Fast, simple and accurate handwritten digit classification using extreme learning machines with shaped input-weights

Mark D. McDonnell,[1,*] Migel D. Tissera,[1] André van Schaik,[2] and Jonathan Tapson[2,†]
[1]*Computational and Theoretical Neuroscience Laboratory,*
Institute for Telecommunications Research, University of South Australia, SA 5095, Australia
[2]*Biomedical Engineering and Neuroscience Group,*
The MARCS Institute, The University of Western Sydney, Australia
(Dated: December 30, 2014)

Deep networks have inspired a renaissance in neural network use, and are becoming the default option for difficult tasks on large datasets. In this report we show that published deep network results on the MNIST handwritten digit dataset can straightforwardly be replicated (error rates below 1%, without use of any distortions) with shallow 'Extreme Learning Machine' (ELM) networks, with a very rapid training time ($\sim$10 minutes). When we used distortions of the training set we obtained error rates below 0.6%. To achieve this performance, we introduce several methods for enhancing ELM implementation, which individually and in combination can significantly improve performance, to the point where it is nearly indistinguishable from deep network 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 percent of weights equal to zero, which is a potential advantage for hardware implementations. 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 suggest that the ease of use and accuracy of ELM should cause it to be given greater consideration as an alternative to deep networks applied to more challenging datasets.

I. 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 common cause to say that deep networks are now considered to be a default option for machine learning on large datasets.

The initial excitement about 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.

In this report, we describe some recent results on the MNIST problem achieved with simple variations on the Extreme Learning Machine algorithm [3]. These results are equivalent or superior to the results achieved by CNN and DBN on this problem, and are achieved with significantly lower network and training complexity. While it is fruitless to speculate on what would have happened if these results were available prior to the publication of the CNN and DBN results, they do pose the important question as to whether the ELM architecture should be a more popular choice for this type of problem, and a more commonplace algorithm as a first step in machine learning.

Table I summarises our results, and shows some comparison points with results obtained by other methods in the past.

A. The Extreme Learning Machine: Notation and Training

The Extreme Learning Machine (ELM) as described by Huang in 2005 [3] consists of a single hidden layer feedforward network (SLFN) which has three key departures from conventional SLFNs. These are that the hidden layer is frequently very much larger than either the input or output layers—a typical heuristic is at least ten times larger; the weights from the input to the hidden layer neurons are randomly initialised and are fixed thereafter (i.e., they are not trained); and the output neurons are linear rather than sigmoidal in response, allowing the output weights to be solved by linear regression.

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

[*]Electronic address: mark.mcdonnell@unisa.edu.au
[†]Electronic address: J.Tapson@uws.edu.au

MNIST problem which is equivalent to prior reported results for similarly-sized SLFN networks [1, 4].

We begin by introducing three parameters that define the dimensions of an ELM used as an N-category classifier:

- L: dimension of input vectors
- M: number of hidden layer units
- N: number of distinct labels for training samples

If the ELM is used to classify P test vectors, then it is convenient to form the matrices:

- $\mathbf{X}_{\text{test}}$, of size $L \times P$, is formed by setting each column to contain a single test vector.
- $\mathbf{Y}_{\text{test}}$, of size $N \times P$, numerically represents the prediction vector of the classifier.

In order to map from input vectors to class prediction vectors, 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:

- $\mathbf{a}_{\text{test}} = \mathbf{W}_{\text{in}}\mathbf{X}_{\text{test}}$, of size $M \times P$, contains the linear projections of the input vectors that are inputs to each of the M hidden-units. Optionally, a bias prior to the hidden layer can be included, in which case we can write $\mathbf{a}_{\text{test}} = \mathbf{W}_{\text{in}}\mathbf{X}_{\text{test}} + \mathbf{B}$, where the bias matrix, $\mathbf{B}$ contains the M bias values repeated in each of P columns. Alternatively, we can expand the size of the input dimension from L to $L+1$, and set the additional input element to always be unity for all training and test data, with the bias then contained in W_{in}. Therefore, we leave out $\mathbf{B}$ from all further descriptions.
- $\mathbf{A}_{\text{test}}$, of size $M \times P$, contains the hidden-unit activations that occur due to each training vector, and is given by
$$\mathbf{A}_{\text{test}} = f(\mathbf{a}_{\text{test}}), \tag{1}$$

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

$$(\mathbf{A}_{\text{test}})_{i,j} = f((\mathbf{a}_{\text{test}})_{i,j}) \tag{2}$$
$$= \frac{1}{1+\exp\left(-(\mathbf{a}_{\text{test}})_{i,j}\right)}. \tag{3}$$

Many nonlinear activation functions can be equally effective, such as the rectifier function, the soft plus function, the absolute value function or the quadratic function. Like standard artificial neural networks, the utility of the nonlinearity is that it introduces hidden-unit responses that represent correlations between input elements, rather than simple linear combinations of them.

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

$$\mathbf{Y}_{\text{test}} = \mathbf{W}_{\text{out}} f(\mathbf{W}_{\text{in}}\mathbf{X}_{\text{test}}). \tag{4}$$

We now described 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 $L \times K$, is formed by setting each column to contain a single training vector.
- $\mathbf{Y}_{\text{label}}$, of size $N \times K$, 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.
- $\mathbf{A}_{\text{train}} = f(\mathbf{W}_{\text{in}}\mathbf{X}_{\text{train}})$, of size $M \times K$, contains the hidden-unit activations that occur due to each training vector.
- $\mathbf{Y}_{\text{train}} = \mathbf{W}_{\text{out}}\mathbf{A}_{\text{train}}$, of size $N \times K$, numerically represents the prediction vector of the classifier when the training data is applied to the input.

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

$$\mathbf{Y}_{\text{label}} = \mathbf{W}_{\text{out}}\mathbf{A}_{\text{train}}. \tag{5}$$

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 $N = K$, 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 standard approach in this situation is to seek the solution that minimises the mean square error. This is a standard regression problem for which the solution can be expressed in the form

$$\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{label}}\mathbf{A}_{\text{train}}^{+}, \tag{6}$$

where $\mathbf{A}_{\text{train}}^{+}$ is the size $K \times M$ Moore-Penrose pseudo inverse corresponding to $\mathbf{A}_{\text{train}}$.

This solution is equivalent to finding $\mathbf{W}_{\text{out}}$ that minimises the sum of all entries in the square error between $\mathbf{Y}_{\text{label}}$ and $\mathbf{Y}_{\text{train}}$, i.e., the solution to the optimisation problem

$$\mathbf{W}_{\text{out}}^{*} = \text{argmin}_{\mathbf{W}_{\text{out}}} \sum_{i=1}^{N}\sum_{j=1}^{K}((\mathbf{Y}_{\text{label}})_{i,j} - (\mathbf{Y}_{\text{train}})_{i,j})^2 \tag{7}$$

$$= \text{argmin}_{\mathbf{W}_{\text{out}}} \sum_{i=1}^{N}\sum_{j=1}^{K}((\mathbf{Y}_{\text{label}})_{i,j} - (\mathbf{W}_{\text{out}}\mathbf{A}_{\text{train}})_{i,j})^2. \tag{8}$$

In practice, any numerical convex optimization method should produce good values for the output weights when solving this problem. However, it can also be useful to regularise the problem; we use a standard method known as ridge regression—see [5, 6] for discussion. Using this approach we can write a simple closed form expression for the output weights,

$$\mathbf{W}_{\rm out} = \mathbf{Y}_{\rm label}\mathbf{A}_{\rm train}^\top(\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top + c\mathbf{I})^{-1}, \quad (9)$$

where $\mathbf{I}$ is the $M \times M$ identity matrix, and c can be optimised using cross-validation techniques.

However, it is computationally more efficient (and avoids other potential problems, such as those described in [7]) to avoid explicit calculation of either the pseudo inverse in (6) or the inverse in (9), and instead treat the following as a set of NM linear equations to be solved for NM unknown variables:

$$\mathbf{Y}_{\rm label}\mathbf{A}_{\rm train}^\top = \mathbf{W}_{\rm out}(\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top + c\mathbf{I}). \quad (10)$$

Fast methods for solving such equations exist, such as the QR factorisation method, which we used here. For large M, the memory and computational speed bottleneck in an implementation then becomes the large matrix multiplication, $\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top$.

As discussed now, there are simple methods that can still enable solution of Eqn. (10) when M is too large for this multiplication to be carried out in one calculation.

B. Computationally efficient methods for training an ELM: iterative methods for large training sets

In practice, it is known that it is generally computationally more efficient 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.

For example, when solving Equation (10) by implemenation in MATLAB, it is computationally efficient to use the overloaded '\' function, which invokes the QR factorisation method. This approach can be used either for the inverse or pseudo-inverse, but we have found it faster to solve (10), which requires the inverse rather than the pseudo-inverse.

Well known software packages such as MATLAB (which we used) exploit multiple CPU cores available in most modern PCs to speed up executation 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 run-time for large hidden-layer sizes becomes the large matrix multiplication required to obtain $\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top$. There is clearly much potential for speeding up this simple but time-consuming operation using GPUs or other hardware acceleration methods.

The description of ELM training describes the standard single-batch approach. There are also online and incremental ELM solutions for real-time and streaming operations and large data sets [16, 21–23]. The use of singular value decomposition offers some additional insight into network structure and further optimization [8]. 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 10000, the $\mathbf{A}_{\rm train}$ 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 $\mathbf{A}_{\rm train}$. Then one of the two key terms in Eqn. (10) can be expressed as

$$\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top = \sum_{j=1}^{K} \mathbf{d}_j\mathbf{d}_j^\top. \quad (11)$$

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 Eqn. (10) by introducing size $N \times 1$ vectors $\mathbf{y}_j,\ j = 1, \ldots K$ to represent the K columns of $\mathbf{Y}_{\rm label}$ and write

$$\mathbf{Y}_{\rm label}\mathbf{A}_{\rm train}^\top = \sum_{j=1}^{K} \mathbf{y}_j\mathbf{d}_j^\top. \quad (12)$$

In this way, the $M \times M$ matrix $\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top$ and the $N \times M$ matrix $\mathbf{Y}_{\rm label}\mathbf{A}_{\rm train}^\top$ can be formed from K training points without need to keep the $\mathbf{A}_{\rm train}$ matrix in memory, and once these are formed, the least squares solution method applied. The matrix $\mathbf{A}_{\rm train}\mathbf{A}_{\rm train}^\top$ still requires a large amount of memory ($M = 120000$ 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, rather than face memory limitations. 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 on the machine used to implement the ELM.

It is important to emphasise that unlike other iterative methods for training ELMs that update the output weights iteratively ([16, 21–23]) the approach described here only updates the $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^{\top}$ iteratively.

II. FASTER AND MORE ACCURATE ELMS BY SHAPING THE INPUT WEIGHTS NON-RANDOMLY

In a conventional ELM, the input weights are randomly chosen, typically from a uniform distribution on the interval $[-1, 1]$, or bipolar binary values from $\{-1, 1\}$.

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 [8]. 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 dot products 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.

However, 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. However, it could also make sense to learn features of the data.

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

1. In the first method, we select input layer weights that are random, but biased towards 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) [4].
2. In the second method, we make use of a recently published approach by Zhu *et. al* [9], where 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).
3. In the third method, we restrict the weights for each hidden layer neuron to be receptive only to 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 [10].
4. In the fourth method, we combine RF-ELM with CIW-ELM, or RF-ELM with C-ELM, and show that the combination is superior to any of the three methods individually.
5. Next, we show that passing the results of a RF-CIW-ELM and a RF-C-ELM into a third standard ELM (thus producing a two-layer ELM system) gives the best overall performance of all methods considered in this paper.
6. Finally, we follow the approach of [11] to highlight that the performance of an ELM can be enhanced by application of backpropagation to adjust 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 the best deep networks on the MNIST problem. First, however, we now describe each method in detail.

A. Computed Input Weights for ELM

The CIW-ELM approach is motivated by considering the standard backpropagation algorithm [12]. 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 Figure 1. While it has been argued, not without reason, that it is a strength of ELM that it is not thusly constrained [6], 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 [4]:

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 $M_n \times K_n$.

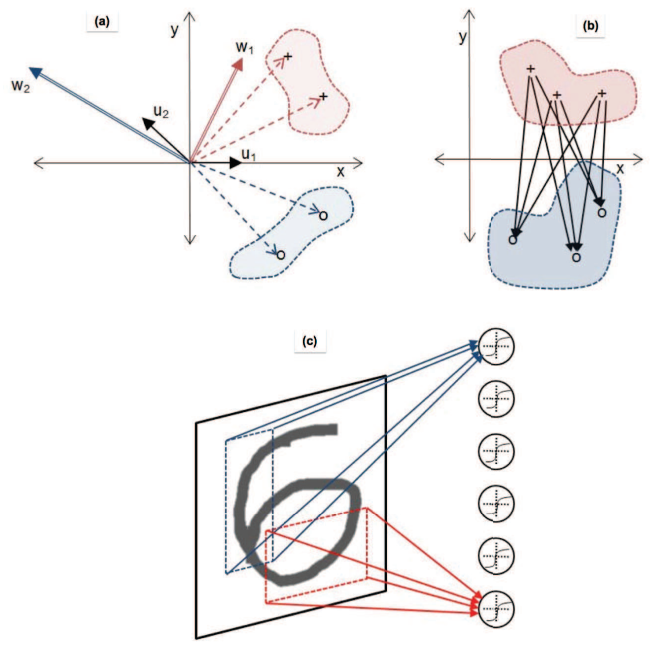


FIG. 1: **Illustration of the three core methods of shaping ELM input weights.** In (a), which is a cartoon of the Computed Input Weights ELM (CIW-ELM) process [4], two classes of input data are indicated by '+' and 'o' symbols. The vectors to the '+' symbols are multiplied by random bipolar binary $\{-1, 1\}$) vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ to produce a biased random weight vector $\mathbf{w}_1$. Similarly the weights to the 'o' class are also multiplied by random vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ to produce a biased random weight vector $\mathbf{w}_2$. Note that in practice we would not use the same random binary vectors. In (b), we show the Constrained ELM (C-ELM) process [9]. 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.

4. Multiply $\mathbf{R}_n$ by the transpose of the input training data set for that class, $X_{\text{train},n}^\top$, 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 $\mathbf{W}_{\text{in}}$.
6. Normalize each row of the input weight matrix, $\mathbf{W}_{\text{in}}$, to unity length.
7. Solve for the output weights of the ELM using standard ELM methods described above.

B. Constrained Weights for ELM

Recently, Zhu *et al.* [9] 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 Figure 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, and largely follows [9], but we found it unnecessary to implement elimination of near-parallel vectors:

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 $\mathbf{W}_{\text{in}}$ to be equal to the difference between each pair of randomly selected training data.
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.

C. Receptive Fields for ELM

We have found that a data-blind 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, such as [10]. 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 Figure 1.

The basic 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 [10] 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.

D. 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, $\mathbf{W}_{\text{in},\mathbf{s}}$.
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) $\mathbf{W}_{\text{in}} = \mathbf{F} \circ \mathbf{W}_{\text{in},\mathbf{s}}$.
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.

E. 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 mulitple-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 Figure 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 I). Note that the middlemost layer shown in Figure 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.

F. Fine Tuning by Backpropagation

In all of the variations of ELM described in this report, the hidden layer weights are learned in an unsupervised mode (in the sense of not being updated or trained according to error), and the output layer weights are solved using a linear regression model. This considerably reduces the trainability of the network, as the capacity in terms of supervised learning is restricted to the output layer weights, which are generally $\sim 10^4$ in number. Given that the capacity, defined as the number of trainable weights, should be able to approach the size of the training data set (60 000 for MNIST), this suggests that the capacity of these networks is less than optimal. However, we are also contending with the diminishing returns offered by a single trainable layer. Following the example of deep networks, and specific ELM versions of backpropagation described by [11], 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 Eqn. (3), 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 $\mathbf{W}_{\text{out}}$ as described in Equations (6)-(10) above.
2. Perform iterative backpropagation as follows:
   (a) Compute the error for the whole training set: $\mathbf{E} = \mathbf{Y}_{\text{label}} - \mathbf{W}_{\text{out}}\mathbf{A}_{\text{train}}$.
   (b) Calculate the weights update, as derived by [11]: $\Delta\mathbf{W}_{\text{in}} = \xi\left[(\mathbf{W}_{\text{out}}^\top\mathbf{E}) \circ (\mathbf{A}_{\text{train}} - \mathbf{A}_{\text{train}} \circ \mathbf{A}_{\text{train}})\right]\mathbf{X}_{\text{train}}^\top$ where ξ is the learning rate, and $\circ$ indicates the Hadamard product (elementwise matrix multiplication).
   (c) Update the weights, $\mathbf{W}_{\text{in}} = \mathbf{W}_{\text{in}} - \Delta\mathbf{W}_{\text{in}}$.
   (d) Re-calculate $\mathbf{A}_{\text{train}}$ with the new $\mathbf{W}_{\text{in}}$.
   (e) Re-solve for $\mathbf{W}_{\text{out}}$ using a linear regression solution, as in (6)-(10), 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.

III. RESULTS AND DISCUSSION FOR THE MNIST BENCHMARK

A. 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 Figure 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 (markers) the actual error rates for each ELM. It can be seen in Figure 3(a) that the error rate decreases approximately log-linearly with the log of M for small M, before slowing as M approaches about 10^4. Figure 3(b) 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 that we found no significant test error improvement for larger M (see Figures 3(c) and 3(d)) we conclude that increasing M further than shown here produces over fitting. This can be verified by cross-validation on the training data.

B. ELM with shaped input-weights and backpropagation

We also trained ELMs using each of the seven methods of 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 (Figure 4(b)), this use of backpropagation is far from optimal, and does not give converged results. However, as shown in Figure 4(a), in comparison with Figure 3(a) these 10 iterations at a fixed learning rate still provide a significant

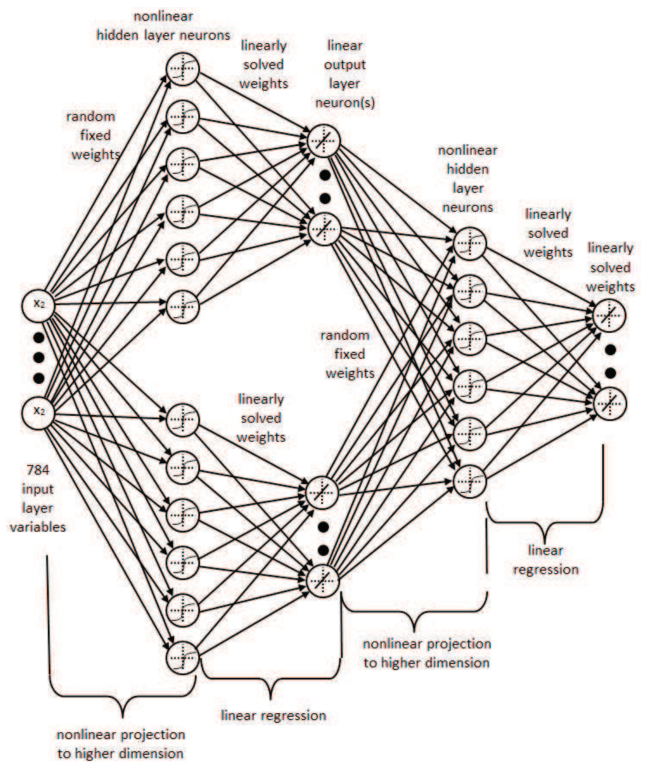


FIG. 2: **Combined two-layer RF-CIW-ELM and RF-C-ELM network.** This figure depicts the structure of our multilayer ELM network that combines a CIW-RF-ELM network with a C-ELM network, using what is effectively an autoencoder output. Note that the middle linear layer of neurons can be removed by combining 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.

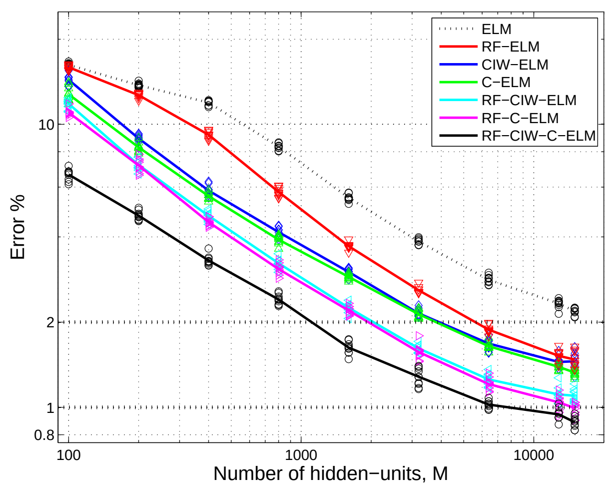
improvement in the error rate for small M. On the other hand, the improvement for $M = 12800$ is minimal, which is not surprising given that the error rate on the training data without using backpropagation for this value of M is already well over 99% (Figure 3(b)).

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 in [11], and we have not used those methods here. However, the best error rate result stated in [11] on the MNIST benchmark was 1.45%, achieved with 2048 hidden units, and the best error rate in [11] for the backpropagation method we used is 3.73% . 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 described in [11]. The latter figure of 3.73% is easily surpassed by all 7 methods for just 800 hidden units. Moreover, the training time reported in [11] 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.

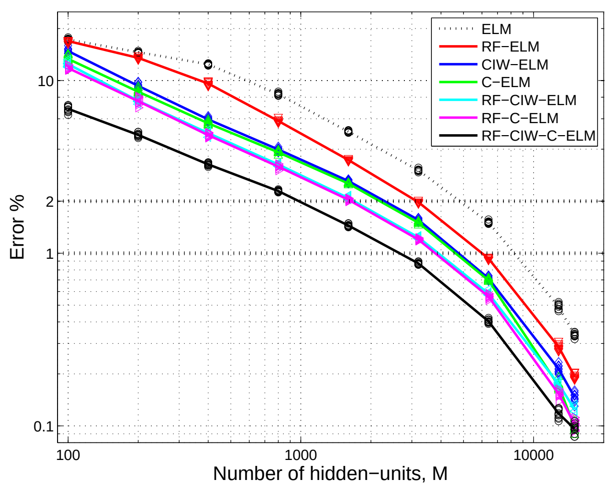
C. Implementation time

The mean runtime for each of our methods is shown in Figure 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 in Figure 5 are the total times for setup, training and testing, 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 matrix $\mathbf{A}_{\text{train}}\mathbf{A}_{\text{train}}^\top$. The solution of the least squares regression problem, perhaps surprisingly, is relatively fast compared to computing the large matrix product. But none of these components depends on the method for setting the input weights, which is why their runtimes all are approximately the same for large M.

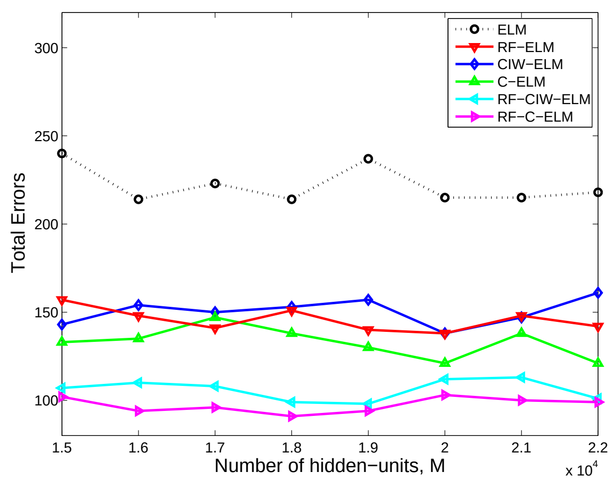
The most important conclusion we draw in terms of runtime is that our best results shown here (for $M = 15000$ hidden units) for RF-CIW-ELM or RF-C-ELM individually take in the order of 15 minutes total runtime and achieve $\sim$99% correct classification on MNIST. In comparison, data tabulated in [11] using backpropagation shows at least 81 minutes in order to achieve 98%



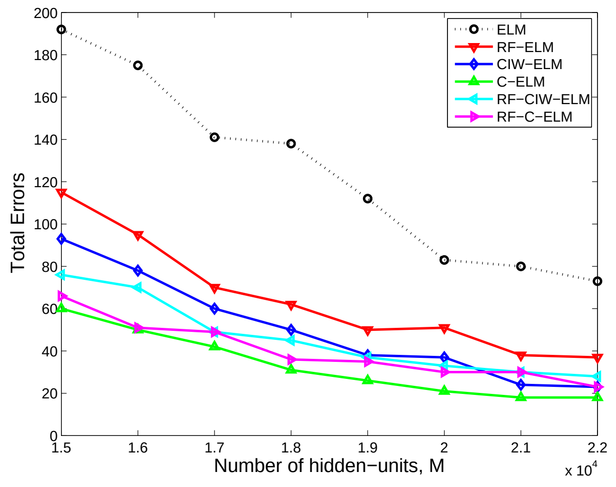
(a) Classification of Test Data



(b) Classification of Training Data



(c) Classification of Test Data: large M



(d) Classification of Training Data: large M

FIG. 3: **Error rates for various SLFN ELM methods with shaped input weights.** The first row shows the mean error percentage from 10 different ELMs applied to classify (a) the 10000-point MNIST test data set, and (b) the 60000-point MNIST training data set used to train the ELMs, for various different sizes of hidden layer, M. Markers show the actual error percentage from each of the 10 ELMs. 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 ELM only.

accuracy, and a best result of 98.55% in 98 minutes. In contrast, the runtimes reported for standard ELM in [11] (28 seconds for 2048 hidden units) 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 implementation time and error rate in comparison to backpropagation.

D. 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 [13–15]. While this is a sensible way to improve the network, the use of non-standard training data invalidates the direct comparison of network performance that we are looking for in this paper. It seems reasonable that any given network's per-

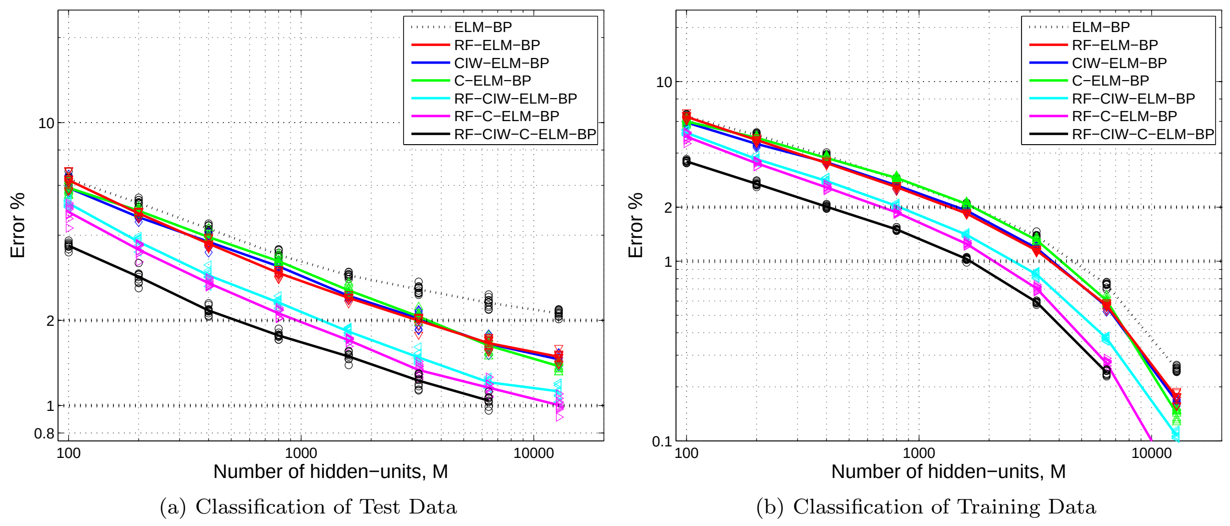


FIG. 4: **ELM-backpropagation error rates for various SLFN ELM methods with shaped input weights.** Each trace shows the mean error percentage from 10 different ELMs applied to classify (a) the 10000-point MNIST test data set, and (b) the 60000-point MNIST training data set used to train the ELMs, for 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 ELMs. In comparison with Figure 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.

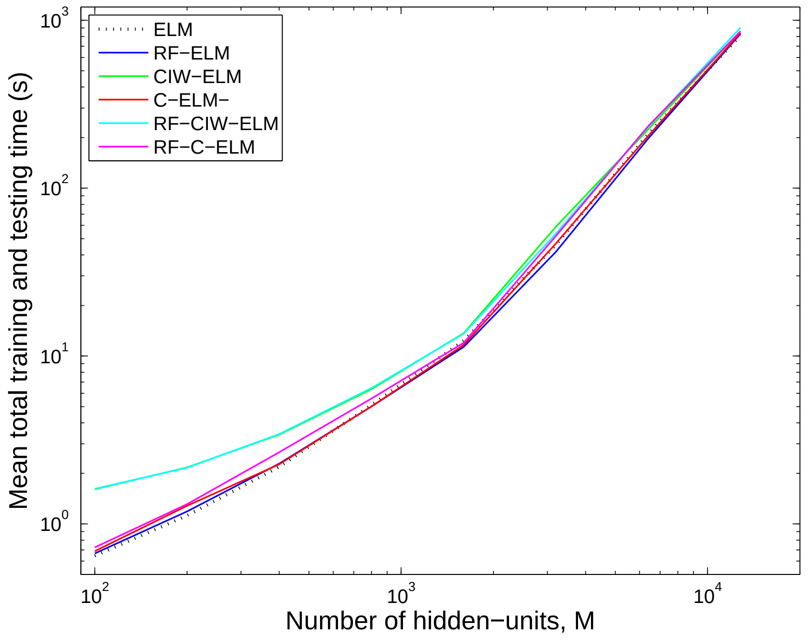


FIG. 5: **Mean implementation times for various SLFN ELM methods with shaped input weights.** Each trace shows the mean run time from 10 different ELMs trained on all 60000 MNIST training data points, and then tested on the 10000 test points, for various different sizes of hidden layer, M. The total time for setup, training and testing 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.

formance on the standard problem is representative of its potential on augmented versions of that problem.

Nevertheless, 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 add random rotations, scaling, shear and elastic distortions, as in [13, 14], we have to date 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 is perfectly capable of using such methods to enhance error rate performance, at the expense of a significant increase in implementation time, as is expected in other non-ELM methods.

IV. CONCLUSIONS

We have shown here that simple SLFNs are capable of achieving the same accuracy as deep belief networks and convolutional neural networks on one of the canonical benchmark problems in deep learning. The most accurate networks we consider here use a combination of several unsupervised learning methods to define a projection from the input space to a very high-dimensional hidden layer. The hidden layer output is then solved simply using linear regression 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 from 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 ELMs regarding matrix inversion discussed in [7] (see also [16, 17]).

The principles used in an ELM, and in particular the use of 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) [18]. Recently this framework [18] was utilized in a very large (2.5 million neuron) model of the functioning brain, known as SPAUN [19, 20]. 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 ELMs.

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 and CIFAR-10. 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. While this is not an unusual mode of argument in the machine learning literature, it may be a case of confirmation bias. 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 does not hold water. We have shown here that results equivalent to those obtained with deep networks and CNNs 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, the current results suggest to us that some more rigorous proof is required that the hierarchical structure will result in greater accuracy, or is in fact necessary at all for this type of machine learning.

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[2] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, pp. 1527–1554, 2006.

[3] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[4] J.Tapson, P. de Chazal, and A. van Schaik, "Explicit computation of input weights in extreme learning machines," in *Proc. ELM2014 conference, Accepted*, 2014, p. arXiv:1406.2889.

[5] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 42, pp. 513–529, 2012.

[6] G.-B. Huang, "An insight into extreme learning machines: Random neurons, random features and kernels," *Cognitive Computation*, vol. In Press, pp. DOI 10.1007/s12 559–014–9255–2, 2014.

[7] B. Widrow, "Reply to the comments on the "No-Prop" algorithm," *Neural Networks*, vol. 48, p. 204, 2013.

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

[9] W. Zhu, J. Miao, and L. Qing, "Constrained extreme learning machine: a novel highly discriminative random feedforward neural network," in *Proc. IJCNN*, 2014, p. XXX.

[10] A. Coates, H. Lee, and A. Y. Ng, "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.

[11] D. Yu and L. Ding, "Efficient and effective algorithms for training single-hidden-layer neural networks," *Pattern Recognition Letters*, vol. 33, pp. 554–558, 2012.

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

[13] P. Y. Simard, D. Steinkraus, and J. C. Platt, "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.

[14] D. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, pp. 3207–3220, 2010.

[15] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. CVPR*, 2012, pp. 3642–3649.

[16] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, "The No-Prop algorithm: A new learning algorithm for multilayer neural networks," *Neural Networks*, vol. 37, pp. 182–188, 2013.

[17] M.-H. Lim, "Comments on the "No-Prop" algorithm," *Neural Networks*, vol. 48, pp. 59–60, 2013.

[18] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press, Cambridge, MA, 2003.

[19] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, C. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, pp. 1202–1205, 2012.

[20] T. C. Stewart and C. Eliasmith, "Large-scale synthesis of functional spiking neural circuits," *Proceedings of the IEEE*, vol. 102, pp. 881–898, 2014.

[21] J. Tapson and A. van Schaik, "Learning the pseudoinverse solution to network weights," *Neural Networks*, vol. 45, pp. 94–100, 2013.

[22] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, pp. 1411–1423, 2006.

[23] A. van Schaik and J.Tapson, "Online and adaptive pseudoinverse solutions for ELM weights," *Neurocomputing*, vol. 149, pp. 233–238, 2015.

Acknowledgement

Mark D. McDonnell's contribution was by supported by an Australian Research Fellowship from the Australian Research Council (project number DP1093425). André van Schaik's contribution was supported by Australian Research Council Discovery Project DP140103001.

Tables

Grouping	Method	Error in testing	Reference
Non-ELM	SLFN, 784-1000-10	4.5%	[1]
	Deep Belief Network	1.25%	[2]
	Convolutional Net LeNet-5	0.95%	[1]
Past ELM	ELM, 784-1000-10 (compares with SLFN above)	6.05%	[4]
	C-ELM	$\sim$5%	[9]
	CIW-ELM, 784-1000-10 (compares with SLFN above)	3.55%	[4]
	ELM, 784-7840-10	2.75%	[21]
	ELM, 784-unknown-10	2.61%	[8]
	CIW-ELM, 784-7000-10	1.52%	[4]
	ELM+backpropagation, 784-2048-10	1.45%	[11]
	Deep ELM, 784-700-15000-10	0.97%	[8]
ELM plus backprop reported here	RF-CIW-ELM + RF-C-ELM parallel, 784-(2$\times$6400)-20-500-10	0.91% (best),1.04% (mean)	This report
SLFN ELM reported here	RF-ELM,784-15000-10	1.36% (best),1.48% (mean)	This report
	CIW-ELM,784-15000-10	1.28% (best), 1.45% (mean)	This report
	C-ELM, 784-15000-10	1.26% (best) 1.33% (mean)	This report
	RF-CIW-ELM , 784-15000-10	1.03% (best), 1.10% (mean)	This report
	RF-C-ELM, 784-15000-10	0.9% (best), 0.99% (mean)	This report
DLFN ELM reported here	RF-CIW-ELM + RF-C-ELM parallel, 784-(2$\times$15000)-20-500-10	0.83% (best), 0.87% (mean)	This report
SLFN ELM reported here	RF-C-ELM, 784-25000-10, Elastic and affine distortions	0.57% (best), 0.62% (mean)	This report

TABLE I: **Comparison of our results on the MNIST data set with published results using other methods. The numbers reported here for the first time indicated as 'mean' are the mean error percentage we obtained from 10 repeats of each method. The numbers indicated as 'best' are from the trained ELM out of the 10 repeats with the best results. Values for each trial shown in Figures 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."**