Echo State Networks

Learning in recurrent Neural Networks

- *Back-Propagation through time:* Unfold the network over time and treat it like a feed-forward network.
- *Recurrent Back-Propagation:* Learn attractors of a recurrent network (similar to Hopfield networks, but supervised).

A simple, yet powerful approach are Echo State Networks (ESN).

They can be used for

- time series prediction,
- inverse modeling,
- pattern generation,
- classification (on time series),
- nonlinear control, ...

Approximation of Dynamics

Consider a dynamical system, getting an (leftwards infinite) input stream

$$\ldots u(n-3), u(n-2), u(n-1), u(n)$$

which has produced the outputs

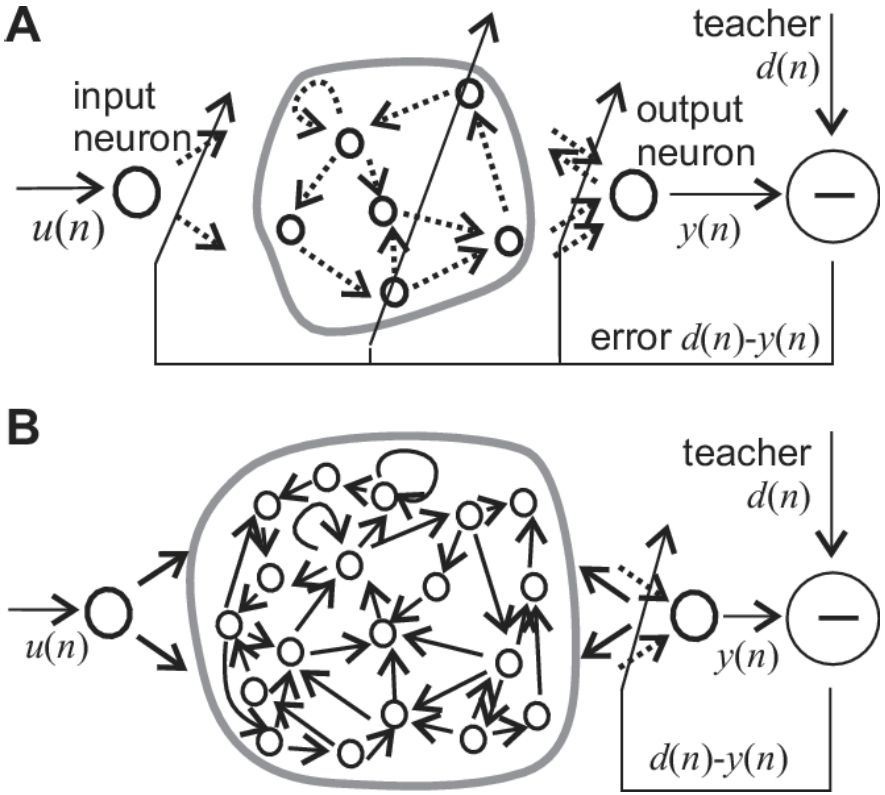
$$\ldots y(n-3), y(n-2), y(n-1).$$

It can be modelled as a function G which yields the next system output, given the input and output history

$$y(n) = G(\ldots, u(n-2), u(n-1), u(n); \quad \ldots, y(n-2), y(n-1))$$

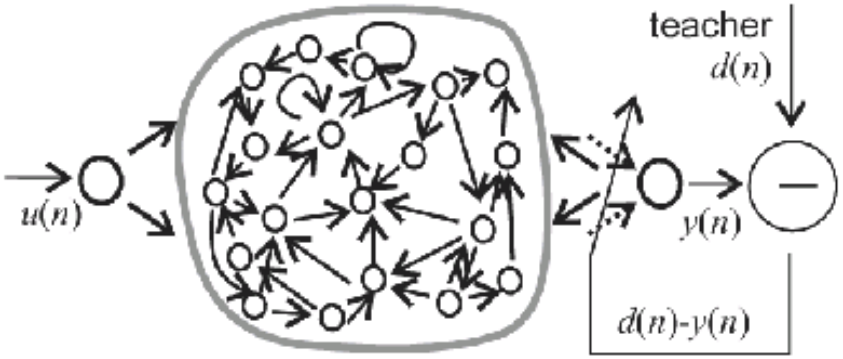
Training Recurrent Neural Networks

We consider *synchronous state updates*, i.e., each neuron updates its state based on its current inputs, and continuous valued neuronal elements (compare to the Hopfield model).



In Echo state networks, only the weights to the output neuron(s) are trained. Recurrent connections are chosen randomly and remain fixed during training.

Training Recurrent Neural Networks



- Inputs $\mathbf{u}(n)$,
- Internal units $\mathbf{x}(n)$, and
- Outputs $\mathbf{y}(n)$.
- $\mathbf{W}$ is the matrix of recurrent weights of internal units.

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

with $\mathbf{f} = (f_1, \ldots f_N)$ being typically the $tanh$.

The network acts as a “dynamic reservoir”.

What can we say about such an Network?

When do such networks behave nicely? For $\bar{\mathbf{u}}_h = \mathbf{u}(n), \ldots, \mathbf{u}(n+h)$, let

$$T(\mathbf{x}(n), \mathbf{y}(n), \bar{\mathbf{u}}_h)$$

denote the network update operator which denotes the network state that results for iterated update with input $\mathbf{u}(n+1), \ldots, \mathbf{u}(n+h)$ when starting in state $\mathbf{x}(n)$ with output $\mathbf{y}(n)$.

The network behaves nicely if it is state contracting: A network is *state contracting* if for all right-infinite input sequences, there exists a null-sequence $(\delta_h)_{h>0}$ such that for all states $\mathbf{x}$, $\mathbf{x}'$, for all $h > 0$ it holds that

$$d(T(x, \bar{\mathbf{u}}_h), T(x', \bar{\mathbf{u}}_h)) < \delta_h,$$

where d is the Euclidean distance and $\bar{\mathbf{u}}_h = \mathbf{u}(n), \ldots, \mathbf{u}(n+h)$.

I.e., the effect of the initial state is washed out.

When is a network state contracting?

The Echo-State property is connected to algebraic porperties of the connection matrix $\mathbf{W}$.

Consider networks with $tanh$ activation functions and weight matrix $\mathbf{W}$.

- Let $\sigma_{max}(\mathbf{W})$ be the largest singular value of $\mathbf{W}$, i.e. it is the largest eigenvalue of $(\mathbf{W}\mathbf{W}^T)^{1/2}$.
- If $\sigma_{max}(\mathbf{W}) = \Lambda < 1$ then $d(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u})) \leq \Lambda d(\mathbf{x}, \mathbf{x}')$. It follows that the network is state contracting.
- Let $|\lambda_{max}|(\mathbf{W})$ be the maximal absolute eigenvalue of $\mathbf{W}$
- If $|\lambda_{max}|(\mathbf{W}) > 1$, then the network is not state contracting.

How to set up a network

Since $\sigma_{max}(\alpha \mathbf{W}) = \alpha \sigma_{max}(\mathbf{W})$, and $|\lambda_{max}|(\alpha \mathbf{W}) = \alpha |\lambda_{max}|(\mathbf{W})$, we simply need to scale W to get a state contracting network.

- Start with a random and sparse matrix $\tilde{\mathbf{W}}$ (connectivity $\approx$ 10%).
- Use a scaling factor α with

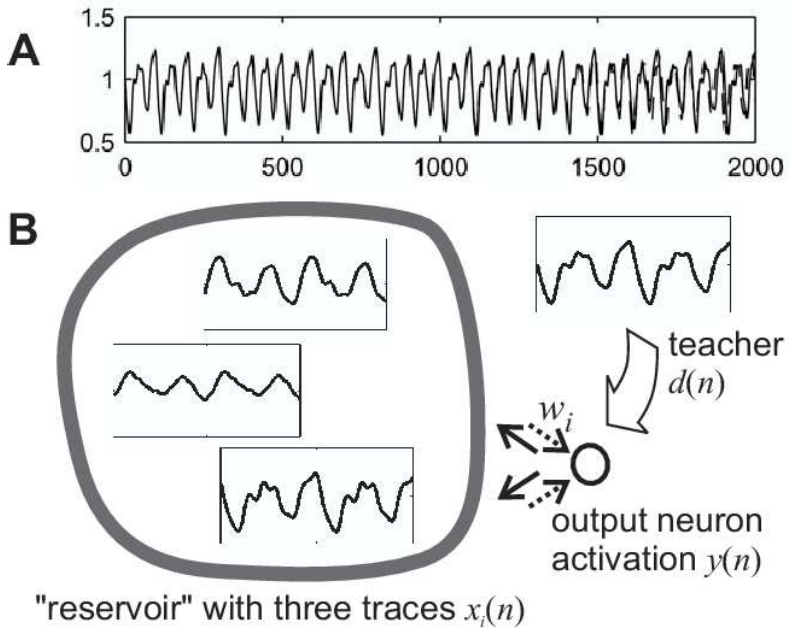
$$\frac{1}{\sigma_{max}(\tilde{\mathbf{W}})} \leq \alpha \leq \frac{1}{|\lambda_{max}|(\tilde{\mathbf{W}})}$$

Training of an ESN

We want the output of the network to be y_{teach}.

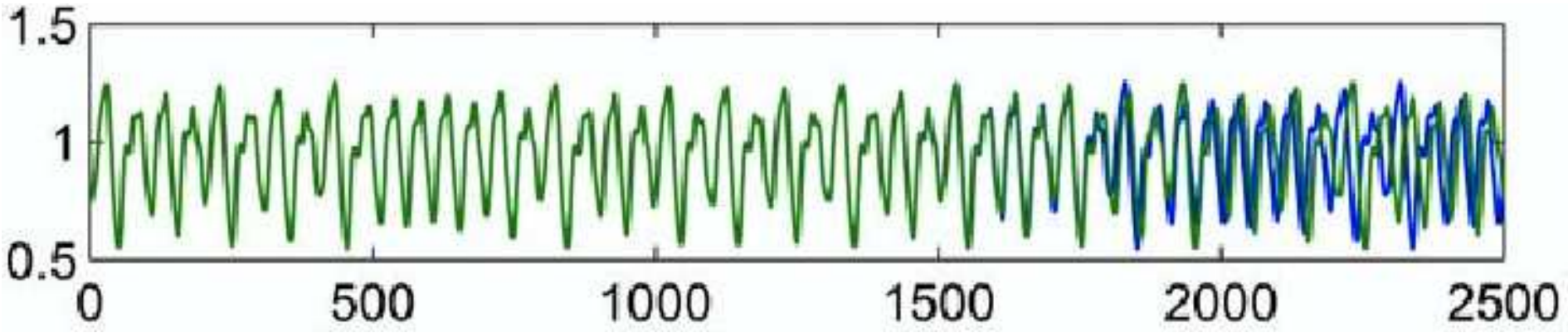
- Let the network run with the training input and a random initial condition.
- Disregard the first network states (they contain information about initial conditions which fades away after sufficiently long time).
- For a linear readout, the output is given by $y(n) = \sum_{i=1}^{N} w_i^{out} x_i(n)$.
- Minimize the error $\epsilon_{train}(n) = y_{teach}(n) - y(n)$ over all n considered in a least mean squares sense (Simple! Just use linear regression).
- If the ouput neurons have feed back connections to the network, one can force y_{teach} onto the output.

Example: Chaotic Time Series Prediction

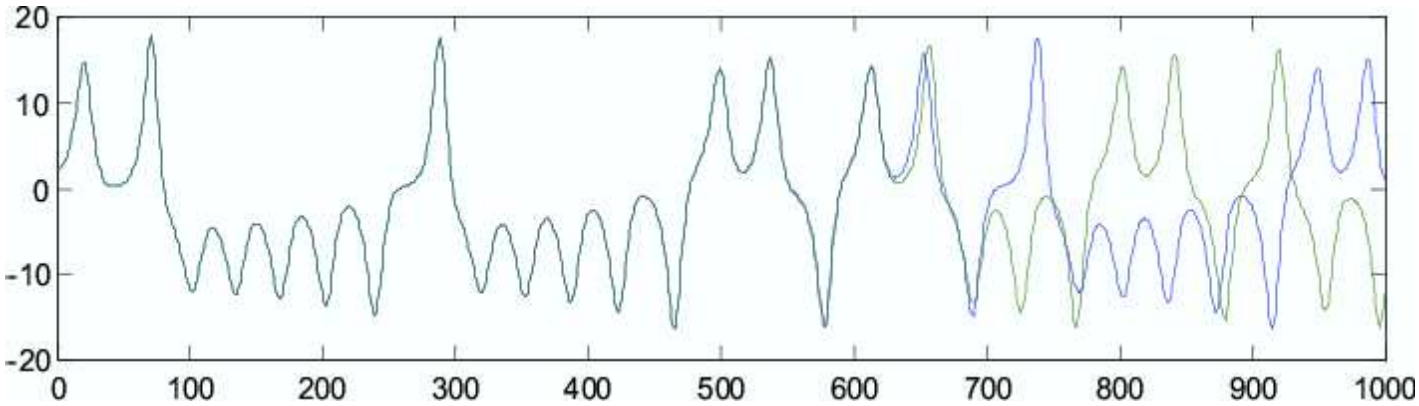


- Force the chaotic attractor of the Mackey-Glass system (MGS) onto the output neuron.
- Let it run for some time to forget the initial state.
- Then train the readout unit on this system.
- For testing, force the system to run for 500 steps on the MGS. Then remove the teacher and let it run autonomously.
- The network predicts the time series.

Example: Chaotic Time Series Prediction



The system improves on all previous methos by a factor of about 2400 in accuracy on the MGS.



Or the Lorenz attractor.