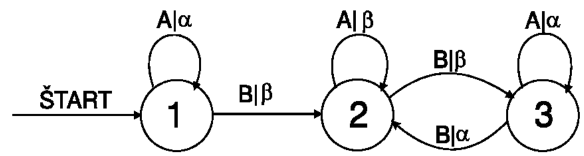
Introduction to artificial neural networks

Recurrent network models

Igor Farkaš
Department of Applied Informatics
Comenius University in Bratislava

Example: Mealy automaton



- Inputs: $\{A,B\}$, outputs: $\{\alpha,\ \beta\}$
- Training set: $A\rightarrow\alpha$, $A\rightarrow\alpha$, $B\rightarrow\beta$, $B\rightarrow\beta$, $B\rightarrow\alpha$, $A\rightarrow\beta$, $A\rightarrow\beta$, $A\rightarrow\beta$
- no sufficient tapped-line can reliably be set, so as to learn the behavior
- State representation of temporal context more appropriate than "past window"

Partially recurrent networks (with context units)

a) Elman (1990) - feedback from hidden layer
- can recognize sequences, make predictions, produce short sequence completion

b) Jordan (1986) - feedback from output layer
- option: decay units $c_i(t+1) = \alpha\ c_i(t) + y_i(t)$ $\alpha<1$
- with fixed input, can generate various output sequences
- with input sequences, can recognize sequences

c) Stornetta (1986) - decay loop on input =>
- moving average of past activations (IIR): $c_i(t+1) = \alpha\ c_i(t) + x_i(t)$
- better suited for recognizing input sequences, than generating or reproducing them

d) Mozer (1986) – input $c_i(t+1) = \alpha\ c_i(t) + f(\sum_j v_{ij} x_j(t))$
- differs from c) in two features: full connectivity b/w inputs and context units, trainable decay links (recurrent)
- requires a learning rule different from BP, similar applicability as c)

Learning algorithms for fully recurrent NNs

- dynamically driven recurrent NNs, global feedback
  - acquire (internal) state representations
- (similarly to spatial tasks) two modes:
  - epochwise training: epoch ~ sequence
  - continuous training
- We mention two gradient based algorithms: BPTT and RTRL
- Heuristics:
  - start with shorter sequences, then increase length
  - update weights only if training error is larger than threshold
  - consider regularization (e.g. weight decay)

Back-propagation through time

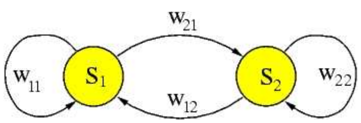
(Werbos, 1990)

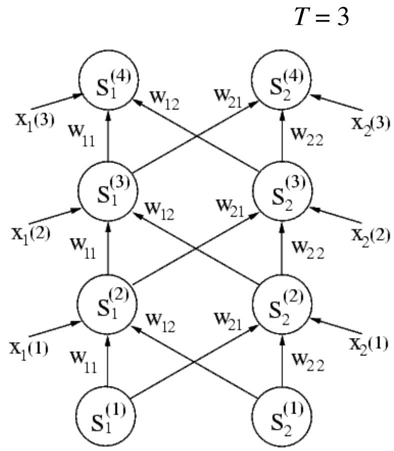
- extension of standard BP algorithm – unfolding in time into a feedforward NN (with identical weights)
- sequence with inputs $\boldsymbol{x}(1), \boldsymbol{x}(2), ..., \boldsymbol{x}(T)$

State equation:

$$s_i(t+1) = f(\textstyle\sum_j w_{ij} s_j(t) + x_i(t)),$$

[*in our example* $i, j = 1,2$]





BPTT algorithm

- applied after processing each sequence (of length T)
- during single forward pass through sequence:
  - record inputs, local gradients δ
- Overall error: $E_{\text{total}}(T) = ½ \sum_{t=1}^{T} \sum_{i\in O} e_i^2(t)$
- for $t = T$: $\delta_i(t) = f'(net_i)\, e_i(t)$

  for $1 < t < T$: $\delta_i(t) = f'(net_i)\, [e_i(t) + \sum_{l\in O} w_{il}\, \delta_l(t+1)]$
- Update weights: $\Delta w_{ij} = - \alpha\, \partial E_{\text{total}}(T) / \partial w_{ij} = \alpha \sum_{t=2}^{T} \delta_i(t)\, x_j(t\text{-}1)$
- impractical for longer sequences (of unknown length)

Real-time recurrent learning (RTRL)

(Williams & Zipser, 1989)

- Instantaneous output error: $e_i(t) = d_i(t) – s_i(t)$; $i\in O$ (targets exist)

  $E(t) = ½ \sum_{i\in O} e_i^2(t)$
- Update weights: $\Delta w_{ij} = - \alpha\, \partial E(t) / \partial w_{ij} = \alpha \sum_{k\in O} e_k(t)\, \partial s_k(t) / \partial w_{ij}$

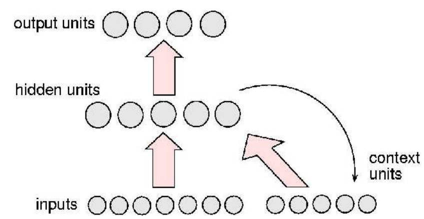
  $\partial s_k(t) / \partial w_{ij} = f'(net_k(t))\, [\delta^{\text{kr}}_{ki}\, s_j(t\text{-}1) + \sum_l w_{kl} \partial s_l(t\text{-}1) / \partial w_{ij}]$

  $l\in$ units feeding to unit k, and $\delta^{\text{kr}}_{ki} = 1$, if $k = i$, else 0.

  - if j pertains to external input, $x_j(t\text{-}1)$ is used instead
- Smaller α recommended, BP "tricks" applicable (e.g. momentum)
- Teacher forcing – replace actual output with desired whenever available
  - may lead to faster training and enhance learning capability
- Very large time and memory requirements (with N neurons, each iteration): N^3 derivatives, $O(N^4)$ updates to maintain

Simple recurrent network

(Elman, 1990)



Implicit representation of time

Hidden state activation:

$$h_k(t+1) = f\left(\sum_j w_{kj} x_j(t) + \sum_l c_{kl} h_l(t)\right)$$

Output: $y_i(t) = f\left(\sum_k v_{ik} h_k(t)\right)$

Unit's activation function:

$$f(u) = \frac{1}{1+\exp(-u)}$$

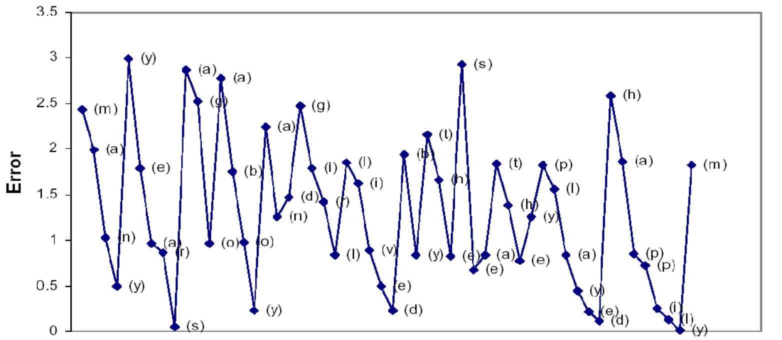
can be trained by BP, RTRL,...

The following examples: symbolic dynamics

Example: Next letter prediction task

Task: letter-in-word prediction, 5-bit inputs
Data: 200 sentences, 4 to 9 words in a sentence
SRN: 5-20-5 units, trained by back-propagation (Rumelhart, Hinton & Williams, 1986)
- NN discovers the notion "word"

Many years ago boy and girl lived by the sea ... (Elman, Cog. Sci., 1990)



Example: Next word prediction task

SRN: 31-150-31
localist encoding of words
no context reset b/w sentences
(Elman, Cog. Sci., 1990)

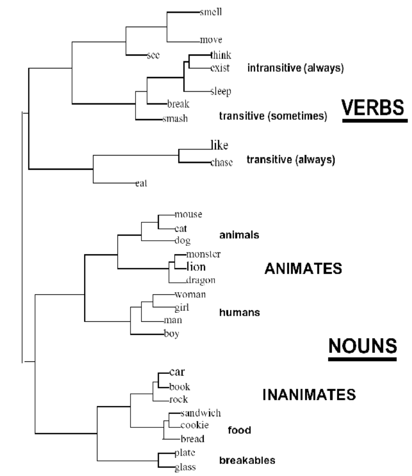
Categories of lexical items used

Category	Examples
NOUN-HUM	man, woman
NOUN-ANIM	cat, mouse
NOUN-INANIM	book, rock
NOUN-AGRESS	dragon, monster
NOUN-FRAG	glass, plate
NOUN-FOOD	cookie, sandwich
VERB-INTRAN	think, sleep
VERB-TRAN	see, chase
VERB-AGPA	move, break
VERB-PERCEPT	smell, see
VERB-DESTROY	break, smash
VERB-EA	eat

Templates for sentence generator

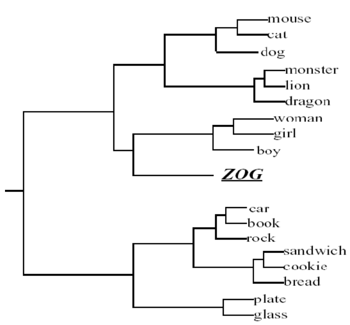
WORD 1	WORDS	WORD 3
NOUN-HUM	VERB-EAT	NOUN-FOOD
NOUN-HUM	VERB-PERCEPT	NOUN-INANIM
NOUN-HUM	VERB-DESTROY	NOUN-FRAG
NOUN-HUM	VERB-INTRAN	
NOUN-HUM	VERB-TRAN	NOUN-HUM
NOUN-HUM	VERB-AGPAT	NOUN-INANIM
NOUN-HUM	VERB-AGPAT	
NOUN-ANIM	VERB-EAT	NOUN-FOOD
NOUN-ANIM	VERB-TRAN	NOUN-ANIM

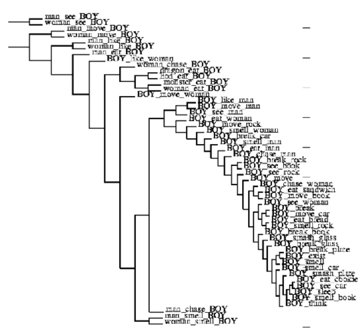
Averaged hidden-unit activation vectors



Properties of hidden-unit activations after training

- activations show structure (clusters)
- types/tokens distinction: types = centroids of tokens
- representations are hierarchically structured
- type vector for a novel word (*zog*) consistent with previous knowledge
- representation space would not grow with a growing lexicon





Example: Modeling recursive processing in humans

(Christiansen & Chater, 1999)

A. Counting recursion — *aabb* — $NNVV$

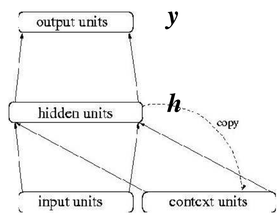
B. Center-embedding recursion — *a b b a* — $S_N P_N P_V S_V$ — *the boy girls like runs*

C. Cross-dependency recursion — *a b a b* — $S_N P_N S_V P_V$ — *the boy girls runs like*

D. Right-branching recursion — *a a b b* — $P_N P_V S_N S_V$ — *girls like the boy that runs*

- Qualitative performance of SRN model matches human behavior, both on relative difficulty of B and C, and between their processing and that of D.
- This work suggests a novel explanation of people's limited recursive performance, without assuming the existence of a mentally represented competence grammar allowing unbounded recursion.
- They compare the performance of the network before and after training – pointing to architectural bias, which facilitates the processing of D over B and C.

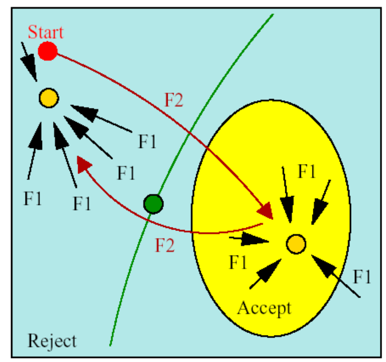
RNN state space organization



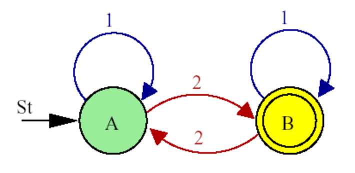
- $y = F(h)$ is a squashed version of a linear transformation => it is smooth and monotonic
- Activations h leading to the same/similar output y are forced to lie close to each other in the RNN state space
- Heuristics for enhancing RNN generalization: cluster RNN state space into a finite number of clusters
- Each cluster will represent an abstract information-processing state => knowledge extraction from RNN (e.g. learning finite state automata with RNNs)

Example: Learning a finite state automaton

- State-space activations in RNN – neural memory – code the entire history of symbols we have seen so far.
- Information latching problem for gradient learning
- To latch a piece of information for a potentially unbounded number of time steps we need attractive sets.



RNN state space

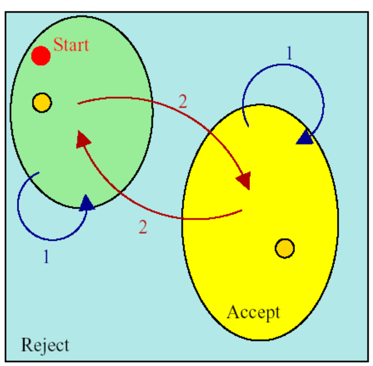


Tiňo (2003)

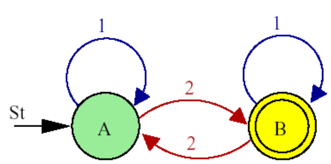
Grammatical:
all strings containing odd number of 2's

Example: string classification task

RNN has a continuous state-space



Automaton can be extracted from trained RNN (clusters of hidden-unit activations correspond to automaton states)

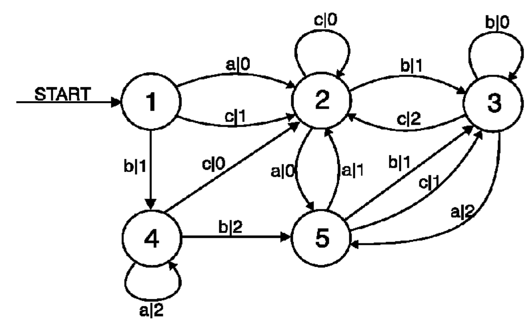


Extracted FSM:
all strings containing odd number of 2's

Tiňo (2003)

Example: temporal association task

3+1 input symbols
3+1 output symbols



Training sequences:

$acccb! \rightarrow 00001x$, $caaab! \rightarrow 10101x$, $bbbbbb! \rightarrow 121000x$, atď.

$acccb!caaab!bbbbbb!\ldots \rightarrow 00001x10101x121000x\ldots$

Suitable scenario: start with simpler sequences
Extracted automaton can generalize training data

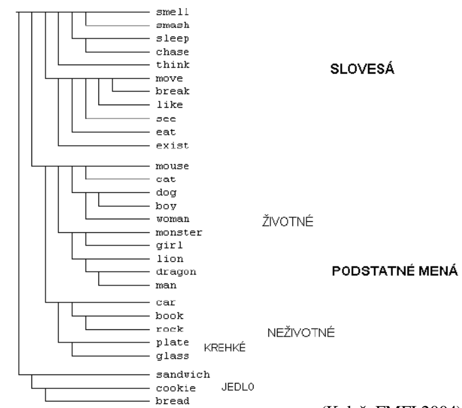
Computational power of recurrent networks

- recurrent NNs can learn to simulate formal automata
  - regular grammars (finite-state machine)
  - context-free grammars (e.g. $a^n b^n$ ~ saddle-point attractive set)
- All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoid activation functions. (Siegelmann, 1991)
- Practically, we may encounter problems with convergence in more complex tasks.
- vanishing gradients problem – difficulty to learn long-term dependencies by gradient-based algorithms
  - approaches: apply heuristics, extended Kalman filtering, elaborate optimization techniques

Properties of hidden-unit activations before training

SRN: 31-1000-31
random initialization of weights
(small values)

Architectural bias:
Structured hidden-unit activations exist prior to training



(Kalaš, FMFI,2004)

Explanation of architectural bias in RNNs

In RNNs with sigmoid activation functions and initialized with small weights (Tiňo et al. (2004):

1) clusters of recurrent activations that emerge prior to training correspond to Markov prediction contexts – histories of symbols are grouped according to the number of symbols they share in their suffix, and

2) based on activation clusters, one can extract from untrained RNNs predictive models – variable memory length Markov models (VLMMs).

RNNs have a potential to outperform finite memory models, but to appreciate how much information has really been induced during training, RNN performance should always be compared with that of VLMMs extracted before training as the "null" base models.

Iterated Function Systems

(Barnsley, 1988)

IFS consists of a complete metric space (X,d), where $X = [0,1]^N$, d is Euclidean metric, together with a set of contractive mappings w_i: $X \to X$

$$w_i(x) = k\,x + (1-k)\,s_i \qquad i = 1,2,\ldots, A$$

Symbols $s_i \in \{0,1\}^N$ $\quad N = ceil(\log_2 A)$ $\quad$ Contraction coef. $k \in (0,0.5]$

Each n-symbol sequence $S = s_1 s_2 \ldots s_n$ is represented by IFS as a point

$$w(x) = w_n(w_{n-1}(\ldots(w_2(w_1(x)))\ldots)), \quad x \in X.$$

Recurrent NN with small random weights also performs contractive mappings in state space (using various k's for each symbol).

IFS – topographic mapping property

Let $S_i^j = s_i\, s_{i+1} \ldots s_j$, then given a sequence $S = s_1 s_2 \ldots$ over $\mathcal{A}$, the chaotic n-block representation of S is defined as a set of points

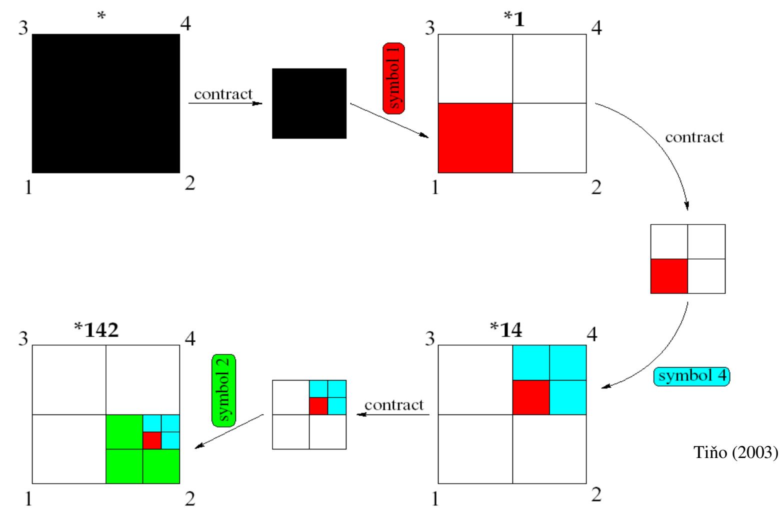
$$CBR_{n,k}(S) = \{S_i^{i+n-1}(x_*)\}_{i \geq 1}$$

where $x_* = \{½\}^M$ is the center of X.

- *CBR* has the property that is temporal analogue of topographic mapping: the longer is common suffix of two sequences, the closer they are mapped in *CBR*.
- On the other hand, the Euclidean distance between points representing two n-sequences that have the same prefix of length n -1 and differ in the last symbol, is at least 1 - k.

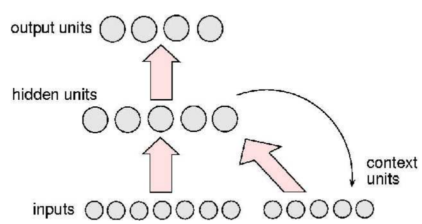
Illustration of contractive mapping (IFS)

Topological ordering with respect to suffixes



Echo state network

(Jaeger, 2001)

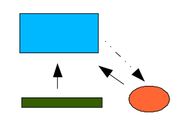


can have architecture of Elman net, but also additional connections are possible

Hidden units:
- linear or sigmoid activation function
- create dynamic reservoir (mapping to high dimensional space),
  - context weights: random sparse matrix, with spectral radius $|\lambda_{max}| < 1$
  - input weights: random, small

- only output weights are trained => fast training
- has Markovian behavior in symbolic dynamics

SOMs for symbolic sequences



- (Markovian) map of suffixes

Neurons: $i = 1,2, \ldots, N$ Winner $i^* = \arg\min_j\{d_j(t)\}$, or $\{\|\mathbf{d}_j(t)\|\}$

Weight update for: $\mathbf{w}_i$ (input weights) $\mathbf{c}_i$ (context weights, optional)

Temporal Kohonen map: $d_i(t) = a.\|\mathbf{x}(t) - \mathbf{w}_i\|^2 + (1\text{-}a)\, d_i(t\text{-}1)$

Recurrent SOM: $\mathbf{d}_i(t) = a.[\mathbf{x}(t) - \mathbf{w}_i] + (1\text{-}a)\, d_i(t\text{-}1)$

Merge SOM: $d_i(t) = (1\text{-}a).\|\mathbf{x}(t) - \mathbf{w}_i\|^2 + a.\|\mathbf{r}(t) - \mathbf{c}_i\|^2$ $\quad \mathbf{x}, \mathbf{r} \in R^d$

$$\mathbf{r}(t) = b.\mathbf{w}_{i^*}(t\text{-}1) + (1\text{-}b)\, \mathbf{r}_{i^*}(t\text{-}1)]$$

Recursive SOM: $d_i(t) = a.\|\mathbf{x}(t) - \mathbf{w}_i\|^2 + b.\|\mathbf{y}(t\text{-}1) - \mathbf{c}_i\|^2$ $\quad y_i = \exp(-d_i)$

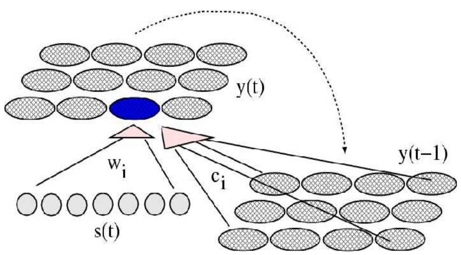
$\mathbf{c}_i, \mathbf{y} \in R^N$

SOMSD: context = winner position in the map

Recursive self-organizing map (RecSOM)

(Voegtlin, 2002)

RecSOM can lead to more complex dynamics compared to other unsupervised models, based on leaky-integrator units.
(Tino, Farkas, van Mourik, 2006)



Quantization error on unit i:

$$E_i = a\|\boldsymbol{s}(t) - \boldsymbol{w}_i\|^2 + b\|\boldsymbol{y}(t-1) - \boldsymbol{c}_i\|^2$$

Output of unit i:

$$y_i = \exp(-E_i)$$

Learning rules:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + z\, h_{ik}[\boldsymbol{s}(t) - \boldsymbol{w}_i(t)]$$

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) + z\, h_{ik}[\boldsymbol{y}(t-1) - \boldsymbol{c}_i(t)]$$

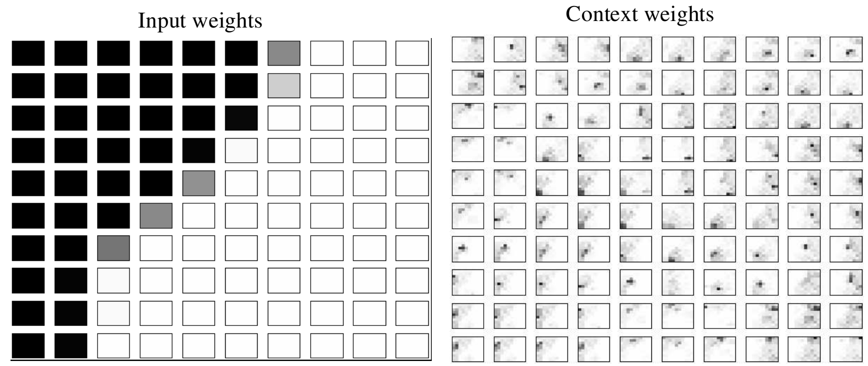
$0 < z \ll 1$ (constant) learning rate

h – (decreasing) neighborhood function

RecSOM: trained on stochastic 2-state automaton: weights

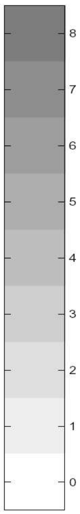
RecSOM: 10x10 units, 1D inputs

Input source: $P(b|a) = 0.3$, $P(a|b) = 0.4$



RecSOM: trained on stochastic 2-state automaton: topographic map of suffixes

baaab	aaaab	aaaab	aaaab	bab	aabab		bbaaa	bbaaa	bbaaa
abaab	aaaab	aaaab	bbaab	bbaab	babab		bbaaa	babaa	babaa
ababb	b	bbab	bbbab	aaaab	aaaab	aaaaa	aaaaa	aabaa	aabaa
baabb	aaabb	bbab	abbbb	baaab	baaaa	aaaaa	aaaaa	aaaaa	aaaaa
aaabb	aaabb	abbbb	abbbb		baaaa	aaaaa	aaaaa	aaaaa	aaaaa
bbabb	bbbbb	bbbbb		bba		abbaa		aaaaa	aaaaa
bbbbb	bbbbb		bbbba	abbba	abbaa	bbbaa	bbbaa	aaaaa	aaaaa
babbb	bbbbb	bbbba	bbbba	bbbba		bbbaa	bbbaa		aaaaa
aabbb	babbb	babba	babba	bbaba	aaaba	baaba	abaaa	abaaa	abaaa
aabbb	bb	aabba	aabba	baaba	aaaba	baba	baaaa	baaaa	baaaa



Summary

- two classes of architectures (time-lagged, partially or fully recurrent)
- time-lagged models are good for tasks with limited memory
- recurrent models with global feedback (via tapped-delay-lines) learn their internal state representations
- existing links to the theory of nonlinear dynamical systems, signal processing and control theory
- More complex learning algorithms: BPTT, RTRL (gradient-based)
- second-order neurons possible – higher computational power
- despite theoretical potential, difficulties to learn more complex tasks
- architectural bias
- novel models: echo-state networks and self-organizing recursive maps