A Recurrent Neural Network for Modelling Dynamical Systems

Coryn A.L. Bailer-Jones[*][†], David J.C. MacKay[‡]
Cavendish Laboratory, University of Cambridge,
Madingley Road, Cambridge, CB3 OHE, England

Philip J. Withers [§]
Department of Materials Science and Metallurgy, University of Cambridge,
Pembroke Street, Cambridge, CB2 3QZ, England

ABSTRACT

We introduce a recurrent network architecture for modelling a general class of dynamical systems. The network is intended for modelling real-world processes in which empirical measurements of the external and state variables are obtained at discrete time points. The model can learn from multiple temporal patterns, which may evolve on different timescales and be sampled at non-uniform time intervals. We demonstrate the application of the model to a synthetic problem in which target data are only provided at the final time step. Despite the sparseness of the training data, the network is able not only to make good predictions at the final time step for temporal processes unseen in training, but also to reproduce the sequence of the state variables at earlier times. Moreover, we show how the network can infer the existence and role of state variables for which no target information is provided. The ability of the model to cope with sparse data is likely to be useful in a number of applications, particularly the modelling of metal forging.

Network: Computation in Neural Systems, 1998, **9**, 531–547

1. Introduction

Many real-world processes can be represented as dynamical systems. A dynamical system can be described in terms of the evolution of one or more *state variables* in response to one or more *external variables*. In this paper we will consider dynamical systems which can be modelled with the equation

$$\frac{\partial \mathbf{v}(\tau)}{\partial \tau} = \mathbf{F}(\mathbf{v}(\tau), \mathbf{x}(\tau)) \tag{1}$$

where $\mathbf{x}$ are the external variables, $\mathbf{v}$ are the state variables, and $\mathbf{F}$ is a non-linear static function. In terms of this definition, the external variables are causally independent of the state variables. Given a set of initial conditions, $\mathbf{v}(\tau = 0)$, and the time sequence of the external variables, $\mathbf{x}(\tau)$, equation 1 determines the evolution of the state variables, i.e. $\mathbf{v}(\tau)$ at $\tau > 0$. It is the problem of learning the time sequence of the state variables which we address in this paper.

An example of a dynamical system is the hot forging of a piece of metal. When a material is forged, its macro and micro structural properties are altered through mechanical deformation. The deformation is described by the forging parameters, such as the strain, strain rate and temperature, all of which are generally functions of time. These are the external variables. Typical state variables of interest are the material grain size and extent of recrystallisation, as well as macroscopic properties such as strength and toughness.

The modelling of materials forging is an important but difficult task (see, e.g. Bailer-Jones et al. 1997, 1998). While it is usually straightforward to measure most of the external variables during forging, many of the state variables are difficult to measure while forging takes place. Therefore we will usually only have measurements of the state variables at the beginning and end of the forging process, giving us relatively few target data with which to develop our model. Additionally, there are some state variables, such as the dislocation density, which are important in describing the evolution of the material, yet cannot be measured at all in most practical applications. As these

[*] email: calj@mpia-hd.mpg.de
[†] Present address: Max-Planck-Institut für Astronomie, Königstuhl 17, D-69117 Heidelberg, Germany
[‡] email: mackay@mrao.cam.ac.uk
[§] email: pjw12@cus.cam.ac.uk

that it is nonetheless possible to infer the existence and role of such "unmeasured" state variables.

The problem we address is the modelling of dynamical systems on the basis of empirical data. That is, we wish to learn the underlying dynamical system for a process from which our only knowledge comes from incomplete measurements of $\mathbf{x}$ and $\mathbf{v}$. In the following sections we shall introduce our recurrent network architecture for modelling equation 1. After deriving the training rule, we shall demonstrate the performance of the model through its application to a synthetic problem.

2. The Model

Many types of recurrent neural network architectures have been proposed for modelling time-dependent phenomena. These include discrete time networks (e.g. Jordan 1986; Rumelhart, Hinton & Williams 1986; Stornetta, Hogg & Huberman 1988; Williams & Zipser 1989; Elman 1990) and continuous time networks (e.g. Pineda 1987, 1988; Pearlmutter 1989). A number of authors have looked specifically at the application of recurrent networks to modelling dynamical processes (e.g. Robinson & Fallside 1991; Parlos, Chong & Atiya 1994; Nerrand et al. 1994). Non-recurrent networks have also been used in some situations to model time series (e.g. Chakraborty et al. 1992). We are interested in modelling real-world dynamical systems in which discrete time measurements are made of the relevant variables at known time points, or *epochs*. Therefore, our model is a discrete time network model of a continuous time system.

Our recurrent network is based on a first-order solution to equation 1. The Taylor expansion about a point, $\mathbf{v}(\tau-\delta\tau)$, is

$$\mathbf{v}(\tau) = \mathbf{v}(\tau-\delta\tau) + \frac{\partial \mathbf{v}(\tau-\delta\tau)}{\partial \tau}\delta\tau \ . \tag{2}$$

(We shall assume that the separations between measurement epochs, $\delta\tau$, are sufficiently small to allow us to drop the additional terms of order $(\delta\tau)^2$.) This solution is modelled using the recurrent network shown in Figure 1, and can be considered as a network in two stages. The first stage is a standard feedforward network which implements function $\mathbf{F}$ in equation 1 directly. The inputs to this stage are the external inputs, $\mathbf{x}(\tau)$, and the state variables, $\mathbf{v}(\tau)$, at a certain epoch τ, and the outputs are

$$\mathbf{y}(\tau) = \mathbf{F}(\mathbf{v}(\tau), \mathbf{x}(\tau)) \ . \tag{3}$$

These outputs are the time derivatives of the state variables at epoch τ. The second stage of the network is the recurrent part, and implements equation 2 via the one-to-one connections from the network outputs to the state variables (or recurrent inputs). The weights of these connections are set to $\delta\tau$. The state variables also feed back into themselves with unit weights. The cycle then repeats for every epoch for which external inputs are defined. What we label as "outputs" in Figure 1 are not outputs in the sense that we usually read values from them. Rather we give them this name to make the analogy with feedforward networks.

In the rest of this paper the indices k, l and m will be used exclusively to count over nodes in the state variable, external input and hidden layers respectively. As there are one-to-one connections between the output and state variable layers, k will also be used to label the nodes in the output layer. These indices will also be used to label weights between the nodes, e.g. w_{km} is the weight between the k^{th} state variable and the m^{th} hidden node. The indices i and j will be used to label arbitrary nodes. V, X and H will denote the sets of nodes in the state variable, external input and hidden layers respectively. We introduce the integer t to enumerate epochs at time τ_t, for $t = 1, 2, \ldots, T$.

The activation of the m^{th} node in the hidden layer is given by

$$h_m(t-1) = f[p_m(t-1)] \tag{4}$$

where

$$p_m(t-1) = \sum_k w_{km} v_k(t-1) + \sum_l w_{lm} x_l(t-1) + w_{bm} x_b \ . \tag{5}$$

The input-hidden transfer function, f, is the tanh function to introduce non-linearity. The bias value, x_b, is fixed to unity. The outputs from the network are given by

$$y_k(t-1) = g[q_k(t-1)] \tag{6}$$

where

$$q_k(t-1) = \sum_m w_{mk} h_m(t-1) + w_{bk} h_b \ . \tag{7}$$

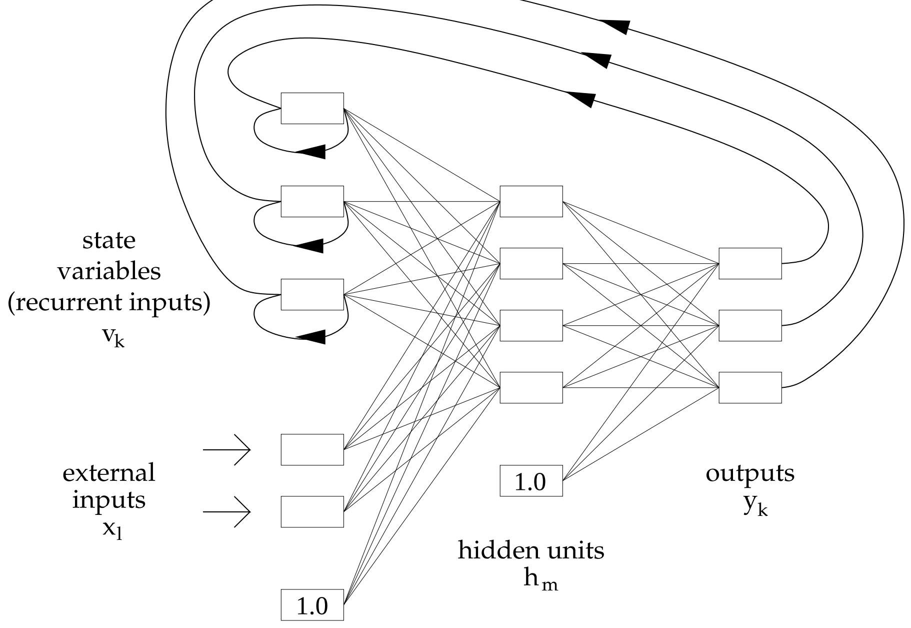


Fig. 1: A recurrent neural network architecture for modelling dynamical systems. Data flows counter-clockwise around the network. An external input vector, $\mathbf{x}(t-1)$, and state variable vector, $\mathbf{v}(t-1)$, are the inputs to the feedforward stage of the network. The "outputs", $\mathbf{y}(t-1)$, are the time derivatives of the state variables. The one-to-one connections from the outputs to the state variables provide the means of evaluating the state variables at the next epoch, $\mathbf{v}(t)$. All connections and the two bias nodes are shown.

The hidden-output transfer function, g, is linear to permit the outputs to be of any size. The bias value, h_b, is fixed to unity.

Everything which occurs in the first stage of the network does so at the same epoch, $t-1$. The recurrent part of the network gives the state variable at the next epoch,

$$v_k(t) = v_k(t-1) + y_k(t-1)\Delta(t) \tag{8}$$

where $\Delta(t) = \tau_t - \tau_{t-1}$. Thus the external inputs and state variables at epoch $t-1$ give rise to the state variable at t.

This network will produce a complete sequence of the state variables, $\mathbf{v}(1), \mathbf{v}(2), \ldots, \mathbf{v}(T)$, given initial conditions on the state variables, $\mathbf{v}(0)$, the external input sequence, $\mathbf{x}(0), \mathbf{x}(1), \ldots, \mathbf{x}(T-1)$, and of course the time steps, $\Delta(1), \Delta(2), \ldots, \Delta(T)$. We shall refer to a single such sequence of the variables as a *temporal pattern*.

3. Training

The network is trained in the conventional supervised manner by minimizing an error function with respect to the network weights. Optimization of the weights is achieved using an extension of the backpropagation scheme of Rumelhart, Hinton & Williams (1986). Our network architecture is similar to that of Jordan (1986), but differs in the important respect that in training our network the error derivatives are propagated to later epochs: although the recurrent weights themselves are not trainable, they can nonetheless be used to propagate errors.

The network is trained with one or more temporal patterns. The necessary training data for a single temporal pattern is the sequence of external inputs and associated epoch separations, the initial state variables, and at least one target value. While target values can be specified on any node in the network, we will limit ourselves to the practical situation in which our targets are values of the state variables. Note that the model does not need targets at every epoch: In modelling forging we are often only able to measure a target state variable at the end of the forge. We shall see in section 6 that this is often sufficient to learn the underlying dynamical system.

We will now consider the propagation of training errors for a single temporal process. The error in the k^{th} state variable at epoch t is

$$e_k(t) = v_k(t) - T_k(t) \tag{9}$$

where $T_k(t)$ is the target at epoch t. If no target is defined, $e_k(t) = 0$. The error at epoch t is

$$E(t) = \frac{1}{2}\sum_k \beta_k[e_k(t)]^2 \tag{10}$$

Differentiating equation 10 with respect to an arbitrary weight w_{ij}

$$\frac{\partial E(t)}{\partial w_{ij}} = \sum_k \beta_k e_k(t) \frac{\partial v_k(t)}{\partial w_{ij}} \quad . \tag{11}$$

The last term is obtained by differentiating equation 8 with respect to w_{ij}

$$\frac{\partial v_k(t)}{\partial w_{ij}} = \frac{\partial v_k(t-1)}{\partial w_{ij}} + \frac{\partial y_k(t-1)}{\partial w_{ij}} \Delta(t) \quad . \tag{12}$$

From equation 6

$$\frac{\partial y_k(t-1)}{\partial w_{ij}} = g'[q_k(t-1)] \frac{\partial q_k(t-1)}{\partial w_{ij}} \tag{13}$$

and from equation 7

$$\begin{aligned} \frac{\partial q_k(t-1)}{\partial w_{ij}} &= \sum_m \left(\frac{\partial w_{mk}}{\partial w_{ij}} h_m(t-1) + w_{mk} \frac{\partial h_m(t-1)}{\partial w_{ij}} \right) + \frac{\partial w_{bk}}{\partial w_{ij}} h_b \\ &= \delta_{kj} \delta_{iH} h_i(t-1) + \sum_m w_{mk} \frac{\partial h_m(t-1)}{\partial w_{ij}} \end{aligned} \tag{14}$$

where δ_{kj} is the conventional delta function and δ_{iH} is defined as

$$\begin{aligned} \delta_{iH} &= 1 \text{ if } i \in H \\ &= 0 \text{ otherwise} \end{aligned}$$

i.e. $\delta_{iH} = 1$ if i is a node in the hidden layer. When $i = b$ (bias node), $h_i(t-1) = h_b = 1$.

From equation 4

$$\frac{\partial h_m(t-1)}{\partial w_{ij}} = f'[p_m(t-1)] \frac{\partial p_m(t-1)}{\partial w_{ij}} \tag{15}$$

and from equation 5

$$\begin{aligned} \frac{\partial p_m(t-1)}{\partial w_{ij}} &= \sum_k \left(\frac{\partial w_{km}}{\partial w_{ij}} v_k(t-1) + w_{km} \frac{\partial v_k(t-1)}{\partial w_{ij}} \right) + \sum_l \frac{\partial w_{lm}}{\partial w_{ij}} x_l(t-1) + \frac{\partial w_{bm}}{\partial w_{ij}} x_b \\ &= \delta_{mj} \delta_{iV} v_i(t-1) + \sum_k w_{km} \frac{\partial v_k(t-1)}{\partial w_{ij}} + \delta_{mj} \delta_{iX} x_i(t-1) \end{aligned} \tag{16}$$

where δ_{mj} is the conventional delta function, and

$$\begin{aligned} \delta_{iV} &= 1 \text{ if } i \in V \\ &= 0 \text{ otherwise} \end{aligned}$$

and

$$\begin{aligned} \delta_{iX} &= 1 \text{ if } i \in X \\ &= 0 \text{ otherwise} \quad . \end{aligned}$$

Writing

$$\frac{\partial v_k(t)}{\partial w_{ij}} \equiv \Phi_k^{ij}(t) \tag{17}$$

and

$$\frac{\partial y_k(t)}{\partial w_{ij}} \equiv \Psi_k^{ij}(t) \tag{18}$$

equation 12 becomes

$$\Phi_k^{ij}(t) = \Phi_k^{ij}(t-1) + \Psi_k^{ij}(t-1) \Delta(t) \quad . \tag{19}$$

$$\Psi_k^{ij}(t-1) = g'[q_k(t-1)] \left(\delta_{kj}\delta_{iH} h_i(t-1) + \sum w_{mk} \frac{\partial h_m(t-1)}{\partial w_{ij}} \right) \tag{20}$$

and equation 16 into 15

$$\frac{\partial h_m(t-1)}{\partial w_{ij}} = f'[p_m(t-1)] \left(\delta_{mj}[\delta_{iV} v_i(t-1) + \delta_{iX} x_i(t-1)] + \sum_{k'} w_{k'm} \Phi_{k'}^{ij}(t-1) \right) \ . \tag{21}$$

Combining equations 20 and 21 we get

$$\Psi_k^{ij}(t-1) = g'[q_k(t-1)] \times \tag{22}$$
$$\left[\delta_{kj}\delta_{iH} h_i(t-1) + \sum_m w_{mk} f'[p_m(t-1)] \left(\delta_{mj}[\delta_{iV} v_i(t-1) + \delta_{iX} x_i(t-1)] + \sum_{k'} w_{k'm} \Phi_{k'}^{ij}(t-1) \right) \right]$$

This equation and equation 19 give a recurrence relation for $\Phi_k^{ij}(t)$, the gradient of v_k with respect to any network weight, in terms of $\Phi_k^{ij}(t-1)$. To solve this, the system must be initialised with an initial value of $\Phi_k^{ij}(t)$, which is taken to be zero because the initial state variables are independent of the network weights. The gradient of the error for a single temporal pattern at epoch t is then

$$\frac{\partial E(t)}{\partial w_{ij}} = \sum_k \beta_k e_k(t) \Phi_k^{ij}(t) \ . \tag{23}$$

These equations allow us to compute an error gradient at every epoch. Note that a target does not have to be defined at every epoch in order to be able to propagate $\Phi_k^{ij}(t)$.

We will often want to train the network on several temporal patterns. This can be done quite simply by applying the above recursion relations to each pattern separately. How we then update the weights is a matter of choice. For example, we could update the weights using the gradient calculated at each epoch for each pattern, or we could cumulate the gradients at all epochs for each pattern before updating. This former method is similar to the Real Time Recurrent Learning method of Williams & Zipser (1989) extended to multiple temporal patterns. We choose instead to do "batch" learning in which the gradient is cumulated over all epochs for all patterns, and then update the weights using a conjugate gradient optimizer.

4. Regularization and a Bayesian Perspective

We regularize training of the network using weight decay, that is by penalizing large weights. The total error which is to be minimized is then

$$\begin{aligned} E_T &= E_d + E(\mathbf{w}) \\ &= \sum_{\mu,t} E_\mu(t) + \frac{1}{2} \sum_g \left(\alpha_g \sum_{i,j \in g} w_{ij}^2 \right) \ . \end{aligned} \tag{24}$$

The first term, E_d, is the sum of errors in equation 10 over all epochs for all patterns (μ). The weight decay term, $E(\mathbf{w})$, is the sum over all the weights collected into four groups: state variable to hidden weights; external input to hidden weights; bias to hidden weights; hidden to output weights. Each group has a value, α_g, associated with it which controls the scale of the weights.

A Bayesian probabilistic interpretation of training identifies $\mathrm{P}(D|\mathbf{w}) = e^{-E_d}$ as the probability of the data given the weights (MacKay 1995). The weight decay term is written as $\mathrm{P}(\mathbf{w}) = e^{-E(\mathbf{w})}$, the prior probability over the weights. Using Bayes' theorem

$$\mathrm{P}(\mathbf{w}|D) \propto \mathrm{P}(D|\mathbf{w})\mathrm{P}(\mathbf{w}) \tag{25}$$

we see that $\mathrm{P}(\mathbf{w}|D) \propto e^{-E_T}$ is the posterior probability of the weights given the data. It is this quantity which we maximise when training the regularized network. From equation 24, we can consider $\mathrm{P}(\mathbf{w})$ as a product of Gaussian prior probability distribution over the different groups of weights, with means zero and standard deviations $\sigma_g = \alpha_g^{-1/2}$. Similarly, $\mathrm{P}(D|\mathbf{w})$ can be considered as a Gaussian distribution over the target values, with standard deviation, or noise level, of $\sigma_k = \beta_k^{-1/2}$. Schemes exist for inferring the optimum values of α_g and β_k directly from the data (MacKay 1995). However, in section 6 we shall fix these hyperparameters to reasonable values. It should be noted that the α parameters for the hidden to output weights are not independent of a rescaling of time or of the state variables, as the outputs ($\mathbf{y}$) are the time derivatives of the state variables. It is for this reason also that the hidden-output transfer function must provide an arbitrary large output range.

The network offers a number of features which may be useful in practical applications. The first is that the network can learn from more than one temporal pattern. Moreover, these temporal processes can consist of different numbers of epochs, and the time interval between the epochs does not have to be constant. This is useful for learning from temporal patterns which evolve on very different time scales but nonetheless correspond to the same dynamical system. An example is the forging of two pieces of material which are identical other than being of very different sizes and masses.

A second feature is that it is not necessary to have a target value at every epoch when training the network: in addition to an initial $\mathbf{v}$ value, only one target $\mathbf{v}$ is required. Furthermore, it is not necessary to have an external input defined at those epochs where a $\mathbf{v}$ target is defined. Again this is useful in practical situations, as it means that we can measure our process (external input) variables independently of the state variables.

In modelling dynamical systems, there may well be important state variables which cannot be measured. For example, the dislocation density is important in determining the evolution of a material microstructure during mechanical processing, but is time consuming to measure in practice. Our model can use state variables for which there are no target values: we shall refer to these as "unmeasured" state variables. Their purpose is to propagate any additional state variable information required to make correct predictions of the measured state variables. As there are no target values for these unmeasured state variables, they may not even correspond to any physical variables, although we may be able to give them a physical interpretation. An example of their use will be given in the next section.

6. Application to Synthetic Problems

We now demonstrate the performance of the model on a synthetic problem in which there are two external input variables, $x_1(\tau)$ and $x_2(\tau)$, and two state variables, $v_1(\tau)$ and $v_2(\tau)$, defined by

$$\frac{\partial v_1}{\partial \tau} = x_1 - 2v_1 + 8v_2 - x_1 v_1 \tag{26}$$

$$\frac{\partial v_2}{\partial \tau} = x_2 - 5v_1 + v_2 - x_2 v_2 \ . \tag{27}$$

The autonomous part of this dynamical system (that with the external inputs set to zero) is a decaying harmonic oscillator, with period 1.0 and e^{-1} damping timescale 2.0. The x terms make the temporal processes somewhat more complex, but the overall oscillatory behaviour is retained.

To mimic real processes in which the external inputs are often constrained to be positive (e.g. temperature), the x_1 and x_2 input sequences were generated from constrained random walks: x_1 (x_2) changes with a probability per unit time of 0.65 (0.999) by a random amount uniformly distributed between -0.5 and $+0.5$ (-1 and $+1$). The modulus of x is then taken to ensure a positive sequence. The initial v values were randomly selected from a uniform distribution between -1 and $+1$.

One hundred synthetic temporal patterns were simulated numerically, with different sequences for x_1 and x_2 and different initial values of v_1 and v_2. The sequences were generated from $\tau = 0$ to $\tau = 8$ inclusive, with a constant epoch separation of $\Delta(t) = 0.1$. In all of the following examples, the network had eight hidden nodes and was trained on 50 of the processes. All results shown are for application to the other 50 processes not seen during training. In training, the network is only ever given the initial state variables and targets at the final epoch: it is never given intermediate targets. The hyperparameters were set to $\beta = 40000$ and $\alpha = 0.01$. Although these are reasonable, no attempt has been made to optimise them, manually or otherwise.

Problem 1 This network has two state variables, one for each of v_1 and v_2. Figure 2 shows that the trained network makes excellent predictions of v_1 and v_2 at the final epoch. It is interesting also to ask whether the network has managed to correctly learn the entire $\mathbf{v}$ sequences for the test data, or whether it has learned some simpler form for $\mathbf{F}$ in equation 1 which just gives correct $\mathbf{v}$ values at the final epoch. Figure 3 shows the $\mathbf{x}$ input sequences, and the predicted $\mathbf{v}$ sequences in comparison with the true sequences, and shows that the sequence predictions are generally very accurate. However, some of the predicted sequences deviate from the true sequences at early times ($\tau \lesssim 2$). This is because the target values (at the final epoch) lie in a smaller region of the state space than do the state variables at earlier epochs (Figure 4): The network has only learned the dynamical system (or rather a good approximation to it) in this smaller region of the state space, and its extrapolations to unfamiliar parts of the state space are somewhat poorer. Nonetheless we see that for this particular problem the network is able to recover after $\tau \approx 2$.

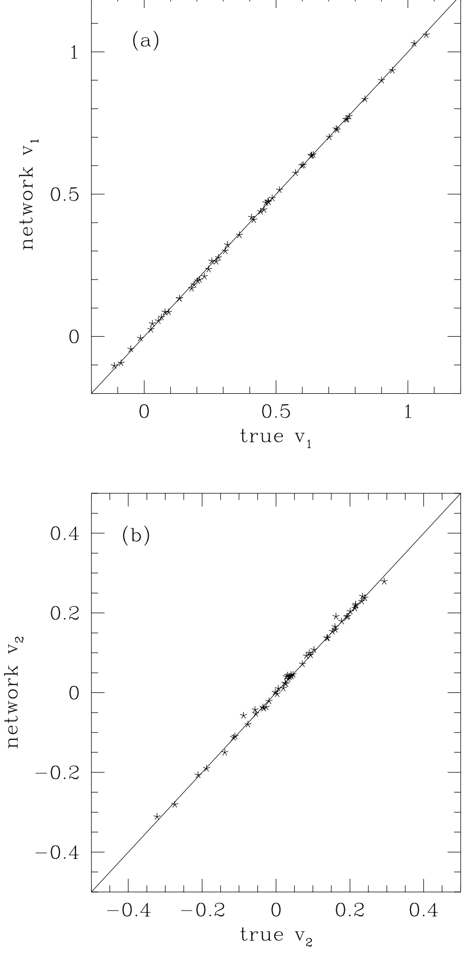


Fig. 2: Network predictions at the final epoch ($\tau = 8$) for a network with two measured state variables (problem 1), (a) v_1 and (b) v_2. The rms errors are 0.0067 and 0.0084 for v_1 and v_2 respectively.

Problem 2 The dynamical system in equations 26 and 27 consists of two coupled state variables. Thus if one state variable (v_1) is removed from the network, we would not expect the network to make good predictions of v_2 either at the final epoch (where there is a target) or at intermediate epochs. We have confirmed this experimentally by training a network with only one state variable with targets for v_2: no v_1 data is seen by the network. Figure 5 shows the scatter plot of the resultant network predictions. The correlation is not zero, probably because v_1 and v_2 are strongly coupled, but the correlation is considerably worse than in Figure 2b. Moreover, Figure 6 shows that the network has failed to learn the underlying dynamical system in any region of the state space.

Problems 3 & 4 Problem 3 is similar to problem 2, but now we add to the network an “unmeasured” state variable for which we provide *no* target information. The goal is that the network will use the unmeasured state variable to propagate any information required to make good predictions for v_2. The initial value of the unmeasured state variable was set to zero for all temporal patterns. The same four target sequences as used in the previous two problems are shown in Figure 7, along with the networks predictions for v_2 and the unmeasured state variable. In order to achieve such good predicted sequences for v_2 (other than at early epochs as explained earlier), the network must be using the unmeasured state variable, because we saw in the previous problem that without this the network is unable to predict the v_2 sequences. While we would not expect the unmeasured state variable to replicate the behaviour of v_1 (as no target information was provided for it), it nonetheless emulates it closely. Learning v_1 exactly is not necessary, as any monotonic transformation of v_1 carries the same information. This can be seen better in Figure 8, in which we have re-trained the same network from different initial random weights (problem 4). Again the sequences for v_2 are very close to the true ones, as are the final values (Figure 9a), but this time the network has discovered a different sequence for the unmeasured state variable. Figure 9b shows that there is an excellent correlation between the values of the unmeasured state variable and the true v_1 variable at the final epoch. This demonstrates that the network can use the extra degree of freedom provided by the unmeasured state variable to

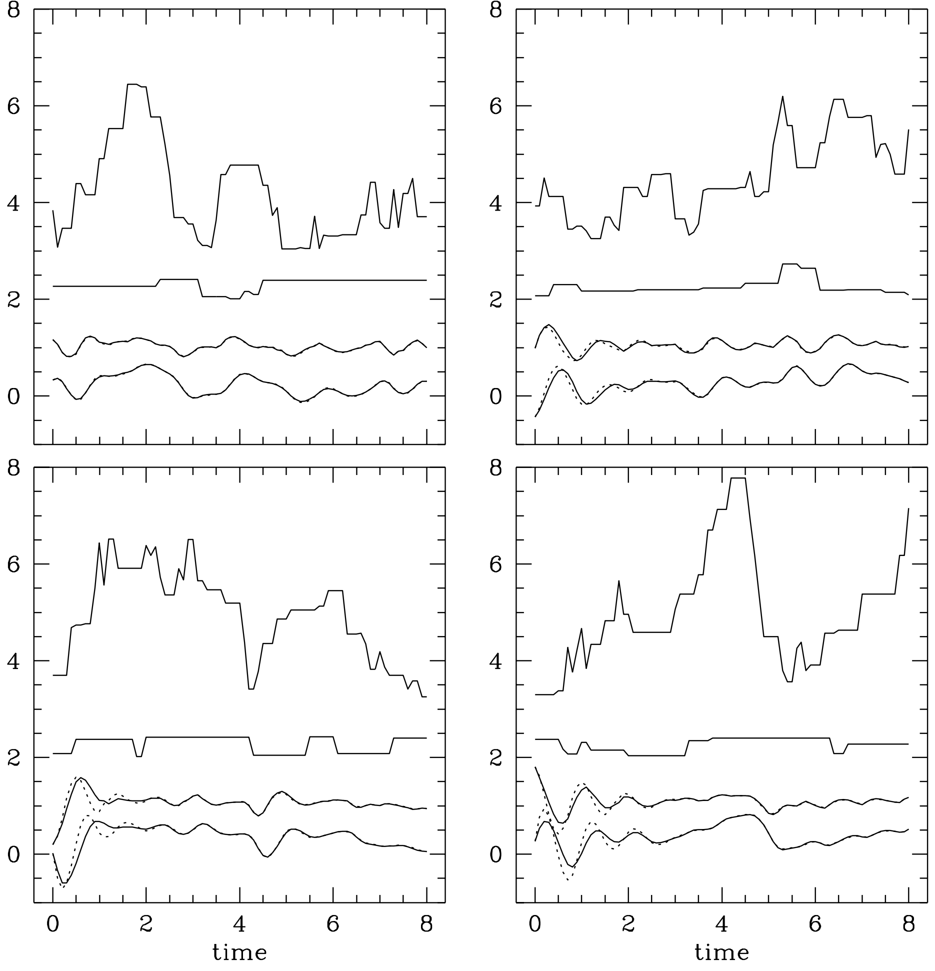


Fig. 3: Four samples of temporal patterns from the synthetic problem in equations 26 and 27. In each panel the solid lines from bottom to top are v_1, v_2, x_1 and x_2. For v_1 and v_2 the solid lines are the network predictions from problem 1. The lower and upper dashed lines are the true v_1 and v_2 sequences respectively. For clarity, the sequences for v_2 (network and true), x_1 and x_2 have been offset from their true positions by 1, 2 and 3 vertical units respectively.

infer the behaviour of a missing variable without being supplied any target information for it.

Problems 5 & 6 In the final two problems, we train a network with one measured state variable (v_2) and two unmeasured state variables, to see what the network does with the extra (redundant) degree of freedom. The sequence predictions are shown in Figure 10. The v_2 sequence is again predicted well. What is interesting here is that the network appears to be using *both* unmeasured state variables. Figure 11a shows the correlations between these two unmeasured variables and v_1. Neither correlation is as strong as in problem 4, when we used only one unmeasured state variable. Indeed, it appears from inspection of Figure 10 that some linear combination of the two unmeasured state variables will give a better correlation with the true v_1, and this is shown in Figure 11b. While the final epoch values are still predicted well (Figure 11c), the performance is slightly worse than in problem 4, indicating that the network may have slightly overfitted. This could be alleviated by increasing the values of the appropriate α weight decay parameters.

If we retrain the same network from different initial random weights, it is possible to get a very different solution for the two unmeasured state variables. One such alternative solution is shown in Figure 12 (problem 6) where we see that the network has made use of only one of the unmeasured state variables. As can be seen in Figure 13, the predictions for v_2 are as good as the previous network solution (problem 5). Indeed, the reproducibility of the v_2 predictions at the final epoch is excellent.

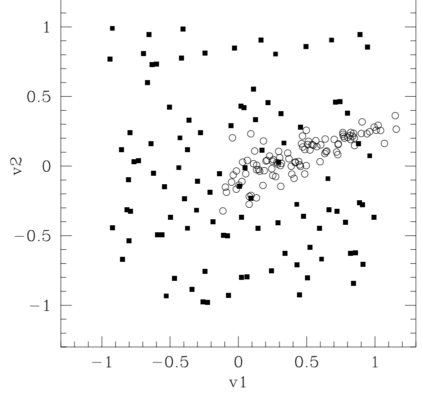


Fig. 4: Distributions of the true state variables for the synthetic problem in equations 26 and 27. The filled squares are the initial $\mathbf{v}$ values, and the open circles the final (target) $\mathbf{v}$ values. The initial (v_1,v_2) values for the the four examples in Figure 3 (0.33,0.17) (top left), (−0.44,−0.01) (top right), (0.02,−0.80) (bottom left) and (0.27,0.80) (bottom right).

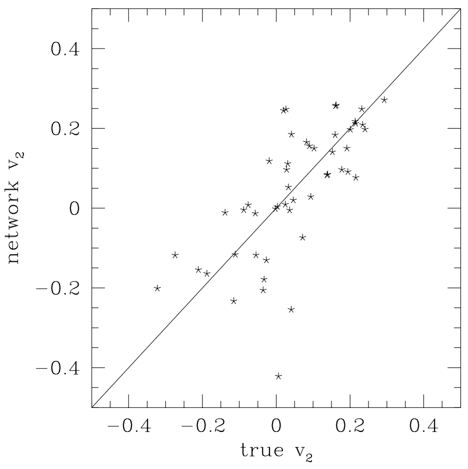


Fig. 5: Network predictions at the final epoch ($\tau = 8$) for a network with only one measured state variable (v_2) (problem 2). The v_1 state variable has been omitted from the network, leading to poorer performance. The rms error is 0.117.

We have tested the network on a number of other problems. In one problem we used a network with two measured state variables and one (redundant) unmeasured state variable. The predictions for v_1 and v_2 were almost as good as in problem 1, and the unmeasured state variable was unused, as with one of the state variables in problem 6. We have also had success using variable $\Delta(t)$ terms and more complex synthetic problems in which the time derivatives were non-linear functions of the state variables.

7. Conclusions

We have introduced a discrete time recurrent neural network for modelling dynamical systems. The network architecture is very general, and should be applicable to a wide range of real-world problems. We have shown that the network is capable of learning a dynamical system based on temporally sparse measurements of the state variables. The network can learn from multiple temporal patterns which may be sampled at non-constant time intervals. We have show how the network can infer the existence of a relevant but omitted state variable using an unmeasured state variable for which no target data is provided. The unmeasured state variable was shown to be well correlated with the omitted state variable, thus allowing us to give it a physical interpretation. This ability is likely to be useful in practical situations, such as forging, in which some important state variables can often not be measured. Our model allows the evolution of these usually “hidden” variables to be monitored.

It should be noted that the synthetic problem we presented was a noise-free problem. Noise could be introduced into any of the external inputs, the initial and target state variables and the $\Delta(t)$ time steps. This, and the application to real forging data, will be the subject of future work.

Successful application of the model requires that the separations between the measurement epochs, $\Delta(t)$, are

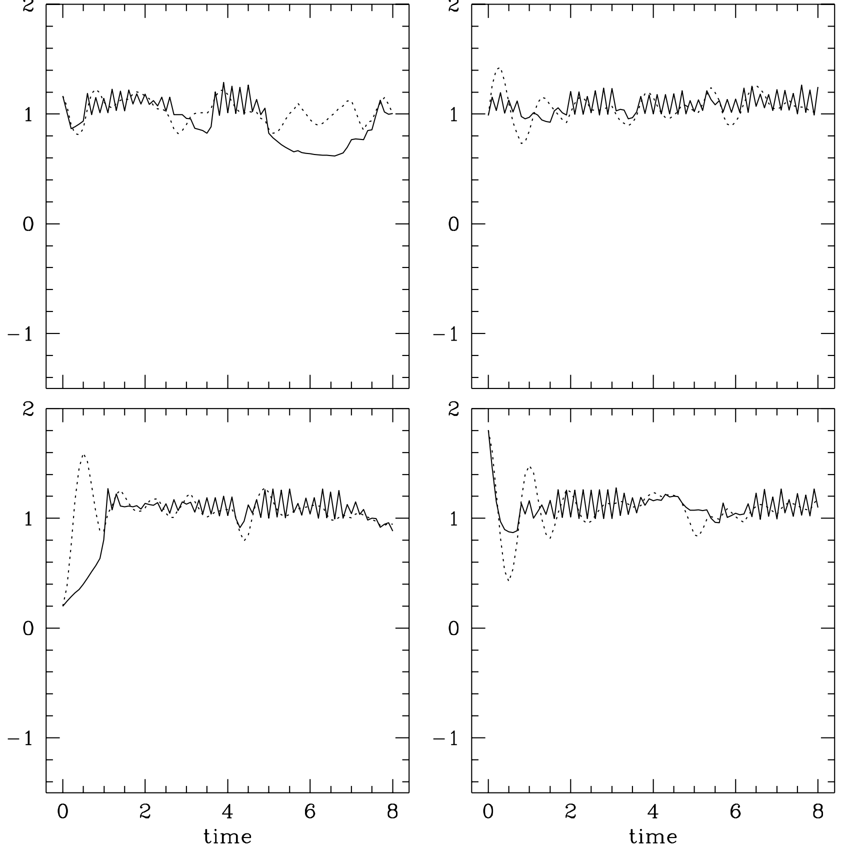


Fig. 6: State variable sequence predictions for a network with only one state variable (v_2) (problem 2). The true v_2 sequences (dashed lines) are the same as in Figure 3. The solid lines are the corresponding v_2 sequences predicted by the network. The sequences have been offset by 1 vertical unit to aid comparison with other diagrams.

small compared to the characteristic time scale of the dynamical system being modelled. This is necessary to satisfy the approximation in equation 2. If measurements cannot be obtained at sufficient frequency, this could be accommodated by adding an extra set of output nodes to give higher order terms in the Taylor expansion: these nodes give $\partial^2 \mathbf{v}(\tau)/\partial \tau^2$ and would be connected to the state variables with connection strength $\Delta(t)^2$.

Acknowledgements

This work was supported by the DERA and the EPSRC (grant number GR/L10239).

References

[1] Bailer-Jones C A L, Sabin T J, MacKay D J C and Withers P J 1997 Prediction of deformed and annealed microstructures using Bayesian neural networks and Gaussian processes *Proceedings of the Australasia Pacific Forum on Intelligent Processing and Manufacturing of Materials* vol 2, eds T Chandra et al. (Watson Ferguson & Co., Brisbane) pp 913–919

[2] Bailer-Jones C A L, MacKay D J C, Sabin T J, and Withers P J 1998 Static and dynamic modelling of materials forging *Australian Journal on Intelligent Information Processing Systems* **5**(1) 10–17

[3] Chakraborty K, Mehrotra K, Mohan C K and Ranka S 1992 Forecasting the behaviour of multivariate time series using neural networks *Neural Networks* **5** 961–970

[4] Elman J L 1990 Finding structure in time *Cognitive Science* **14** 179–211

[5] Jordan M I 1986 Attractor dynamics and parallelism in a connectionist sequential machine *Proc. of the Eight Ann. Conf. of the Cognitive Science Society* (Erlbaum, Hillsdale NJ) 531–546

[6] D J C MacKay 1995 Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks *Network: Computation in Neural Systems* **6** 469–505

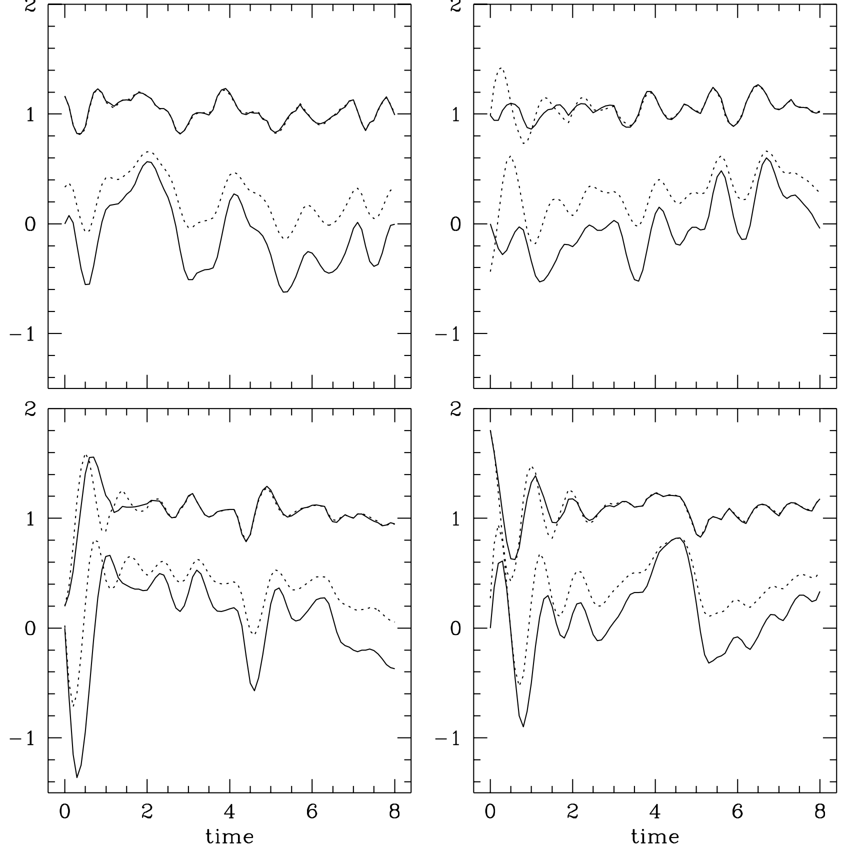


Fig. 7: State variable sequence predictions for a network with one measured state variable (v_2) and an unmeasured state variable (problem 3). The lower and upper dashed lines in each panel are the same true v_1 and v_2 sequences (respectively) as shown in Figure 3. The lower solid line is the sequence of the unmeasured state variable. The upper solid line is the sequence for v_2 as predicted by the network. The v_2 sequences (network and true) have been offset by 1 vertical unit.

[7] Nerrand O, Roussel-Ragot P, Urbani D, Personnaz L and Dreyfus G 1994 Training recurrent neural networks: why and how? An illustration in dynamical process modeling *IEEE Transactions on Neural Networks* **5**(2) 178–184

[8] Parlos A G, Chong K T and Atiya A F 1994 Application of the recurrent multilayer perceptron in modelling complex process dynamics *IEEE Transactions on Neural Networks* **5**(2) 255–266

[9] Pearlmutter B A 1989 Learning state space trajectories in recurrent neural networks *Neural Computation* **1** 263–269

[10] Pineda F J 1987 Generalization of back-propagation to recurrent neural networks *Physical Review Letters* **59**(19) 2229–2232

[11] Pineda F J 1988 Dynamics and architecture for neural computation *Journal of Complexity* **4** 216–245

[12] Robinson A J and Fallside F 1991 A recurrent error propagation network speech recognition system *Computer Speech and Language* **5** 259–274

[13] Rumelhart D E, Hinton G E and Williams R J 1986 Learning internal representations by error propagation *Parallel distributed processing: explorations in the microstructure of cognition* eds D E Rumelhart, J L McClelland and the PDP Research Group (MIT Press, Cambridge MA) pp 318–362

[14] Stornetta W S, Hogg T and Huberman B A 1988 A dynamical approach to temporal pattern processing *Neural Information Processing Systems* ed D Z Anderson (AIP, New York) pp 750–759

[15] Williams R J and Zipser D 1989 A learning algorithm for continually running fully recurrent neural networks *Neural Computation* **1** 270–280

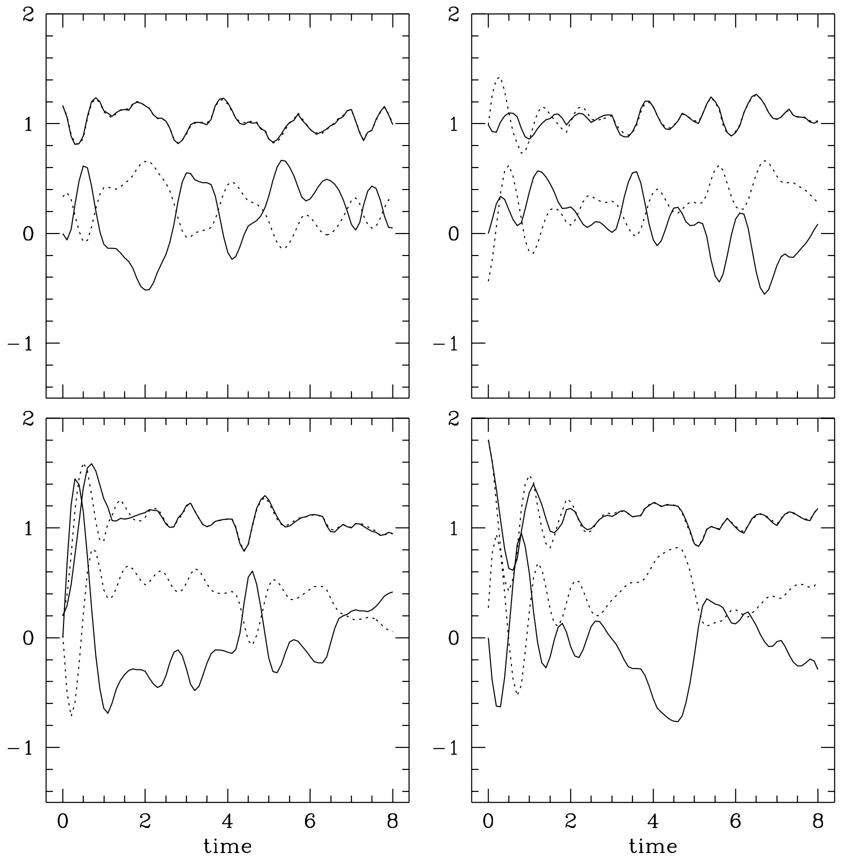
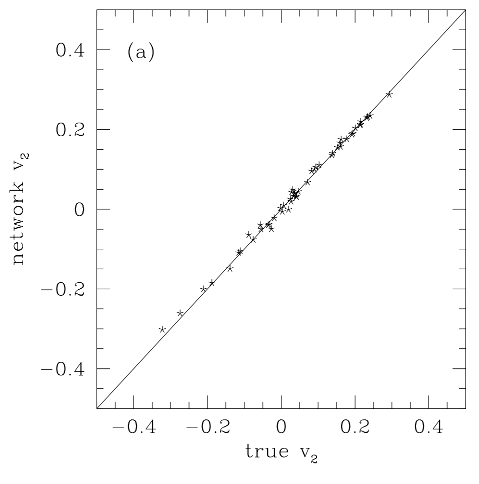


Fig. 8: Same plot as in Figure 7 but using a network trained from different initial random weights (problem 4).



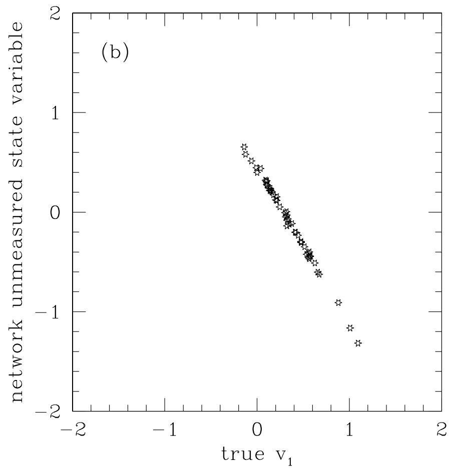


Fig. 9: Network predictions at the final epoch ($\tau = 8$) for a network with one measured state variable (v_2) and an unmeasured state variable (problem 4). (a) Network v_2 plotted against true v_2. The rms error is 0.0095 (b) The unmeasured state variable learned by the network plotted against the v_1 state variable, no values of which were given to the network in training.

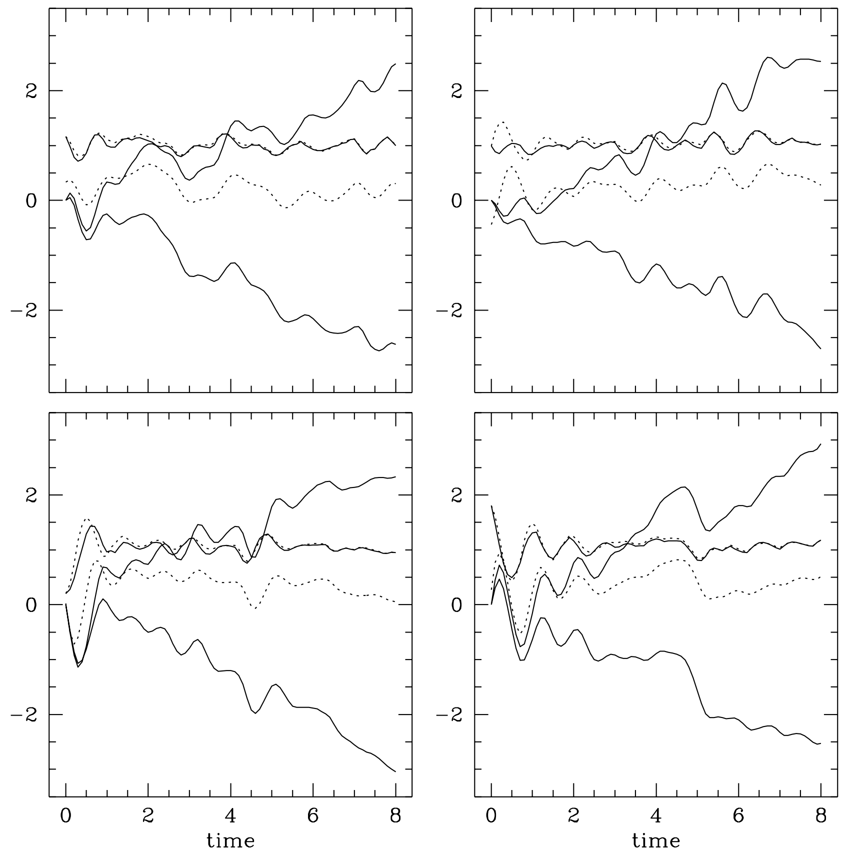


Fig. 10: State variable sequence predictions for a network with one measured state variable (v_2) and two unmeasured state variables (problem 5). The lower and upper dashed lines in each panel are the same true v_1 and v_2 sequences (respectively) as shown in Figure 3. The upper solid line at $\tau = 0$ is the sequence for v_2 as predicted by the network. The other two solid lines are the sequences of the unmeasured state variables. The v_2 sequences (network and true) have been offset by 1 vertical unit.

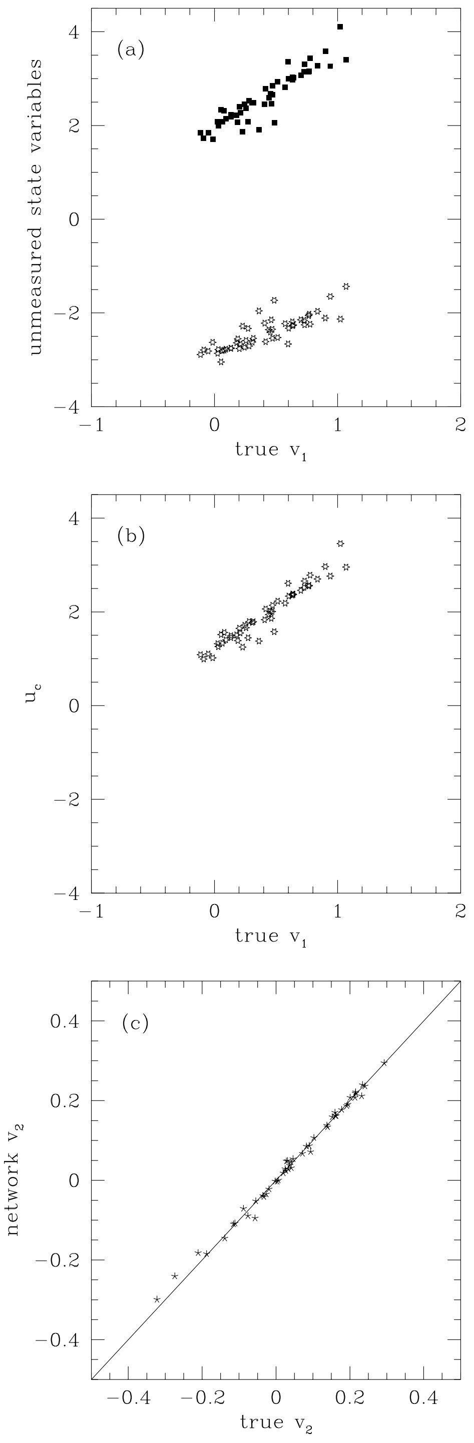


Fig. 11: Network predictions at the final epoch ($\tau = 8$) for a network with one measured state variable (v_2) and two unmeasured state variables (problem 5). (a) The two unmeasured state variables (u_1 as squares, u_2 as stars) plotted against the true v_1 state variable. (b) $u_c = 0.244u_1 + 0.970u_2$ is the optimum linear combination of u_1 and u_2 which gives the best correlation with v_2. (c) Network v_2 against true v_2. The rms error is 0.0119.

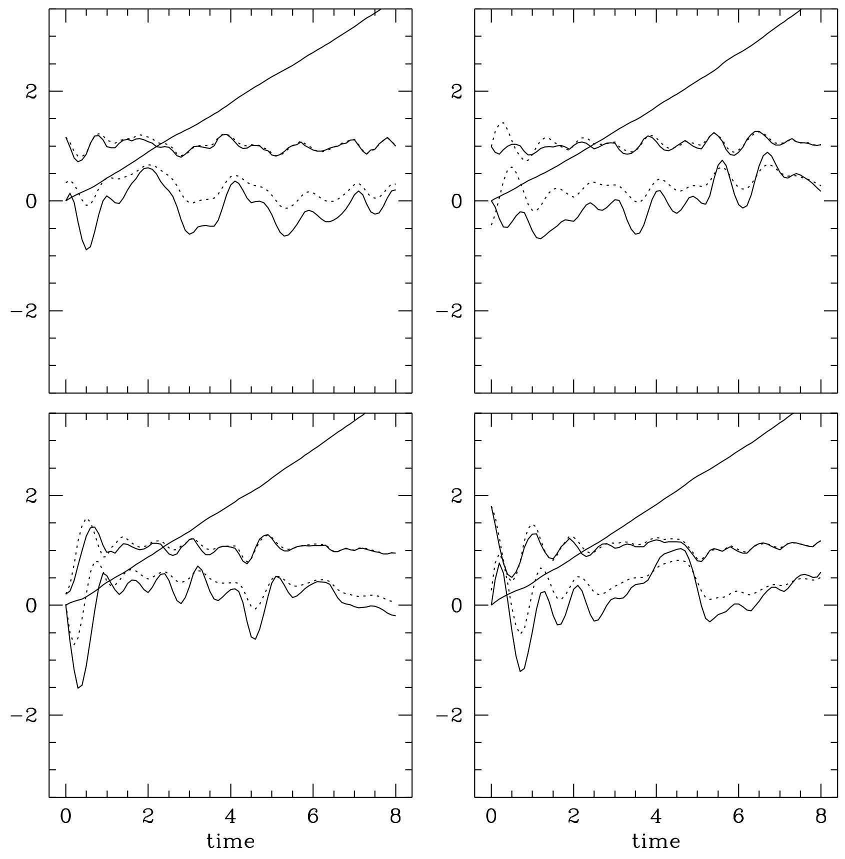
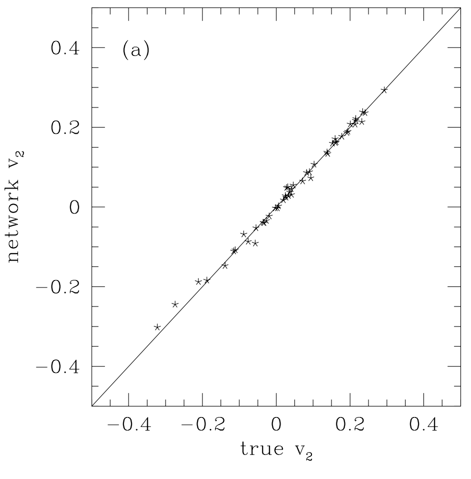


Fig. 12: Same plot as in Figure 10 but using a network trained from different initial random weights (problem 6).



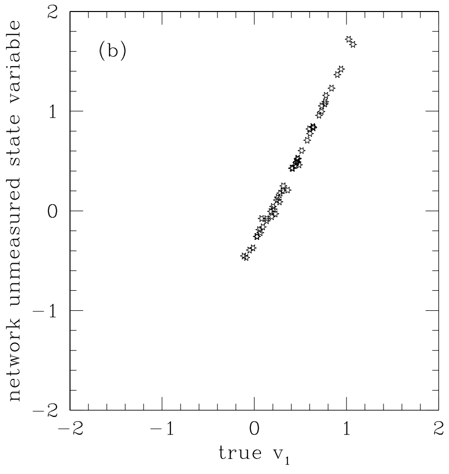


Fig. 13: Network predictions from problem 6 at the final epoch ($\tau = 8$). (a) Network v_2 plotted against true v_2. The rms error is 0.0111. (b) One of the unmeasured state variables learned by the network plotted against the v_1 state variable, no values of which were given to the network in training.