Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Neural Networks

Lecture 11

Neural networks for sequential data

Igor Farkaš

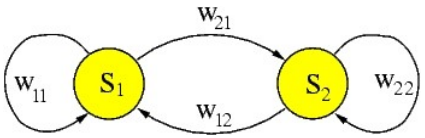
2018

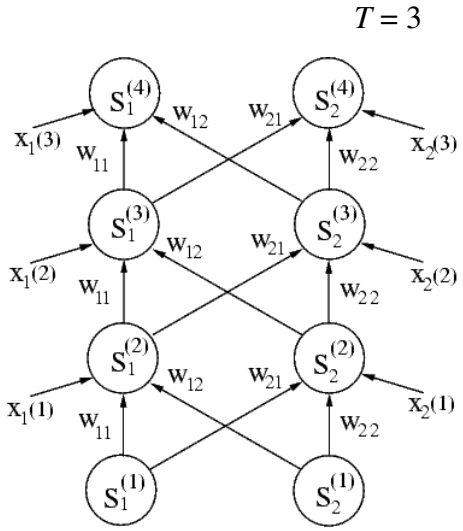
Back-propagation through time

- 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:

$$s_i(t+1) = f\left(\sum_j w_{ij} s_j(t) + x_i(t)\right),$$
$$[in\ our\ example\ \ i, j \in \{1,2\}]$$





(Werbos, 1990)

BPTT algorithm

- applied after processing each sequence (of length T)
- during single forward pass through a sequence: $\boldsymbol{x}(1), \boldsymbol{x}(2), ..., \boldsymbol{x}(T)$
  - record inputs, local gradients δ
- Overall error: $E_{\text{total}}(T) = \frac{1}{2} \sum_{t=1}^{T} \sum_{i \in O} e_i^2(t)$
- 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: $\Delta w_{ij} = -\alpha\, \partial E_{\text{total}}(T) / \partial w_{ij} = \alpha \sum_{t=2}^{T} \delta_i(t)\, s_j(t-1)$

- impractical for longer sequences (of unknown length)
- truncated BPTT possible (Williams and Peng, 1990)

Real-time recurrent learning (RTRL)

- Units in discrete time: $s_i(t+1) = f\left(\sum_j w_{ij} s_j(t) + x_i(t)\right)$
- 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:

$$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha \sum_{k \in O} e_k(t) \frac{\partial s_k(t)}{\partial w_{ij}}$$

$$\frac{\partial s_k(t)}{\partial w_{ij}} = f'(net_k(t)).\left[\delta_{ik}^{kr}\, s_j(t-1) + \sum_l w_{kl} \frac{\partial s_l(t-1)}{\partial w_{ij}}\right]$$

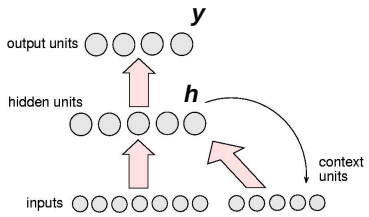
  $l \in$ units feeding to unit k, and if $k == i$, $\delta^{kr}_{ik} = 1$, otherwise 0.

  - 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)

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)
- Symbolic dynamics helps understand RNNs

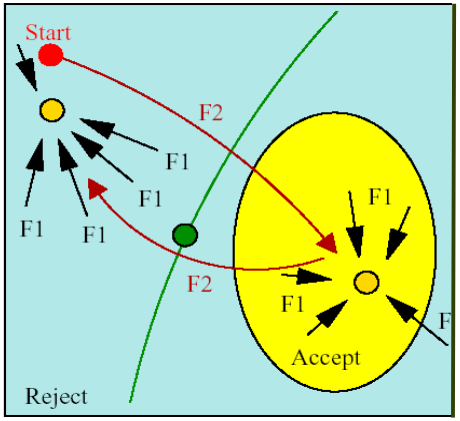
Difficulties in training RNN

- vanishing gradients problem – difficulty to learn long-term dependencies by gradient-based algorithms
- For many practical applications, it is necessary that a RNN stores state information for an arbitrary duration and to do so in the presence of noise (Bengio et al., 1994)
- Robust information latching in a recurrent network is accomplished if the states of the network are contained in the reduced attracting set of a hyperbolic attractor, (Bengio et al., 1994)
  - i.e. in the basin of attraction for which all the eigenvalues of the associated Jacobian (of the linearized model) have an absolute value < 1.
- What helps: second-order optimization techniques, Kalman filtering, LSTM...

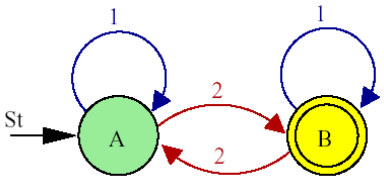
Example: Learning a finite state automaton

• 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

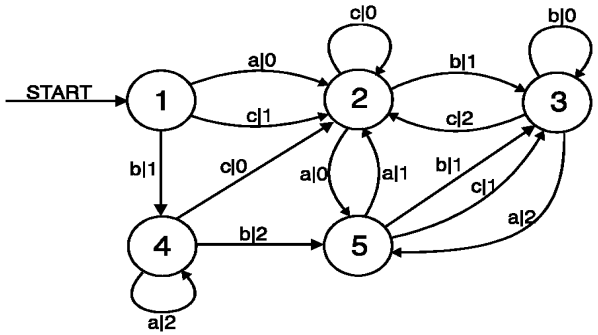


Tiňo (2003)

Grammatical:
all strings containing odd number of 2's

Example: temporal association task

3+1 input symbols
3+1 output symbols



Training sequences:

$$acccb! \rightarrow 00001x,\ caaab! \rightarrow 10101x,\ bbbbbb! \rightarrow 121000x,\ \text{atd'}.$$

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

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

Example: Modeling recursive processing in humans

A. Counting recursion $a^n b^n$ $N^n V^n$

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.

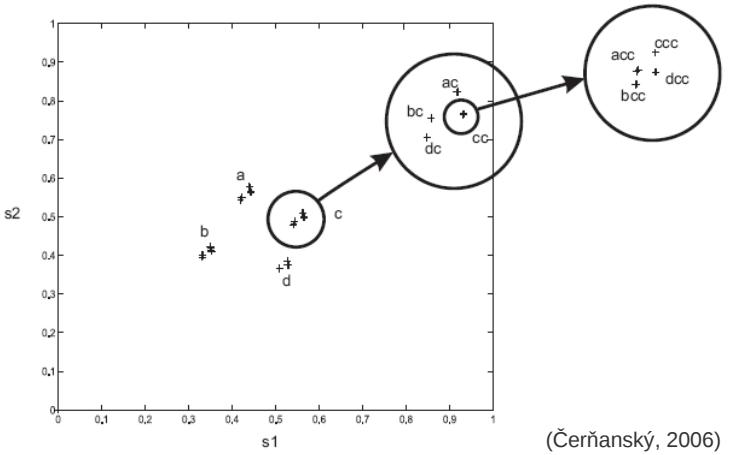
(Christiansen & Chater, 1999)

Architectural bias

- Structured hidden-unit activations in RNN exist prior to training

(Kolen, 1994; Tiňo, Čerňanský, Beňušková, 2004; Čerňanský, Makula, Beňušková, 2007)

Hidden unit activations (2D space), 4 symbols in input alphabet



(Čerňanský, 2006)

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).

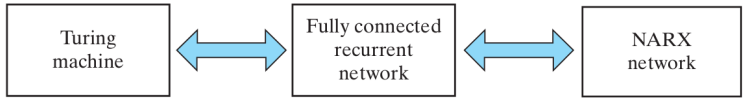
RNNs have a potential to outperform finite memory models, but to appreciate how much information has really been induced during training, RNN performance should always be compared with that of VLMMs extracted before training as the "null" base models.

Computational power of recurrent networks I

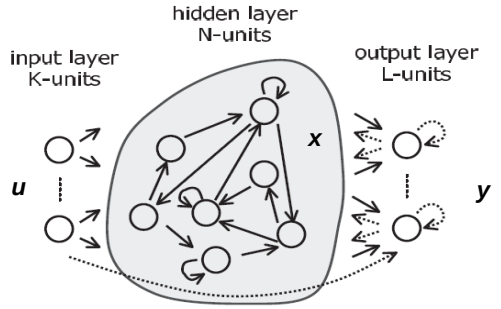
- recurrent NNs (e.g. NARX or SRN) are able to simulate finite-state automata, e.g. by learning
  - regular grammars
  - context-free grammars (e.g. $a^n b^n$ ~ saddle-point attractive set)
- "Every finite-state machine is equivalent to, and can be 'simulated' by, some neural net (NN). That is, given 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 I:* "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

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.



Reservoir computing – echo-state network



(Jaeger, 2001)

ESN can have an SRN architecture, but also additional connections are possible (useful for some tasks).

Reservoir units: usually nonlinear (tanh), can also be linear.

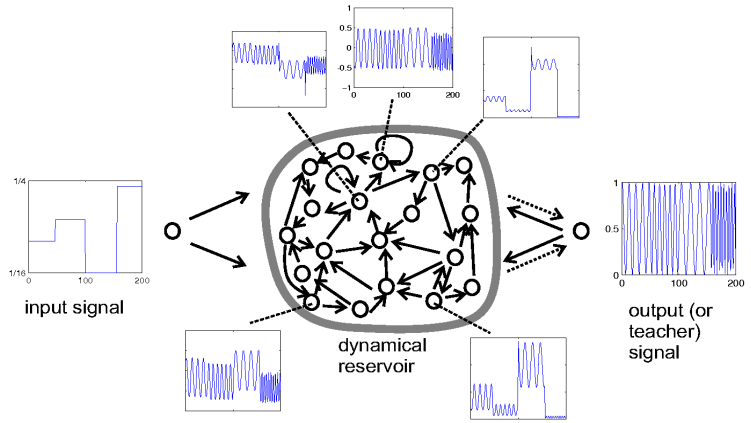
System equations:

$$\boldsymbol{x}(t)=f\left(\mathbf{W}\,\boldsymbol{x}(t-1)+\mathbf{W}^{\text{inp}}\boldsymbol{u}(t)+\mathbf{W}^{\text{fb}}\boldsymbol{y}(t)\right)$$

$$\boldsymbol{y}(t)=f^{out}\left(\mathbf{W}^{\text{out}}\,\boldsymbol{z}(t)\right) \qquad \mathbf{W}^{out}\ \sim L\times(N+K)]$$

$$\boldsymbol{z}(t)=[\boldsymbol{x}(t);\boldsymbol{u}(t)]$$

Echo State Network (ctd)



- studied issues: memory capacity, information transfer, ...
- edge of stability = interesting regime (may be optimal w.r.t. info processing)

ESN training

- Initialize
  - create the reservoir with echo-state property (asymptotic properties of reservoir dynamics are related to driving signal): (Jaeger, 2001)

    Network F: $X \times U \rightarrow X$ (with compactness condition) has the **echo state property** w.r.t. U, if for any left infinite input sequence $u^{-\infty} \in U^{-\infty}$ and any two state vector sequences $x^{-\infty}, y^{-\infty} \in X^{-\infty}$ compatible with $u^{-\infty}$, it holds that $x_0 = y_0$.
  - small random input weights (from uniform or gaussian distribution)
- Collect reservoir states
  - feed the input sequence into the network (apply state equation)
- Compute output weights
  - Supervised learning, via pseudoinverse of **X,** or RLS

• ESN reservoir has a Markov property (in symbolic dynamics)

ESN properties

Echo-state property (ESP): depends on spectral properties of **W** = (typically) random *sparse* matrix, measures:

- spectral radius: ρ(**W**) = $|\lambda_{max}|$, i.e. largest absolute eigenvalue,
- spectral norm: $s_{max}(\mathbf{W})$ = largest singular value , relation: $0 \le \rho(\mathbf{W}) \le s_{max}(\mathbf{W})$

- Criteria for ESP: $s_{max}(\mathbf{W}) < 1$ → too strict, ρ(**W**) < 1 not sufficient
- New recipe (Yildiz & Jaeger, 2012): (i) random $w_{ij} \ge 0$, (ii) scale **W** so that ρ(**W**) < 1, (iii) change the signs of a desired number of entries to get some $w_{ij} < 0$ as well.
- ρ(**W**) ≈ 1 tends to be a "turning point" in behavior (e.g. memory capacity)

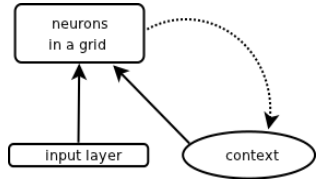
Memory capacity (MC): reflects the ability to retrieve input data from the reservoir

- scalar i.i.d. inputs assumed, MC depends on **W**, $\mathbf{W}^{inp}$, reservoir size *N*, sparsity,...

$$\mathrm{MC} = \sum_{k=1}^{k_{\max}} \mathrm{MC}_k = \sum_{k=1}^{k_{\max}} \frac{cov^2(u(t-k), y_k(t))}{var(u(t)) \cdot var(y_k(t))} \qquad y_k(t) = \mathbf{w}_k^{\mathrm{out}}\mathbf{x}(t) = \tilde{u}(t-k)$$
$$k_{\max} = L$$

Reservoir stability – measured by characteristic Lyapunov exponent (Sprott, 2003), that quantifies the average divergence of state space trajectories under perturbations.

SOMs for sequential data



Extensions of SOMs for sequential data

What can they learn to represent?

- (Markovian) map of suffixes only?

Neurons: $i = 1,2, ..., N$, input dim = d Winner $i^* = \arg\min_j\{d_j(t)\}$

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

Examples of models:

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

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

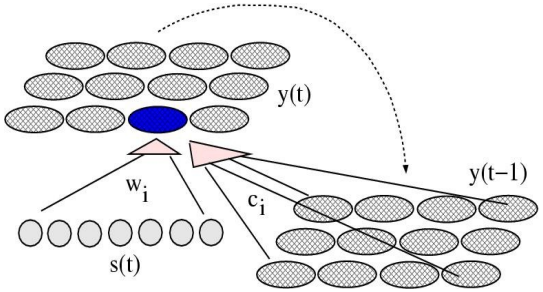
Recursive SOM: $d_i(t) = a.\|\boldsymbol{x}(t) - \boldsymbol{w}_i\|^2 + b.\|\boldsymbol{r}(t) - \boldsymbol{c}_i\|^2$ $y_i = \exp(-d_i)$

$$\boldsymbol{r}(t) = [y_1(t-1), \ldots, y_N(t-1)] \qquad \boldsymbol{c}_i, \boldsymbol{r} \in R^N$$

Recursive self-organizing map (RecSOM)

RecSOM can lead to more complex dynamics compared to other unsupervised models.
(Tiňo, Farkaš, van Mourik, 2006)

(Voegtlin, 2002)



Quantization error on unit *i*:

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

Output of unit *i*: $y_i = \exp(-d_i)$

Learning rules:

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

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

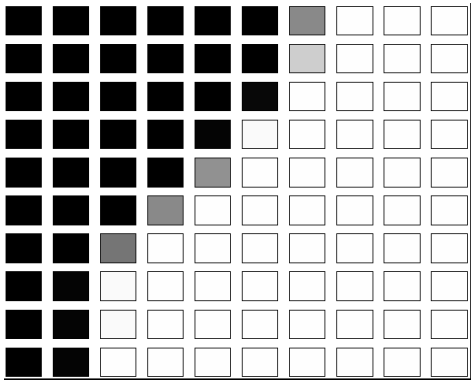
$0 < z \ll 1$ (constant) learning rate h_{ik} – (shrinking) neighborhood function

RecSOM: trained on stochastic 2-state automaton: weights

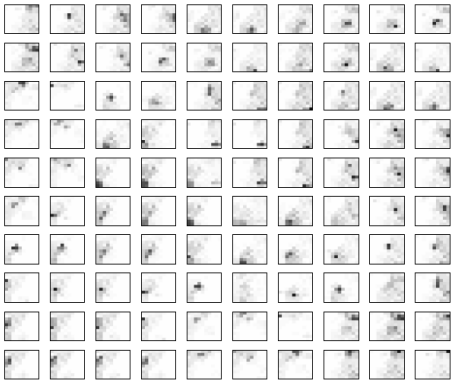
RecSOM: 10x10 units, 1D inputs

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

Input weights



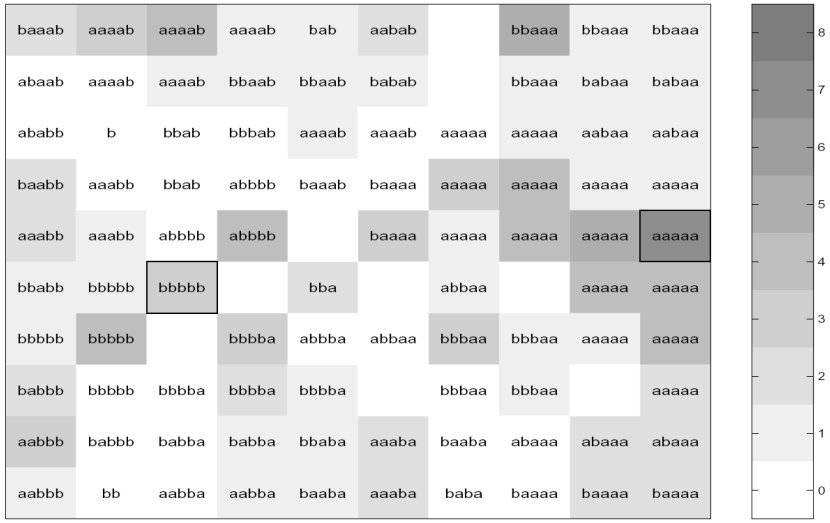
Context weights



(Tiňo, Farkaš, van Mourik, 2006)

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

- Units' receptive fields of length 5 shown, color denotes distances b/w units' weights



Long short-term memory (LSTM)

- Introduced to deal with long-term dependencies (Hochreiter & Schmidhuber, (1997)
  - cf. regular grammar with CFG
- Using trained gates, it introduces self-loops to produce paths where the gradient can flow for long (neither exploding nor vanishing)
- the cell state is the core of the LSTM, controlled by the gates
- Various LSTM variants proposed
- Trainable with 2nd order methods, Nesterov gradient, but also SGD…
- Clipping the gradient found useful, e.g. element-wise (Mikolov, 2012); or by L2 norm (Pascanu et al, 2013).
- Very successful in various applications

LSTM dynamics

Input gates

$$\boldsymbol{g}(t)=\sigma(\mathbf{U}^{\text{inp}}\boldsymbol{x}(t)+\mathbf{W}^{\text{inp}}\boldsymbol{h}(t-1)+\mathbf{b}^{\text{inp}})$$

Forget gates

$$\boldsymbol{f}(t)=\sigma(\mathbf{U}^{\text{fgt}}\boldsymbol{x}(t)+\mathbf{W}^{\text{fgt}}\boldsymbol{h}(t-1)+\boldsymbol{b}^{\text{fgt}})$$

Cell state

$$\boldsymbol{s}(t)=\boldsymbol{f}(t).*\boldsymbol{s}(t-1)+ \boldsymbol{g}(t).*\sigma(\mathbf{U}^{\text{fgt}}\boldsymbol{x}(t)+\mathbf{W}^{\text{fgt}}\boldsymbol{h}(t-1)+\boldsymbol{b})$$

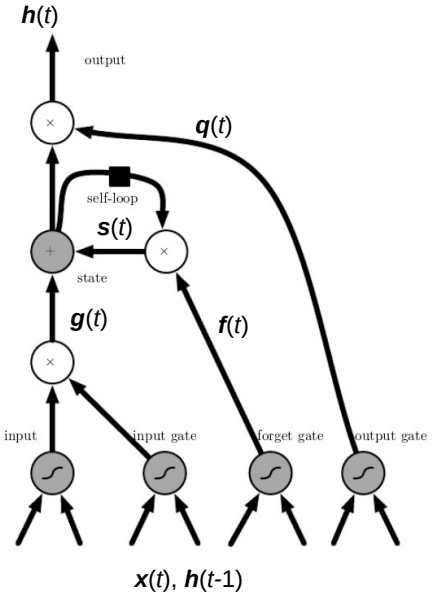
Output gates

$$\boldsymbol{q}(t)=\sigma(\mathbf{U}^{\text{out}}\boldsymbol{x}(t)+\mathbf{W}^{\text{out}}\boldsymbol{h}(t-1)+\boldsymbol{b}^{\text{out}})$$

LSTM output

$$\boldsymbol{h}(t)=\tanh(\boldsymbol{s}(t)).*\boldsymbol{q}(t)$$

.* = elementwise multiplication



(Goodfellow et al, 2015)

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
- existence of architectural bias
- echo-state networks
- recursive self-organizing maps
- more complex RNN architecture – LSTM, superior in performance