Training Spiking Neuronal Networks With Applications in Engineering Tasks

Phill Rowcliffe and Jianfeng Feng

***Abstract*—In this paper, spiking neuronal models employing means, variances, and correlations for computation are introduced. We present two approaches in the design of spiking neuronal networks, both of which are applied to engineering tasks. In exploring the input–output relationship of integrate-and-fire (IF) neurons with Poisson inputs, we are able to define mathematically robust learning rules, which can be applied to multilayer and time-series networks. We show through experimental applications that it is possible to train spike-rate networks on function approximation problems and on the dynamic task of robot arm control.**

***Index Terms*—Integrate-and-fire (IF), kernel, mean interspike interval (ISI), robot arm, variance.**

I. Introduction

In recent years, there has been significant growth in the field of biological computation. In that time, a closer unification between neuroscience and artificially intelligent computational models has been observed. As a result, computational neural models exist, owing more to their biological counterparts than previous classical artificially intelligent models. Voltage threshold models such as integrate-and-fire (IF) model [17], [22], [29]–[31], and the more biophysical Hodgkin–Huxley (HH) model [16], [21], all incorporate more of the dynamics of actual biological neurons than the traditional classical approach to neural modeling, such as the perceptron [25].

In trying to understand the computational properties of the brain, it is necessary to understand the biophysical mechanisms involved in the process. Defining these mechanisms with computational models allows us to further explore some of the complex and adaptive processes, which may be employed by biological neural systems.

As such, we have seen the field of *computational neuroscience* grow considerably in recent years. A result of this is the emergence of a variety of engineering applications, and learning rules, which now employ biologically plausible computational models. Indeed, we have seen many successful applications within the fields of human arm movement [19], computer vision [4], and speech recognition [23], [32], to name just a few. In considering the application of biologically plausible neural models, we need to consider which type of model to use, and how best to address the issue of training.

Many of the engineering applications that have applied biophysical models have used the IF model as the main computational unit. In using this model, it is often the temporal sensitivity of the neuron that is exploited within computation, i.e., the time interval between successive spikes. In fact, Bohte *et al.* [3] used this principle to develop an *error regression* learning rule to train a network of IF neurons. It is worth noting at this point that the rule developed by Bohte *et al.* [3] is one of the few learning rules, applied to spiking networks, which is not based on a Hebbian [15] approach to synaptic weight modification. Indeed, many of the learning rules, developed for use on spike-time-dependent models [13], rely on Hebbian correlation as the principal means for synaptic weight modification. However, as Bohte *et al.* have shown, with their backpropagation learning rule, Hebbian learning need not be the only approach to training IF neurons for use within engineering.

In [26], a single biologically plausible spike-rate model used a mathematically derived backpropagation learning rule, to solve a nonlinear tractable task. Like Bohte *et al.* in [3], the learning rule was based on error regression. However, the neuron model used in [26] represented the IF model in terms of its firing rate. Defining the model in these terms provided a relationship between the synaptic input of a neuron and the firing rate output of the model. The spike-rate model as presented in [26] provided both first- and second-order statistical representation of the synaptic input. As such, computational information is shown to be present in both the mean and the variance of synaptic input. When plotting the spiking rate output of the model against its synaptic input, Rowcliffe *et al.* [26] presented a series of firing rate output profiles, kernel-like in nature. One of the main observation about this model is that its output firing rate appears to owe more to radial basis function (RBF) model than its classical predecessor: the perceptron.

As a unit of computation though, the single spike-rate model has advantages over some classical models with the inclusion of both the mean firing rate and the variance of the firing rate. With many classical models, like the perceptron, if there is an equal balance of excitatory and inhibitory inputs, the mean effect on the model is zero. With the spike-rate model, this is not the case. The spike-rate model includes both first- and second-order statistics: the mean and the variance. Indeed, it was shown in [9] and [26] that even when the mean input vanishes, the model and the learning rule still function due to the synaptic variance. As such, the spike-rate neuron is a computing model of both the mean and variance. The output surface planes for this model have been shown to be controllable by synaptic modification through the use of the derived learning rule [26].

Manuscript received May 12, 2007; revised December 18, 2007; accepted December 30, 2008. First published August 15, 2008; current version published September 4, 2008.

P. Rowcliffe is with the Department of Informatics, School of Science and Technology (SciTech), University of Sussex, Brighton, East Sussex BN1 9QH, U.K. (e-mail: phillipr@sussex.ac.uk).

J. Feng is with the Centre for Computational System Biology, Fudan University, Shanghai, China and also with the Center for Scientific Computing and Computer Science, University of Warwick, Coventry CV4 7AL, U.K. (e-mail: jffeng@fudan.edu.cn).

Digital Object Identifier 10.1109/TNN.2008.2000999

The spike-rate model, therefore, has potential engineering applications, which we will introduce as part of this paper. The advantages of using second-order statistics over the classical approach, which mainly use first-order statistics, have been known in literature for many years. In [11], for example, Feng and Tuckwell introduced an optimal control task based upon the control of second-order statistics, the variance, which presented some interesting properties.

Introducing second-order statistics in computations is though a minimal requirement if we intend to implement stochastic computations; an example being the Bayesian approach. Hence, the framework we present here opens up the possibility of carrying out a random computation in neuronal networks.

The computational power of spiking neuronal networks have already been explored in the liquid state machine [23] and the echo state machine [18]. In their work, the computational performance was achieved due to the high-dimensional projection of the low-dimensional input space (a kernel property). However, in the approach we present here, though we employ the kernel property as a natural result of the spiking neuronal network, we do not use the inefficient computational projection of the input space to a high-dimensional space.

In this paper, therefore, we consider the design and structure of a network of spike-rate neurons and what is involved in training these networks for use in engineering tasks. We propose two applications of these models, for use on specific tasks, as a basis for investigating their computational properties. Our approach has been to derive learning algorithms based on a multilayered network of spike-rate neurons. We have expanded on the learning rule introduced in [26], where we identified the input–output relationship of the spike-rate model and applied an error minimization technique to train the model. The network designs introduced here are specific to the tasks of function approximation and robot control, and have similar structures to RBF networks.

We, therefore, present one of the first applications of a network of spike-rate neurons and show that it is possible to train these networks with a mathematically derived learning rule. Though the networks take a longer period of time to train than their classical counterparts, they do offer a significant advantage over classical artificial intelligence (AI) models in that they include both the mean μ and the variance σ of the input signal.

This paper is set out as follows. Section II will define the model of the single spike-rate neuron, the basis of which we will use in the building of our neuronal network that we will define in Section III. A network learning rule is introduced in Section IV together with the results from the function approximation tasks, which we present in Section V. Finally, in Sections VI and VII, we detail the approach we took in applying the spike-rate model to the task of robot arm control, defining the recurrent structure of a time-series network, together with the modified backpropagation through time version of our network learning rule.

II. Model Description

First, we present the definition of the spike-rate model. We begin by considering the IF model [6], [13], [31], defined many times in literature and presented here as follows.

Suppose a cell receives excitatory postsynaptic potentials (EPSPs) at n of its synapses, and inhibitory postsynaptic potentials (IPSPs) at m of its inhibitory synapses. When the membrane potential $V(t)$ is between its resting state V_{rest} and its threshold V_{thre}, it satisfies the following:

$$dV(t) = -\tau(V(t) - V_{\text{rest}})dt + d\bar{I}_{\text{syn}}(t) \tag{1}$$

where τ is the decay rate of the membrane and $\bar{I}_{\text{syn}}(t)$ is the synaptic input

$$\bar{I}_{\text{syn}}(t) = \sum_{j=1}^{n} w_{ij}^{E} E_j(t) - \sum_{j=1}^{m} w_{ij}^{I} I_j(t). \tag{2}$$

Here, $E_j(t)$ and $I_j(t)$ are renewal processes for $t \geq 0$, and $w_{ij}^{E} > 0$ and $w_{ij}^{I} > 0$ are the magnitudes of the EPSP and IPSP, respectively. The total current input into the neuron is summed over all n excitatory and m inhibitory synapses. When $V(t)$ crosses the membrane threshold V_{thre} from below, a spike is generated and the membrane resets to its resting potential V_{rest}.

However, in [31], Tuckwell showed that jump processes, such as $E_j(t)$ and $I_j(t)$ in (2), can be approximated using diffusion approximations, such that

$$E_j(t) \approx \lambda_j^{E} t + \lambda_j^{E^{\alpha/2}} B_j^{E}(t)$$
$$I_j(t) \approx \lambda_j^{I} t + \lambda_j^{I^{\alpha/2}} B_j^{I}(t)$$

where B_j^{E} and B_j^{I} are Brownian motions, λ_j^{E} and λ_j^{I} are the synaptic input renewal process rates with $\lambda_j^{E} = (\lambda_1^{E}, \ldots, \lambda_n^{E})$ and $\lambda_j^{I} = (\lambda_1^{I}, \ldots, \lambda_m^{I})$, and $\alpha > 0$ is the parameter, discussed in [11], which produces a Poisson process input when $\alpha = 1$.

We have seen in [6] that (1) can be approximated as

$$dv(t) = -\tau(v(t) - V_{\text{rest}})dt + d\bar{i}_{\text{syn}}(t) \tag{3}$$

where

$$\bar{i}_{\text{syn}}(t) = \mu_i t + \sigma_i B(t) \tag{4}$$

and

$$\begin{aligned}\mu_i &= \sum_{j=1}^{n} \lambda_j^{E} w_{ij}^{E} - \sum_{j=1}^{m} \lambda_j^{I} w_{ij}^{I} \\ \sigma_i^2 &= \sum_{j=1}^{n} \left(\lambda_j^{E}\right)^{\alpha} \left(w_{ij}^{E}\right)^2 + \sum_{j=1}^{m} \left(\lambda_j^{I}\right)^{\alpha} \left(w_{ij}^{I}\right)^2 \\ &\quad + \rho \sum_{j \neq k=1}^{n,m} \left(\lambda_j{}^{E}\right)^{\frac{\alpha}{2}} \left(\lambda_k{}^{E}\right)^{\frac{\alpha}{2}} w_{ij}^{E} w_{ik}^{E} \\ &\quad + \rho \sum_{j \neq k=1}^{n,m} \left(\lambda_j{}^{I}\right)^{\frac{\alpha}{2}} \left(\lambda_k{}^{I}\right)^{\frac{\alpha}{2}} w_{ij}^{I} w_{ik}^{I}.\end{aligned} \tag{5}$$

Here, ρ is the correlation coefficient between the ith and jth input, a detailed discussion of which is given in [28]. It is worth noting here, that the model's description of the synaptic input, as given in (4), presents it in terms of input mean and variance.

For simplicity of notation, we next consider excitatory and inhibitory inputs to be independent, with $w_{ij} = w_{ij}^{E} = w_{ij}^{I}$, $m = n$, and $\lambda_j^{I} = r\lambda_j^{E} = r\boldsymbol{\lambda}$. Here, $r = 0$ if the unit only receives purely excitatory inputs and $r = 1$ when the unit receives

equal excitatory and inhibitory inputs. Equation (5) can now be rewritten as

$$\mu_i = \sum_{j=1}^{n} \lambda_j w_{ij}(1-r)$$
$$\sigma_i^2 = \left(\sum_{j=1}^{n} \lambda_j^{\alpha} w_{ij}^2 + \rho \sum_{j \neq k=1}^{n} {\lambda_j}^{\frac{\alpha}{2}} {\lambda_k}^{\frac{\alpha}{2}} w_{ij} w_{ik} \right) (1 + r^{\alpha}). \tag{6}$$

Equations (3), (4), and (6) represent the model in terms of its membrane potential. As stated previously, the synaptic input used in this model, given in (4), presents a synaptic input in terms of the mean input μ_i and the variance σ_i^2, about this mean. The importance of this can be seen in the case when $r = 1$, i.e., when a neuron has an equal balance of excitatory and inhibitory synaptic inputs. In this case, the μ_i term in (6), i.e., the mean input, disappears. However, the variance term σ_i^2 remains and as a result the neuron still receives synaptic activity.

Next, we represent this model in terms of its firing rate. With the model represented in this way, we are able to see a direct input–output relationship, previously shown to exist in [7], in which computational information is encoded within the *firing rate* of the model.

To achieve this, first let us define $f_i(\boldsymbol{\lambda})$ as the firing rate output of the IF unit i, subject to synaptic input rates $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$. We can write the firing rate in terms of the interspike interval as

$$f_i(\boldsymbol{\lambda}) = \frac{1}{T_{\text{ref}} + \langle T_i(r) \rangle} \tag{7}$$

where T_{ref} is the refractory period, $\langle T_i(r) \rangle$ is the mean interspike interval of output unit i; note that i typically covers the output space say of $i = 1, \ldots, N$.

The definition of the mean interspike interval has previously been given in [6] as

$$\langle T_i(r) \rangle = \frac{2}{\tau} \int_{\frac{V_{\text{rest}}\tau - \mu_i}{\sigma_i}}^{\frac{V_{\text{thre}}\tau - \mu_i}{\sigma_i}} g(x)\, dx \tag{8}$$

where τ is the decay rate of the IF model, and $g(x)$, known as *Dawson's integral*, is defined as

$$g(x) = \exp(x^2) \int_{-\infty}^{x} \exp(-u^2) du. \tag{9}$$

Now, the model, as presented in (3) and (4), gives the neuron in terms of its *mean firing rate* and its *standard deviation*. As stated earlier, in classical artificial neural networks, for a model with equally balanced excitatory and inhibitory synaptic inputs, the firing rate activity is silent. In the neural model presented here, it can clearly be seen that this is not the case.

In Fig. 1, we see an example of the case when $r = 1$ in (6), where the mean term μ_i disappears, but the variance σ_i^2 term remains. The figure plots of the firing rate output of a model with two synapses with varying inputs $\lambda_1 = (-2, \ldots, 2)$ and $\lambda_2 = (-2, \ldots, 2)$. Equation (7) presents the IF neuron in terms of its firing rate output and its mean interspike interval.

It is this aspect of the model that has motivated much of the remainder of this paper. By examining the variance term of a neuronal model, rather than trying to smooth out any "noise" element, we will examine its computational properties in its application to a series of engineering tasks. As such, we will focus our experiments on the case of $r = 1$. In doing so, we will effectively have a firing rate model with synaptic input terms

$$\mu_i = 0$$
$$\sigma_i^2 = 2 \left(\sum_{j=1}^{n} \lambda_j^{\alpha} w_{ij}^2 + \rho \sum_{j \neq k=1}^{n} {\lambda_j}^{\frac{\alpha}{2}} {\lambda_k}^{\frac{\alpha}{2}} w_{ij} w_{ik} \right). \tag{10}$$

We will, however, define the network and learning rules, in terms of a general approach, because such a learning rule, and network design, should hold for the case when $r = 0, \ldots, 1$.

III. Spike-Rate Networks

The kernel-like structure of the spike-rate model, similar to that presented in Fig. 1, is an interesting property of the neuron model. These kernel-like output profiles were previously explored in [26], where similar outputs were observed for differing synaptic weight configurations, i.e., differing values of r.

These neural models appear to have more in common with RBF models than their classical predecessors, such as the perceptron. The spike-rate model's output for $r = 1$ is very similar to that of a multiquadratic function used in some RBF models. In comparing this spike-rate model's output with that of a multiquadratic RBF, two important similarities have been observed.

First, multiquadratic RBF networks use basis functions similar to those introduced in [14], and are generally of the form

$$y(x) = (x^2 + c^2)^{\frac{1}{2}} \tag{11}$$

where x is the input, $c > 0$, and $x \in \mathbb{R}$. The output of the spike-rate model defined in (7) is equivalent to this function, when $0 < c < 1$ in (11).

Second, in [24], it was proven that for a distinct set of N points in $\mathbb{R}^{m_0}$ there exists an $N \times N$ interpolation matrix Φ, if the jith element $\phi_{ji} = \phi(\|x_j - x_i\|)$ is nonsingular. For a multiquadratic function to be nonsingular, $\{x_i\}_{i=1}^{N}$ must be distinct. This is true for both the multiquadratic RBF model and the spike-rate model over all real inputs.

These two common features serve as a basis for the development of a network model based on the principles of RBF network design. In the numerical models presented here, spike-rate neurons replace the basis functions presented in traditional RBF network architectures, and the network's output comprises a linear summation of these spike-rate neuron's outputs.

A. Kernel Neural Networks

We now introduce our first design of a network of spike-rate neurons based on an RBF-style architecture. The output profile of the spike-rate model, presented in Fig. 1, shows a kernel centered at zero when input $\boldsymbol{\lambda} = 0$. We see that a similar output is obtained in a multiquadratic RBF if c, in (11), is set to *zero*. So we find that by including *centers* for each spike-rate neuron, similar to the approach taken in the positioning of basis units in RBF networks, we ensure the spike-rate network is positioned

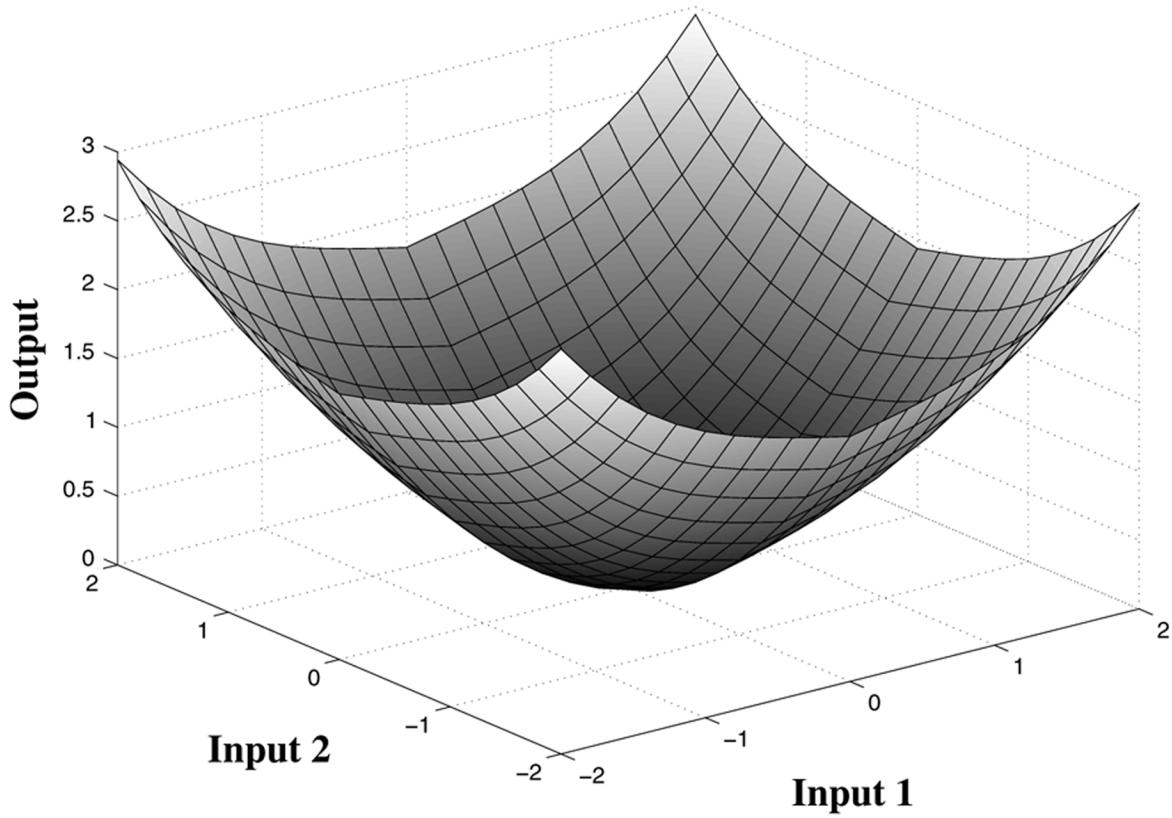

Fig. 1. Graphical plot of the firing rate output plane of a single neuron with two synaptic inputs, where $\alpha = 2$ and $r = 1$. Input values were varied between -2 and 2.

across its input space. To achieve this, therefore, (6) is modified as follows:

$$\mu_i = \sum_{j=1}^{n} (\lambda_j - \lambda^i) w_{ij} (1 - r)$$
$$\sigma_i^2 = \left(\sum_{j=1}^{n} (\lambda_j - \lambda^i)^{\alpha} w_{ij}^2 + \rho \sum_{j \neq k=1}^{n} (\lambda_j - \lambda^i)^{\frac{\alpha}{2}} \times (\lambda_k - \lambda^i)^{\frac{\alpha}{2}} w_{ij} w_{ik} \right) (1 + r^{\alpha}) \quad (12)$$

where λ^i is the center for neuron i over the dimensions of the input space $j = 1, \ldots, n$ specific to neuron i, and $T_{\text{ref}} > 0$. It is now possible to place individual neurons across the neuron's input space by a suitable choice of λ^i. For n input nodes on the input layer, and M spike-rate neurons on layer i, the output from this RBF-*style* network is defined as

$$y_h(\boldsymbol{\lambda}) = \sum_{i=1}^{M} w_{hi} f_i(\boldsymbol{\lambda}) \quad (13)$$

where y_h is the output from output node h connected to neuron i by the weight connection w_{hi}, as shown diagrammatically in Fig. 2.

We observe that the output $y_h(\boldsymbol{\lambda})$ is a special case of the spike-rate model presented in (7) if r is set to 0, i.e., when there are purely excitatory weight connections for the neurons in the output layer of the network. Equation (6), therefore, can be rewritten in the form

$$\mu_h = \sum_{i=1}^{M} f_i(\boldsymbol{\lambda}) w_{hi}$$
$$\sigma_h^2 = \sum_{i=1}^{M} f_i(\boldsymbol{\lambda})^{\alpha} w_{hi}^2 + \rho \sum_{i \neq k=1}^{M} f_i(\boldsymbol{\lambda})^{\frac{\alpha}{2}} f_k(\boldsymbol{\lambda})^{\frac{\alpha}{2}} w_{hi} w_{hk}. \quad (14)$$

In the model defined in (13) and (14), the *special case* spike-rate neuron differs from the spike-rate model, discussed in Section II, because it takes as inputs the *firing rates* from

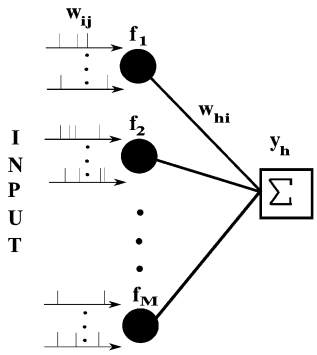

Fig. 2. Schematic plot of an RBF-style spike-rate network. Each of the $i = 1, \ldots, M$ spike-rate neurons receives $j = 1, \ldots, n$ inputs $\boldsymbol{\lambda}$, with weights w_{ij}. The output of the RBF-style network y_h is a summation of the product of the firing rate of each of the spike-rate neurons $f_i(\boldsymbol{\lambda})$ and the output layer's weight connection w_{hi}.

neurons in the previous layer. In a biological framework, neurons emit and receive spikes. This is not the case here though. We have instead chosen to model an RBF-style architecture as our initial step in the examination of the computational performance of a network of spike-rate neurons. It should be noted, however, that in [8], a theoretical framework has been presented for a network of IF neurons in terms of the first- and second-order statistics of the interspike interval (ISI). In the case there, Feng *et al.* have shown that it is possible to build a neural framework with a diffusion approximation for the renewal inputs, and thus an approximation to describe the behavior of a network of IF neurons.

IV. Network Training

To support the learning rules we will present in this paper, we have included in Appendix I the derivation of a learning rule for a single spike-rate neuron, first introduced in [26]. This shows how the synaptic weights are updated in order to reduce the output error, during training. It is included here so as to provide a broader picture in the training of the spike-rate neuron, at both a single-neuron level and a network level.

In Section III, we introduced a spike-rate network design similar to that of an RBF network, i.e., with an input layer, a hidden layer of nonlinear basis functions (or spike-rate neurons in this case), and a linear output layer. We, therefore, choose to apply a similar learning algorithm to those used in training RBF networks. This algorithm involves training the network in two separate stages. The first stage consists of centering each of the spike-rate neurons across the input space. A k-means algorithm is used to identify the number of subsets within the training data. Once k-means is complete, the output from this stage will identify the optimum number of spike-rate neurons to use in the network, and where their centers should be positioned so as to cover the range of the input space.

The second stage in the training consists of a learning rule, which is used to update the weights in the network. The network is trained on the same data set used in the k-means section of the algorithm. Training is complete when the network's output falls within a specified error tolerance.

A. Stage One

In this first stage, the algorithm partitions the input training data sets into subgroups. A set of training data is first identified, which traditionally covers the input–output space. For this type of network, the input data is a set of synaptic inputs $\boldsymbol{\lambda}$ as defined in Section II. The remainder of the algorithm is defined as follows.

Let $\boldsymbol{a}^b$ be the set of input data points, where $b = 1, \ldots, B$. For these data points, employ the k-means algorithm [5] to partition this data into a set of k vectors $\tilde{\mu}^j$ where $j = 1, \ldots, k$.

1) Partition the input data $\{\boldsymbol{a}^b\}$ into K initial sets that cover this input space.
2) Calculate the mean point of each of the k sets
$$\tilde{\mu}^j(t) = \frac{1}{B} \sum_{b \in S_j} \boldsymbol{a}^b$$
where $\tilde{\mu}^j(t)$ is the mean of the data points in set S_j at iteration t.
3) For each data point in $\{\boldsymbol{a}^b\}$, calculate its distance from each set's center $\tilde{\mu}^j(t)$, and reassign each point to the set with the closest mean.
4) Calculate the new mean for each of the updated sets
$$\tilde{\mu}^j(t+1) = \frac{1}{B} \sum_{b \in S_j} \boldsymbol{a}^b.$$
5) Repeat until changes in the groupings stabilize.

The resulting centers of each of the sets are then used as the centers for each of the spike-rate neurons as presented in (12). Here, $\tilde{\mu}^j = \boldsymbol{\lambda}^j$ with the number of spike-rate neurons M, in the network design, equal to the number of sets partitioned with the k-means algorithm, i.e., $M = k$.

Once the centers of the neurons have been identified, the unsupervised section of the training algorithm is concluded. This stage of the algorithm has now provided the optimum number neurons for the network, i.e., $M = k$ in (13) and the position of their centers across the input space. The network as defined in (13) can now be trained on the input data using stage two of the algorithm.

B. Stage Two

This second stage of the algorithm consists of a supervised learning rule. An approach similar to backpropagation [27], [34] is used to adjust the weights between the output layer and the hidden layer.

1) The Output Layer of an RBF-Style Network: The error function E, also known as the *sum of squares* error, is defined as
$$E = \frac{1}{2} \sum_{h=1}^{P} (d_h - y_h(\boldsymbol{\lambda}))^2 \tag{15}$$
where $y_h(\boldsymbol{\lambda})$ is the output of the network at neuron h, as defined in (13) and d_h is the desired target response of output neuron h. The error is calculated over the total number of output neurons P, in the output layer. The output error here depends upon the weights w_{hi} and so any correction of these weights is proportional to the partial derivative of this error. This gradient change is obtained using the definition for $y_h(\boldsymbol{\lambda})$ in (13), such that
$$\frac{\partial E}{\partial w_{hi}} = \frac{\partial E}{\partial y_h(\boldsymbol{\lambda})} \frac{\partial y_h(\boldsymbol{\lambda})}{\partial w_{hi}} = (d_h - y_h(\boldsymbol{\lambda})) f_i(\boldsymbol{\lambda}). \tag{16}$$
To compliment the learning rule presented above, in Appendix II, we have included a proof concept for a version of this learning rule that can be applied to RBF-type networks, which use spike-rate neurons in the output layer. Though it is not biologically realistic for a spike-rate neuron to have *firing rates* as synaptic inputs, the proof completes the learning rule in terms of the backpropagation of the network error.

V. Applying the Learning Rule

We now present a series of experimental results to illustrate the spike-rate network's ability to perform function approxima-

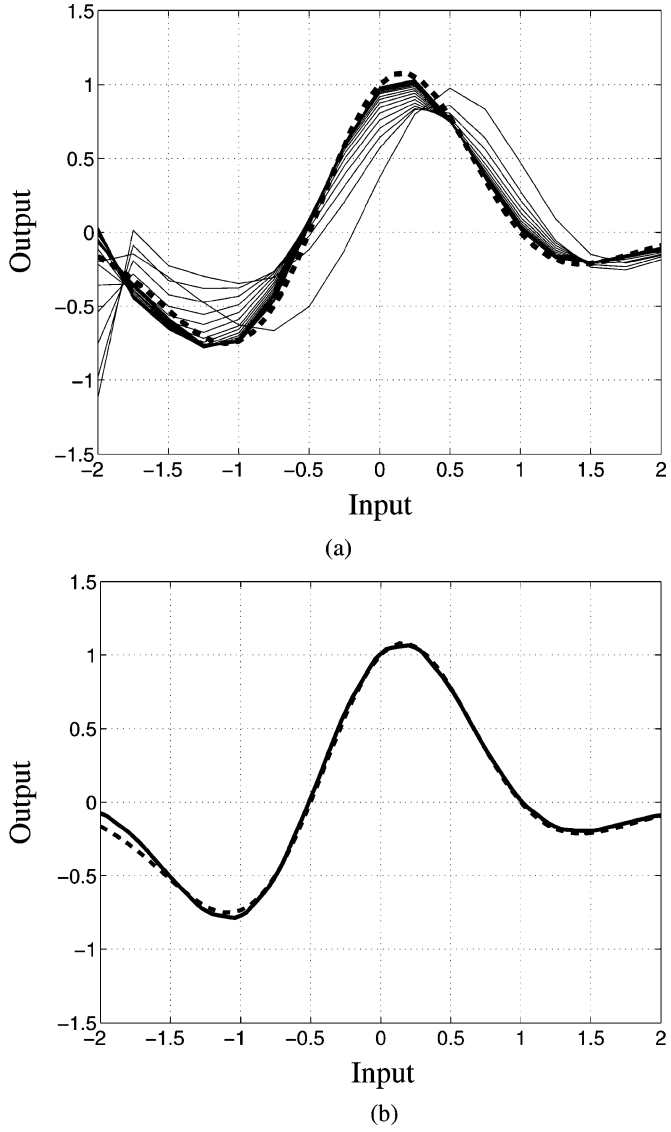

Fig. 3. (a) Network's output when trained to fit the curve defined by function (17). The change in the network's output (the thin gray line) is shown at 100 iterations intervals during the training. The target in both (a) and (b) is represented by the dotted line. (b) Target output (the dotted line) and the network's output (the thick black line) after training. Here, the network is tested on 100 data points within the range $[-2, 2]$, which were not part of the data set used in training the network.

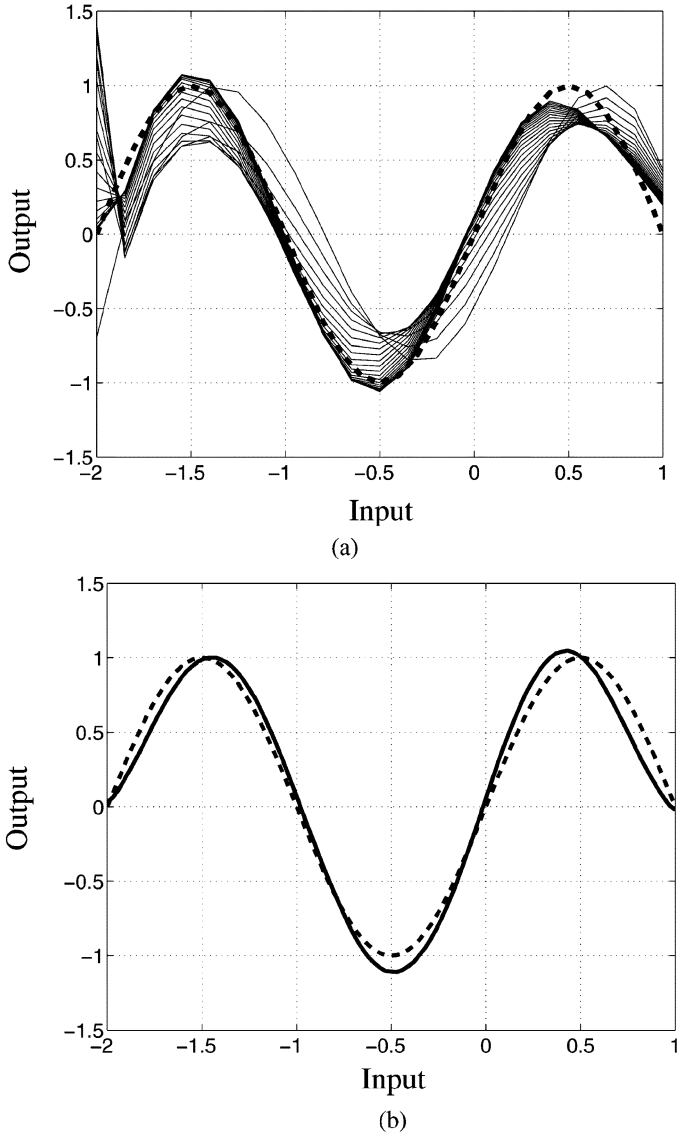

Fig. 4. (a) Network's output when trained to fit the curve defined by function (18). The change in the network's output (the thin gray line) is shown at 100 iterations intervals during the training. The target in both (a) and (b) is represented by the dotted line. (b) Target output (the dotted line) and the network's output (the thick black line) after training. Here, the network is tested on 100 data points within the range $[-2, 1]$, which were not part of the data set used in training the network.

tion. The network was tested on a variety of functions and we include two examples here for consideration.

The two functions the network was trained to approximate are

$$y(x) = (1 + x - 2x^2)e^{-x^2} \tag{17}$$

$$y(x) = \sin(x). \tag{18}$$

In all experimental trials, a single-layered network of 20 spike-rate neurons was used. Each spike-rate neuron had an equal balance of excitatory and inhibitory synaptic inputs, i.e., $r = 1$. These neurons were connected to an output node by a linear summation of the weighted connection between them and the output node. An optimal learning rate of 0.2 was used, and a sample of input points taken from the curves in (17) and (18) were used to train the network.

In Fig. 3(a), a graphical representation is presented, showing the network's output throughout the training process as it was trained to approximate function (17). Initially, the network was set up with equal weight values of 0.5. The training algorithm described in Section IV was used to train the network, which converged the output to within an error of 0.01 by the 1994th iteration.

Fig. 4(a) shows the network's output when trained to approximate function (18). Initially, the network was set up with equal weight values of 0.5. The algorithm trained the network to within an error convergence of 0.01 in 1976 iterations.

Once training was complete, each network was tested on the remaining input data, i.e., data which had not initially been sampled for use in the network training. Figs. 3(b) and 4(b) show how the networks generalized this data, for functions (17) and

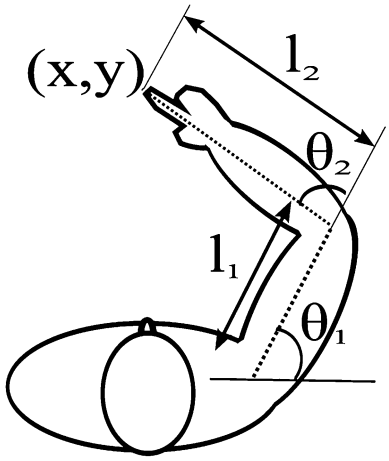

Fig. 5. A 2-D representation of a human arm as seen from above. It shows the position of the two joint angles θ_1 and θ_2, which the network outputs to control the movement of the robot arm. The length of the two arm segments are l_1 and l_2, and the target position of the arm is represented by the Cartesian coordinate pair (x, y).

(18), respectively. In both figures, the dotted lines represent the original target output and the thick black lines represent the network's approximation. Both networks produced reasonable approximations using the data.

VI. Robot Control

Following on from this, a dynamic application for the spike-rate network is investigated, namely, a network designed to act as a control mechanism for a simulated robot arm, in the task of goal locating. To accomplish this, an *artificial environment* was designed, enabling arm data to be collected for use in the initial training of the network. The artificial environment permits accurate and detailed observations to be made of the robot arm's performance and movement, both during and after training.

A. Two Joint Robot Arm

The robot controller and its environment are designed for the task of *goal location*. The controller's input data is the set of Cartesian coordinates representing the target object's location within a 2-D plane at Fig. 5. The network's objective is to move the robot's arm towards these target coordinates. The arm itself consists of two sections of length l_1 and l_2 and two joints represented by the angles θ_1 and θ_2. The first joint fixates the arm at a point in the plane, with θ_1 describing the circular movement of l_1 about this point. The second joint is between the two sections l_1 and l_2 with θ_2 describing the angular position of l_2 about this joint. The network's outputs are the two joint angles θ_1 and θ_2 of the robot arm. Fig. 5 shows a diagrammatic representation of the arm. A change in these angles directs the movement of the robot arm

$$\theta_2 = \arccos\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}\right)$$
$$\theta_1 = \arctan\left(\frac{y}{x}\right) - \arctan\left(\frac{l_2\sin\theta_2}{l_1+l_2\cos\theta_2}\right) \quad (19)$$

where x and y are the trajectory coordinates the arm traces during the movement it makes towards its target. It is possible to rewrite (19) in terms of these Cartesian coordinates points

$$x = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2)$$
$$y = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2) \quad (20)$$

where l_1 and l_2 are the length of the robot arm segments.

B. Design of the Neural Network

In designing the network for use in this task, we began by identifying the set of inputs–outputs, which would be employed by a robot controller. For the inputs, these would be the location coordinates of the object in the environment. For the outputs, these would be the robot arm's angular movement over time. How this information is generated is discussed in Section VI-D.

We chose, therefore, to expand on our RBF-style network, presented in Section III-A, but with the inclusion of recurrent connections. This approach, together with the application of a backpropagation through time (BPTT) learning rule [35], is one which is perfectly suited to dynamic tasks, and one which is commonly applied, to similar problems, within the field of AI. By unfolding the network in time, we are able to treat the entire network as one large feedforward network.

Using this type of network, each time step in the movement of the robot arm, i.e., $t = 1, \ldots, T$, is represented by a duplicate network. Each network, at each of the time steps, receives external inputs from the robot environment, as well as recurrent connections from neurons in previous time steps, as represented in Fig. 6. A neuron's output at time step t is used as part of the input for its equivalent neuron at the next time step $t+1$. So for an RBF-style network, as described in Section III-A, synaptic inputs into a spike-rate neuron i at time step t comprise two main elements: the normal synaptic input as introduced in (4) and a recurrent input, such that $\overline{i}_{\text{syn}}$ in (4) is rewritten as (21) shown at the bottom of the page, where $\widetilde{i}_{\text{syn}}$ is the synaptic input for a spike-rate neuron in a BPTT network, $\boldsymbol{\lambda}$ is the input into neuron i which is external to the network, $f_i^{(t-1)}(\boldsymbol{\lambda}_i)$ is the output from neuron i at the time step $t-1$ [note for $t=0$, $f_i^{(0)}(\boldsymbol{\lambda}_i)=0$], $w_{ij}^{(t)}$ is the magnitude of the postsynaptic potential for the network at time t, and $\tilde{w}_{ii}^{(t-1)}$ is the time delay weight connection between spike-rate neuron i at time $t-1$ and spike-rate neuron

$$\widetilde{i}_{\text{syn}}^{(t)} = \sum_{j=1}^{n}\lambda_j w_{ij}^t(1-r)t + \sqrt{\left(\sum_{j=1}^{n}\lambda_j^{\alpha}\left(w_{ij}^{(t)}\right)^2 + \rho\sum_{j\neq k=1}^{n}\lambda_j{}^{\frac{\alpha}{2}}\lambda_k^{\frac{\alpha}{2}}w_{ij}^{(t)}w_{ik}^{(t)}\right)(1+r^{\alpha})\cdot B(t)} + f_i^{(t-1)}(\boldsymbol{\lambda})\tilde{w}_{ii}^{(t-1)} \quad (21)$$

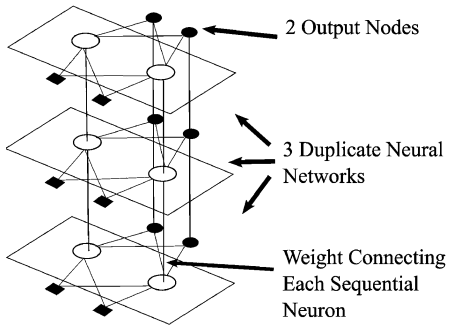

Fig. 6. Graphical representation of time-series network, showing the network unfolded over three time steps. Each network has two input nodes (the black squares), a single layer of spike-rate neurons (the white circles), and two output nodes (the black circles). Here, the network is duplicated over just three time steps, with each unit in each network connected to its corresponding unit in the subsequent network by the weight connection $\bar{w}_{ii}^{(t-1)}$, as represented in (21).

i at time t. In this example, there is no time-delayed recurrent connection between the external inputs in the previous time step and the neurons in the current time step, because inputs external to the network remain constant for all t i.e., $\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}^{(t-1)}$ for all $t = 1, \ldots, T$, and so we refer to these synaptic inputs as $\boldsymbol{\lambda}$. Also for all t, $\lambda^{i,(t)} = \lambda^{i,(t-1)}$, i.e., the center of each spike-rate neuron is constant for all t.

To incorporate the time-delayed inputs, μ and σ are rewritten

$$\begin{aligned}\tilde{\mu}_i^{(t)} &= \left(\sum_{j=1}^{n}\left(\lambda_j - \lambda^i\right) w_{ij}^{(t)}\right)(1-r) \\ &\quad + f_i^{(t-1)}(\boldsymbol{\lambda})\tilde{w}_{ii}^{(t-1)} \\ \left(\tilde{\sigma}_i^{(t)}\right)^2 &= \left(\sum_{j=1}^{n}(\lambda_j - \lambda^i)^{\alpha}\left(w_{ij}^{(t)}\right)^2\right. \\ &\quad \left. + \rho \sum_{j\neq k=1}^{n}\left(\lambda_j - \lambda^i\right)^{\frac{\alpha}{2}}\left(\lambda_k - \lambda^i\right)^{\frac{\alpha}{2}} w_{ij}^{(t)} w_{ik}^{(t)}\right) \\ &\quad \cdot (1 + r^{\alpha}). \end{aligned} \tag{22}$$

For an output node of an RBF-style network, as introduced in (13), the output $y_h^{(t)}$ at time step t will, therefore, be

$$y_h^{(t)}(\boldsymbol{\lambda}) = \sum_{i=1}^{M} w_{hi}^{(t)} z_i^{(t)} \tag{23}$$

with

$$z_i^{(t)} = f_i^{(t)}(\boldsymbol{\lambda}) \tag{24}$$

and

$$f_i^{(t)}(\boldsymbol{\lambda}) = \frac{1}{T_{\text{ref}} + < T_i^{(t)}(r) >} \tag{25}$$

where

$$\langle T_i^{(t)}(r)\rangle = \frac{2}{\tau}\int_{\frac{V_{\text{rest}}\tau - \tilde{\mu}_i^{(t)}}{\sigma_i}}^{\frac{V_{\text{thre}}\tau - \tilde{\mu}_i^{(t)}}{\sigma_i}} g(x)\,dx \tag{26}$$

with h being one of P units on the output layer and $y_h^{(0)} = 0$. For each time step $t = 1, 2, \ldots, T$, each network's weight matrix can be represented as $w^1, w^2, \ldots, w^T$. So we see from (21) that not only does each network receive inputs from external sources and other neurons in the network, but also from neurons in previous time step.

The output node h is the weighted sum of the output it receives from all the neurons in its current network at time t, plus its recurrent connections with previous output neurons h over the previous $t-1$ iterations. In definitions of time-delayed neural networks [33], the weight connections below $w_{hi}^{(t)}$, in our definition given previously, are set to zero. Because this network is a feedforward time-series network, the omission of $w_{hi}^{(t-1)}$ and below is appropriate since only information from the previous time step is directly passed forward in the network.

C. Dynamic Training

The training of this type of recurrent network will require the development of a BPTT-style algorithm. To achieve this, the error function defined in (15) is now defined for a single-layered network. Using the network's output as shown in (23), the total error over the time periods $t = 1, \ldots, T$ is given as

$$E[1,T] = \frac{1}{2}\sum_{t=1}^{T}\sum_{h=1}^{P}\left(d_h^{(t)} - y_h^{(t)}(\boldsymbol{\lambda})\right)^2 \tag{27}$$

where $d_h^{(t)}$ is the desired response of the network to the input pattern $\boldsymbol{\lambda}$ at time t. To minimize this error with respect to the synaptic weights, a modified version of the BPTT algorithm presented in [36] is applied.

1) Propagate information through the network for the time interval $t = 1, \ldots, T_n$ $(1 < T_n \leq T)$, noting at each stage the network's *inputs*, *desired response*, and *synaptic weights*.
2) Perform a backwards pass to calculate the local gradient change on the output layer

$$\delta_h^{(t)} = -\frac{\partial E[1,T]}{\partial\left\langle T_h^{(t)}(r)\right\rangle} \tag{28}$$

for each time step t, beginning with $t = T$ and working backwards.
3) Adjust the weights accordingly using

$$w_{hi}^{(t)} = w_{hi}^{(t)} - \eta F\_w_{hi}^{(t)} \tag{29}$$

where $F\_$ is the ordered derivative [35] of the error function (27) with respect to the weights in the network. This is the feedback of any error in the network's output to those weights $w_{hi}^{(t)}$ that are responsible for that output error. Each of the networks for $t = 1, \ldots, T$ will have their weights adjusted using (29) where

$$\begin{aligned} F\_w_{hi}^{(t)} &= \frac{\partial E[1,T]}{\partial\left\langle T_h^{(t)}(r)\right\rangle}\frac{\partial\left\langle T_h^{(t)}(r)\right\rangle}{\partial w_{hi}^{(t)}} \\ &= \sum_{t=1}^{T}\delta_h^{(t)}\frac{\partial\left\langle T_h^{(t)}(r)\right\rangle}{\partial w_{hi}^{(t)}}. \end{aligned} \tag{30}$$

$(\partial\langle T_h^{(t)}(r)\rangle)/(\partial w_{hi}^{(t)})$ is as defined in (40) and inputs are as stated in (21).

D. Generating Robot Data

To generate a set of robot arm data for training and testing purposes, we tried to approximate some of the dynamics observed in human arm movement. In [1], [2], [12], and [20], it was shown that arm trajectories, between an initial starting point and a target goal, form an approximate straight line. When measuring the velocities of the arm movement along these trajectories they appear *bell-like* in shape. The movement begins with an initial acceleration as the arm moves from its starting point towards the goal, and then begins to decelerate as it approaches this goal.

To achieve an approximation of this type of movement, to allow for the generation of training data, a series of (x_i, y_i) coordinates are generated, which map out the arm's trajectory as it moves from its starting position towards its target goal.

Given the initial starting point coordinates $(0, 0)$ and the final target coordinates (x, y) this straight line is split into n segments

$$x_{\text{seg}} = \frac{x}{n}$$

where x_{seg} is a segment of the line. Using this, a series of coordinate pairs is calculated using

$$x_i = i \times x_{\text{seg}}$$
$$y_i = x_i \frac{y}{x}.$$

As each layer of the network represents an iterative time step of equal length, velocity is, therefore, represented by the variation of the outputs of θ_1 and θ_2 from one time step to the next. Selecting data points from the trajectory data set, will generate the required velocity if the appropriate spacing is chosen between corresponding data points in order for them to produce the desired bell-shaped velocity.

VII. Robot Trials

Graphical results are now presented showing the output when a feedforward time-series network was trained in robot arm control for use in goal location. In these experimental trials, networks of varying sizes were trained to locate a variety of points within the robot arm environment. In each case, the BPTT algorithm presented in Section VI-C was used to train the network. In most experiments, the learning rate η remained fixed at 0.01.

1) Single-Goal Locating: The results presented in Fig. 7 are from one of the experiments performed to test the network's performance when trained to locate a single goal in its environment. In these experiments, the network consisted of two input nodes that were fully connected to two spike-rate neurons, which were in turn connected to two output units. The recurrent connections were only between neurons on similar layers. The time series of the network was for 15 equally spaced time steps. Fig. 6 shows a similar network architecture, unfolded over three time steps.

Fig. 7(a) shows the desired output of the network. Each of the 15 time steps is shown indicating the position and trajectory of the robot arm from a starting point at $(0, 0)$ to its goal at $(1, 1)$. The thick black lines represent the position of the robot's angled two joint arm at each of the 15 time steps. The circles represent the robots end point along the trajectory path. The spacing of these points are included to help represent the velocity change along this trajectory.

The BPTT training algorithm was used to train the network on this trajectory, and Fig. 7(b) shows the network's performance in locating the goal, after 39 training iterations. At this point, the algorithm had trained the network to produce an output within an error tolerance of 0.01. On examining the output in Fig. 7(b), a small error is observed in the line of the trajectory as it approaches the goal.

The error plots during the training are shown in Fig. 7(c) and (d). Fig. 7(c) shows the change in the output error averaged over all the time steps (i.e., each one of the time-series feedforward networks). Early on in the training cycle there was a large output error when compared to the network's target output. However, this rapidly reduced during the training process. Fig. 7(d) presents a more detailed plot of this error, showing the output error for each of the 15 feedforward networks, at each point in the 39 stages of the training process. It can clearly be seen that the most substantial errors occur at those networks which represent the latter half of the time series, with the network at time step 15 contributing the largest error. Network errors are passed forward in the series of feedforward networks, accumulating almost exponentially in networks further along the time sequence.

In Fig. 7(e) and (f), the change in the variance, σ is presented during the training process. An important aspect of this model over classical models is that information is contained within the variance and is affected in the training process. In the experimental models presented here, $r = 1$ effectively making $\mu_i = 0$, thus it is the information contained within this variance, which is used in the control of the robot arm.

2) Multiple Goals: The results presented in Fig. 8 are from the experiment performed to test the network's performance when trained to locate multiple goals in its environment. In these experiments, the networks consisted of two input nodes that were fully connected to a series of 15 spike-rate neurons, which were in turn connected to two output units. The recurrent connections were only between neurons on similar layers. The time series of the network was tested for 15 equally spaced time steps. Due to the computational intensity of the network, we restricted the number of time steps to 15 so that we would be able to test the dynamics of the network within a reasonable time frame.

In the first series of the experiments, the network was trained to locate one goal and then trained to locate a second goal. The results in Fig. 8(a) and (b) show that the network *forgot* how to locate the first goal once it had learned how to locate the second goal. In both instances, the trajectory outputs were identical to those produced by networks that had only been trained on a single goal.

In the second series of the experiments, the network was trained alternatively on the two sets of trajectory data. The training process was allowed to continue until convergence had occurred within a predetermined error tolerance of 0.05. Fig. 8(c) shows two output trajectory paths for the robot arm after training

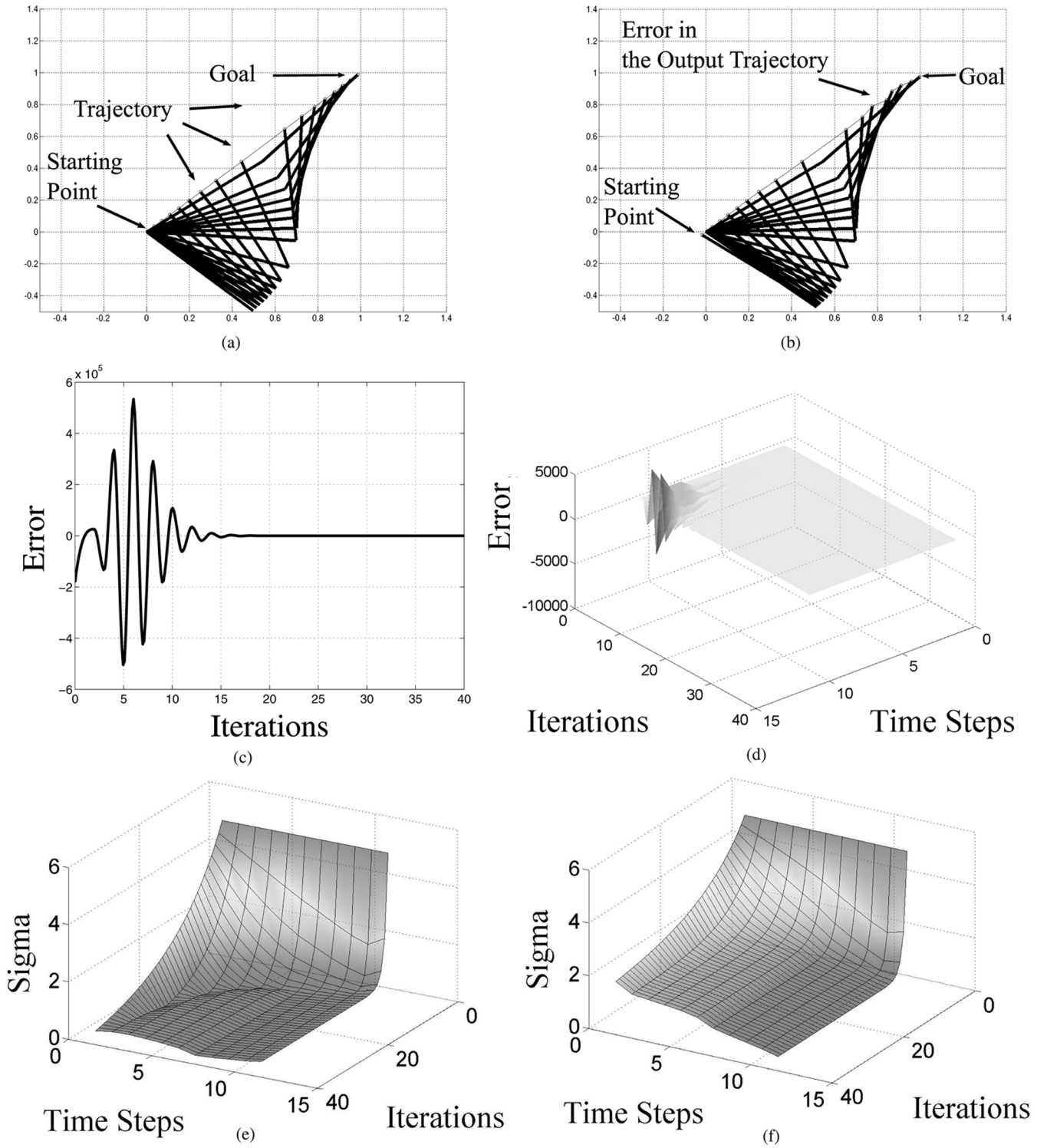

Fig. 7. (a) Target position and trajectory of the robot arm in locating the object goal over 15 equal time intervals. The position of the robot arm, at each of the 15 time steps, is indicated by the thick black line that represents the segments l_1 and l_2 as indicated in Fig. 5. (b) Trajectory the robot arm traces out after 39 training iterations. (c) Total error output of the network during the training process. (d) Error over for each network, at each time step, is shown over the 39 training iterations. (e) and (f) Change in σ for each of the two output units during the 39 training iterations.

the network for 600 iterations. The input goals were the pairs of coordinate points $(1, -1)$ and $(1, 0)$. Both trajectory paths show a degree of error when compared to the expected straight line paths of the desired response. There is a small error still present towards the end of the trajectory path in both trials; however, the spacing of the arm's position for each of the time step intervals is more accurate. We saw evidence of this in the function approximation experiments in Section V. In those experiments, the basic shape of the target curve was quickly modeled by the network, but the exact positioning of the output curve required longer training before it exactly matched the target data. In the robot arm experiments, the shape of the curve would represent the

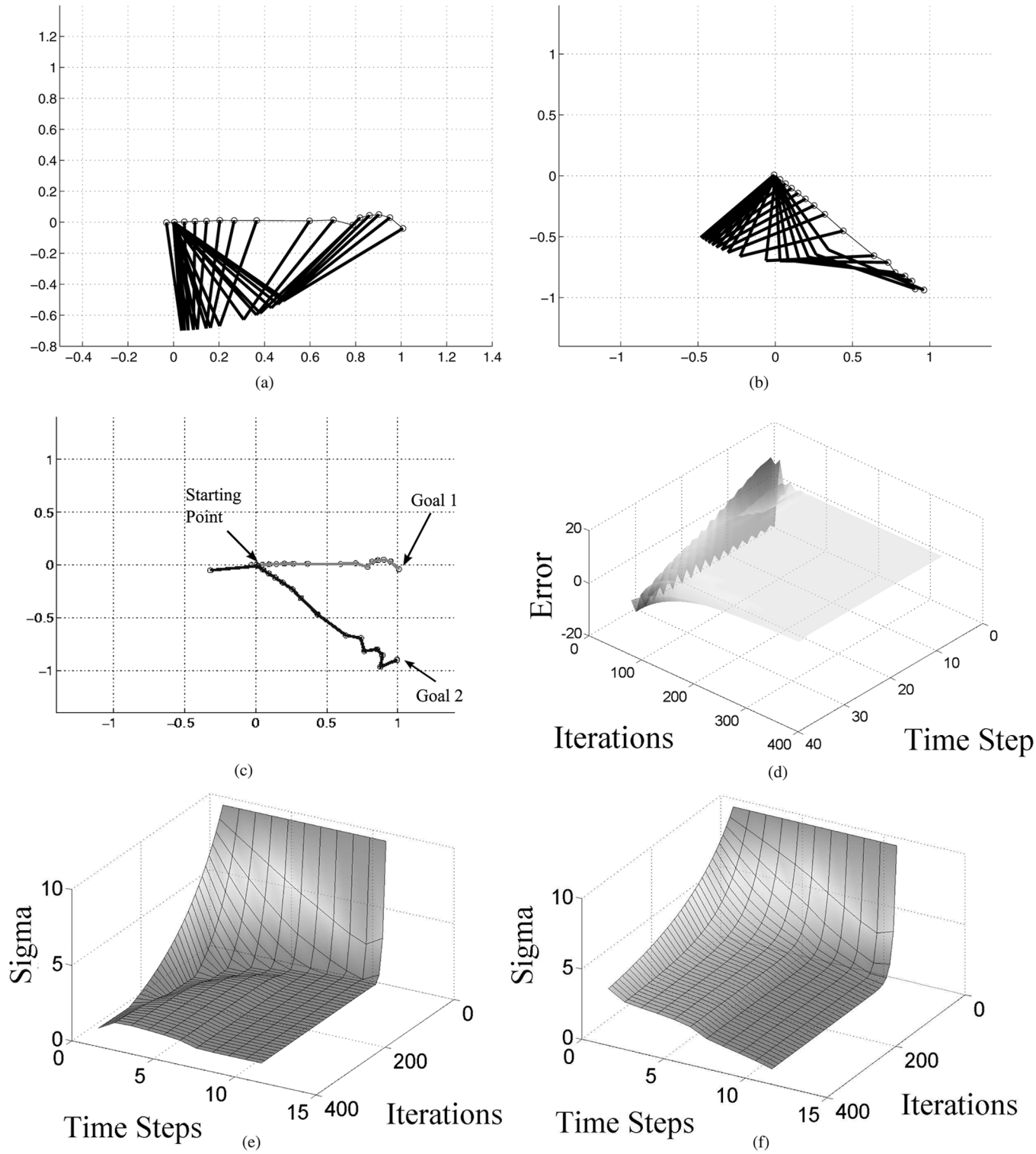

Fig. 8. (a) Trajectory traced out by the robot arm in locating the goal at the Cartesian coordinates $(1, 0)$. (b) Trajectory traced out by the robot arm in locating the goal at the Cartesian coordinates $(1, -1)$. The position of the robot arm, at each of the 15 time steps, is indicated by the thick black line, which represents the segments l_1 and l_2 as indicated in Fig. 5. The controller was initially trained to locate goal 1 and then goal 2. (c) Trajectory paths when the controller was alternatively trained on goals 1 and 2. (d) Error output during this training process. (e) and (f) Change in σ for each of the two output units over 400 training iterations.

angular change of the robot arm's joints over time. This general shape appears to be modeled at the earlier stages of the network's training. The majority of the training appears to be matching the detail of the movement rather than just the trend in the data, i.e., the path of the trajectory.

Fig. 8(d) is a graphical plot of the overall network error occurring during the process the network was trained to locate the two goals. The training cycle took 600 iterations. However, a large *acceptable* error of 0.05 was used to ensure convergence within a reasonable time period.

Fig. 8(e) and (f) shows the change in the variance σ during the training process. As in the single-goal location task, these biologically plausible models use the information, contained within the variance, as part of their computation.

VIII. Conclusion

We have expanded on the single-neuron model we defined and tested in [26], and developed learning algorithms for use with spike-rate neuronal networks, in particular, possible applications for use in engineering tasks. The definition of the spike-rate model showed a relationship between the synaptic stimulus and the neuron's spike-rate activity.

We investigated the similarity between the single spike-rate neuron model's output and the basis function of an RBF model, which we supported in our design of an RBF-style network of spike-rate neurons. The element of synaptic variance σ^2, included in the model, proved to be an important component in the computational abilities of the spike-rate neuron. When we examined the case of the mean input $\mu = 0 (r = 1)$, we saw that σ^2 alone had computational properties sufficient enough for the model to be applied to a series of nonlinear tasks. Indeed, it was this computational component of synaptic input, which we used in the two sets of experiments included here.

In presenting a generalized learning algorithm similar in approach to the error minimization used in backpropagation, we have shown how it is possible to train a spike-rate network model, and apply it to engineering tasks. Though the network structures were less biologically plausible than the original design of the spike-rate model, the networks proved to be good approximators capable of learning a variety of mathematical functions. These networks quite quickly learned the trends in sets of training data, though the dynamics of the model meant that exact data matching took a large number of training iterations.

We were also able to show that this single-layered network model can be extended into a recurrent time-series structure, and that it is possible to solve a dynamic control task with biologically plausible neuron models. By expanding on our general definition of a spike-rate training algorithm introduced in [26], we adopted a similar BPTT style training algorithm applicable for use on time-series spiking networks. Though computationally expensive, the algorithm and network model were able to solve the robot arm control task and locate the target in the experiment.

The unfolded feedforward network structure presented in Section VI-B lacked the real biological plausibility that one might have expected when dealing the spike-rate neuron model. In these networks, the recurrent connections used the firing rate outputs from one layer, as part of the synaptic input in the subsequent layer. Though the firing rate has an important role to play in understanding the input–output relationship of a neuron, its relevance as part of the synaptic input is meaningless. By constructing the networks in this way, we were, however, able to show a possible approach to a dynamic problem using spike-rate neuron.

Both these experiments present us with the opportunity to expand on our initial designs of a spike-rate network. By retaining more of the features of the IF neuron, i.e., diffusion approximations for the synaptic input renewal processes, we would want to have similar approximations for the outputs from the neural model, retaining μ and σ in the output spike train. We would then have an opportunity of building networks of IF neurons, where inputs–outputs from each layer in the network retain more of the biologically observed spiking features. We could then develop the spike-rate learning rules presented here, for use on more biologically plausible networks of spiking neurons.

Thus far, however, the model has primarily been tested on spike-rate neurons with $r = 1$. This was done in order to test the computational effectiveness of the variance term. We propose an initial extension to the work by including the case when $r = 0.5$ and 0. This will have the effect of including the mean *and* the variance in the computation. Comparing how these models perform in engineering tasks, with results from similar classical AI models, will help us identify any possible computational advantage the spike-rate model has over its classical counterpart. Could modeling the mean signal and the "noise" add to the model's computational power?

Furthermore, the idea of employing the variance in computation is not new. It has been extensively discussed in the literature on stochastic resonance. In a typical scenario of stochastic resonance, the output is maximized when the variance is, usually, very small in value. In our set up here, however, the variance and the mean are coupled, which implies that we cannot arbitrarily reduce the noise, while keeping the mean (the first-order statistics) unchanged. Although we do use the important properties of second-order statistics, our approach is very different from that of stochastic resonance [10].

Appendix I
Learning Rule for the Single-Neuron Model

We include here the learning rule for the single-neuron model. The model consists of a single layer of $i = 1, \ldots, N$ spike-rate neurons, each with $j = 1, \ldots, m$ synaptic inputs λ_j. The learning algorithm, shown here, seeks to minimize the error between the spike-rate network's output $f_i(\lambda_j)$ and the desired target output d_i, i.e.,

$$E = \sum_{i=1}^{N} (f_i(\lambda) - d_i)^2. \tag{31}$$

To minimize such an error, as in biological systems, we apply an approach similar to backpropagation

$$\Delta w_{ij} = -\eta \Delta_{ij} E$$

where η is the learning rate and

$$\Delta_{ij} E = \frac{\partial E}{\partial w_{ij}} \tag{32}$$

with w_{ij} being the synaptic weight connections between units i and j. Here we are seeking to minimize E.

Taking (32) and differentiating, we get

$$\frac{\partial E}{\partial w_{ij}} = 2(f_i(\lambda) - d_i) \frac{\partial f_i(\lambda)}{\partial w_{ij}}. \tag{33}$$

Using f as defined in (7)

$$\frac{\partial f_i(\lambda)}{\partial w_{ij}} = -\frac{1}{(T_{\text{ref}} + < T_i(r) >)^2} \frac{\partial < T_i(r) >}{\partial w_{ij}}. \tag{34}$$

Now, using the mean ISI $< T_i(r) >$, as defined in (8), we get

$$\begin{aligned}\frac{\partial < T_i(r) >}{\partial w_{ij}} &= \frac{2}{\tau}\frac{\partial}{\partial w_{ij}}\left[g\left(\frac{V_{\text{thre}}\tau - \mu_i}{\sigma_i}\right) - g\left(\frac{V_{\text{rest}}\tau - \mu_i}{\sigma_i}\right)\right] \\ &= \frac{2}{\tau}\left[g\left(\frac{u_i}{\sigma_i}\right)\left(\frac{u'_{i,j}\sigma_i - u_i\sigma'_{i,j}}{\sigma_i^2}\right) \right. \\ &\qquad \left. -g\left(\frac{v_i}{\sigma_i}\right)\left(\frac{v'_{i,j}\sigma_i - v_i\sigma'_{i,j}}{\sigma_i^2}\right)\right] \end{aligned} \tag{35}$$

with $u_i = V_{\text{thre}}\tau - \mu_i$, $v_i = V_{\text{rest}}\tau - \mu_i$, $u'_{i,j} = \partial u_i/\partial w_{ij}$, and $\sigma'_{i,j} = \partial \sigma_i/\partial w_{ij}$. Taken together, we have obtained a learning rule (34) and (35) for a single-layered spike-rate neuronal network. The weight update rule in (35) uses both the mean and the variance of synaptic input. The important feature is that any change in the firing rate depends on the changes in these two components of synaptic input.

Appendix II
The Output Layer of a Multilayered Spike-Rate Network

As a proof concept, we present the case for the second stage of the learning rule, discussed in Section IV-B1, when $y_h(\boldsymbol{\lambda}) = f_h(\boldsymbol{\lambda})$, i.e., the case when output units are not linear weighted sums of hidden spike rate neurons as defined in (13) but are themselves spike-rate neurons. In this situation, (16) now becomes

$$\begin{aligned}\frac{\partial E}{\partial w_{hi}} &= \frac{\partial E}{\partial f_h(\boldsymbol{\lambda})}\frac{\partial f_h(\boldsymbol{\lambda})}{\partial \langle T_h(r)\rangle}\frac{\partial \langle T_h(r)\rangle}{\partial w_{hi}} \\ &= \delta_h \frac{\partial \langle T_h(r)\rangle}{\partial w_{hi}} \end{aligned} \tag{36}$$

where

$$\delta_h = \frac{\partial E}{\partial f_h(\boldsymbol{\lambda})}\frac{\partial f_h(\boldsymbol{\lambda})}{\partial \langle T_h(r)\rangle}. \tag{37}$$

Equation (37) is the local error gradient for output neuron h. The amount by which w_{hi} must change is given by the *delta rule* where a proportion η of the rate change in error E is taken. This is defined as the rate of change of the error with respect to the synaptic weight connection w_{hi}

$$\Delta w_{hi} = -\eta \frac{\partial E}{\partial w_{hi}}. \tag{38}$$

For the output layer, the delta rule is defined as

$$\Delta w_{hi} = -\eta \delta_h \frac{\partial \langle T_h(r)\rangle}{\partial w_{hi}} \tag{39}$$

so taking $< T_h(r) >$, where $< T_h(r) >$ is defined in (8)

$$\begin{aligned}\frac{\partial < T_h(r) >}{\partial w_{hi}} &= \frac{2}{\tau}\frac{\partial}{\partial w_{hi}}\left[g\left(\frac{V_{\text{thre}}\tau - \mu_h}{\sigma_h}\right) - g\left(\frac{V_{\text{rest}}\tau - \mu_h}{\sigma_h}\right)\right] \\ &= \frac{2}{\tau}\left[g\left(\frac{u_h}{\sigma_h}\right)\left(\frac{u'_{h,i}\sigma_h - u_h\sigma'_{h,i}}{\sigma_h^2}\right) \right. \\ &\qquad \left. -g\left(\frac{v_h}{\sigma_h}\right)\left(\frac{v'_{h,i}\sigma_h - v_h\sigma'_{h,i}}{\sigma_h^2}\right)\right] \end{aligned} \tag{40}$$

with $u_h = V_{\text{thre}}\tau - \mu_h$, $v_h = V_{\text{rest}}\tau - \mu_h$, $u'_{h,i} = \partial u_h/\partial w_{hi}$, and $\sigma'_{h,i} = \partial \sigma_h/\partial w_{hi}$, and where

$$\begin{aligned}\mu'_{h,i} &= \lambda_i(1-r) \\ \sigma'_{h,i} &= \frac{(\lambda_i^{\alpha} w_{hi} + \lambda_i{}^{\frac{\alpha}{2}}\lambda_{i_k}{}^{\frac{\alpha}{2}}\sum_{i_k \neq i} w_{hi_k})(1 + r^{\alpha})}{\sigma}. \end{aligned} \tag{41}$$

Here, λ_i is the input to neuron h from another neuron i, but where neuron $i_k \neq i$.

1) The Hidden Layer: For those spike-rate neurons on the hidden layer, the local error gradient will be

$$\begin{aligned}\delta_i &= \frac{\partial E}{\partial f_i(\boldsymbol{\lambda})}\frac{\partial f_i(\boldsymbol{\lambda})}{\partial \langle T_i(r)\rangle} \\ &= -\frac{\partial E}{\partial f_i(\boldsymbol{\lambda})}\frac{1}{(T_{\text{ref}} + \langle T_i(r)\rangle)^2} \end{aligned} \tag{42}$$

and so using the chain rule, the partial derivative of the error E becomes

$$\begin{aligned}\frac{\partial E}{\partial f_i(\boldsymbol{\lambda})} &= \frac{\partial E}{\partial f_h(\boldsymbol{\lambda})}\frac{\partial f_h(\boldsymbol{\lambda})}{\partial \lambda_i} \\ &= -\sum_{h=1}^{P}(d_h - f_h(\boldsymbol{\lambda}))\frac{\partial f_h(\boldsymbol{\lambda})}{\partial \lambda_i} \end{aligned} \tag{43}$$

where λ_i is the output from spike-rate neuron i. In this definition of a multilayered feedforward spike-rate network, the output from the $i = 1, \ldots, M$ spike-rate neurons on the hidden layer is represented by the term λ_i, and this is the synaptic input for the $h = 1, \ldots, P$ spike-rate neurons on the output layer.

Differentiating the rate output for unit h with respect to the input it receives from i, as shown in (43)

$$\begin{aligned}\frac{\partial f_h(\boldsymbol{\lambda})}{\partial \lambda_i} &= \frac{-1}{(T_{\text{ref}} + \langle T_h(r)\rangle)^2}\frac{\partial \langle T_h(r)\rangle}{\partial \lambda_i} \\ &= \frac{-2}{\tau(T_{\text{ref}} + \langle T_h(r)\rangle)^2} \\ &\quad \times \left[g\left(\frac{u_h}{\sigma_h}\right)\left(\frac{\tilde{u}'_{h,i}\sigma_h - u_h\tilde{\sigma}'_{h,i}}{\sigma_h^2}\right) \right. \\ &\qquad \left. -g\left(\frac{v_h}{\sigma_h}\right)\left(\frac{\tilde{v}'_{h,i}\sigma_h - v_h\tilde{\sigma}'_{h,i}}{\sigma_h^2}\right)\right] \end{aligned} \tag{44}$$

with $u_h = V_{\text{thre}}\tau - \mu_h$, $v_h = V_{\text{rest}}\tau - \mu_h$, $\tilde{v}'_{h,i}, \tilde{u}'_{h,i} = -\partial\mu_h/\partial\lambda_i$, and $\tilde{\sigma}'_{h,i} = \partial\sigma_h/\partial\lambda_i$. Denoting these derivatives with respect to λ_i

$$\begin{aligned}\tilde{\mu}'_{h,i} &= w_{hi}(1-r) \\ \tilde{\sigma}'_{h,i} &= \frac{\alpha[2\lambda_i^{\alpha-1}w_{hi}^2 + \lambda_i{}^{\frac{\alpha}{2}-1}w_{hi}\sum_{i_k \neq i}\lambda_i{}^{\frac{\alpha}{2}}w_{hi_k}](1 + r^{\alpha}).}{4\sigma} \end{aligned}$$

Using the definition of δ_h given in (37), δ_i, for hidden neuron i, is given as

$$\begin{aligned}\delta_i &= -\frac{\partial f_i(\boldsymbol{\lambda})}{\partial\langle T_i(r)\rangle}\sum_{h=1}^{P}\frac{(d_h - f_h(\lambda_i))}{(T_{\text{ref}} + \langle T_h(r)\rangle)^2}\frac{\partial\langle T_h(r)\rangle}{\partial\lambda_i} \\ &= -\frac{\partial f_i(\boldsymbol{\lambda})}{\partial\langle T_i(r)\rangle}\sum_{h=1}^{P}\delta_h\frac{\partial\langle T_h(r)\rangle}{\partial\lambda_i} \qquad (45)\end{aligned}$$

and thus, the weight correction for the hidden neuron

$$\Delta w_{ij} = -\eta\delta_i\frac{\partial\langle T_i(r)\rangle}{\partial w_{ij}}. \qquad (46)$$

APPENDIX III
LOCAL GRADIENT CHANGE IN BPTT

The computation of the $\delta_h^{(t)}$'s in (28) first require a calculation at time $t = T$, then the remainder can be calculated *backwards* to time $t = 1$. This is calculated as follows:

$$\delta_h^{(t)} = \left[d_h^{(t)} - y_h^{(t)}\left(v_h^{(t)}\right)\right]y'^{(t)}_h\left(v_h^{(t)}\right) \qquad (47)$$

where $y'^{(t)}_h(\cdot)$ is the derivative of $y_h^{(t)}$, as defined in (23), with respect to its activation $v_h^{(t)}$, as substituted here for clarity. Calculating the δ's for the networks in layers $t = T-1, \ldots, 1$

$$\begin{aligned}\delta_h^{(t)} &= \frac{\partial E[t, t+1]}{\partial v_h^{(t)}} \\ &= \frac{\partial E(t)}{\partial v_h^{(t)}} + \frac{\partial E(t+1)}{\partial v_h^{(t)}} \\ &= \left[d_h^{(t)} - y_h^{(t)}\left(v_h^{(t)}\right)\right]\frac{\partial y_h^{(t)}\left(v_h^{(t)}\right)}{\partial v_h^{(t)}} + \frac{\partial E(t+1)}{\partial v_h^{(t)}} \\ &= \left[d_h^{(t)} - y_h^{(t)}\left(v_h^{(t)}\right)\right]y'^{(t)}_h\left(v_h^{(t)}\right) \\ &\quad + \sum_{k=1}^{P}\left[d_k^{(t+1)} - y_k^{(t+1)}\left(v_k^{(t+1)}\right)\right]\frac{\partial y_k^{(t+1)}\left(v_k^{(t+1)}\right)}{\partial v_h^{(t)}} \\ &= y'^{(t)}_h\left(v_h^{(t)}\right)\left(d_h^{(t)} - y_h^{(t)}\left(v_h^{(t)}\right) + \sum_{k=1}^{P}w_{kh}^{(t)}\delta_k^{(t+1)}\right)\end{aligned}$$

substituting for (47) and rewriting (23) such that

$$y_h^{(t)}(\boldsymbol{\lambda}) = \sum_{i=1}^{M}w_{hi}^{(t)}y_i^{(t)}\left(v_i^{(t-1)}\right) \qquad (48)$$

where

$$y_i^{(t)}\left(v_i^{(t-1)}\right) = z_i^{(t)}. \qquad (49)$$

REFERENCES

[1] W. Abend, E. Bizzi, and P. Morasso, *Human Arm Trajectory Formation*, vol. 105, pp. 331–348, 1982.

[2] C. G. Atkenson and J. M. Hollerbach, "Kinematic features of unrestrained vertical arm movements," *J. Neurosci.*, vol. 5, pp. 2318–2330, 1985.

[3] S. M. Bohte, J. N. J. N. Kok, and H. H. L. Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 7–37, 2002.

[4] C. Christodoulou, G. Bugmann, and T. G. Clarkson, "A spiking neuron model: Applications and learning," *Neural Netw.*, vol. 15, no. 7, pp. 891–908, 2002.

[5] R. O. Duda and P. E. Hart, *Pattern Classification And Scene Analysis*. New York: Wiley, 1973.

[6] J. Feng, "Is the integrate-and-fire model good enough?," *Neural Netw.*, vol. 14, pp. 955–975, 2001.

[7] J. Feng, H. Buxton, and Y. C. Deng, "Training the integrate-and-fire model with the informax principle I," *J. Physics*, vol. 35, pp. 2379–2394, 2002.

[8] J. Feng, Y. C. Deng, and E. Rossoni, "Dynamics of moment neuronal networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 73, 2006, 049106-1–049106-17.

[9] J. Feng, Y. Sun, H. Buxton, and G. Wei, "Training the integrate-and-fire model with the informax principle II," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 326–336, Mar. 2003.

[10] J. Feng and B. Tirozzi, "Stochastic resonance tuned by correlations in neural models," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 61, no. 4, pp. 4207–4211, 2000.

[11] J. Feng and H. C. Tuckwell, "Optimal control of neuronal activity," *Phys. Rev. Lett.*, vol. 91, 2003, 018101-1–018101-6.

[12] T. Flash and N. Hogan, "The coordination of arm movements: An experimentally confirmed mathematical model," *J. Neurosci.*, vol. 5, no. 7, pp. 1688–1703, 1985.

[13] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[14] R. L. Hardy, "Multiquadratic equations of topography and other irregular surfaces," *J. Geophys. Res.*, vol. 76, pp. 1905–1915, 1971.

[15] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley, 1949.

[16] A. L. Hodgkin and A. F. Huxley, "A quantitative description of ion currents and its applications to conductance and excitation in nerve membranes," *J. Physiol. (London)*, vol. 117, pp. 500–544, 1952.

[17] J. J. B. Jack, D. Nobel, and R. Tsien, *Electric Current Flow in Excitable Cells*, 1st ed. Oxford, U.K.: Oxford Univ. Press, 1975.

[18] H. Jäger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, pp. 78–80, 2004.

[19] P. Joshi and W. Maass, "Movement generation with circuits of spiking neurons," *Neural Comput.*, vol. 17, no. 8, pp. 1715–1738, 2005.

[20] Y. Uno, M. Kawato, and R. Suzui, "Formation and control of optimal control trajectories in human multijoint arm movements: Minimum torque change model," *Biol. Cybern.*, vol. 61, pp. 89–101, 1989.

[21] W. M. Kistler, W. Gerstner, and J. L. van Hemmen, "Reduction of Hodgkin-Huxely equations to a single-variable threshold model," *Neural Comput.*, vol. 9, pp. 1015–1045, 1997.

[22] B. Knight, "Dynamics of encoding in a population of neurons," *J. General Physiol.*, vol. 59, pp. 734–766, 1972.

[23] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.

[24] C. A. Micchelli, "Interpolation of scattered data: Distance matrices and conditionally positive definitive functions," *Constructive Approx.*, vol. 2, pp. 11–22, 1986.

[25] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, DC: Spartan, 1962.

[26] P. Rowcliffe, J. Feng, and H. Buxton, "Spiking perceptions," *IEEE Trans. Neural Netw.*, vol. 17, no. 3, pp. 803–807, May 2006.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. William, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McCelland, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, the PDF Research Group.

[28] E. Salinas and T. J. Sejnowski, "Correlated neuronal activity and the flow of neural information," *Nature Rev. Neurosci.*, vol. 2, pp. 539–550, Aug. 2001.
[29] R. Stein, "Some models of neuronal variability," *J. Biophysics*, vol. 7, pp. 37–68, 1967.
[30] R. B. Stein, "The frequency of nerve action potentials generated by applied currents," *Proc. R. Soc. Lond. A, Math. Phys. Sci.*, vol. 167, pp. 64–86, 1967.
[31] H. C. Tuckwell, *Introduction to Theoretical Neurobiology*. Cambridge, U.K.: Cambridge Univ. Press, 1988, vol. 2.
[32] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: A case study," *Inf. Process. Lett.*, vol. 95, no. 6, pp. 521–528, 2005.
[33] R. L. Watrous and L. Shastri, "Learning phonetic features using connectionist networks: An experimentation in speech recognition," in *Proc. 10th Int. Joint Conf. Artif. Intell.*, 1987, pp. 851–854.
[34] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Dept. Appl. Math., Harvard Univ., New Haven, CT, 1974.
[35] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[36] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Comput.*, vol. 2, pp. 490–501, 1990.

Phill Rowcliffe, received the D.Phil. degree from the Department of Informatics, Sussex University, Sussex, U.K.

His research interests, within computational neuroscience, are in the information transmission within the interspike interval and the application of mathematical learning rules.

Jianfeng Feng, received the B.Sc., M.Sc., and Ph.D. degrees from the Department of Probability and Statistics, Peking University, China.

Currently, he is the Director of the Centre for Computational System Biology, Fudan University, China and the Professor in Computational Biology, Warwick University, Warwick, U.K. He has published around 120 journal papers, and aims for publications exclusively in journal with an impact factor around three or above.