A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach

-Herbert Jaeger

Presented by: Shaowei Wang

Outline

- Recurrent neural network (RNN)
- Algorithms on training RNN
- Echo state network – special case of RNN

Recurrent neural network (RNN)

- Used to do sequence processing
- The output is fed back as input to others
- Allows loop

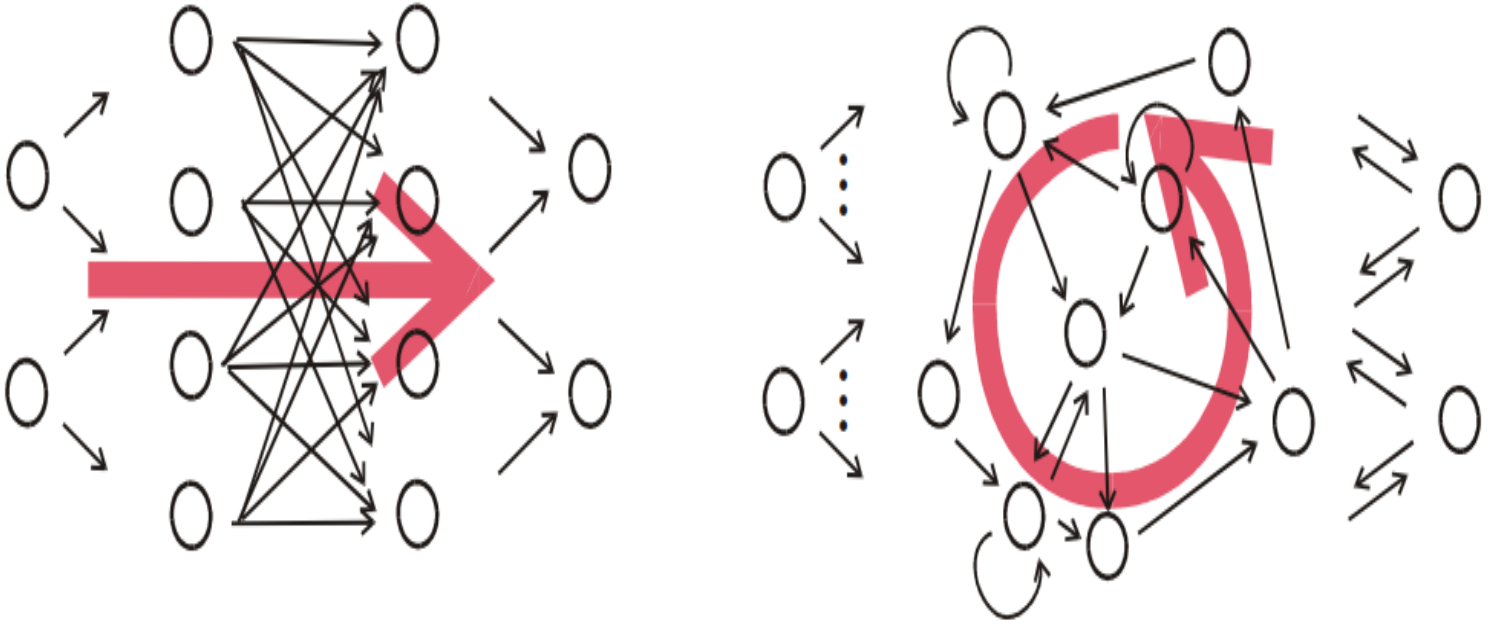


Figure 1.1: *Typical structure of a feedforward network (left) and a recurrent network (right).*

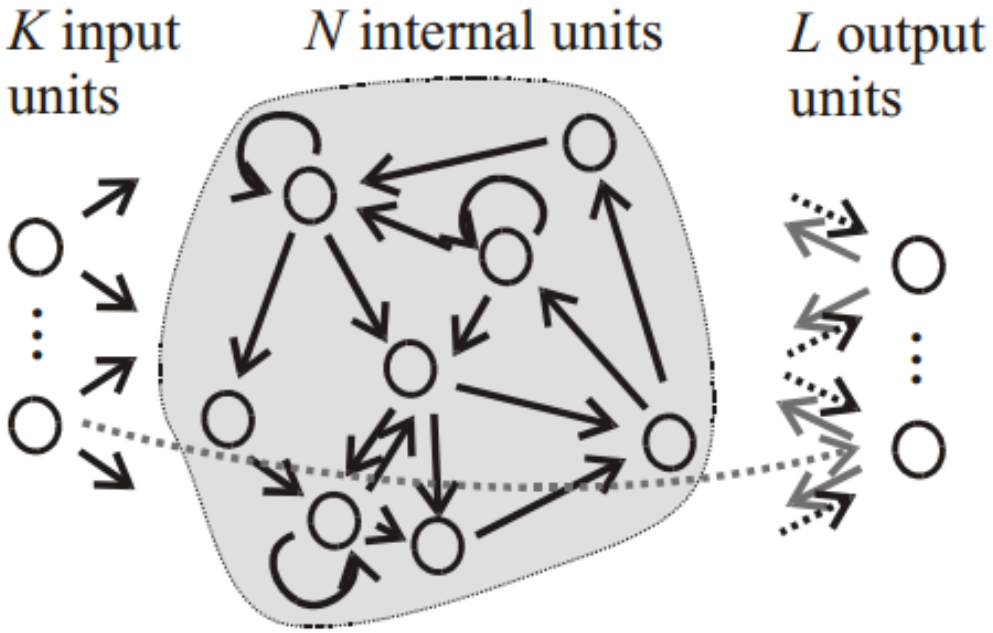
Short characterization comparison

	Recurrent neural network	Feedforward network
Structure	At least contain cycles Can “remember” states	No cycle and the layers are clear
Input-output	time sequential data	no time series
publications	5%	95%
Training approaches	No clear winner	Most popular: backpropagation algorithm

Category

- Discrete-time recurrent neural network (DTRNN)
  - The processing occurs in discrete steps, as if the network was driven by an external clock
- Continuous-time recurrent neural network (CTRNN)
  - The processing occurs in continuous time

The difference in activation updating



$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)), \quad (1.6)$$

n could be understood as time step

Training approaches

- Backpropagation though time (BPTT)
- Real-time recurrent learning
- Extended Kalman filter

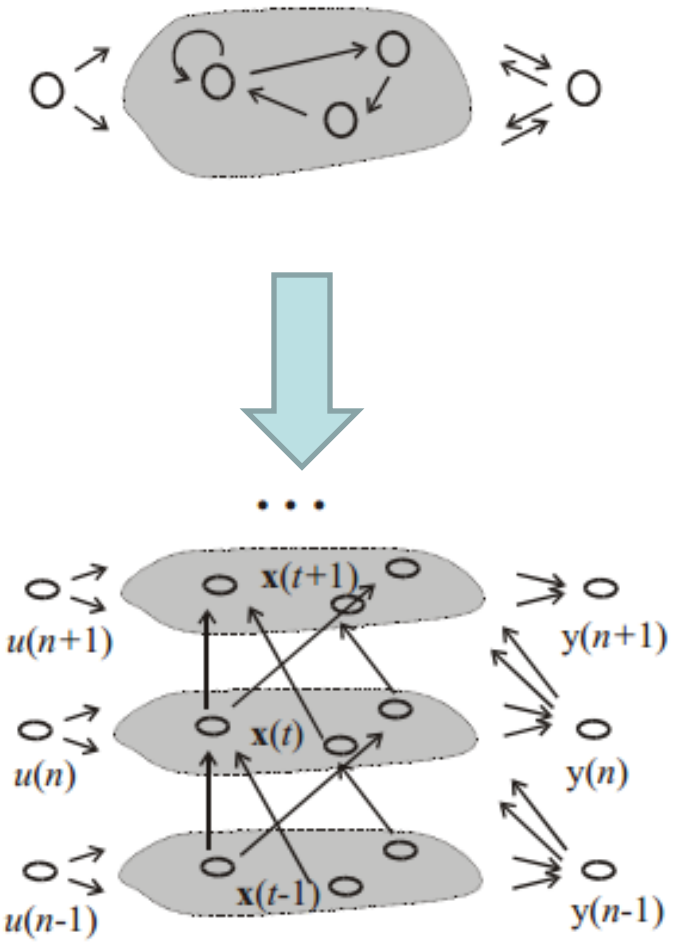
Backpropagation though time

•*Unfold the* discrete-time recurrent neural network into a multilayer feedforward neural network (FFNN) each time a sequence is processed.

•FFNN has a layer for each ``time step'' in the sequence, as if the ``time step'' is the index of the layer

•Redirect the connection between layers

•Use the standard backpropagation algorithm to train each FFNN form top to bottom



Algorithm to update weights

$$new\ w_{ij} = w_{ij} + \gamma \sum_{n=1}^{T} \delta_i(n)\, x_j(n-1) \quad [use\ x_j(n-1) = 0\ for\ n = 1]$$

$$new\ w_{ij}^{in} = w_{ij}^{in} + \gamma \sum_{n=1}^{T} \delta_i(n)\, u_j(n)$$

(2.18)

$$new\ w_{ij}^{out} = w_{ij}^{out} + \gamma \times \begin{cases} \sum_{n=1}^{T} \delta_i(n)\, u_j(n), \text{ if } j \text{ refers to input unit} \\ \sum_{n=1}^{T} \delta_i(n)\, x_j(n-1), \text{ if } j \text{ refers to hidden unit} \end{cases}$$

$$new\ w_{ij}^{back} = w_{ij}^{back} + \gamma \sum_{n=1}^{T} \delta_i(n)\, y_j(n-1) \quad [use\ y_j(n-1) = 0\ for\ n = 1]$$

- Since layers have been obtained by replicating the DTRNN over and over, weights in all layers should be the same.
- BPTT updates all equivalent weights using the sum of the gradients obtained for weights in all layers.

Drawback

- It's hard to be used in the application where online adaption is required as the entire time series must be used.
  - One option (p-BPTT): truncate part of time instead of entire time.
    - Drawback: the 'memory' beyond truncated time can't be captured by the model

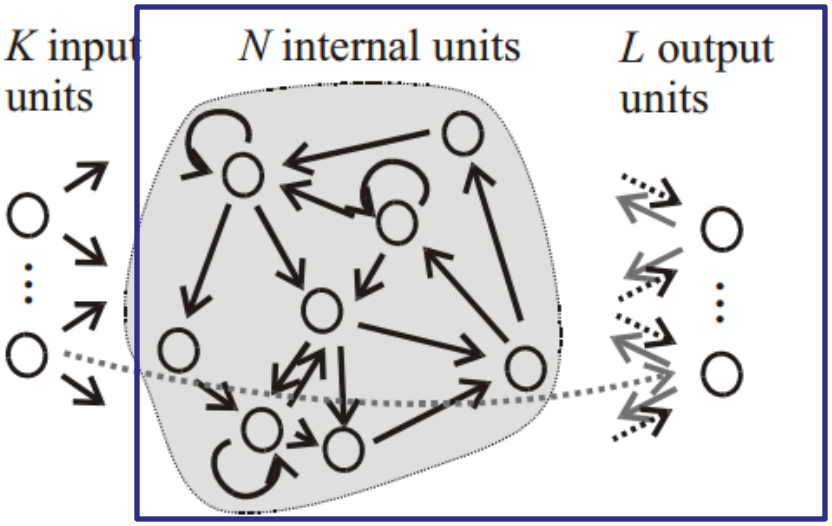
Training approaches

- Backpropagation though time (BPTT)
- Real-time recurrent learning
- Extended Kalman filter

Real-time recurrent learning

- Compute the error gradient and update weights for each time step

- During forward step, it compute the gradient of internal and output nodes with respects to all weights as the network



Objective function

Some of the units in U are output units, for which a target is defined. A target may not be defined for every single input however. For example, if we are presenting a string to the network to be classified as either grammatical or ungrammatical, we may provide a target only for the last symbol in the string. In defining an error over the outputs, therefore, we need to make the error time dependent too, so that it can be undefined (or 0) for an output unit for which no target exists at present. Let $T(t)$ be the set of indices k in U for which there exists a target value $d_k(t)$ at time t. We are forced to use the notation d_k instead of t here, as t now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and define our error function for a single time step as

$$E(\tau) = \frac{1}{2} \sum_{k \in U} [e_k(\tau)]^2 \tag{5}$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{\text{total}}(t_0, t_1) = \sum_{\tau=t0+1}^{t1} E(\tau) \tag{6}$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla_{\mathbf{W}} E_{total}(t_0, t+1) = \nabla_{\mathbf{W}} E_{total}(t_0, t) + \nabla_{\mathbf{W}} E(t+1) \tag{7}$$

As a time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}} \tag{8}$$

http://www.willamette.edu/~gorr/classes/cs449/rtrl.html

Training approaches

- Backpropagation though time (BPTT)
- Real-time recurrent learning
- Extended Kalman filter

Extended Kalman filter

- Kalman filter
  - a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error on linear system.
- Extented Kalman filter
  - A version working on nonlinear system

Objective function of Kalman filter

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-)$$

$$e_k \equiv x_k - \hat{x}_k.$$

$$P_k = E[e_k e_k^T].$$

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

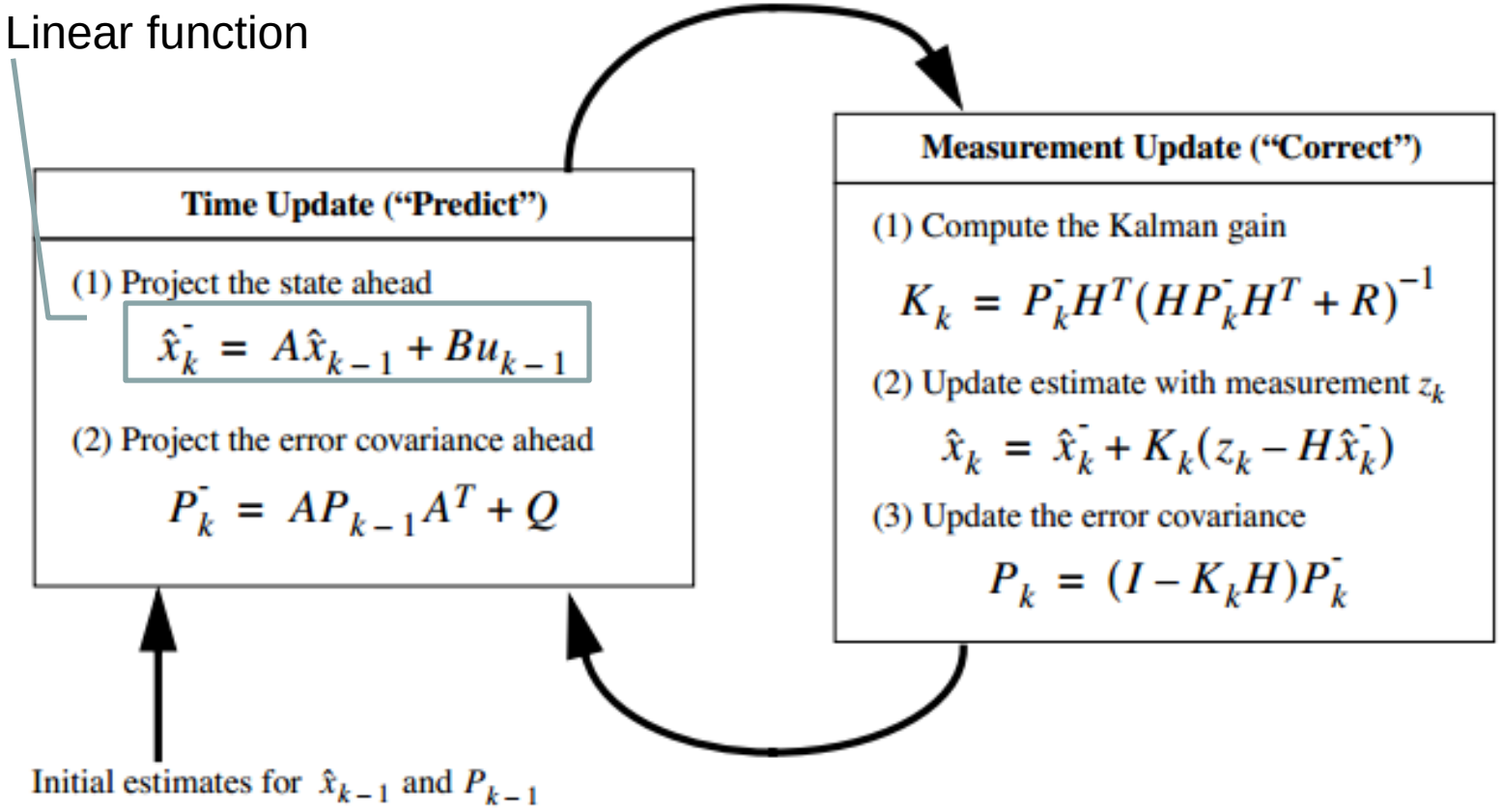
$$= \frac{P_k^- H^T}{H P_k^- H^T + R}.$$

X_k the state of network at time-step k,
$X^_K$ is the posteriori state estimate of X_K,
$X^_K^-$ is the priori state estimate of X_K
Z_k is the actual measurement, H $X^_K^-$ predict measurement
e_k is the posteriori estimate error

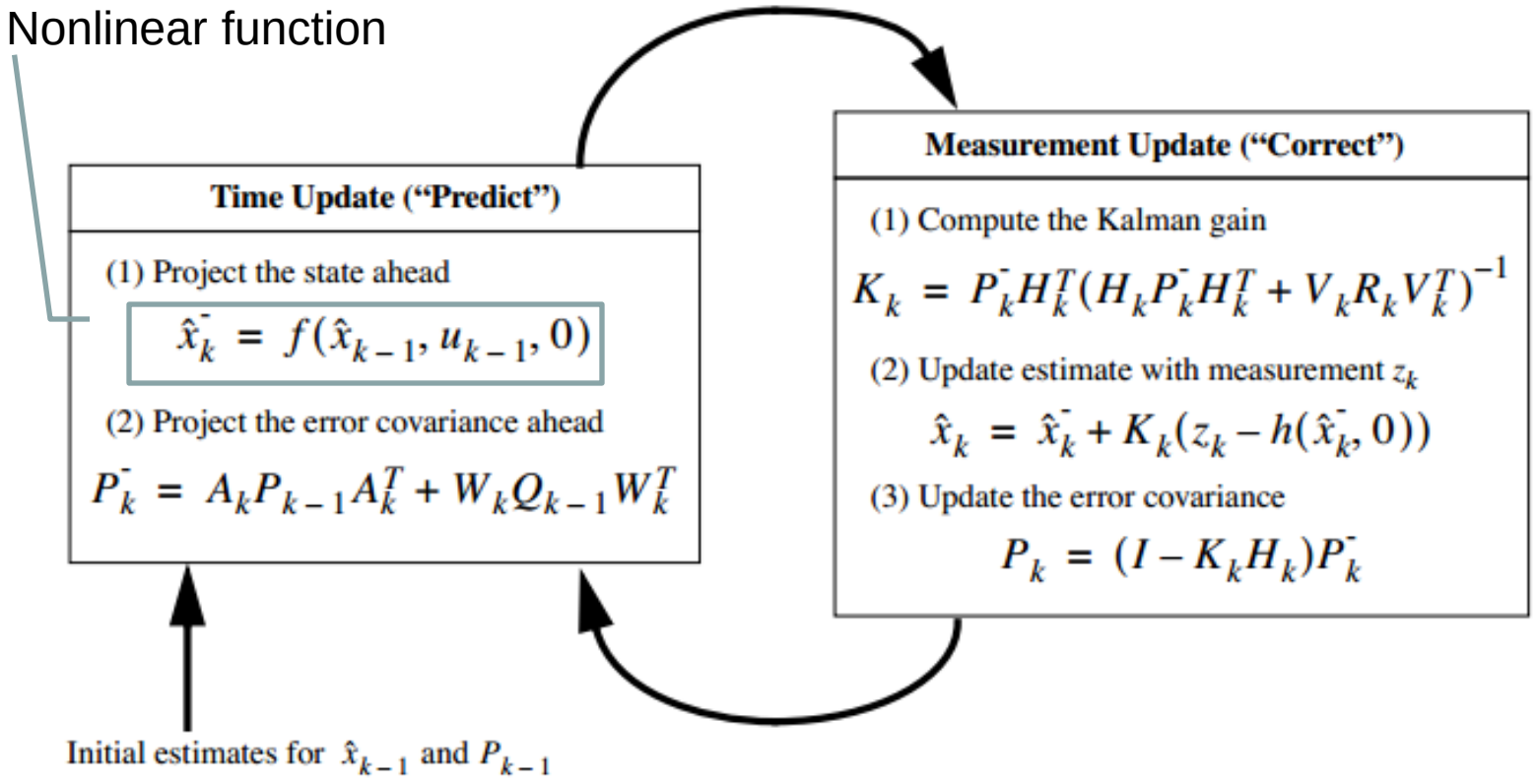
Minimize Pk (error covariance) as the objective function and get Kk

http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

Discrete Kalman filter cycle



Discrete extended Kalman filter cycle



Adapt to RNN

Time update (predict)

(5.8)
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{q}(n)$$
$$\mathbf{d}(n) = \mathbf{h}(\mathbf{w}, \mathbf{u}(0), \ldots, \mathbf{u}(n))$$
$$e = z(n) - d(n)$$

Measurement update (correct)

(5.9)
$$\mathbf{K}(n) = \mathbf{P}(n)\mathbf{H}(n)[\mathbf{H}(n)^t \mathbf{P}(n)\mathbf{H}(n)]^{-1}$$
$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mathbf{K}(n)\xi(n)$$
$$\mathbf{P}(n+1) = \mathbf{P}(n) - \mathbf{K}(n)\mathbf{H}(n)^t \mathbf{P}(n) + \mathbf{Q}(n)$$

(5.10)
$$\mathbf{K}(n) = \mathbf{P}(n)\mathbf{H}(n)[(1/\eta)\mathbf{I} + \mathbf{H}(n)^t \mathbf{P}(n)\mathbf{H}(n)]^{-1},$$

Interpret the weights w of the RNN as the state of a dynamical system

where d(n) is the desired output, w(n) is the weights. n is the time step. H(n) is the derivative of h(). The information of d(n) is incorporated into P(n) to update the Kalman gain function K(n).

Outline

- Recurrent neural networks
- Algorithms on training RNN
- Echo state network – special case of RNN

Example – sinewave

- D(n) =1/2sin(n/4)
- Task: remember the d(n) by using 300-step sequence of d(n) as the teacher signal (training data), without input.

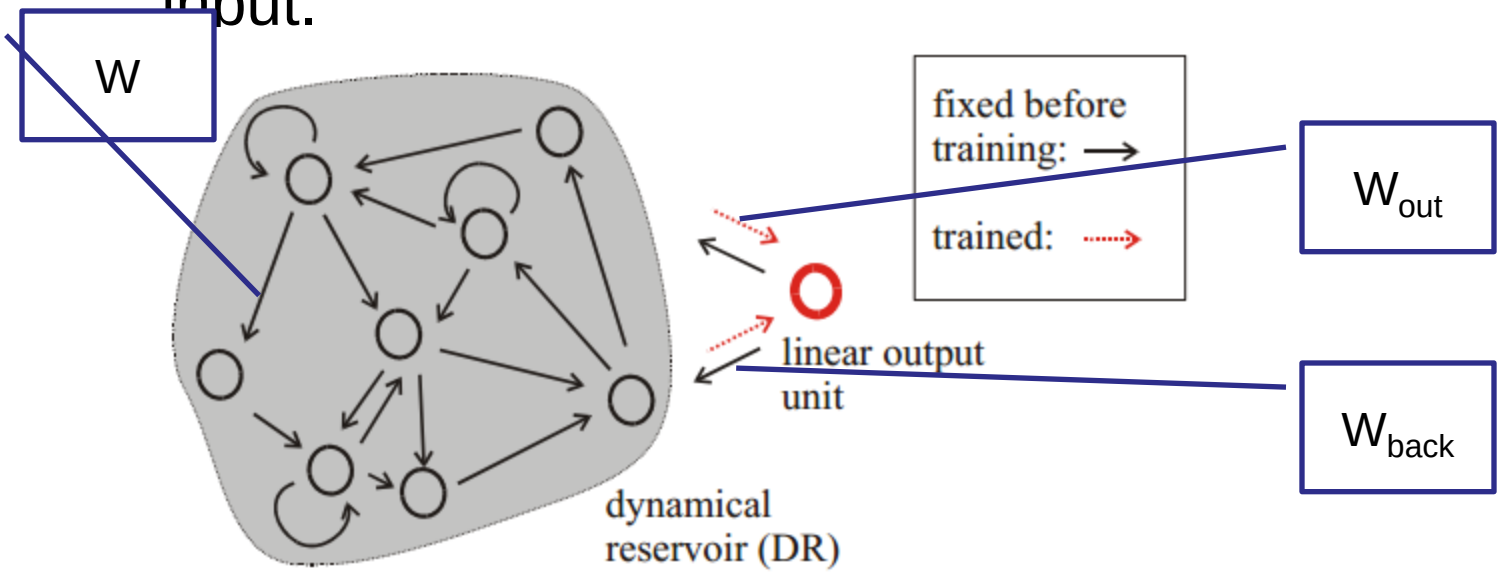
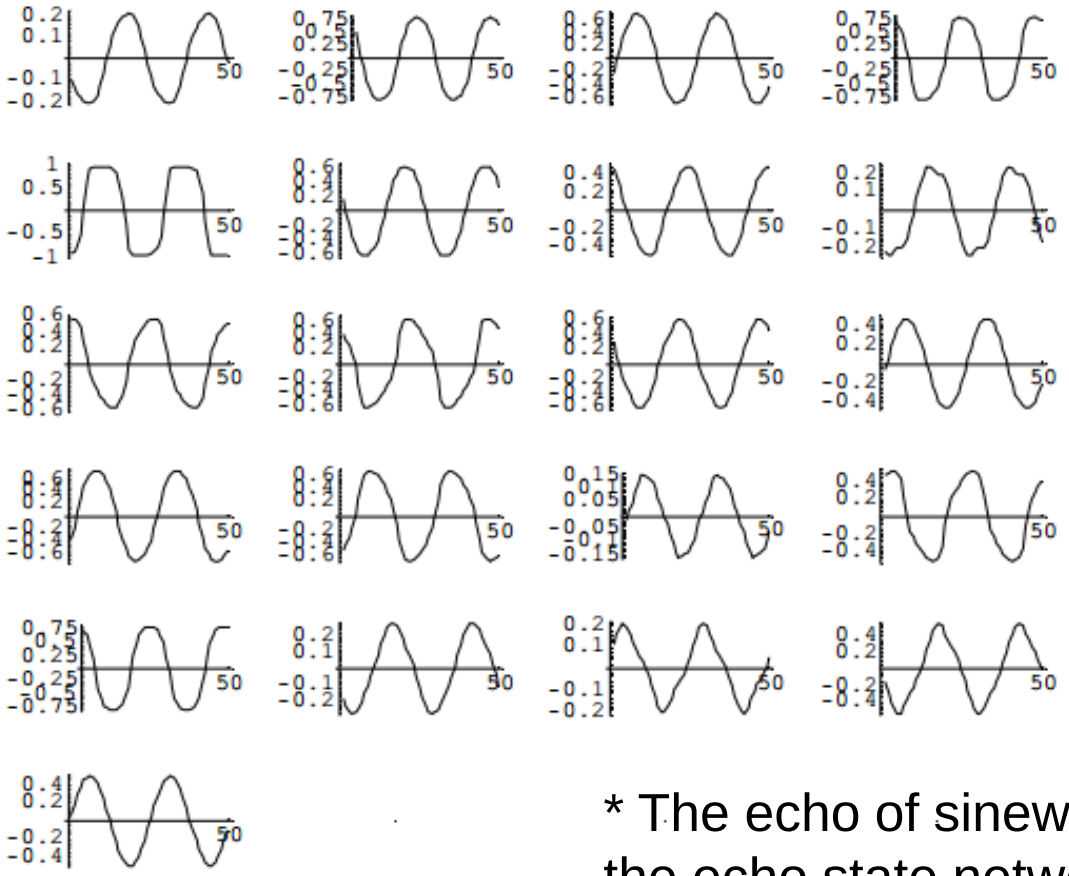


Figure 6.2: *Schematic setup of ESN for training a sinewave generator.*



* The echo of sinewave d(n)is kept in the echo state network

Figure 6.3. The dynamics within the DR induced by teacher-forcing the sinewave $d(n)$ in the output unit. 50-step traces of the 20 internal DR units and of the teacher signal (last plot) are shown.

Features of ESN

- It has a sparsely connected hidden layer (with typically 1% connectivity).
- The weights of hidden neurons are randomly assigned and are fixed.
- The weights of output neurons can be learned and produce (echo) specific pattern.

http://en.wikipedia.org/wiki/Echo_state_network

Whether a network has echo state property?

- The property is only dependent on W (weights of hidden layers), not dependent on W_{back} and W_{in}.
- If spectral radius $|\lambda_{max}| > 1$, no echo state property, where λ_{max} the max value of eigenvector of W.
- If spectral radius $|\lambda_{max}| < 1$, have echo state property.

How to estimate the W^{out}?

- The desired output weights $\mathbf{W}^{out}$ are the linear regression weights of the desired outputs $\mathbf{d}(n)$ on the states of network $\mathbf{x}(n)$
- Minimize the mean squared error (MSE)

$$MSE = \sum_{n=1}^{n=T} (d(n) - \sum_{i=1}^{|W^{out}|} w_i^{out} x_i(n))^2$$

Learning algorithm - 1

Step 1. Procure an untrained DR network ($\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{back}$) which has the echo state property, and whose internal units exhibit mutually interestingly different dynamics when excited.

This step involves many heuristics. The way I proceed most often involves the following substeps.

1. Randomly generate an internal weight matrix $\mathbf{W}_0$.
2. Normalize $\mathbf{W}_0$ to a matrix $\mathbf{W}_1$ with unit spectral radius by putting $\mathbf{W}_1 = 1/|\lambda_{max}| \mathbf{W}_0$, where $|\lambda_{max}|$ is the spectral radius of $\mathbf{W}_0$. Standard mathematical packages for matrix operations all include routines to determine the eigenvalues of a matrix, so this is a straightforward thing.
3. Scale $\mathbf{W}_1$ to $\mathbf{W} = \alpha \mathbf{W}_1$, where $\alpha < 1$, whereby $\mathbf{W}$ obtains a spectral radius of α.
4. Randomly generate input weights $\mathbf{W}^{in}$ and output backpropagation weights $\mathbf{W}^{back}$. Then, the untrained network ($\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{back}$) is (or more honestly, has always been found to be) an echo state network, regardless of how $\mathbf{W}^{in}$, $\mathbf{W}^{back}$ are chosen.

Small a for fast teacher dynamics, otherwise big a

Learning algorithm - 2

Step 2. Sample network training dynamics.

This is a mechanical step, which involves no heuristics. It involves the following operations:

1. Initialize the network state arbitrarily, e.g. to zero state $\mathbf{x}(0) = \mathbf{0}$.
2. Drive the network by the training data, for times $n = 0, ..., T$, by presenting the teacher input $\mathbf{u}(n)$, and by teacher-forcing the teacher output $\mathbf{d}(n\text{-}1)$, by computing

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{d}(n)) \quad (6.10)$$

3. At time $n = 0$, where $\mathbf{d}(n)$ is not defined, use $\mathbf{d}(n) = \mathbf{0}$.
4. For each time larger or equal than an initial washout time T_0, collect the network state $\mathbf{x}(n)$ as a new row into a state collecting matrix $\mathbf{M}$. In the end, one has obtained a state collecting matrix of size $(T - T_0 + 1)$ x $(K + N + L)$.
5. Similarly, for each time larger or equal to T_0, collect the sigmoid-inverted teacher output $\tanh^{-1}\mathbf{d}(n)$ row-wise into a teacher collection matrix $\mathbf{T}$, to end up with a teacher collecting matrix $\mathbf{T}$ of size $(T - T_0 + 1)$ x L.

Note: Be careful to collect into $\mathbf{M}$ and $\mathbf{T}$ the vectors $\mathbf{x}(n)$ and $\tanh^{-1}\mathbf{d}(n)$, not $\mathbf{x}(n)$ and $\tanh^{-1}\mathbf{d}(n\text{-}1)$!

Learning algorithm - 3

Step 3: Compute output weights.

1. Concretely, multiply the pseudoinverse of $\mathbf{M}$ with $\mathbf{T}$, to obtain a $(K+N+L)$ x L sized matrix $(\mathbf{W}^{out})^t$ whose i-th column contains the output weights from all network units to the i -th output unit:

   (6.11) $$(\mathbf{W}^{out})^t = \mathbf{M}^{-1}\mathbf{T} .$$

   Every programming package of numerical linear algebra has optimized procedures for computing pseudoinverses.
2. Transpose $(\mathbf{W}^{out})^t$ to $\mathbf{W}^{out}$ in order to obtain the desired output weight matrix.

Thank you!