



Neural Networks

Lecture 7

Sequential modeling

Igor Farkaš

2021

Temporal processing with neural networks

Required for tasks with temporal structure – where the same input can be associated with different outputs

e.g. $A \rightarrow \alpha$, $B \rightarrow \beta$, $B \rightarrow \alpha$, $B \rightarrow \gamma$, ...

Types of tasks:

- Sequence recognition (classification)
- Sequence prediction
- Sequence generation

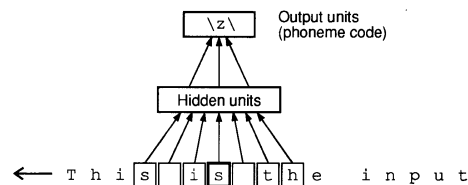
Incorporating time into a NN:

- tapped-delay input ~ time as spatial dimension
- using recurrent NN architecture
 - associative memory
 - temporal input-output mapping

Time as spatial dimension: NETtalk

NN learns to read English text

Input 7×29 units encoding 7 chars,
80 hidden and 26 output units
encoding phonemes.

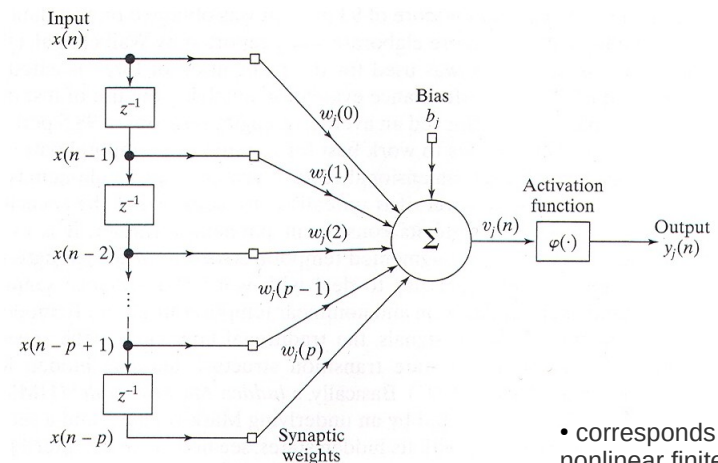


- training on 1024 words, intelligible speech after 10 epochs, 95% accuracy after 50 epochs
- 78% generalization accuracy, quite intelligible speech
- NN first learned gross features (e.g. word boundaries) and then gradually refined its discrimination (psycholinguistic plausibility).
- graceful degradation
- meaningful internal representations (e.g. vowel-consonant distinction)
- General question: How to estimate delay from training data?

(Sejnowski & Rosenberg, 1987)

3

Focused neuronal filter

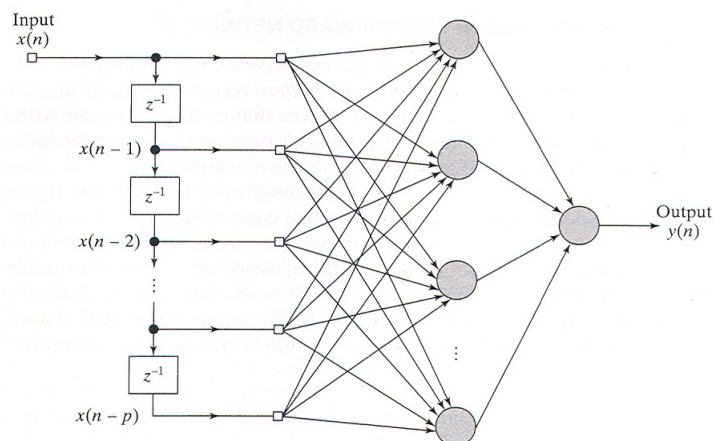


- focused because all memory comes from the input
- can be trained as ordinary feedforward NN

• corresponds to a nonlinear finite impulse response (FIR) filter (in digital signal processing)

4

Focused time feedforward NN

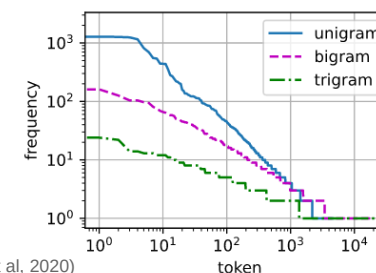


- usable for stationary input-output mapping tasks
- can be trained as ordinary feedforward NN
- time as a spatial dimension

5

Link to statistical n-gram models

- Assume 1D time series $X = x_1, x_2, \dots, x_{t-1}, \dots$
- Autoregressive predictive model: $p(x_t | x_{t-1}, \dots, x_1)$
- Simplified under Markov assumption: $p(x_t | x_{t-1}, \dots, x_{t-n})$ for $n > 0$
- Useful in language models: estimate $p(w_1, w_2, \dots, w_T)$
- Probability of the next word (in text): $w_t \sim p(w_t | w_{t-1}, \dots, w_{t-n})$
- Zipf's law governs the word distribution for not only unigrams but also the other n -grams:
- high accuracy of NLP models thank to large corpora
- time-lagged NNs \sim n -grams



(Zhang et al, 2020)

6

Tasks for time-lagged NNs

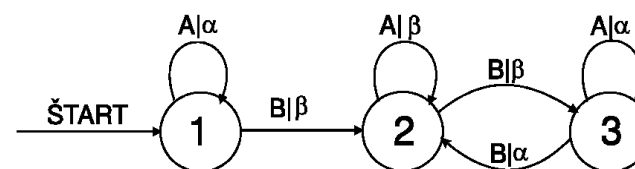
- time series prediction (single-step, multi-step)
- noise cancellation
- adaptive control
- speech recognition
- system identification

BUT... what if the required memory may be unlimited?

- time-lagged NNs have no feedback
- global feedback = facilitator of computational intelligence (Haykin, 2009)

7

Example: Mealy automaton

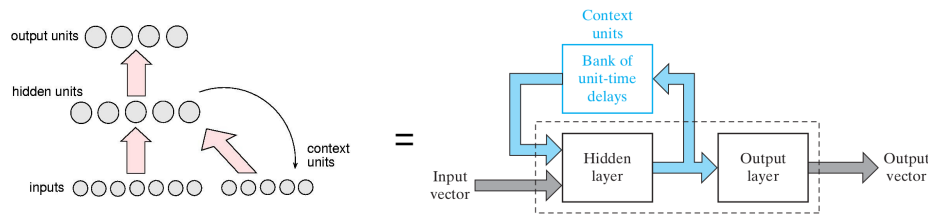


(Tiño, 1997)

- 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 is more appropriate than "past window" \rightarrow **recurrent models**

8

Simple recurrent network (SRN)



Hidden state activation:

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

Output activation:

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

Unit's activation function:

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

Latent variable model: $p(x_t | x_{t-1}, \dots, x_1) \approx p(x_t | h_{t-1})$

Can be trained by various gradient algorithms (BP, BPTT, RTRL,...)

- SRN is a partially recurrent, state-space model
- time is implicitly represented

(Elman, 1990)

9

Example: Next letter prediction task

Task: letter-in-word prediction (sequence of symbols), 5-bit inputs

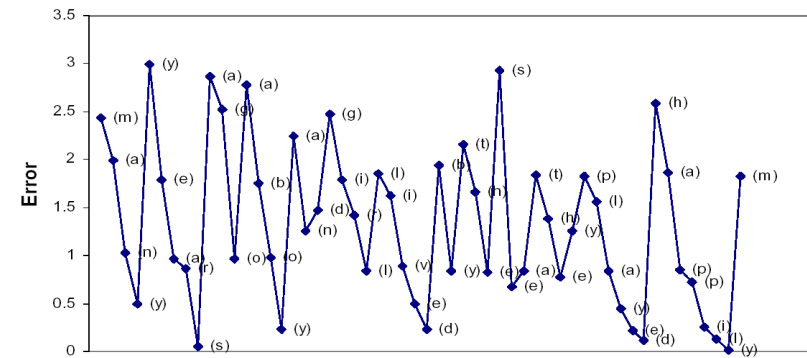
Data: 200 sentences, 4 to 9 words in a sentence

SRN: 5-20-5 units, trained by **back-propagation** (Rumelhart, Hinton & Williams, 1986)

- SRN discovers the notion of "word"

Many years ago boy and girl lived by the sea ...

(Elman, 1990)



10

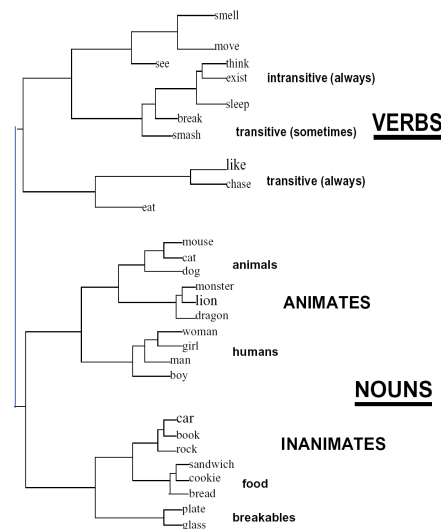
Example: Next word prediction task

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

SRN:
31-150-31,
localist
encoding
of words

Averaged hidden-unit activation vectors

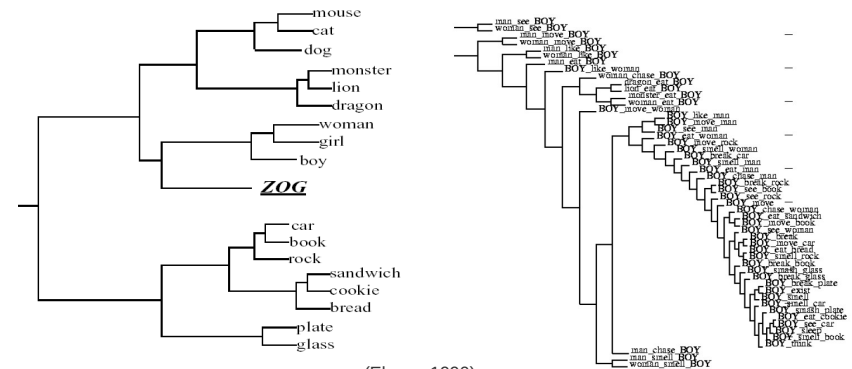


(Elman, 1990)

11

Properties of hidden-unit activations after training

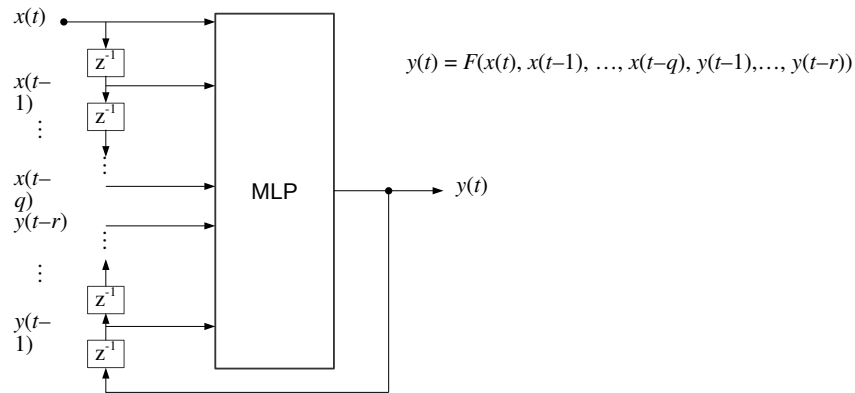
- activations show structure (clusters) in 150-dim. space
- types/tokens distinction: types = centroids of tokens
- representations are hierarchically structured
- representation space would not grow with a growing lexicon
- type vector for a novel word (*zog*) consistent with previous knowledge



(Elman, 1990)

12

Input-output recurrent model (NARX)



- Generic single-input single-output recurrent architecture
- NARX (nonlinear autoregressive model with exogenous inputs (Chen et al, 1990))

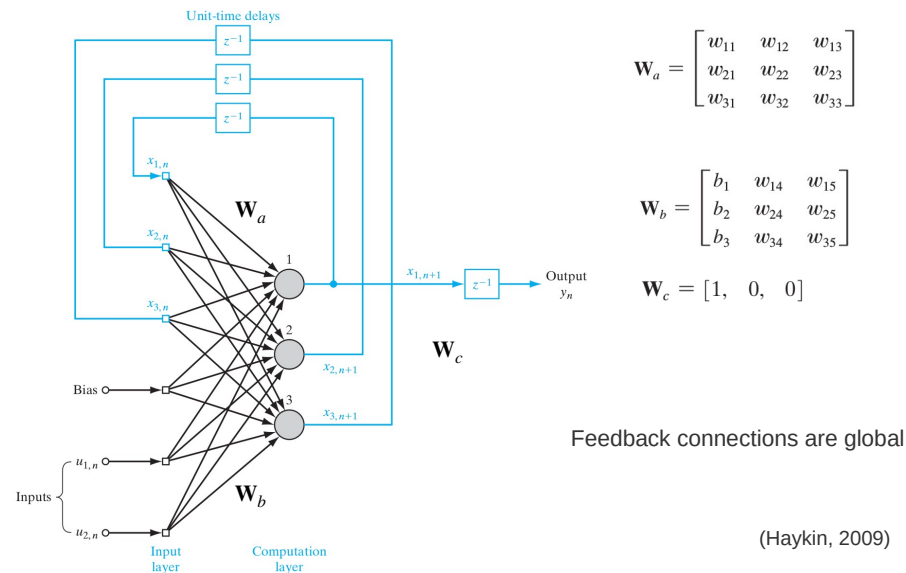
13

Learning algorithms for fully recurrent NNs

- dynamically driven RNNs, with a global feedback
 - acquire (internal) state representations
- (similarly to spatial tasks) two modes: (Williams & Zipser, 1995)
 - epochwise training: epoch = sequence
 - continuous training
- We mention two gradient based algorithms:
 - BPTT (epochwise or continuous), and RTRL (continuous)
- **Heuristics:** (Giles, 1996)
 - start with shorter sequences, then increase length
 - update weights only if training error is larger than threshold
 - consider regularization (e.g. weight decay)

14

Example of a fully recurrent NN



15

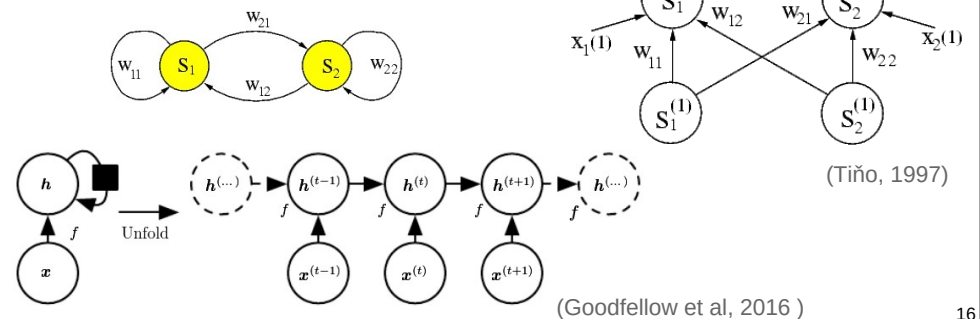
Back-propagation through time

- extension of standard BP algorithm – unfolding in time into a feedforward NN with **identical** weights (Werbos, 1990)
- sequence with inputs $x(1), x(2), \dots, x(T)$

State equation:

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

[in example $i, j \in \{1, 2\}$]



16

BPTT algorithm

- applied after processing each sequence (of length T)
- during **single forward pass** through a sequence: $x(1), x(2), \dots, x(T)$
 - record inputs, local gradients δ
- Overall error: $E_{\text{total}}(T) = \frac{1}{2} \sum_{t=1}^T \sum_{i \in \text{Output}} e_i^2(t)$
- for $t = T$: $\delta_i(t) = f'(net_i) e_i(t)$
for $1 < t < T$: $\delta_i(t) = f'(net_i) [e_i(t) + \sum_{l \in \text{Output}} w_{il} \delta_l(t+1)]$
- **Update weights**: $\Delta w_{ij} = -\alpha \partial E_{\text{total}}(T) / \partial w_{ij} = \alpha \sum_{t=2}^T \delta_i(t) s_j(t-1)$
- impractical for longer sequences (of unknown length)
- truncated BPTT possible (Williams and Peng, 1990)

17

Real-time recurrent learning (RTRL)

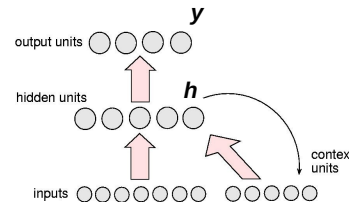
- Units in discrete time: $s_i(t+1) = f(\sum_j w_{ij} s_j(t) + x_i(t))$
- Instantaneous output error: $e_i(t) = d_i(t) - s_i(t)$; $i \in O$ (where targets exist)
 $E(t) = \frac{1}{2} \sum_{k \in O} e_k^2(t)$
- **Update weights**: $\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha \sum_{k \in O} e_k(t) \frac{\partial s_k(t)}{\partial w_{ij}}$
 $\frac{\partial s_k(t)}{\partial w_{ij}} = f'(net_k(t)) \cdot [\delta_{ik}^{\text{kr}} s_j(t-1) + \sum_l w_{kl} \frac{\partial s_l(t-1)}{\partial w_{ij}}]$
 $l \in \text{units feeding to unit } k, \text{ and } \delta_{ik}^{\text{kr}} = 1, \text{ if } k = i, \text{ else } 0.$
– if j pertains to an external input, $x_j(t-1)$ is used instead of $s_j(t-1)$
- **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

(Williams & Zipser, 1989)

18

RNN state space organization

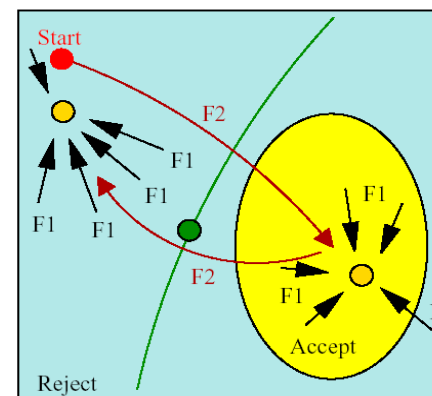
- $y = F(h)$ is a squashed version of a linear transformation \Rightarrow it is smooth and monotonic
- Activations h leading to the same/similar output y are forced to lie **close to each other** in the RNN state space
- Heuristics for enhancing RNN generalization: **cluster** RNN state space into a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



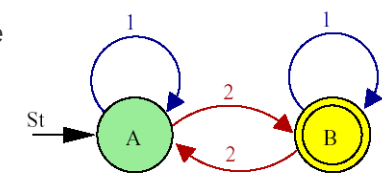
19

Example: Learning a finite state automaton

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



RNN
state
space



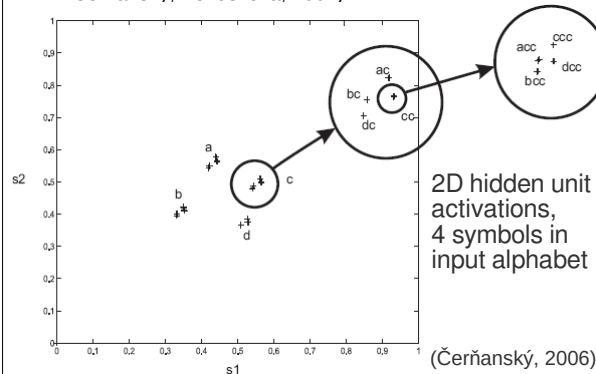
Tiño (2003)

Grammatical:
all strings containing odd number of 2's

20

Architectural bias

- Structured hidden-unit activations in RNN exist prior to training, i.e. model predictions overcome random guess:
- clusters of recurrent activations that emerge prior to training correspond to Markov prediction contexts – histories of symbols are grouped according to number of symbols they share in their suffix (Tíňo, Čerňanský, Beňušková, 2004)



RNNs can outperform finite memory models, but to assess the contribution of training, RNN accuracy should be compared with that of variable-length Markov models (extracted before training as the “null” base models).

21

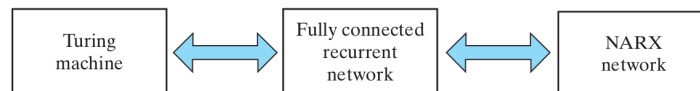
Computational power of recurrent networks I

- recurrent NNs (e.g. NARX, SRN...) are able to simulate finite-state automata, e.g. by learning
 - regular grammars
 - context-free grammars (e.g. $a^n b^n$ ~ saddle-point attractive set)
 - simple context-sensitive grammars (LSTM)
- “Every finite-state machine is equivalent to, and can be ‘simulated’ by, some neural net (NN). That is, given any finite-state machine M, we can build a certain NN which, regarded as a black-box machine, will behave precisely like M!” (Minsky, 1967)
- Pivotal work of learning Reber grammar with a SRN (Cleeremans, Servan-Schreiber & McClelland, 1989)
- *Theorem 1*: “All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions.” (Siegelmann & Sontag, 1991)
 - TM is a more abstract mathematical model than FSM

22

Computational power of recurrent networks II

- *Theorem II*: “NARX networks with one layer of hidden neurons with bounded, one-sided saturated activation functions and a linear output neuron can simulate fully connected RNN with bounded, one-sided saturated (BOSS) activation functions, except for a linear slowdown.” (Siegelmann, 1997)
 - A “linear slowdown” means that if the fully connected RNN with N neurons computes a task of interest in time T , then the total time taken by the equivalent NARX network is $(N+1)T$.
- *Corollary*: NARX networks with one hidden layer of neurons with BOSS activation functions and a linear output neuron are Turing equivalent.



23

Summary

- two classes of architectures (time-lagged, partially or fully recurrent)
- time-lagged models are good for tasks with limited memory
- link to statistical n-gram models
- recurrent models with global feedback learn internal state representations
- links to 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 allows Markovian predictions

24