Neural Networks 45 (2013) 94–100

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2013 Special Issue

Learning the pseudoinverse solution to network weights

J. Tapson *, A. van Schaik

Bioelectronics and Neuroscience Group, The MARCS Institute, University of Western Sydney, Building XB, Kingswood Campus, Kingswood 2751 NSW, Australia

ARTICLE INFO

Keywords:
Moore–Penrose pseudoinverse
Neural engineering
Extreme learning machine
Biological plausibility

ABSTRACT

The last decade has seen the parallel emergence in computational neuroscience and machine learning of neural network structures which spread the input signal randomly to a higher dimensional space; perform a nonlinear activation; and then solve for a regression or classification output by means of a mathematical pseudoinverse operation. In the field of neuromorphic engineering, these methods are increasingly popular for synthesizing biologically plausible neural networks, but the "learning method"—computation of the pseudoinverse by singular value decomposition—is problematic both for biological plausibility and because it is not an online or an adaptive method. We present an online or incremental method of computing the pseudoinverse precisely, which we argue is biologically plausible as a learning method, and which can be made adaptable for non-stationary data streams. The method is significantly more memory-efficient than the conventional computation of pseudoinverses by singular value decomposition.

1. Introduction

One of the core problems for neural network practitioners, in both the neuromorphic and machine learning fields, is the following: given a known input–output relationship, or input and output data sets, how do we synthesize a network that will optimally model the relationship? This synthesis problem is generally divided into two parts—what the architecture or structure of the network should be, and how we should establish the weights for connections between the neurons.

In the last decade we have seen the independent and parallel emergence, in the neuroscientific and machine learning fields, of similar architectures and weight-learning algorithms that neatly and efficiently solve both of these synthesis problems for a large class of relationships. The methodologies, named by their discoverers the *Neural Engineering Framework* (Eliasmith & Anderson, 2003) in computational neuroscience and the *Extreme Learning Machine* (ELM) (Huang, Zhu, & Siew, 2006) in the machine learning field, both synthesize three-layer feedforward networks which are superficially similar to the classic multilayer perceptron—with input, hidden and output layers (the neural engineering approach also makes use of recurrent connections in the hidden layer). What makes these architectures unique is that the input layer signals are connected to an unusually large number of hidden layer neurons, using randomly initialized connection weights. This has the effect of randomly spreading or projecting the inputs from their original input dimensionality to a hidden layer of very much higher dimensionality. It is then possible to find a hyperplane in the higher dimensional space which approximates a desired function regression solution, or represents a classification boundary for the input–output relationship. The output neurons need therefore compute only a linearly weighted sum of the hidden layer values in order to solve the problem. These linear weights can then be determined analytically by computing the product of the pseudoinverse of the hidden layer values with the desired output values. The methodology can be summarized very simply as follows:

1. Connect the input layer of neurons to a much larger hidden layer, using random weights.
2. Analytically solve the output weights (between the large hidden layer and linear output neurons) by calculating the pseudoinverse of the product of the hidden layer activations and the target outputs.

The method works for both continuous-valued and spiking neural networks, where the spiking signals are rate encodings of an underlying variable (Eliasmith & Anderson, 2003). It has the advantages of having no learning parameters, and requiring no learning as such—the full and final solution is obtained with one analytical calculation step.

We will refer broadly to this class of methods as linear solutions to higher dimensional interlayer networks (LSHDI). The ELM form of the method has been rapidly and widely adopted in the machine learning field because it offers, in a single forward computational step, a least-squares optimal, non-parameterized solution to the problem of learning network weights; and it is computationally equivalent to (but much faster to train than) the widely used support vector machine (SVM—see below). It is very quick to

* Corresponding author. Tel.: +61 2 4736 0238; fax: +61 2 4736 0833.
E-mail address: j.tapson@uws.edu.au (J. Tapson).

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.neunet.2013.02.008

compute, and avoids the problems of stability and convergence on local minima which plague the users of error backpropagation in multilayer networks, for example. A similar method has also been found to work well as a preprocessing filter for feature recognition using convolutional pooling architectures (Saxe et al., 2011).

The first step (randomization) in the LSHDI method performs the same function as the "kernel trick" used in most kernel methods, with the hidden layer encoding the feature vectors. Cho and Saul (2010) have shown that deep architecture (multilayer) neural networks, and specifically networks with a single hidden layer, can be recast as SVMs if appropriate kernels are used. Similarly, Rahimi and Recht (2009) noted that "randomization is computationally cheaper than optimization", and applied randomized kernels to classical networks—a process they described as "Weighted Sums of Random Kitchen Sinks". It should be noted that in these cases the randomization was not in the weights, but in the kernel functions; and that the resulting networks are not LSHDI (as described here) for any cost functions other than linear least-square error. This raises the issue of whether the LSHDI solution hyperplane is optimal in terms of maximizing the margin between classes (in a classifier application) as would be the case in a traditional SVM. Barak and Rigotti (2011) have shown that the pseudoinverse solution to the weight problem indeed converges to the optimal margin solution as the size of the margin increases, and that it approaches optimality for relatively low values of the margin, which suggests that at least in some cases it would be optimal.

In bio-inspired and neuromorphic networks, the LSHDI method solves a significant modeling problem, which is that it has been extremely difficult to construct simulated biological neural networks to model specific relationships—while the neuronal elements of these networks are well defined, there has been no widely applicable method to synthesize a network to solve a given problem. In one form (the *Neural Engineering Framework*, or NEF) the method is emerging as the core of a generic compiler for silicon neural systems (Choudhary et al., 2012; Galluppi, Davies, Furber, Stewart, & Eliasmith, 2012). These silicon neural systems have reached the point where they are not limited by the number or complexity of the neurons or interconnects on the hardware platform, but by the ease with which they can be programmed. The NEF approach has been enthusiastically adopted by neuromorphic engineers working on silicon-based neural networks.

Significantly, we are starting to see some evidence from neurophysiology that structures embodying the LSHDI principle may exist in the brain. For example, recent work by Rigotti, Ben Dayan Rubin, Wang, and Fusi (2010) in modeling recorded cortical activity in monkeys performing context-sensitive tasks shows that complex rule-based tasks require both sensory stimuli and internal representation of states; and that a significant number of random connections placed between input sources and a hidden interlayer, and random recurrent connections between interlayer neurons, are necessary for optimal performance. They describe these interlayer neurons as generating *mixed selectivity*, which is equivalent to increasing the dimensionality of the state representation.

An obstruction to acceptance of the LSHDI method as being biologically plausible is the necessity to compute the pseudoinverse of a matrix in order to synthesize the network. The Moore–Penrose pseudoinverse is a relatively recent addition to linear algebra (Penrose, 1955), and is usually computed using singular value decomposition (SVD). It seems intrinsically unlikely that we would find in the cortex a plausible equivalent to this mathematically complex process.

In this paper we demonstrate an LSHDI algorithm which incrementally learns the pseudoinverse solution to the weights problem, in the context of online presentation of new training input, and we show that it is plausible as a physiological process in real neurons. The solution to the network weights is exactly the same as that which would be calculated by singular value decomposition. It converges with a single forward iteration per input data sample, and as such is ideal for real-time online computation of the pseudoinverse solution. It requires significantly less memory than the SVD method, as its memory requirement scales as the square of the size of the hidden layer, whereas the SVD memory requirement scales with the product of the hidden layer size and the size of the training data set. Given that the data set size should significantly exceed the hidden layer size, the former is advantageous. We call the algorithm OPIUM (Online PseudoInverse Update Method).

OPIUM is adapted from an iterative method for computing the pseudoinverse, known as Greville's method (Greville, 1960). The existence of OPIUM, and its biological plausibility, suggests that there is no reason why a biological neural network would not arrive at the same weights that are computed using a singular value decomposition, and therefore that this method of synthesizing network structures may be used without the fear that it is biologically implausible.

In addition, we show for the first time that a single modification to this algorithm allows it to adapt over time to changes in the modeled relationship. Given that in the real world, input–output relationships are seldom stationary processes, an adaptive network is considerably more robust (and plausible) than one that finds a static solution.

Pseudoinverse methods were widely used in an earlier era of neural network research, to the extent that a significant class of Kohonen-type linear associative memories were known as pseudoinverse neural networks (PINNs) (Kohonen, 1988, 1989). The key variations in the LSHDI use of the method are in the initial spreading to higher dimension, and the associated nonlinear activation performed on the higher dimension signals. We are not aware of any prior recasting of the pseudoinverse method into a biologically plausible framework.

There have been a number of neural network methods described in which variations of gradient descent or delta learning have been used to achieve mathematically interesting results; for example, Oja and colleagues have shown that neural networks can perform Principal Component Analysis (PCA) and Independent Component Analysis (ICA) with variations on a Hebbian learning rule (Oja, 1989). Similarly, Eliasmith and colleagues have shown that Hebbian learning rules may be used to successfully generate weights when modeling biologically realistic networks using the Neural Engineering Framework (MacNeil & Eliasmith, 2011; Stewart, Bekolay, & Eliasmith, 2012). We note that conventional backpropagation, applied only to the output layer, should in principle converge to the same solution as the pseudoinverse, given that it is also minimizing the least-square error; but that it is unlikely to converge optimally for a single presentation of each input–output pair, as is the case for the OPIUM method.

We begin by describing OPIUM, demonstrate it in practice, and then motivate its biological plausibility.

2. Mathematical foundations

The structure of LSHDI networks can be generalized from that of a standard multilayer perceptron—see for example Bishop (2008). If we consider a particular sample of input data to be $x_t \in \mathbb{R}^{L\times 1}$ where t is a time or series index, then forward propagation of the local signals through the network can be described by:

$$y_{n,t} = \sum_{j=1}^{M} w_{nj}^{(2)}\, g\left(\sum_{i=1}^{L} w_{ji}^{(1)} x_{i,t}\right), \tag{1}$$

where $y_t \in \mathbb{R}^{N\times 1}$ is the output layer vector corresponding to input x_t, and each element $y_{n,t}$ is a linear sum of the M hidden

layer outputs weighted by $w_{nj}^{(2)}$. n is the output vector index, j the hidden layer index, and i the input vector index. The hidden layer outputs depend on the neuron's activation function $g()$ and the randomly determined interconnecting weights $w_{ji}^{(1)}$ between input and hidden layer. (Weights are usually uniformly distributed in some appropriate range.) The superscripts on the weight matrices indicate the layer. In this representation all the weights are static. The number of hidden layer neurons, M, is deliberately chosen to be large compared to the size of the input layer, which is denoted L; for example, $M > 10L$ is considered the norm in the ELM literature.

The random selection of static input weights $w_{ji}^{(1)}$ reduces the network training requirement to be the optimization of the output weights $w_{nj}^{(2)}$, and the linear output summation reduces this problem to one of linear regression. This is why the ELM has found such favor with practitioners; training consists of a single parameter-less analytical calculation, rather than some type of gradient descent with backpropagation, or a stochastic learning algorithm.

The weight optimization problem can be stated as follows: given a series of hidden layer outputs

$$a_{j,t} = g\left(\sum_{i=1}^{L} w_{ji}^{(1)} x_{i,t}\right), \tag{2}$$

we can form a matrix $A = [a_1 \cdots a_k]$ where $A \in \mathbb{R}^{M\times k}$ in which each column contains the output of the hidden layer at one instant in the series, with the last column containing the most recent instant in the series; and then given a similar matrix $Y \in \mathbb{R}^{N\times k}$ consisting of the corresponding output values $Y = [y_1 \cdots y_k]$: what set of weights $W \in \mathbb{R}^{N\times M}$ will minimize the error in:

$$WA = Y? \tag{3}$$

This may be solved analytically by taking the Moore–Penrose pseudoinverse $A^+ \in \mathbb{R}^{k\times M}$ of A:

$$W = YA^+. \tag{4}$$

When A and Y are explicitly and exhaustively known (static) data sets, then a singular value decomposition suffices to determine A^+. In this paper, we show that A^+ may be learnt incrementally in a real-world or real-time application as new data becomes available; and with a modification, it may be made adaptable for non-stationary data sets.

There are a number of methods for incremental calculation of the pseudoinverse solution to this problem (Greville, 1960; Kohonen, 1989) and we are aware of an existing method by Huang, Liang, Rong, Saratchandran, and Sundararajan (2005) for incremental solution of the ELM. (We note however that current online or incremental methods focus on learning through the addition of further neurons, whereas this research focuses on learning by online recalculation of network weights.) We have selected one incremental solution method, Greville's algorithm, for use in this work.

Greville's original method for calculating the pseudoinverse (Greville, 1960) is well known, and has been used by several groups (e.g. Guo & Lyu, 2003) to synthesize neural networks in which extra hidden layer neurons are added incrementally. This approach does not lend itself to applications in which the network structure is static and the number of hidden layer neurons is fixed. Inspired by a variation presented by Ben-Israel and Greville (2003, p. 238) we adapt the Greville method as follows. Given the input and output data streams a and y respectively for some process sampled k times, when the next set of data a_k, y_k becomes available, we form two vectors partitioned as follows (for real-valued data):

$$A_k = [A_{k-1}\ a_k], \tag{5}$$

$$Y_k = [Y_{k-1}\ y_k]. \tag{6}$$

We want to calculate W_k given W_{k-1} where

$$W_k = Y_k {A_k}^+, \qquad W_{k-1} = Y_{k-1} {A_{k-1}}^+. \tag{7}$$

According to Greville's theorem, the pseudoinverse A^+ is given by:

$${A_k}^+ = [A_{k-1}\ a_k]^+ = \begin{bmatrix} {A_{k-1}}^+ (I - a_k b_k^T) \\ b_k^T \end{bmatrix}, \tag{8}$$

where $b_k \in \mathbb{R}^{Mx1}$ is given by:

$$b_k = \frac{\left({A_{k-1}}^+\right)^T {A_{k-1}}^+ a_k}{1 + {a_k}^T \left({A_{k-1}}^+\right)^T {A_{k-1}}^+ a_k}, \tag{9}$$

if

$$\left(I - A_{k-1}{A_{k-1}}^+\right) a_k = 0. \tag{10}$$

If the condition of Eq. (10) is not met, a different expression for b_k applies. The condition of Eq. (10) implies that a_k is in the column space of A_{k-1}. When the number of columns in A_{k-1} is larger than the dimension of a_k, i.e. $k > M$, then the vectors a_k are linearly dependent and the condition of Eq. (10) will hold. This will typically be the case in the online regression defined in Eq. (3), after appropriate initialization. We will return to the problem of initialization later in this section.

We can define a symmetric square matrix $\theta_{k-1} \in \mathbb{R}^{M\times M}$:

$$\theta_{k-1} = \left({A_{k-1}}^+\right)^T {A_{k-1}}^+ = \left(A_{k-1}{A_{k-1}}^T\right)^+, \tag{11}$$

and simplify the expression for b_k:

$$b_k = \frac{\theta_{k-1} a_k}{1 + {a_k}^T \theta_{k-1} a_k}, \tag{12}$$

θ_k is the pseudoinverse of the correlation matrix of the hidden layer activation. To update θ_k when a new pair a_k, y_k is observed, we can write:

$$\theta_k = \left(A_k {A_k}^T\right)^+ = \left(A_{k-1}{A_{k-1}}^T + a_k {a_k}^T\right)^+. \tag{13}$$

Kovanic (1979) shows that when (10) holds the matrix inversion lemma may be applied to (13) to obtain an update rule for θ_k:

$$\theta_k = \theta_{k-1} - \frac{\theta_{k-1} a_k {a_k}^T \theta_{k-1}^T}{1 + {a_k}^T \theta_{k-1} a_k} = \theta_{k-1} - \theta_{k-1} a_k {b_k}^T, \tag{14}$$

which can be readily computed from θ_{k-1} and a_k.

To update the weights, we use:

$$\begin{aligned} W_k &= Y_k {A_k}^+ = [Y_{k-1} y_k] \begin{bmatrix} {A_{k-1}}^+ (I - a_k b_k^T) \\ b_k^T \end{bmatrix} \\ &= Y_{k-1}{A_{k-1}}^+ - Y_{k-1}{A_{k-1}}^+ a_k b_k^T + y_k b_k^T \\ &= Y_{k-1}{A_{k-1}}^+ + (y_k - Y_{k-1}{A_{k-1}}^+ a_k) b_k^T \\ &= W_{k-1} + (y_k - W_{k-1} a_k) b_k^T, \end{aligned} \tag{15}$$

which can be readily computed from y_k, a_k, W_{k-1}, and θ_{k-1}. In this equation, $(y_k - W_{k-1}a_k)$ represents the error in the mapping from a_k to y_k using the weight matrix W_{k-1} from the previous step. This error is weighted by $b_k = \left(1 + {a_k}^T \theta_{k-1} a_k\right)^{-1} \theta_{k-1} a_k$ to update W_{k-1}. From this we see that the hidden layer activation a_k is spread through θ_{k-1} (the inverse of the hidden layer correlation matrix) normalized to $\left(1 + {a_k}^T \theta_{k-1} a_k\right)$, to calculate the error weighting. The spreading of a_k through θ_{k-1} serves to distributes the energy of the hidden layer activation inversely proportional to the correlation in past hidden layer activations.

In order for Eqs. (14) and (15) to be valid, condition (10) must hold, which it will once the number of a_k, y_k pairs significantly exceeds the dimension M of a_k. However, this does not hold if we

start A_0 and Y_0 as empty matrices. In that case, Greville's alternative expression for b_k should be used whenever (10) does not hold. Unfortunately, given noise in the input vectors in any real world scenario, testing for (10) is not straightforward, as it is vanishingly likely to hold. An alternative possibility for initialization, which we use here, is to modify the problem slightly and ask for W_k to not only map a_k to y_k, but also to map arbitrarily small individual activations of hidden layer neurons ($\in I^{M\times M}$) to the zero matrix $O^{N\times M}$. This gives us:

$$A_0 = \in I^{M\times M}, \tag{16}$$

$$Y_0 = O^{N\times M}, \tag{17}$$

$$\theta = \frac{1}{\in^2} I^{M\times M}. \tag{18}$$

With this initialization, A_0 is of rank M and the new vector a_1 will thus be in the column space of A_0, so that condition (10) is met.

We have so far assumed that there is a static relation between the input vectors and output vectors. For non-stationary data, condition (10) will not always hold. We could use Greville's alternative expression for b_k, but we again run into the problem that we cannot reliably test (10). Instead, we have implemented a modification of (14) that pushes θ_k towards its initial value θ_0 whenever the error $e_k = (y_k - W_{k-1}a_k)$ is high, i.e., when the model is no longer accurate.

$$\theta_k = \gamma\left(\theta_{k-1} - \theta_{k-1}a_k{b_k}^T + I\left(\frac{1-e^{-|E|}}{\in}\right)\right), \tag{19}$$

where $|E|$ is the L_2 norm of the error vector e_k, normalized to activation levels:

$$E = \frac{{e_k}^T e_k/M}{1 + {a_k}^T\theta_{k-1}a_k}, \tag{20}$$

and γ is a normalization factor given by:

$$\gamma = \frac{1}{1 + \left(\frac{1-e^{-|E|}}{\in}\right)}. \tag{21}$$

The change in the calculation of θ_k has the consequence that more recent inputs have a higher effect on the weights than earlier inputs, as in the general class of infinite-impulse response (IIR) filters. Similarly, it shares with IIR filters the potential for instability in recursion, although we have found this to be very uncommon.

We note that the stationary solution produces exactly the same pseudoinverse solution that would be computed using batch processing of data by means of the singular value decomposition (SVD). Obviously, the adaptive solution for non-stationary data is no longer a mathematically exact pseudoinverse solution to Eq. (3) for the complete data sets A and Y. It is, however, much more useful for non-stationary real-world processes, as will be seen in the examples.

3. Applications of the method: handwritten digits

Recognition or classification of handwritten digits is a standard machine learning problem, and in the form of the MNIST database LeCun and Cortes (2012) it has become a benchmark problem. For the LSHDI methods, this problem represents a massive computational burden: the digits are 28 × 28 pixels, and the standard learning set contains 60 000 sample digits. Given a standard rule-of-thumb "fan-out" of 10 hidden-layer neurons per input, this would require a network of 784 input neurons, 7840 hidden-layer neurons, and the pseudoinversion of a 7840 × 60 000 matrix. At ∼5 × 10^9 elements, this significantly

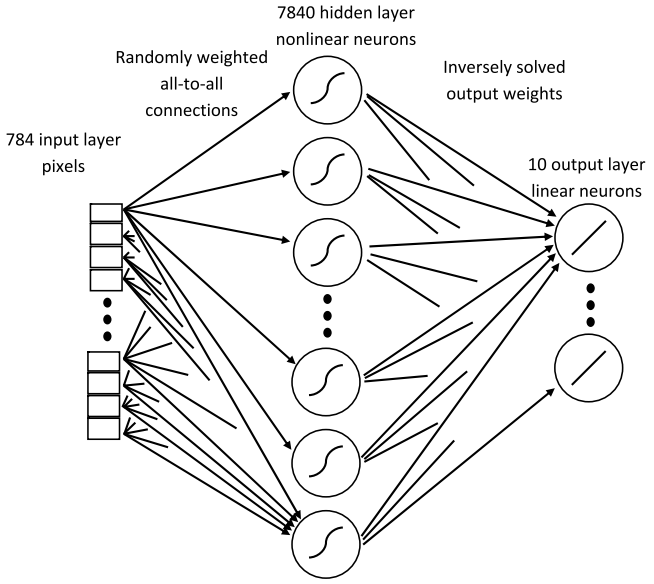

Fig. 1. Structure of an LSHDI network as used to classify the MNIST handwritten digit database. The inputs are the pixels; they are connected to a higher-dimensional interlayer of nonlinear neurons, using randomly weighted connections. The output layer consists of linear neurons and the output layer weights are solved analytically using the pseudoinverse operation.

exceeds the size of matrix that can be inverted without incurring significant memory issues on most computational platforms (particularly given that the SVD method requires solution of three matrices of approximately this size); in the absence of several hundred Gb of RAM it would require extensive disk read/write operations which would impact severely on computation time. This massive pseudoinversion could in principle be avoided by first calculating the full autocorrelation matrix, and the input–output cross-covariance matrix, and multiplying them together before performing a 7840 × 7840 element SVD, but the burden of computing these matrices (including a 7840 × 60 000 cross-correlation) would not be trivial. By using OPIUM, we can perform the LSHDI classification of this data set in RAM on a standard notebook computer. We are only aware of one prior effort to use the standard LSHDI method (in ELM form) on the MNIST classification problem and the computational resources of Microsoft Research in Redmond, Washington were apparently necessary for success in that case (Huang, 2012).

Using the structure shown in Fig. 1, with random weights in the input layer uniformly distributed between −0.5 and 0.5, the typical test error on the raw MNIST data set is 2.75%, which compares favorably with most results from 2-layer feedforward networks on this problem (LeCun, Bottou, Bengio, & Haffner, 1998). Given that the structure has been applied absolutely naïvely to this problem – in fact, in the same way as it would be applied to any generic problem with similar input–output pairs – the result reinforces the usefulness of this technique. We note that apart from the number of hidden-layer neurons, there are no parameters to be tuned and there is no learning process except for the single pseudoinversion operation. The nonlinear hidden-layer neurons used the logistic function to produce sigmoidal activation.

The progression of online training is also illustrated in the MNIST database example. We can present the training data one at a time, and update the linear solution with each presentation. The progressive improvement for this method is illustrated in Fig. 2. This shows networks of three sizes, in which training data consisting of single digits randomly selected from the test set, were presented one at a time and the new solution computed after each new input.

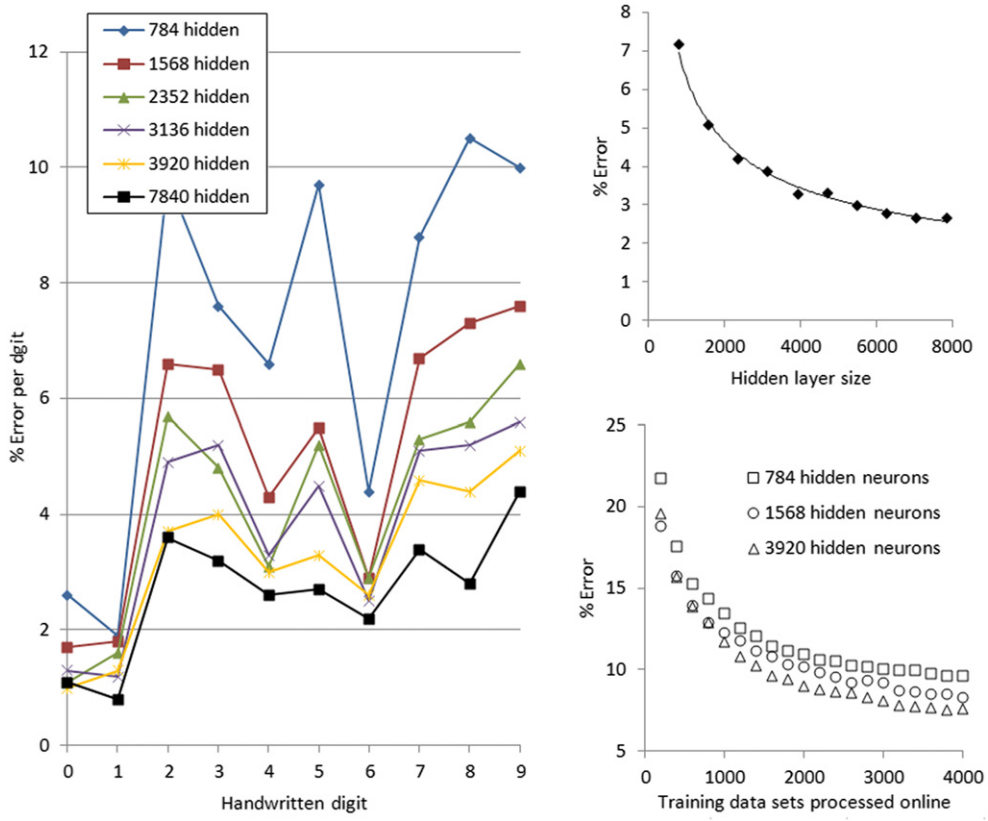

Fig. 2. Performance of the LSHDI network in classifying the MNIST handwritten digit benchmark problem. The left hand graph shows improving performance per digit, as the number of hidden layer neurons increases. Simple forms such as 0 and 1 are more accurately classified than complex forms such as 3 and 5. The RH upper figure shows overall improvement in accuracy with hidden layer size M; the errors decrease in proportion to $M^{-0.5}$. The RH lower figure shows the online learning in progress, for three network sizes; it can be seen that after just 2000 randomly selected digits from the 60 000 digit training set, the test set is already classified with 90% accuracy by networks with large hidden layer size; accuracy will continue to increase (to the levels on the left hand figure) as the training set size increases.

4. Applications of the method: predicting a chaotic time series

We demonstrate the application of the adaptive method with a simple and well-known time series prediction problem, which is to predict the output of the Mackey–Glass (MG) time series (Mackey & Glass, 1977). The MG time series is generated by a nonlinear delay differential equation, shown below, which displays chaotic oscillations:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} - bx(t). \tag{22}$$

This is typically solved using a 4th order Runge–Kutta method. Standard parameters, used by other research groups, are $a = 0.2$; $b = 0.1$; $\tau = 17$; and a computational timestep of 0.1 arbitrary intervals is used.

We have used the LSHDI method to model the series in the usual fashion, which is to tap the series at a fixed set of delays, and then predict the value of the series at some interval in the future. Once again, we have used a standard set of taps and delays in order to produce results comparable with other methods. The data in Fig. 3 were modeled with a network using four taps in the range 100–1 lagging time intervals, and a hidden layer of 100 neurons. The task was to predict the value 50 intervals ahead.

As in the MNIST solution, the accuracy increases with the number of interlayer neurons, although with a diminishing return. An examination of the solved output weights typically indicates a power-law variation in significance (weighting) for the interlayer neurons, with perhaps 15% of neurons contributing 50% of the output weight; which suggests that an algorithm that pruned relatively insignificant interneurons could be used to reduce the interlayer size.

Fig. 3 shows the MG time series and the corresponding predictions for the LSHDI method, in the stationary and the adaptive OPIUM forms. It can be seen that the adaptive method produces a significantly better prediction. It should be noted that this chaotic time series is not in any real sense being modeled globally by the adaptive method, as it is by the static method; the adaptive LSHDI method is finding a local solution to the problem and "forgetting" prior data with an exponential decay of memory. As with the MNIST problem, the error magnitude shows an exponential decay with increasing number of interlayer neurons.

We note that none of the examples above use spiking neurons. Eliasmith and Anderson (2003) have used the LSHDI method extensively with rate-encoded spiking neurons, without any apparent loss of utility or accuracy; the weights in these networks are solved using rate variables rather than actual spike trains. This begs the question whether there is any point in using rate-encoded spikes rather than the underlying rate variable (in this use of the LSHDI method; the use of time-encoded spikes is critical in many other situations, such as in spatio-temporal pattern recognition problems). In practice, there may be dynamical effects such as synchrony which occur when LSHDI coding is performed with spikes, which will not occur directly with the underlying variable.

5. Biological plausibility

The question as to whether a given computational neural process is biologically plausible may seem ill-posed, but it has been addressed in some detail by O'Reilly (1998). He defines biological realism as an overarching principle, and then identifies five

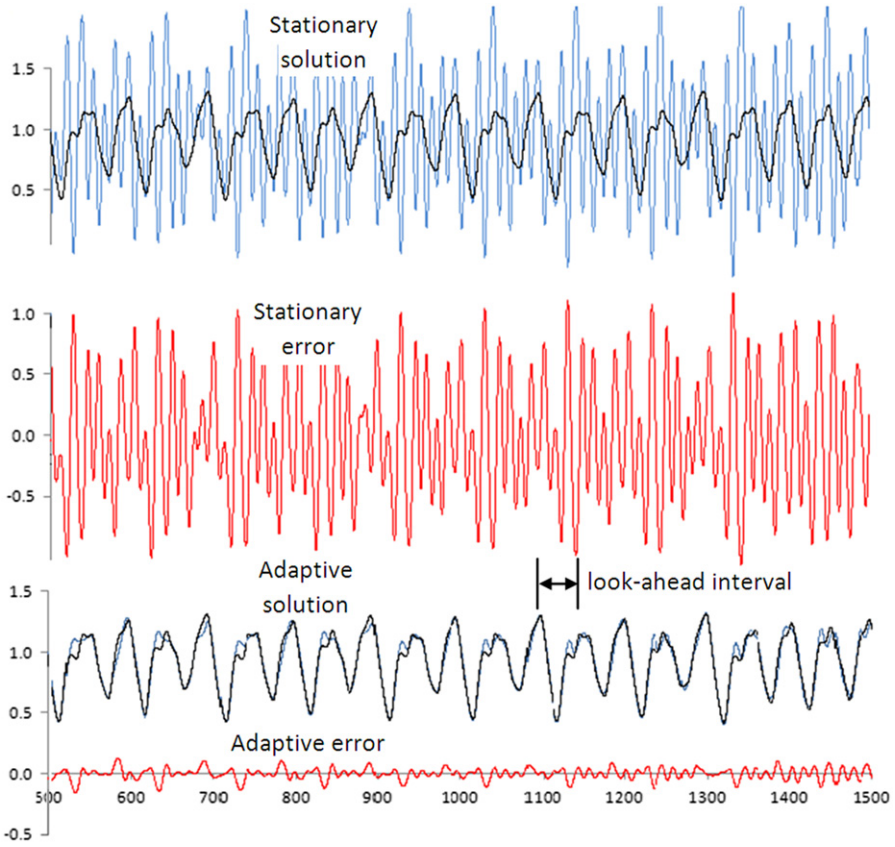

Fig. 3. The Mackey–Glass chaotic time series predicted with stationary and adaptive LSHDI solutions (spaced vertically for clarity); the magnitude of the forward prediction is indicated by the look-ahead interval as shown. It can be seen that by focusing on an adaptive local prediction rather than a global model, the adaptive method is significantly more accurate. The RMS error for the stationary solution was 0.502 and for the adaptive solution was 0.046. The function was approximated by using as input the historical sequence tapped at present, −60, −120 and −180 timesteps; and predicting the value at 50 intervals by incrementally predicting the output at each timestep from the present to the desired look-ahead point. 100 sigmoidal interneurons were used.

architectural principles which can provide converging evidence of biological realism. We list the architectural principles below, and comment on the adaptive OPIUM in that context.

1. *Distributed representation of information*
   The general LSHDI model gives an excellent framework for distributed representation of information, in that input information is spread into a higher dimensional space, and then reduced in dimensionality by construction from that space. Each input variable is represented to some extent in the output of every hidden layer neuron, and loss of a single hidden layer neuron would not significantly degrade the reconstruction of the input variables (the architecture is, after all, a type of associative memory). While it is not a core issue in this research, the architecture convincingly embodies the principle of distributed representation of information.
2. *Inhibitory competition between units*
   The OPIUM algorithm uses the θ matrix to distribute the hidden layer activation energy competitively between weights (the competition being driven by the divisive normalization of activation energy, so that the total energy available is limited). This decomposition by divisive normalization has been identified and described by Schwartz and Simoncelli (2001) in sensory neural systems, amongst others. It has the effect of creating a soft winner-take-all effect, and in the context of Boltzmann machines has been suggested to act as a kind of principle component analysis (Kohonen, 2000).
3. *Bidirectional activation propagation*
   While the network itself is feedforward only, the θ matrix represents a modulation of the output gains of the hidden layer, for the purposes of learning the output layer weights, and can be recast as a backpropagation of error. It should be noted that the backpropagation of error in this model is only local, i.e., the error is used to adjust only the weights immediately prior to the output layer. This avoids the biological-plausibility weakness of the standard backpropagation algorithm, in which output error is backpropagated through several layers in a biologically implausible fashion.
4. *Error-driven task learning (supervised learning)*
   The weight update step in the OPIUM algorithm presented here is a very direct form of error-driven weight modification—it can be summarized as:

   $$\triangle w = E b_k^T$$

   where $\triangle w$ is the weight change, E is the error and b_k a normalization factor. It is particularly suitable that the weight update is local and does not depend on reverse propagation of an error through multiple layers, as is required in conventional backpropagation.
5. *Hebbian learning (unsupervised learning)*
   The θ matrix is relatively close to the identity matrix in our experiments. This means that we can interpret b_k as a renormalized version of the hidden layer activation a_k (see Eq. (12)). The weight update then has the effect of encouraging strong weight development between strongly firing hidden layer neurons and the output layer, which is a straightforward form of Hebbian learning.

This suggests to us that the incremental learning of the pseudoinverse meets O'Reilly's criteria to a reasonable degree, and that it can be recast as a combination of processes which are recognized to occur in biological neural systems.

6. Conclusions

The incremental learning method presented here has two significant uses. The first is largely symbolic; by presenting a biologically-plausible learning method which derives pseudoinverse solutions, we have opened the door to the use of the associated LSHDI methods in biological models, in such a way that awkward questions about the physiological likelihood of the pseudoinverse computation can be addressed with some confidence. The second use of this method is practical; by presenting a new algorithm for online learning and adaptive updating of the pseudoinverse solution in the face of non-stationary processes, we have enabled its use with greater simplicity and accuracy in real-world situations. In addition, the method uses significantly less computer memory than standard SVD based techniques.

Acknowledgments

The authors would like to thank the organizers and funders of the CapoCaccia and Telluride Cognitive Neuromorphic Engineering Workshops, at which the bulk of this work was performed. The authors also thank Gregory Cohen for help with running the MNIST classifications.

References

Barak, O., & Rigotti, M. (2011). A simple derivation of a bound on the perceptron margin using singular value decomposition. *Neural Computation*, *23*, 1935–1943.

Ben-Israel, A., & Greville, T. N. E. (2003). *Generalized inverses: theory and applications* (2nd ed.). New York: Springer.

Bishop, C. M. (2008). *Neural networks for pattern recognition*. Oxford: Oxford University Press.

Cho, Y., & Saul, L. K. (2010). Large-margin classification in infinite neural networks. *Neural Computation*, *22*, 2678–2697.

Choudhary, S., Sloan, S., Fok, S., Neckar, A., Trautmann, E., & Gao, P. et al. (2012). Silicon neurons that compute. In *Int. conf. artificial neural networks*, ICANN 2012. *Lecture Notes in Computer Science, 7552.* (pp. 121–128).

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: computation, representation and dynamics in neurobiological systems*. MIT Press.

Galluppi, F., Davies, S., Furber, S., Stewart, T., & Eliasmith, C. (2012). Real time on-chip implementation of dynamical systems with spiking neurons. In *Proc. int. joint conf. neural networks*, IJCNN 2012. (pp. 1–8).

Greville, T. N. E. (1960). Some applications of the pseudoinverse of a matrix. *SIAM Review*, *2*, 15–22.

Guo, P., & Lyu, M. R. (2003). A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. *Neurocomputing*, *56*, 101–121.

Huang, G. -B. (2012). Extreme learning machine: learning without iterative tuning. In *Tutorial presentation at the World congress on computational intelligence*, WCCI 2012 (p. 96). Available at: http://www.ntu.edu.sg/home/egbhuang/ (on 05.12.12).

Huang, G. -B., Liang, N. -Y., Rong, H. -J., Saratchandran, P., & Sundararajan, N. (2005). On-line sequential extreme learning machine. In *Proc. IASTED international conference on computational intelligence*, CI 2005.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*, 489–501.

Kohonen, T. (1988). Correlation matrix memories. *IEEE Transactions on Computers*, *C-21*(4), 353–359.

Kohonen, T. (1989). *Self-organization and associative memory* (3rd ed.). Berlin: Springer-Verlag (Chapter 6).

Kohonen, T. (2000). *Self-organizing maps* (3rd ed.). Berlin: Springer-Verlag (Chapter 2).

Kovanic, P. (1979). On the pseudoinverse of a sum of symmetric matrices with applications to estimation. *Kybernetika*, *15*(5), 341–348.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., & Cortes, C. (2012). The MNIST database of handwritten digits. Available at: http://yann.lecun.com/exdb/mnist/ (last accessed: 12.07.12).

Mackey, M., & Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science*, *197*, 287.

MacNeil, D., & Eliasmith, C. (2011). Fine-tuning and the stability of recurrent neural networks. *PLoS One*, *6*(9), e22885.

Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, *1*, 61–68.

O'Reilly, R. C. (1998). Six principles for biologically-based computational models of cortical cognition. *Trends in Cognitive Sciences*, *2*, 455–462.

Penrose, Roger (1955). A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, *51*, 406–413.

Rahimi, A., & Recht, B. (2009). Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems, 21* (pp. 1313–1320). Cambridge, MA: MIT Press.

Rigotti, M., Ben Dayan Rubin, D., Wang, X.-J., & Fusi, S. (2010). Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. *Frontiers in Computational Neuroscience*, *4*, 24, http://dx.doi.org/10.3389/fncom.2010.00024.

Saxe, A., Koh, P., Chen, Z., Bhand, M., Suresh, B., & Ng, A. (2011). On random weights and unsupervised feature learning. In *Proc. 28th int. conf. on machine learning*.

Schwartz, O., & Simoncelli, E. (2001). Natural signal statistics and sensory gain control. *Nature Neuroscience*, *4*, 819–825.

Stewart, T. C., Bekolay, T., & Eliasmith, C. (2012). Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Decision Neuroscience*, *6*, http://dx.doi.org/10.3389/fnins.2012.00002.