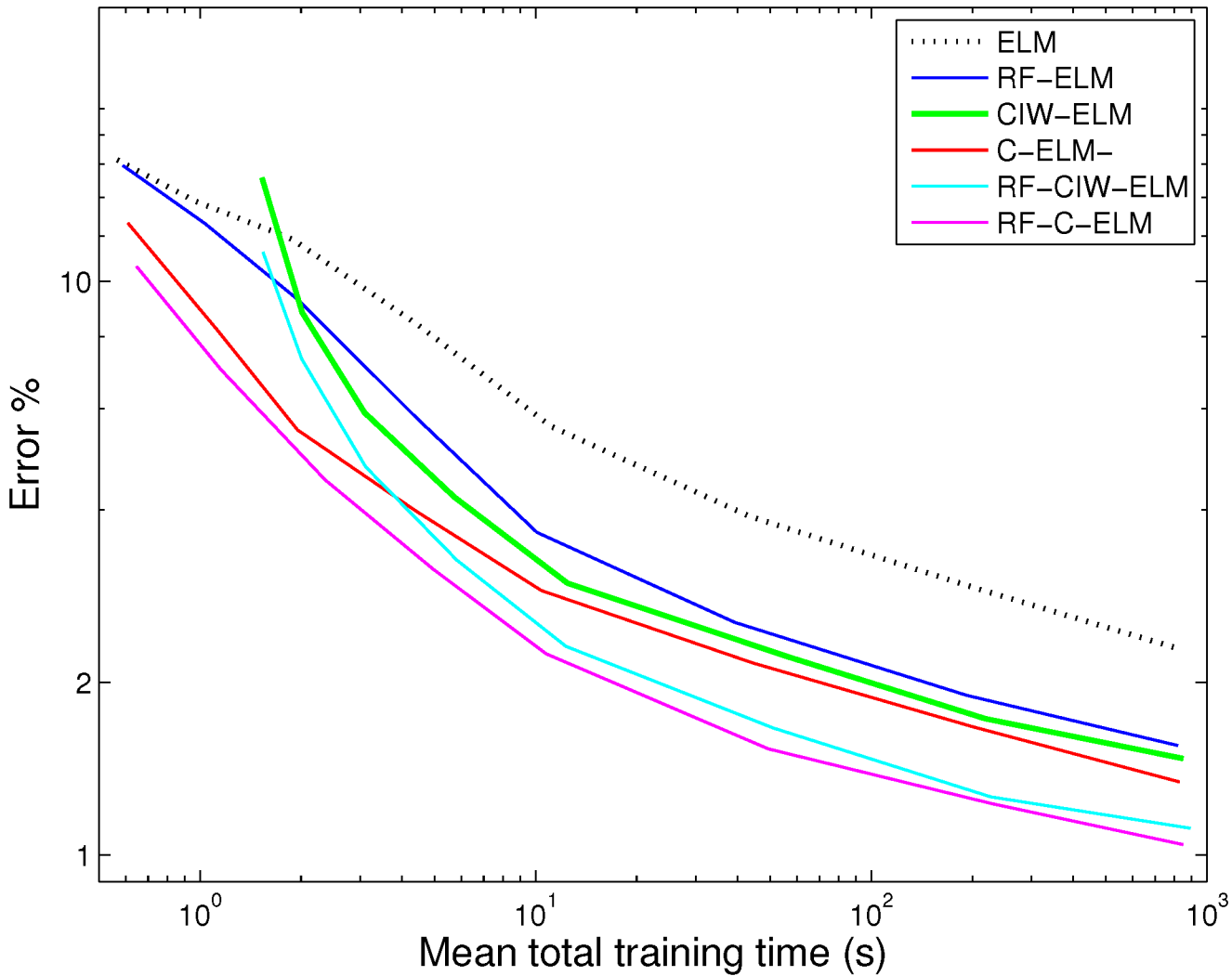
Fig 5. Mean training times for MNIST for various SLFN ELM training methods with shaped input weights. Each trace shows the mean run time from 10 different networks trained on all 60000 MNIST training data points, to achieve the test-date error rates shown in Fig 4. The total time for setup and training are shown, excluding time to load the MNIST data from files. When backpropagation is applied, the runtime scales approximately linearly with the number of iterations, but each backpropagation iteration is slower than each trace shown here, because both input and output weights are updated in each iteration. The time for testing is not included in the figure, but was approximately 10 seconds for $M = 12,800$, and increases only linearly with M.

doi:10.1371/journal.pone.0134254.g005

matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$. The time for testing was not included in the shown data. We found, predictably, that this scaled linearly with M, and was about 10 seconds for $M = 12,800$.

The most important conclusion we draw in terms of runtime is that our best results shown here (for $M = 15000$ hidden units) for RF-CIW-ELM or RF-C-ELM individually take in the order of 15 minutes total runtime and achieve $\sim 99\%$ correct classification on MNIST. In comparison, data tabulated previously for backpropagation shows at least 81 minutes in order to achieve 98% accuracy, and a best result of 98.55% in 98 minutes [23]. In contrast, runtimes reported for the standard ELM algorithm previously (28 seconds for 2048 hidden units [23]) are comparable to ours (12 seconds for 1600 hidden units and less than 1 minute for 3200 hidden units). This illustrates that improving error rate by shaping the input weights as we have done here has substantial benefits for runtime and error rate in comparison to backpropagation.

Distorting and pre-processing the training set

Many other approaches to classifying MNIST handwritten digits improve their error rates by preprocessing it and/or by expanding the size of the training set by applying affine (translation, rotation and shearing), and elastic distortions of the standard set [4, 5, 28]. We have also experimented with distorting the training set to improve on the error rates reported here. For example, with 1 and 2 pixel affine translations, we are able to achieve error rates smaller than 0.8%. When we added random rotations, scalings, shears and elastic distortions, we achieved a best repeatable error rate of 0.62%, and an overall best error rate of 0.57%. However, adding distortions of the training set substantially increases the runtime for two reasons. First, more training points generally requires a larger hidden layer size. For example, when we increase the size of the training set by a factor of 10, we have found we need $M > 20000$ to achieve error rates smaller than 0.7%. This significantly affects the run time through the $O(M^2)$ matrix multiplication required.

At this stage, we have chosen to not systematically continue to improve the way in which we implement distortions to approach state of the art MNIST results, but our preliminary results show that ELM training is capable of using such methods to enhance error rate performance, at the expense of a significant increase in runtime, as is expected in other non-ELM methods.

Results on NORB

We briefly present some results on a second well-known image classification benchmark: the NORB-small database [29]. This database consists of 48600 stereo greyscale images from five classes, and there is a standard set of 24300 stereo images for training, and a standard set of 24300 for testing. Each of the images in the two stereo channels for each sample is of size 96×96 pixels.

Given the large size of each image relative to MNIST images, we preprocessed all images by spatially lowpass filtering using a 9×9 pixel Gaussian kernel, with standard deviation of 4, and then decimating to 13×13 pixel images. We then contrast-normalised each image by subtracting its mean, and dividing by its standard deviation.

Fig 6 shows results for the error rate (ten repeats and the ensemble mean are shown) on the test set from application of the RF-C-ELM method, as the number of hidden units increases. We set the minimum receptive field size to 1, and the ridge regression parameter to $c = 5 \times 10^{-6}$. Our test results peak at close to 95% correct, which is within 3% of state-of-the-art [30], and superior to some results for deep convolutional networks [31].

Computationally efficient methods for ELM training: iterative methods for large training sets

In practice, it is known to be generally computationally more efficient (and avoids other potential problems, such as those described in [32]) to avoid explicit calculation of matrix inverses or pseudo-inverses when solving linear sets of equations. The same principle applies when using the ELM training algorithm, and hence, it is preferable to avoid explicit calculation of the inverse in Eq (5), and instead treat the following as a set of NM linear equations to be solved for NM unknown variables:

$$\mathbf{Y}_{\text{label}}\mathbf{A}_{\text{train}}^{\top} = \mathbf{W}_{\text{out}}\left(\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top} + c\mathbf{I}\right). \tag{6}$$

Fast methods for solving such equations exist, such as the QR decomposition method [18], which we used here. For large M, the memory and computational speed bottleneck in an implementation then becomes the large matrix multiplication, $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$. However, there are
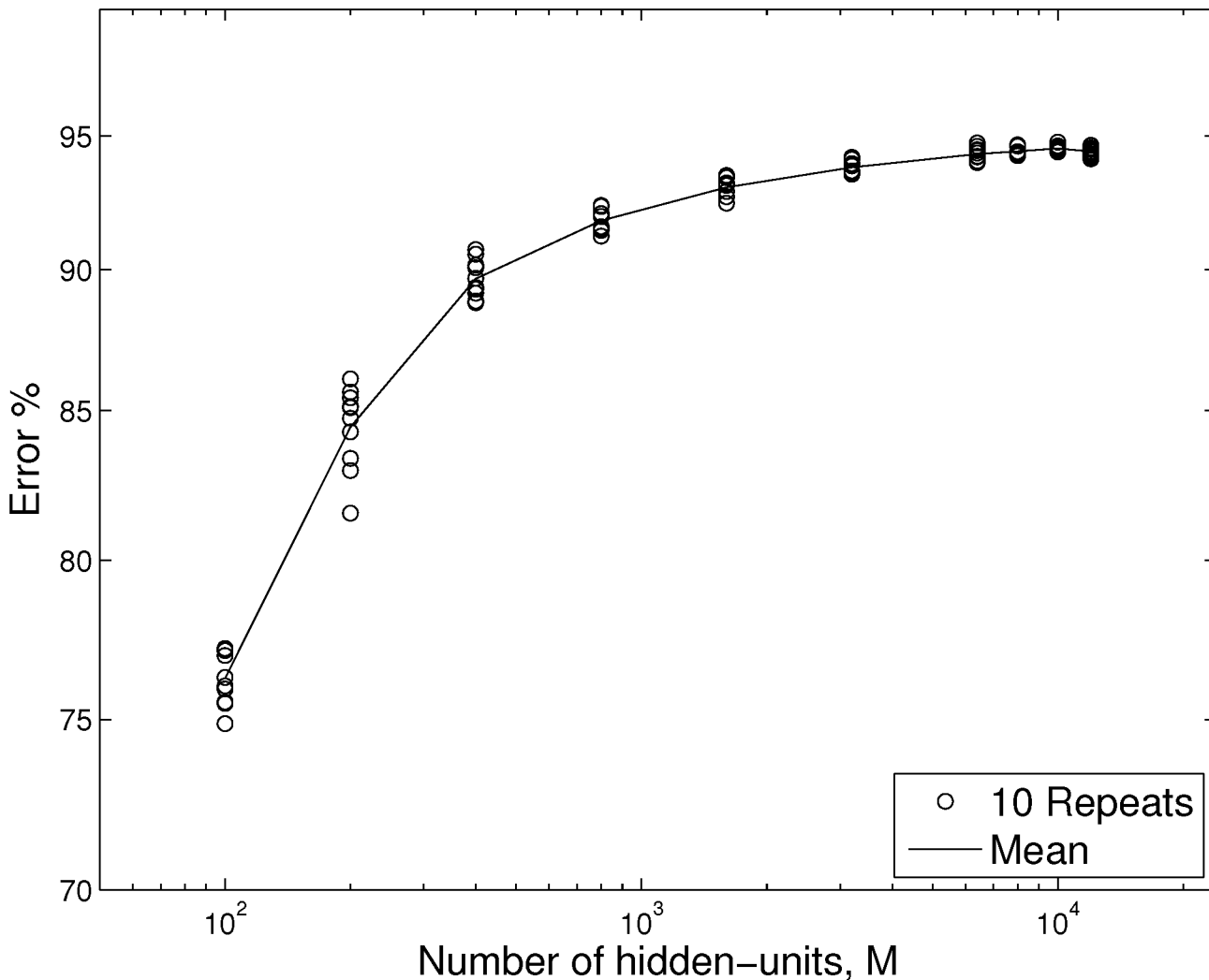
Fig 6. Error rates for NORB-small for RF-C-ELM. The error rate on the 24300 stereo-channel NORB-small test images as a function of the number of hidden-units, M. The data was preprocessed by downsampling each channel of each image to 13×13 pixels, and then contrast normalising. Our best result from all repeats was 94.76%, for $M = 10000$.

simple methods that can still enable solution of Eq (6) when M is too large for this multiplication to be carried out in one calculation.

For example, when solving Eq (6) by implementation in MATLAB, it is computationally efficient to use the overloaded '\' function, which invokes the QR decomposition method. This approach can be used either for the inverse or pseudo-inverse, but we have found it faster to solve (6), which requires the inverse rather than the pseudo-inverse.

Well known software packages such as MATLAB (which we used) automatically exploit multiple CPU cores available in most modern PCs to speed up execution of this algorithm using multithreading. Alternative methods like explicitly calculating the pseudo-inverse, or singular value decomposition, are in comparison significantly (sometimes several orders of magnitude) slower.

When using the linear equation solution method, the main component of training runtime for large hidden-layer sizes becomes the large matrix multiplication required to obtain $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$. There is clearly much potential for speeding up this simple but time-consuming operation, such as by using GPUs or other hardware acceleration methods.

The above text discusses the standard single-batch approach. There are also online and incremental ELM solutions for real-time and streaming operations and large data sets [26, 27, 33, 34]. The use of singular value decomposition offers some additional insight into network structure and further optimization [20]. Here we describe an iterative method that offers advantages in training where the output weight matrix need not be calculated more than once.

One potential drawback of following the standard ELM method of solving for the output weights using all training data in one batch is the large amount of memory that is potentially required. For example, with the MNIST training set of 60000 images, and a hidden layer size of $M = 10000$, the $\mathbf{A}_{\text{train}}$ matrix has 6×10^8 elements, which for double precision representations requires approximately 4.5 GB of RAM. Although this is typically available in modern PCs, the amount of memory required becomes problematic if training data is enhanced by distortions, or if the amount of hidden units needs to be increased significantly.

We have identified a simple solution to this problem, which is as follows. First, we introduce size $M \times 1$ vectors $\mathbf{d}_j$, $j = 1, \ldots, K$, formed from the columns of $\mathbf{A}_{\text{train}}$. Then one of the two key terms in Eq (6) can be expressed as

$$\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top} = \sum_{j=1}^{K} \mathbf{d}_j \mathbf{d}_j^{\top}. \tag{7}$$

That is, the matrix that describes correlations between the activations of each hidden unit is just the sum of the outer products of the hidden-unit activations in response to all training data. Similarly, we can simplify the other key term in Eq (6) by introducing size $N \times 1$ vectors $\mathbf{y}_j$, $j = 1, \ldots K$ to represent the K columns of $\mathbf{Y}_{\text{label}}$ and write

$$\mathbf{Y}_{\text{label}}\mathbf{A}_{\text{train}}^{\top} = \sum_{j=1}^{K} \mathbf{y}_j \mathbf{d}_j^{\top}. \tag{8}$$

In this way, the $M \times M$ matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$ and the $N \times M$ matrix $\mathbf{Y}_{\text{label}}\mathbf{A}_{\text{train}}^{\top}$ can be formed from K training points without need to keep the $\mathbf{A}_{\text{train}}$ matrix in memory, and once these are formed, the least squares solution method applied. The matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$ still requires a large amount of memory ($M = 12000$ requires over 1 GB of RAM), but using this method the number of training points can be greatly expanded and incur only a runtime cost. In practice, rather than form the sum from K training points, it is more efficient to form batches of subsets of training points and then form the sums: the size of the batch is determined by the maximum RAM available.

It is important to emphasise that unlike other iterative methods for training ELMs that update the output weights iteratively [26, 27, 33, 34], the approach described here only iteratively updates $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$.

Conclusions

We have shown here that simple SLFNs are capable of achieving the same accuracy as deep belief networks [2] and convolutional neural networks [1] on one of the canonical benchmark problems in deep learning: image classification. The most accurate networks we consider here use a combination of several non-iterative learning methods to define a projection from the input space to a hidden layer. The hidden layer output is then solved simply using least squares

regression applied to a single batch of all training data to find the weights for a linear output layer. If extremely high accuracy is required, the outputs of one or more of these SLFNs can be combined using a simple autoencoder stage. The maximum accuracy obtained here is comparable with the best published results for the standard MNIST problem, without augmentation of the dataset by preprocessing, warping, noising/denoising or other non-standard modification. The accuracies achieved for the basic SLFN networks are in some cases equal to or higher than those achieved by the best efforts with deep belief networks, for example.

Moreover, when using the receptive field (RF) method to shape inputs weights, the resulting input weight matrix becomes highly sparse: using the RF algorithm above, close to 90% of input weights are exactly zero.

We note also that the implementations here were for the most part carried out on standard desktop PCs and required very little computation in comparison with deep networks. It should be highlighted that we have found significant speed increases for training by avoiding explicit calculation of matrix inverses. Moreover, we have shown that it is possible to circumvent memory difficulties that could arise with large training sets, by iteratively calculating the matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$, and then still only computing the output weights once. This method could also be used in streaming applications: the matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$ could be updated with every training sample, but the output weights only updated periodically. In these ways, we can avoid previously identified potential limitations of the ELM training algorithm, regarding matrix inversion discussed in [32] (see also [26, 35]).

The principles implemented in the ELM training algorithm, and in particular the use of single-batch least squares regression in a linear output layer, following random projection to nonlinear hidden-units, parallel a principled approach to modelling neurobiological function, known as the *neural engineering framework* (NEF) [14]. Recently this framework [14] was utilized in a very large (2.5 million neuron) model of the functioning brain, known as SPAUN [36, 37]. The computational and performance advantages we have demonstrated here could potentially boost the performance of the NEF, as well as, of course, the many other applications of neural networks.

Although deep networks and convolutional networks are now standard for hard problems in image and speech processing, their merits were originally argued almost entirely on the basis of their success in classification problems such as MNIST. The argument was of the form that because no other networks were able to achieve the same accuracy, the unique hierarchy of representation of features offered by deep networks, or the convolutional processing offered by CNNs, must therefore be necessary to achieve these accuracies. However, If there exists a neural network that does not use a hierarchical representation of features, and which can obtain the same accuracy as one that does, then this argument may be a case of confirmation bias. We have shown here that results equivalent to those originally obtained with deep networks and CNNs on MNIST can be obtained with simple single-layer feedforward networks, in which there is only one layer of nonlinear processing; and that these results can be obtained with very quick implementations. While the intuitive elegance of deep networks is hard to deny, and the economy of structure of multilayer networks over single layer networks is proven, we would argue that the speed of training and ease of use of ELM-type single layer networks makes them a pragmatic first choice for many real-world machine learning applications.

Acknowledgments

Author Contributions

Conceived and designed the experiments: MM MT TV AvS JT. Performed the experiments: MM MT JT. Analyzed the data: MM MT JT. Wrote the paper: MM MT TV AvS JT.

References

1. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998; 86:2278–2324. doi: 10.1109/5.726791

2. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural computation. 2006; 18:1527–1554. doi: 10.1162/neco.2006.18.7.1527 PMID: 16764513

3. LeCun Y, Cortes C, Burges CJC. The MNIST database of handwritten digits; Accessed August 2014. http://yann.lecun.com/exdb/mnist/

4. CireŞan D, Meier U, Gambardella LM, Schmidhuber J. Deep, big, simple neural nets for handwritten digit recognition. Neural Computation. 2010; 22:3207–3220. doi: 10.1162/NECO_a_00052 PMID: 20858131

5. CireŞan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. In: Proc. CVPR; 2012. p. 3642–3649.

6. Wan L, Zeiler M, Zhang S, LeCun Y, Fergus R. Regularization of neural networks using DropConnect. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA; JMLR: W&CP volume 28; 2013.

7. Zelier MD, Fergus R. Stochastic pooling for regularization of deep convolutional neural networks. In: In Proc. International Conference on Learning Representations, Scottsdale, USA, 2013; 2013.

8. Goodfellow IJ, Warde-Farley D, Mirza M, Courville A, Bengio Y. Maxout networks. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA; JMLR: W&CP volume 28; 2013.

9. Lee CY, Xie S, Gallagher P, Zhang Z, Tu Z. Deeply-supervised nets. In: Deep Learning and Representation Learning Workshop, NIPS; 2014.

10. Huang GB, Zhu QY, Siew CK. Extreme learning machine: Theory and applications. Neurocomputing. 2006; 70:489–501. doi: 10.1016/j.neucom.2005.12.126

11. Schmidt PF, Kraaijveld MA, Duin RPW. Feed forward neural networks with random weights. In: Proc. 11th IAPR Int. Conf. on Pattern Recognition, Volume II, Conf. B: Pattern Recognition Methodology and Systems (ICPR11, The Hague, Aug.30—Sep.3), IEEE Computer Society Press, Los Alamitos, CA, 1992, 1–4; 1992.

12. Chen CLP. A rapid supervised learning neural network for function interpolation and approximation. IEEE Transactions on Neural Networks. 1996; 7:1220–1230. doi: 10.1109/72.536316 PMID: 18263516

13. Eliasmith C, Anderson CH. Developing and applying a toolkit from a general neurocomputational framework. Neurocomputing. 1999; 26:1013–1018. doi: 10.1016/S0925-2312(99)00098-3

14. Eliasmith C, Anderson CH. Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. MIT Press, Cambridge, MA; 2003.

15. Tapson J, de Chazal P, van Schaik A. Explicit computation of input weights in extreme learning machines. In: Proc. ELM2014 conference, Accepted; 2014. p. arXiv:1406.2889.

16. Nair V, Hinton GE. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML), Haifa, Israel; 2010.

17. Marquardt DW, Snee RD. Ridge Regression in Practice. The American Statistician. 1975; 29:3–20. doi: 10.2307/2683673

18. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C: The Art of Scientific Computing. 2nd ed. Cambridge University Press, Cambridge, UK; 1992.

19. Huang GB, Zhou H, Ding X, Zhang R. Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics. 2012; 42:513–529. doi: 10.1109/TSMCB.2011.2168604

20. Kasun LLC, Zhou H, Huang GB, Vong CM. Representational learning with extreme learning machine for big data. IEEE Intelligent Systems. 2013; 28:31–34.

21. Zhu W, Miao J, Qing L. Constrained extreme learning machines: A study on classification cases; 2015. ArXiv:1501.06115.

22. Coates A, Lee H, Ng AY. An analysis of single-layer networks in unsupervised feature learning. In: Proc.14th International Conference on Artificial Intelligence and Statistics (AISTATS), 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR:W&CP 15; 2011.

23. Yu D, Ding L. Efficient and effective algorithms for training single-hidden-layer neural networks. Pattern Recognition Letters. 2012; 33:554–558. doi: 10.1016/j.patrec.2011.12.002

24. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986; 323:533–536. doi: 10.1038/323533a0

25. Huang GB. An insight into extreme learning machines: Random neurons, random features and kernels. Cognitive Computation. 2014; 6:376–390. doi: 10.1007/s12559-014-9255-2

26. Widrow B, Greenblatt A, Kim Y, Park D. The No-Prop algorithm: A new learning algorithm for multilayer neural networks. Neural Networks. 2013; 37:182–188. doi: 10.1016/j.neunet.2012.09.020 PMID: 23140797

27. Tapson J, van Schaik A. Learning the pseudoinverse solution to network weights. Neural Networks. 2013; 45:94–100. doi: 10.1016/j.neunet.2013.02.008 PMID: 23541926

28. Simard PY, Steinkraus D, Platt JC. Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003); 2003.

29. LeCun Y, Huang FJ, Bottou L. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. In: Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 2; 2004. p. 97–104.

30. CireŞan DCD, Meier U, Masci J, Gambardella LM, Schmidhuber J. Flexible, high performance convolutional neural networks for image classification. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence; 2011. p. 1237–1242.

31. Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y. What is the best multi-stage architecture for object recognition? In: In Proc. IEEE 12th International Conference on Computer Vision; 2009.

32. Widrow B. Reply to the comments on the "No-Prop" algorithm. Neural Networks. 2013; 48:204. doi: 10.1016/j.neunet.2013.11.003 PMID: 24314855

33. Liang NY, Huang GB, Saratchandran P, Sundararajan N. A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Transactions on Neural Networks. 2006; 17:1411–1423. doi: 10.1109/TNN.2006.880583 PMID: 17131657

34. van Schaik A, Tapson J. Online and adaptive pseudoinverse solutions for ELM weights. Neurocomputing. 2015; 149:233–238. doi: 10.1016/j.neucom.2014.01.071

35. Lim MH. Comments on the "No-Prop" algorithm. Neural Networks. 2013; 48:59–60. doi: 10.1016/j.neunet.2013.07.004 PMID: 23945268

36. Eliasmith C, Stewart TC, Choo X, Bekolay T, DeWolf T, Tang C, et al. A Large-Scale Model of the Functioning Brain. Science. 2012; 338:1202–1205. doi: 10.1126/science.1225266 PMID: 23197532

37. Stewart TC, Eliasmith C. Large-scale synthesis of functional spiking neural circuits. Proceedings of the IEEE. 2014; 102:881–898. doi: 10.1109/JPROC.2014.2306061