Sequence learning and Recurrent Neural Networks

Machine Learning

Jürgen Schmidhuber

Michael Wand, Paulo Rauber

TAs: Róbert Csordás, Aleksandar Stanic, Xingdong Zuo, Francesco Faccio

Sequence Learning

- Up to now we have looked at *static mappings*:

$$y = f(x(t)),\ \forall t$$

where *t,* time, just imposes an ordering on the input patterns.

- It doesn’t matter **when** *x* was presented to the network

Sequence Learning

- Now we look at sequential inputs where the output *y* can depend on more than just the immediate input:

$$y = f(s(t)) = F(x(t), x(t-1), \ldots, x(1))$$

where *s(t)* is the ***state***, which can change every time a new input arrives:

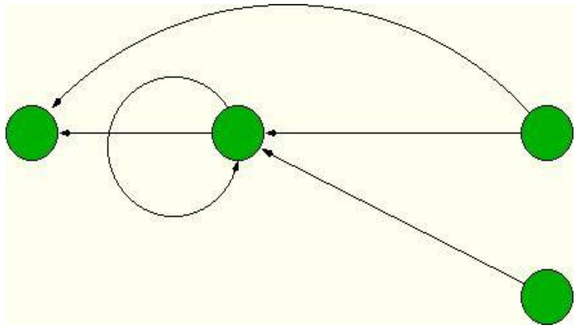
$$s(t+1) \leftarrow g(s(t), x(t))$$

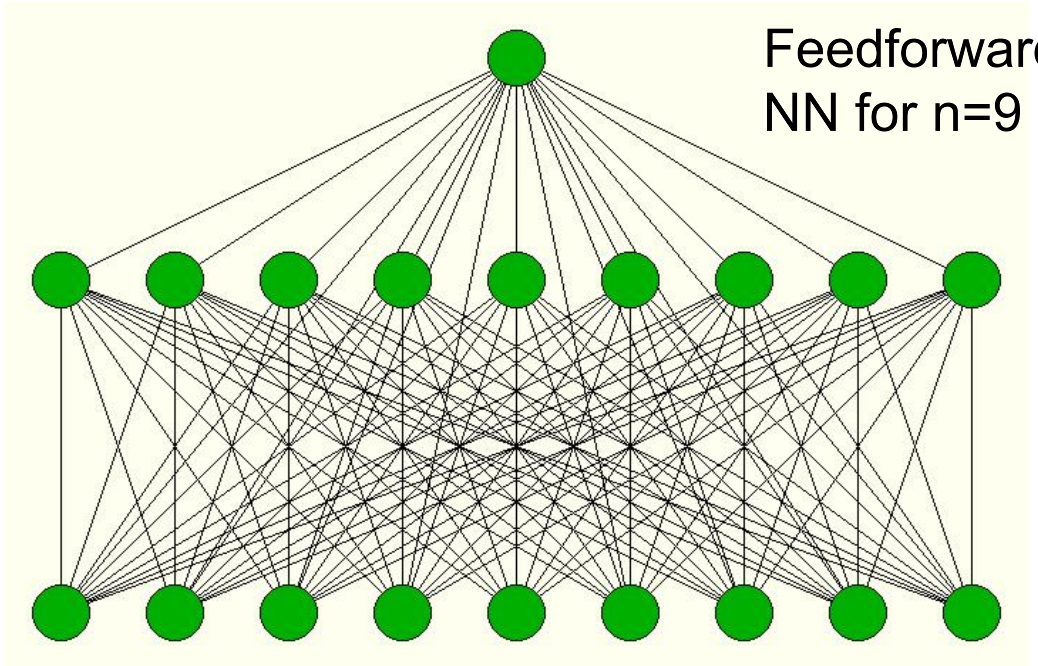
- State provides memory, which in NNs is implemented by *feedback* or *recurrent* connections

Why study sequences?

- Many natural processes are inherently sequential
  - Speech
  - Vision
  - Natural language
  - DNA
- In robotics, tasks short-term memory can be essential for determining the state of the world, due to limited sensor information

Even static problems may profit from RNNs, e.g., *n*-bit parity: number of 1 bits odd?





- RNN much faster - random weights: only 1000 trials!
- fewer parameters
- generalizes to *all n*
- natural solution

Sequential Training Set

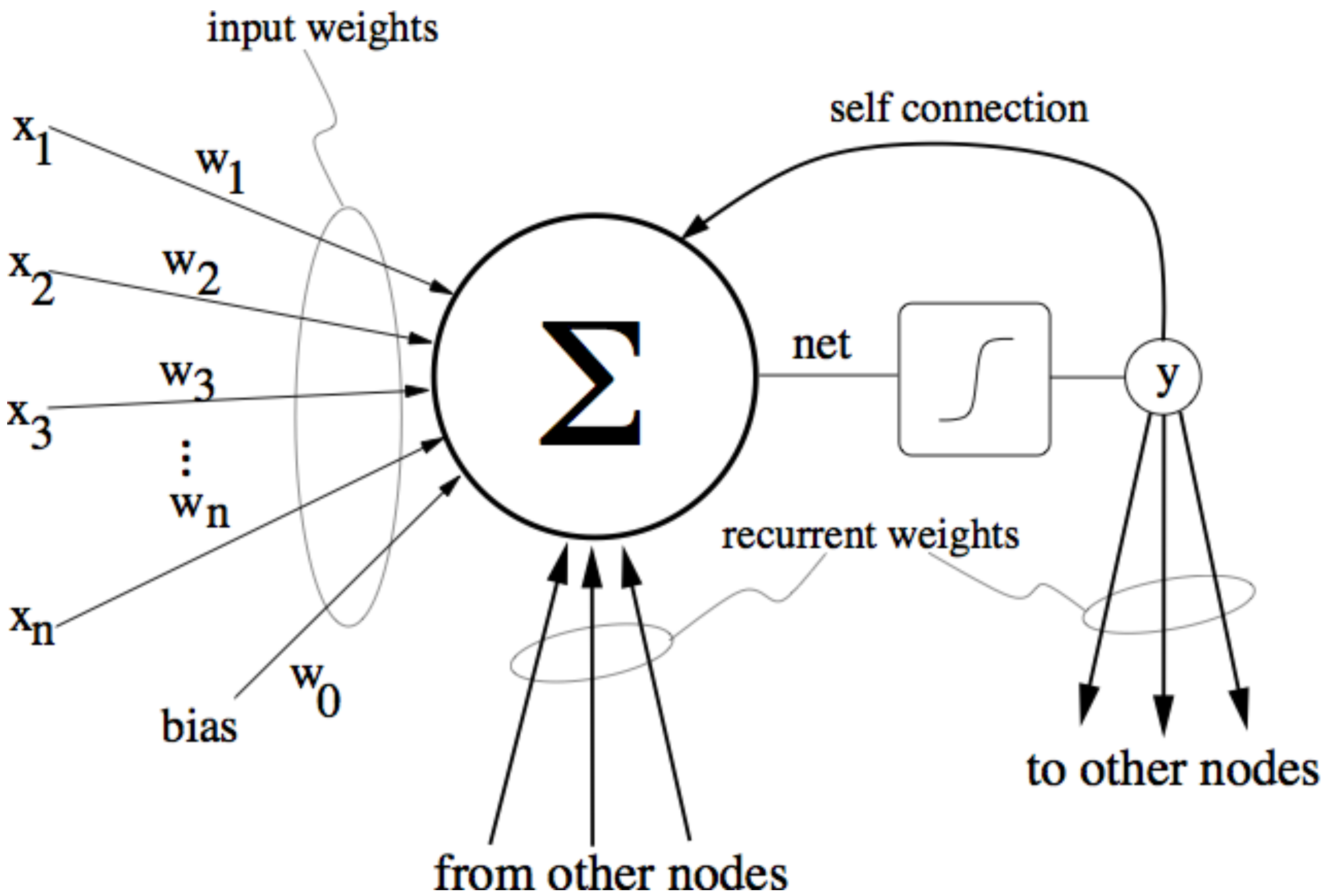
$$\text{example} \equiv [[\text{input sequence}],\ \text{target}]$$

$$\text{input sequence} \equiv \left[\mathbf{x}(t), \mathbf{x}(t-1), \mathbf{x}(t-2), \ldots, \mathbf{x}(1)\right]$$

$$\begin{gathered}
\left[[\mathbf{x}^1(t_1), \mathbf{x}^1(t_1-1), \mathbf{x}^1(t_1-2), \ldots, \mathbf{x}^1(1)],\ \mathbf{d}^1\right] \\
\left[[\mathbf{x}^2(t_2), \mathbf{x}^2(t_2-1), \mathbf{x}^2(t_2-2), \ldots, \mathbf{x}^2(1)],\ \mathbf{d}^2\right] \\
\left[[\mathbf{x}^3(t_3), \mathbf{x}^3(t_3-1), \mathbf{x}^3(t_3-2), \ldots, \mathbf{x}^3(1)],\ \mathbf{d}^3\right] \\
\vdots \\
\left[[\mathbf{x}^N(t_N), \mathbf{x}^N(t_N-1), \mathbf{x}^N(t_N-2), \ldots, \mathbf{x}^N(1)],\ \mathbf{d}^N\right]
\end{gathered}$$

where t_i is the length of sequence i,
x and **d** are vectors

Recurrent Non-Linear Neuron



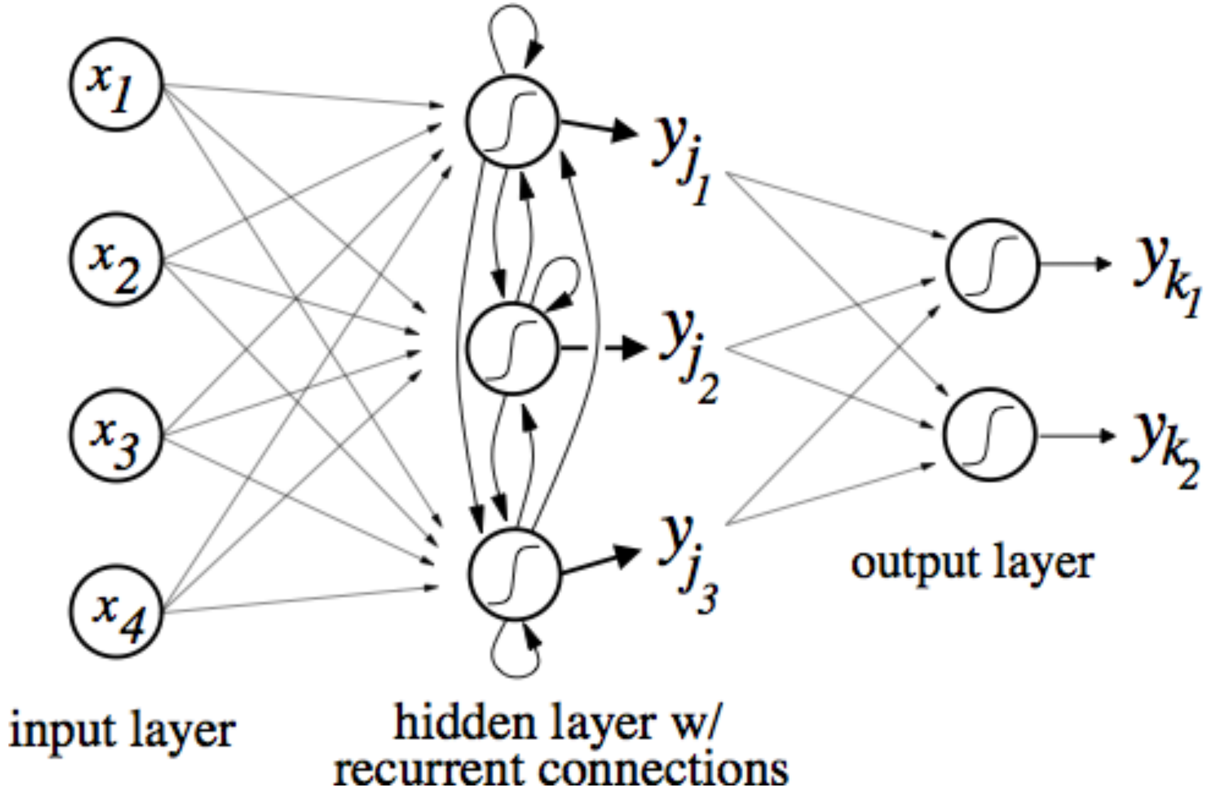
Recurrent Non-Linear Neuron

Now the output of a unit depends on both the input and also the output from other neurons

$$y_k = \sigma\left(\underbrace{\sum_{i=1}^{I} w_{ik} x_i}_{\text{input connections}} + \underbrace{\sum_{j=1}^{H} w_{jk} y_j}_{\text{recurrent connections}} + b \right)$$

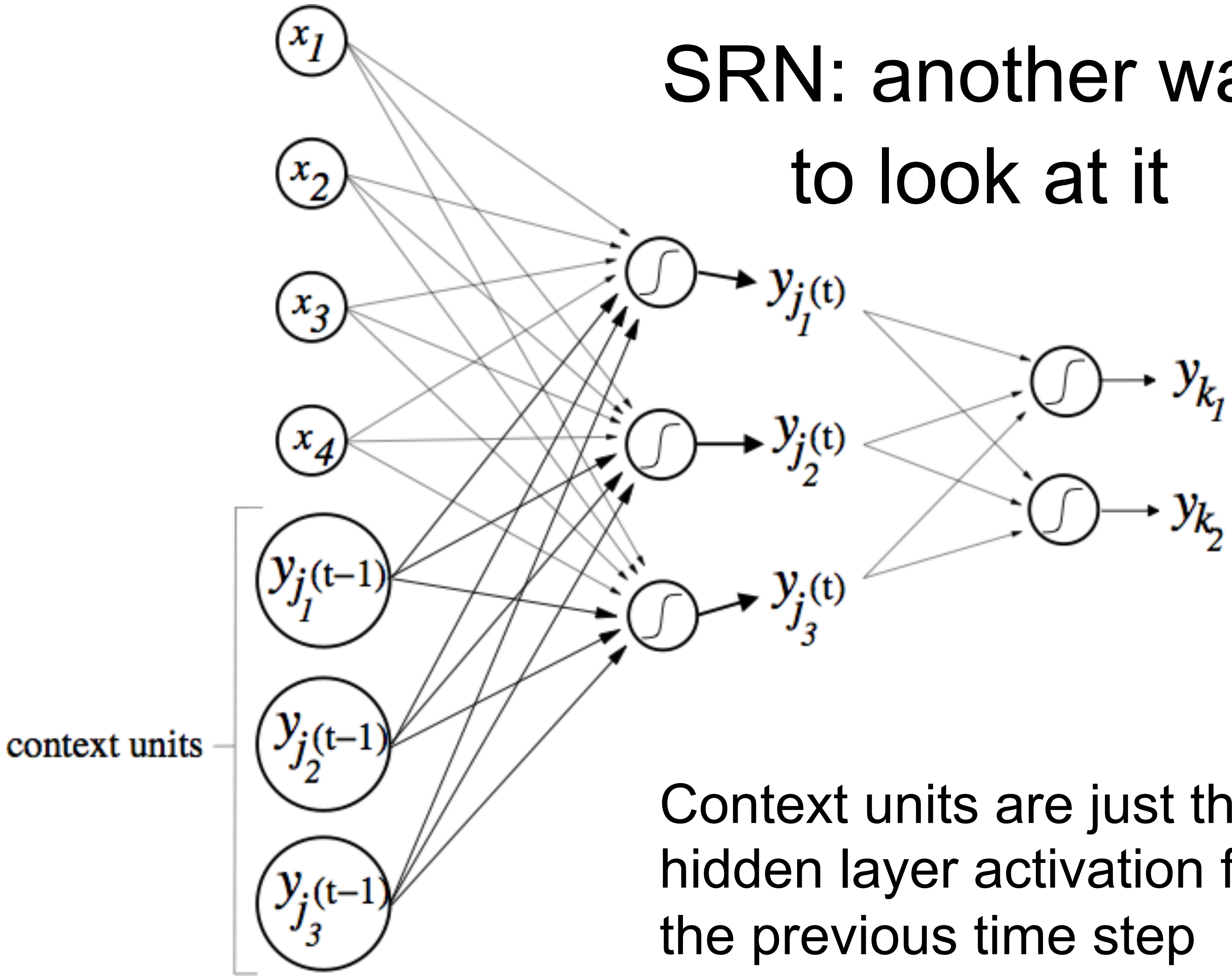
where I is the number of inputs to neuron y_k and
H is the number of hidden units in the same layer as y_k

Simple Recurrent Network (SRN)



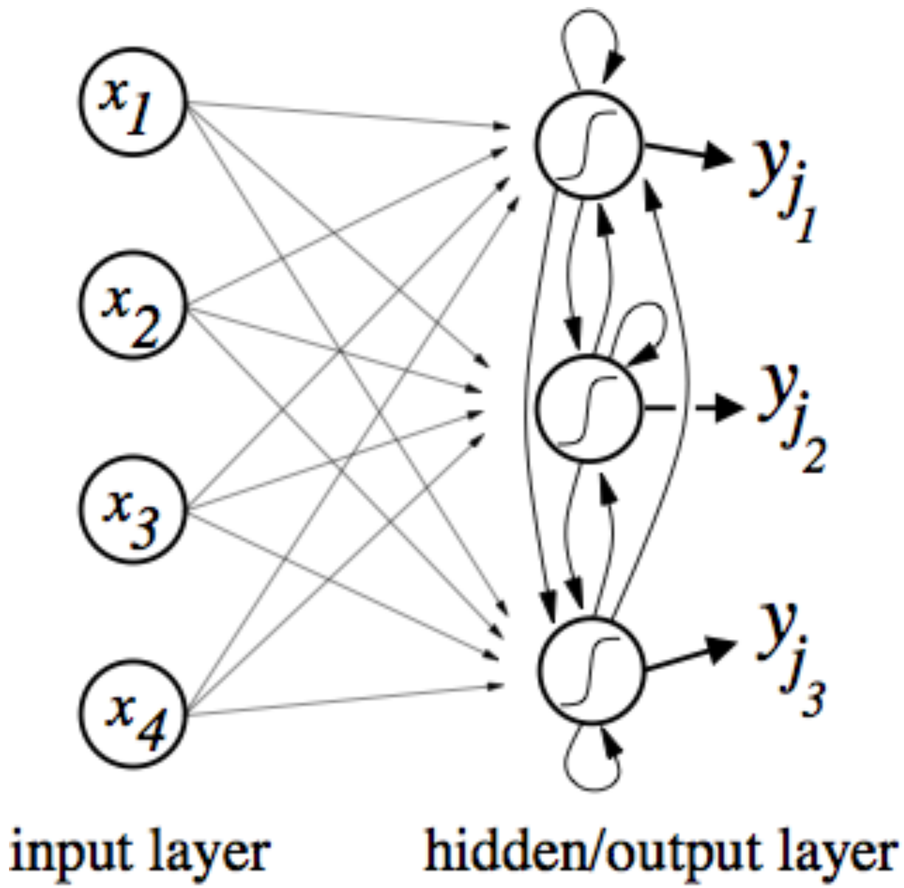
Hidden units now have state, or ***memory***,
which is dependent on all previous inputs

SRN: another way to look at it



Context units are just the hidden layer activation from the previous time step

Fully Connected RNN



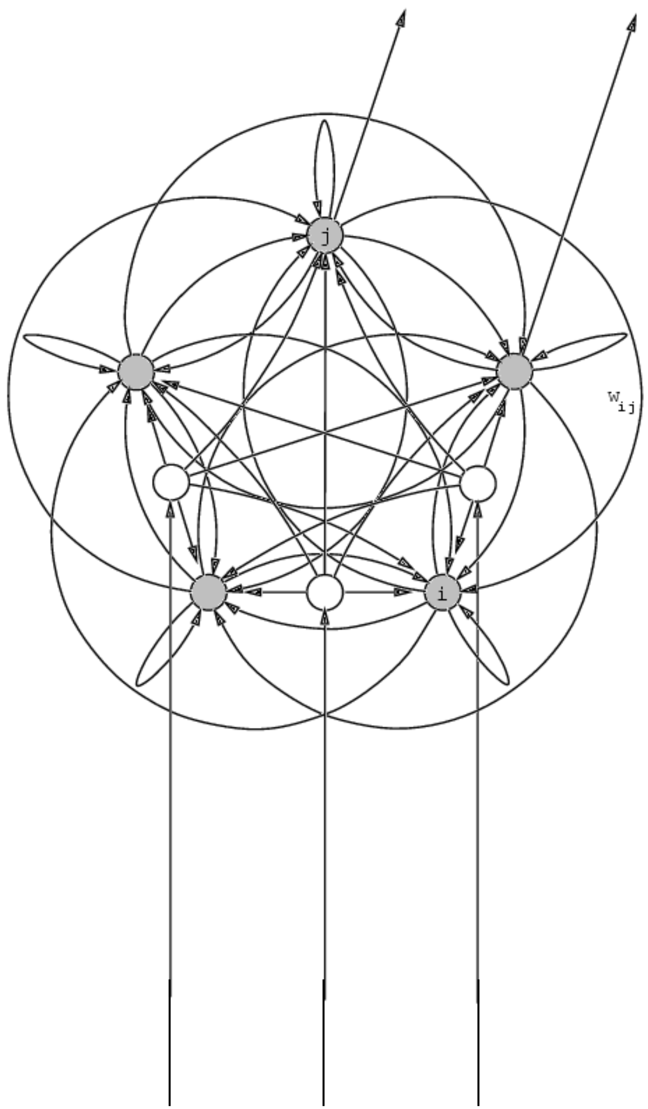
Can approximate any differentiable trajectory
Same as SRN but without output layer

Recurrent networks are Turing-complete!

- All the models we've talked about until today can only implement input output mappings
- Recurrent nets are dynamical systems
- A large enough feedforward net (e.g. MLP) can approximate any continuous function
- A recurrent network can implement any algorithm (modulo storage size)
- In practice, easier to learn some algorithms than others...

Gradient-based RNNs:
∂ wish / ∂ program

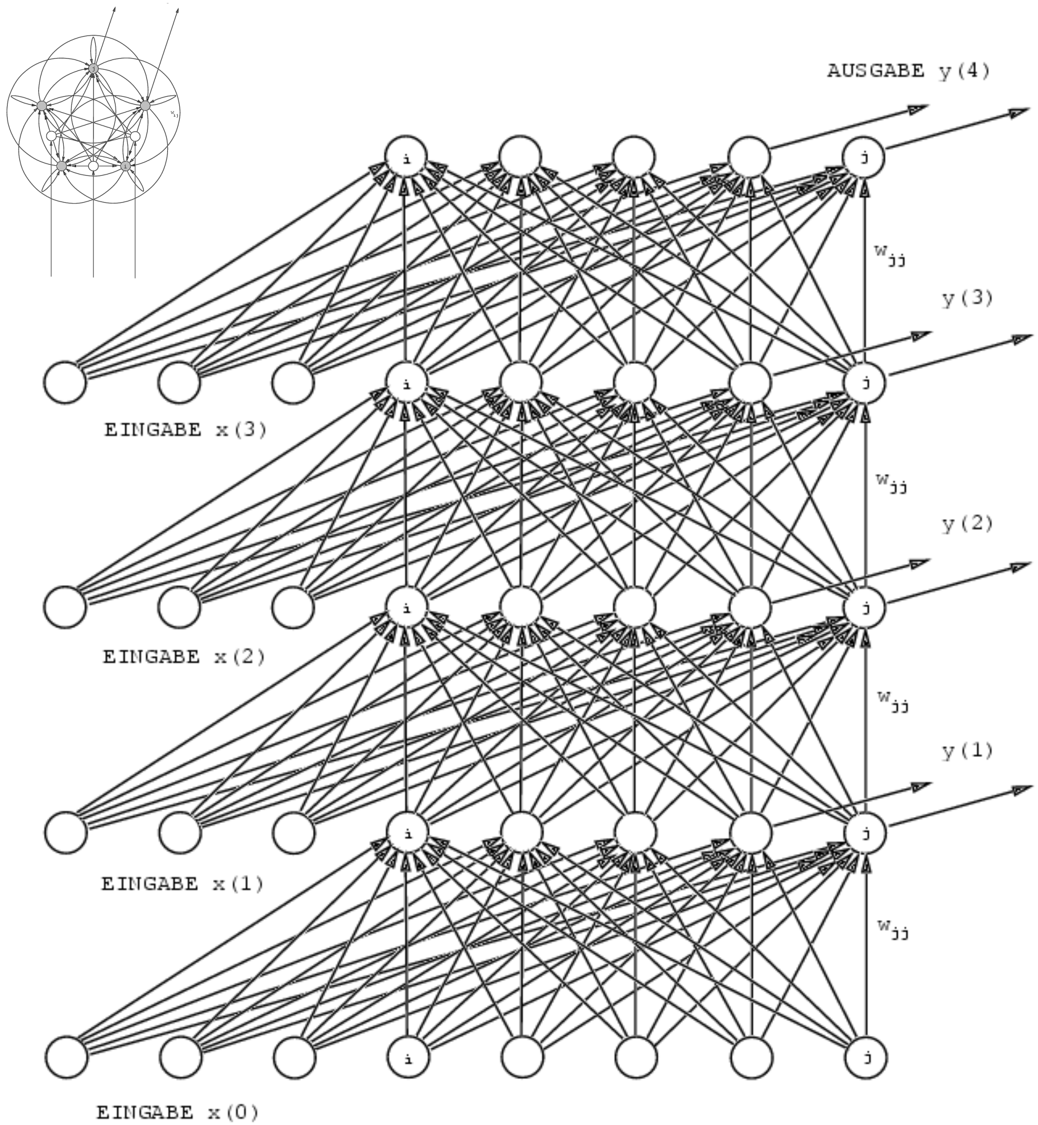
- RNN weight matrices = general algorithm space
- Differentiate objective with respect to program
- Obtain gradient or search direction in program space



- $net_k(t) = \Sigma_i w_{ki}\, y_i(t-1)$
- Forward: $y_k(t)=f_k\,(net_k(t))$
- Error: $e_k(t)=f_k'(net_k(t))\Sigma_i\, w_{ik}\, e_i(t+1)$

80s: BPTT, RTRL - gradients based on "unfolding" etc.

(Williams, Werbos, Robinson)



$$E = \sum_{seq} \sum_{s} \sum_{t} \sum_{o_i} (o_i^s(t) - d_i^s(t))^2$$

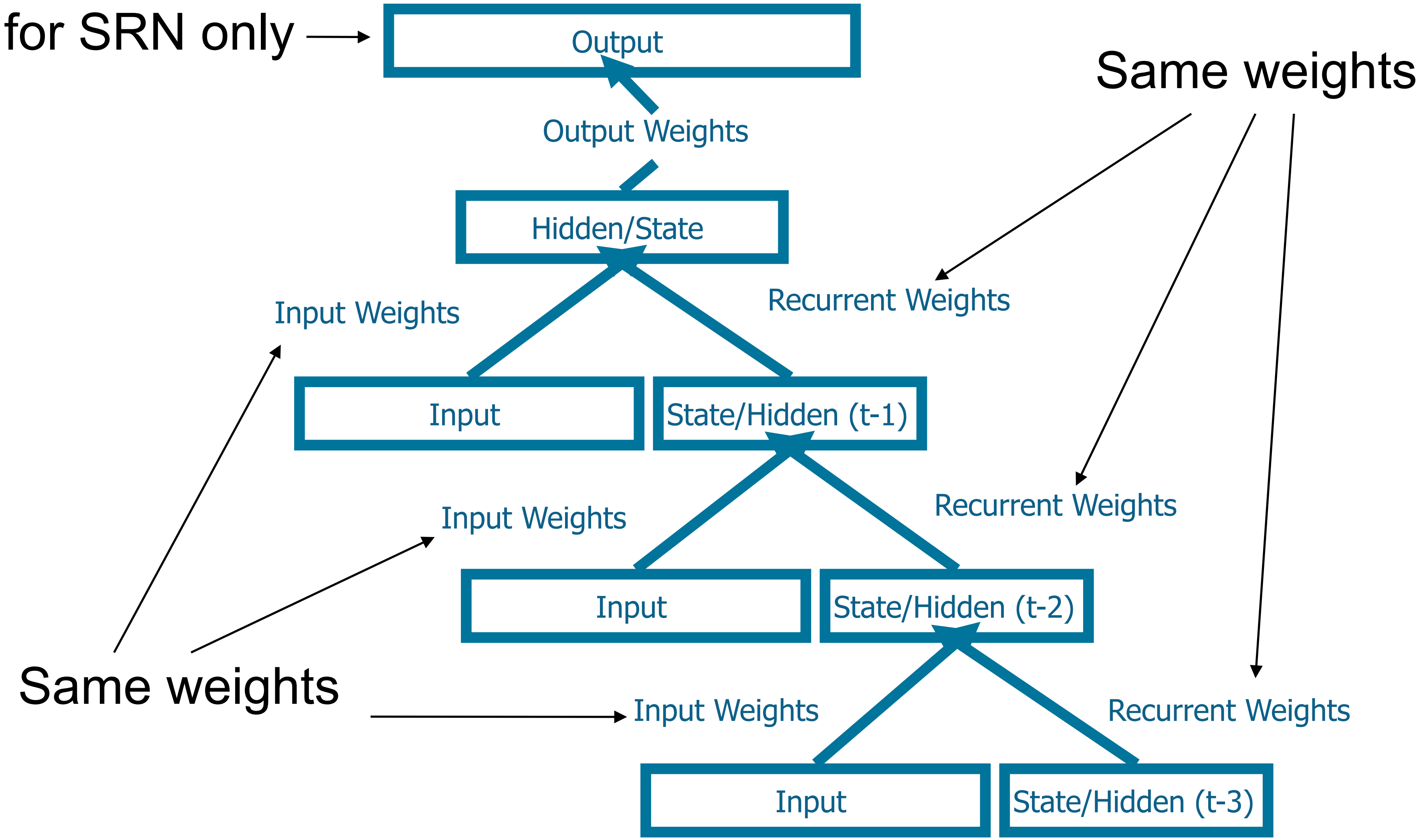
$$\Delta w \propto \frac{\partial E}{\partial w}$$

Backpropagation Through Time

- Just like backpropagation but network is "unfolded" spatially for each time-step in input sequence
- For an *n*-step sequence, we get a network with *n*-layers
- Each layer has the same weights
- Error at output is propagated back through all layers

Backpropagation Through Time

Propagate error further back

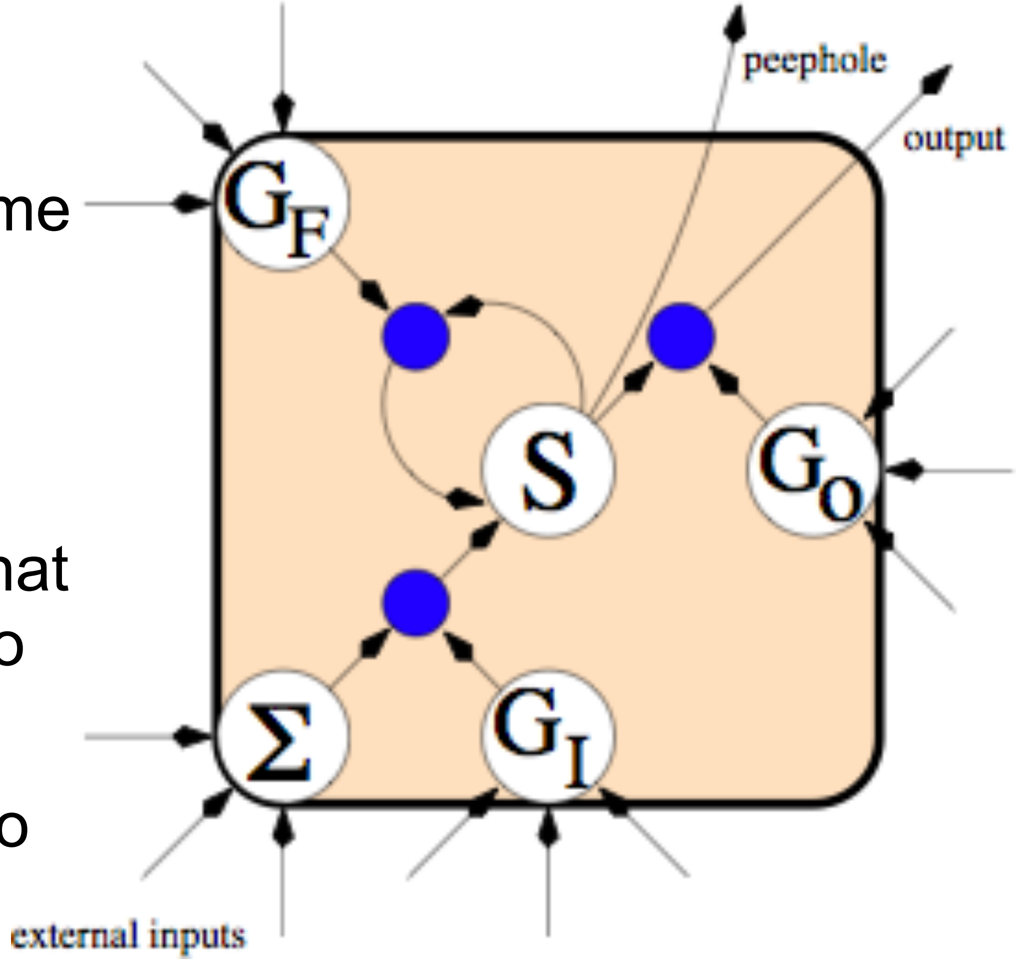


Vanishing error gradient

- Although RNNs can represent arbitrary sequential behavior, the training suffers from dimensionality
- Once the output depends on some input more than around 10 time-steps in the past, they become very difficult to train
- The error gradient becomes very small, so that the weights cannot be adjusted to respond to events far in past
- We might as well use an MLP with a input layer *n* time-steps wide… if you know *n* in advance!
- **Solutions**:
  - Long Short-Term Memory

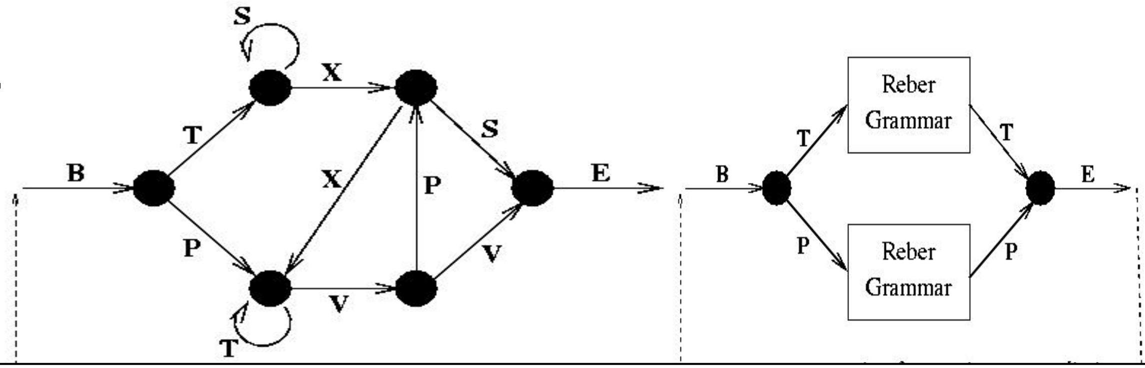
Long Short-Term Memory (LSTM)

- LSTM nets have *memory cells* with a linear state S that keeps error flowing back in time and is controlled by 3 gates
- Input gate (Gi) controls what information enters the state
- Output gate (Go) controls what information leaves the state to other cells
- Forget gate (Gf) allows cell to forget state when no longer needed



LSTM Cell

Regular Grammars: LSTM vs Simple RNNs & RTRL



method	hidden units	# weights	learning rate	% of success	success after
RTRL	3	≈ 170	0.05	"some fraction"	173,000
RTRL	12	≈ 494	0.1	"some fraction"	25,000
ELM	15	≈ 435		0	>200,000
RCC	7-9	≈ 119-198		50	182,000
LSTM	4 blocks, size 1	264	0.1	100	39,740
LSTM	3 blocks, size 2	276	0.1	100	21,730
LSTM	3 blocks, size 2	276	0.2	97	14,060
LSTM	4 blocks, size 1	264	0.5	97	9,500
LSTM	3 blocks, size 2	276	0.5	100	8,440

Contextfree / Contextsensitive Languages

	Train[n]	% Sol.	Test[n]
A^nB^n			
Wiles & Elman 95	1...11	20%	1...18
LSTM	1...10	100%	1...1000
$A^nB^nC^n$			
LSTM	1...50	100%	1...500

What this means:	LSTM + Kalman: n=22,000,000 (Perez, 2002)!!!

--------------------*LEGAL:*---------------------

aaaaa…..aaabbbbb…..bbbccccc…..ccc

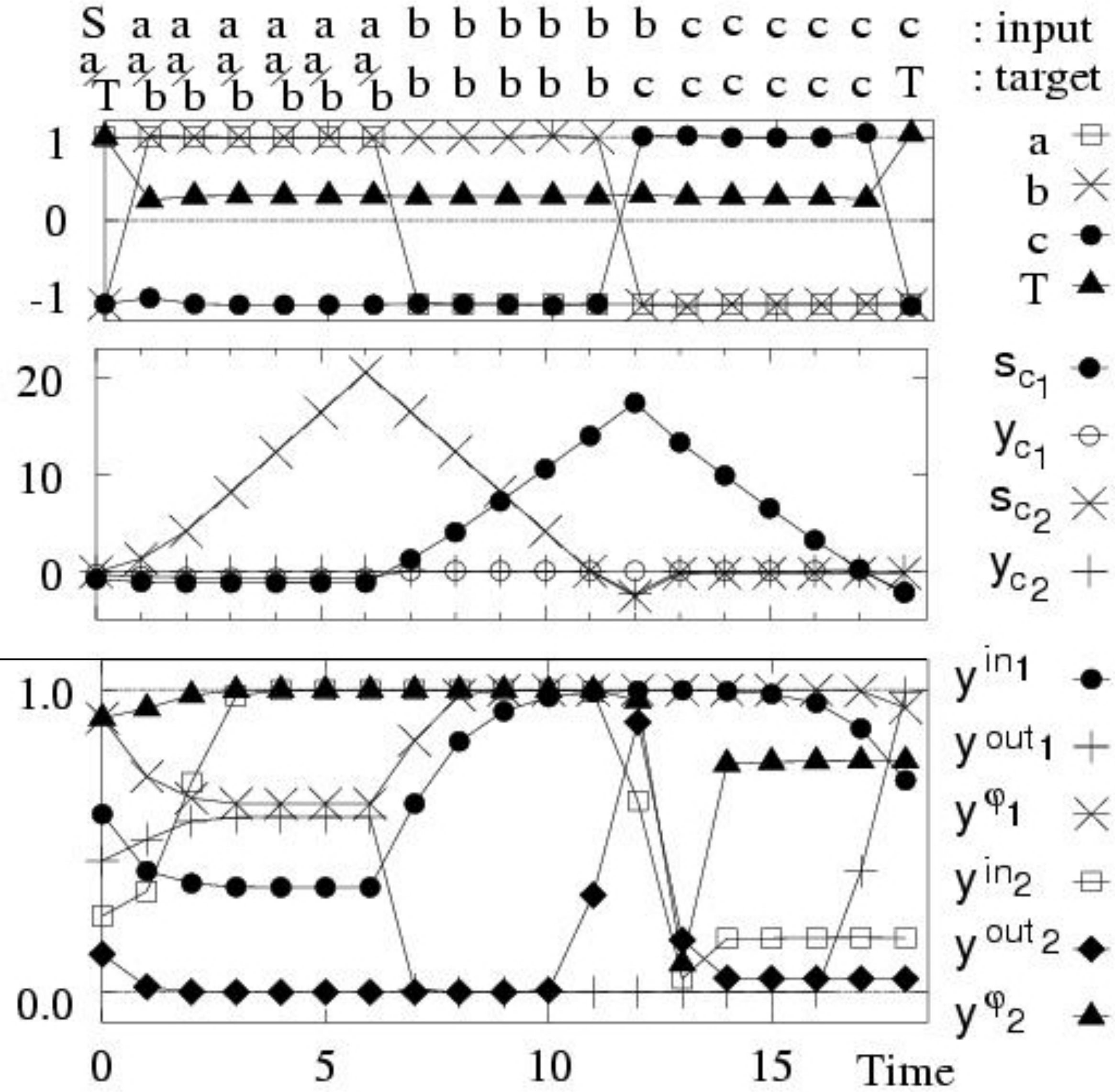
500 500 500

------------------*ILLEGAL:*---------------

aaaaa…..aaabbbbb…..bbbccccc…..ccc

500 499 500

Typical evolution of activations



Storing & adding real values

t_1, t_2 and t_e are randomly chosen.

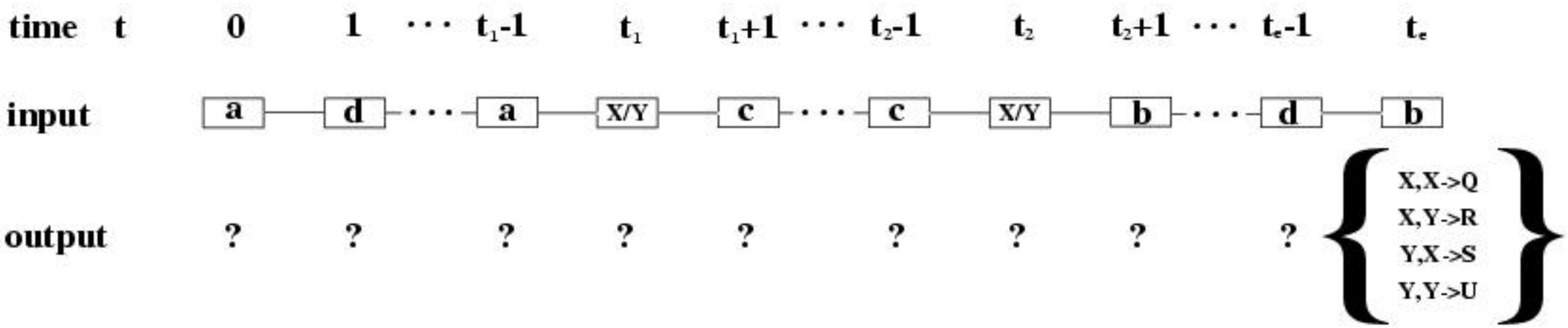
time	t	0	1	· · ·	t_1-1	t_1	t_1+1	· · ·	t_2-1	t_2	t_2+1	· · ·	t_e-1	t_e
						X_1				X_2				
input	I_1	-0.5	0.9	· · ·	0.1	0.6	-0.3	· · ·	-0.7	-0.1	0.5	· · ·	0.1	0.4
input	I_2	-1	0	· · ·	0	1	0	· · ·	0	1	0	· · ·	0	-1
output		?	?		?	?	?		?	?	?		?	$0.5 + \frac{X_1 + X_2}{4}$

target of this example: 0.625

- T=100: 2559/2560; 74,000 epochs
- T=1000: 2559/2560; 850,000 epochs

Noisy temporal order

t_1, t_2 and t_e are randomly chosen. At time t_1 and t_2 an input is randomly chosen from {X,Y}.



- T=100: 2559/2560 correct;
- 32,000 epochs on average

Noisy temporal order II

- Noisy sequences such as aabab...dcaXca...abYdaab...bcdXdb….
- 8 possible targets after 100 steps:
- X,X,X → 1; X,X,Y → 2; X,Y,X → 3; X,Y,Y → 4; Y,X,X → 5; Y,X,Y → 6; Y,Y,X → 7; Y,Y,Y → 8;

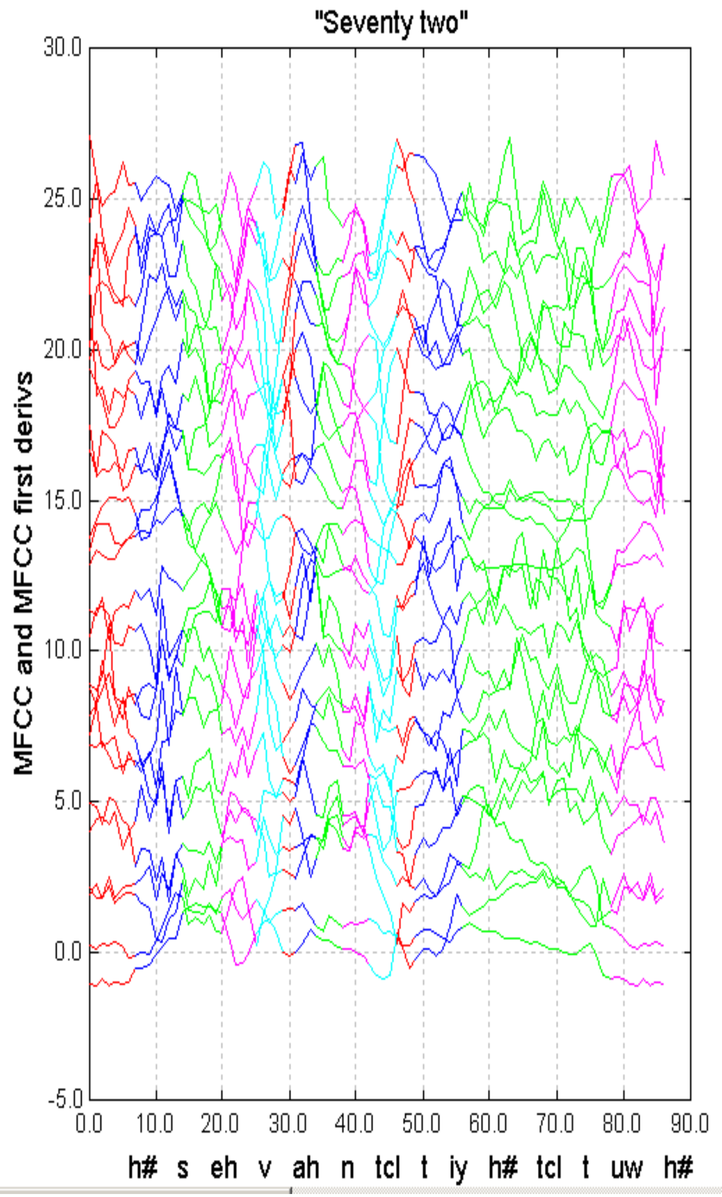
- 2558/2560 correct (error < 0.3)
- 570,000 epochs on average

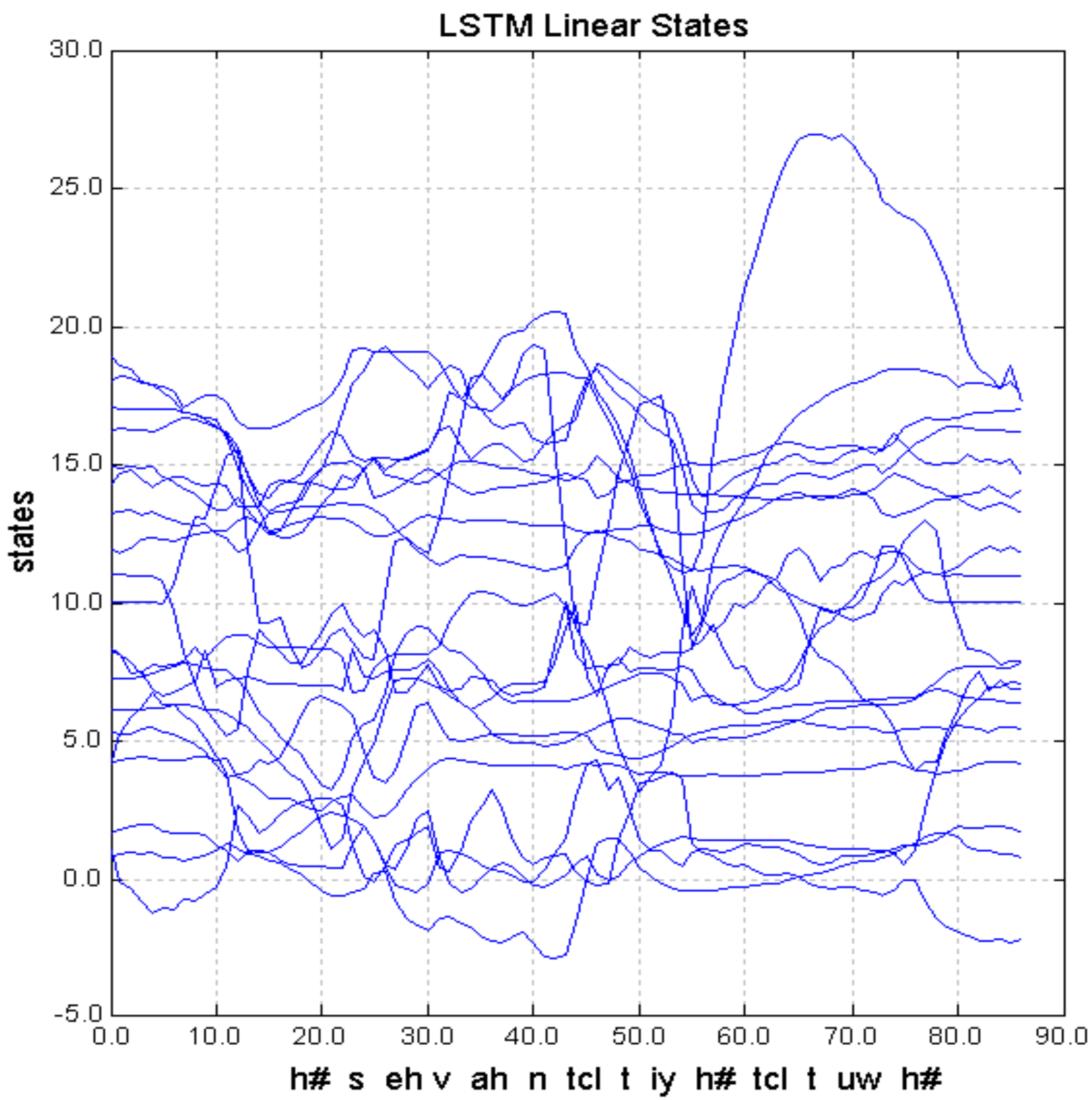
Example: phoneme classification (Graves, 2007)

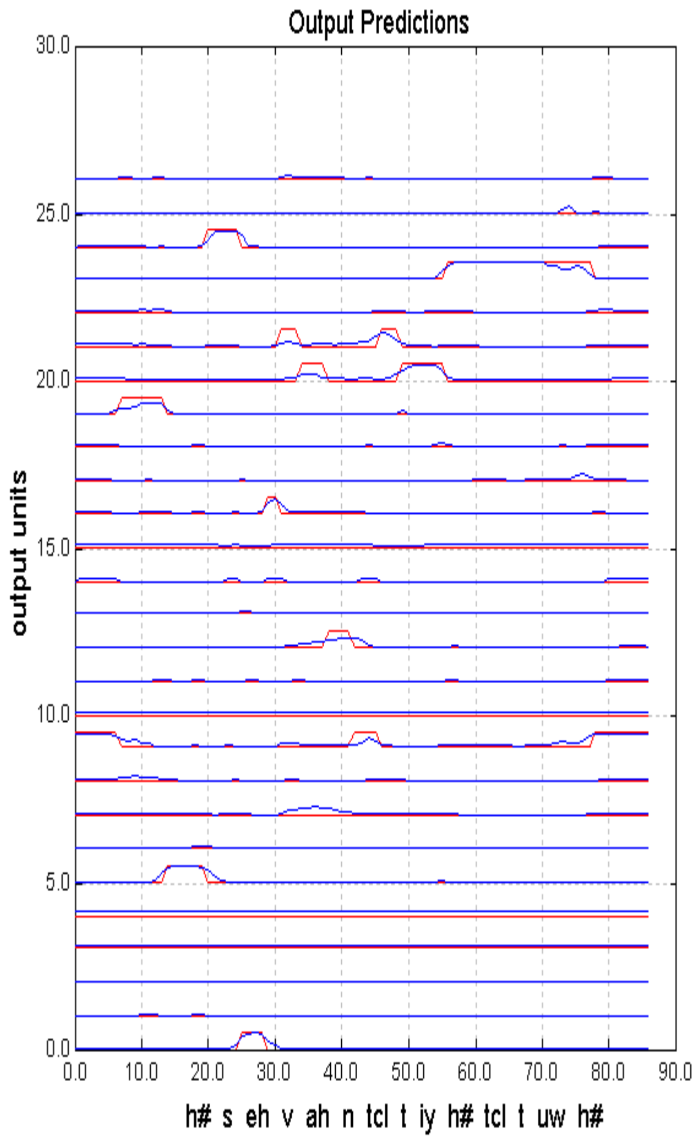
- Input: a stream audio data; 26 spectral coefficients per time-step
- Target: which phoneme is being sounded at each time-step
- Training set: TIMIT corpus of a wide variety of American dialects

Phoneme Identification

- *Numbers 95* database: street numbers / zip codes (Bengio)
- 13 MFCC values + 1st derivative = 26 inputs
- 27 possible phonemes
- ~=4500 sentences
  ~=77000 phonemes
  ~= 666,000 10ms frames







State trajectories suggest a use of history.

Bidirectional RNNs

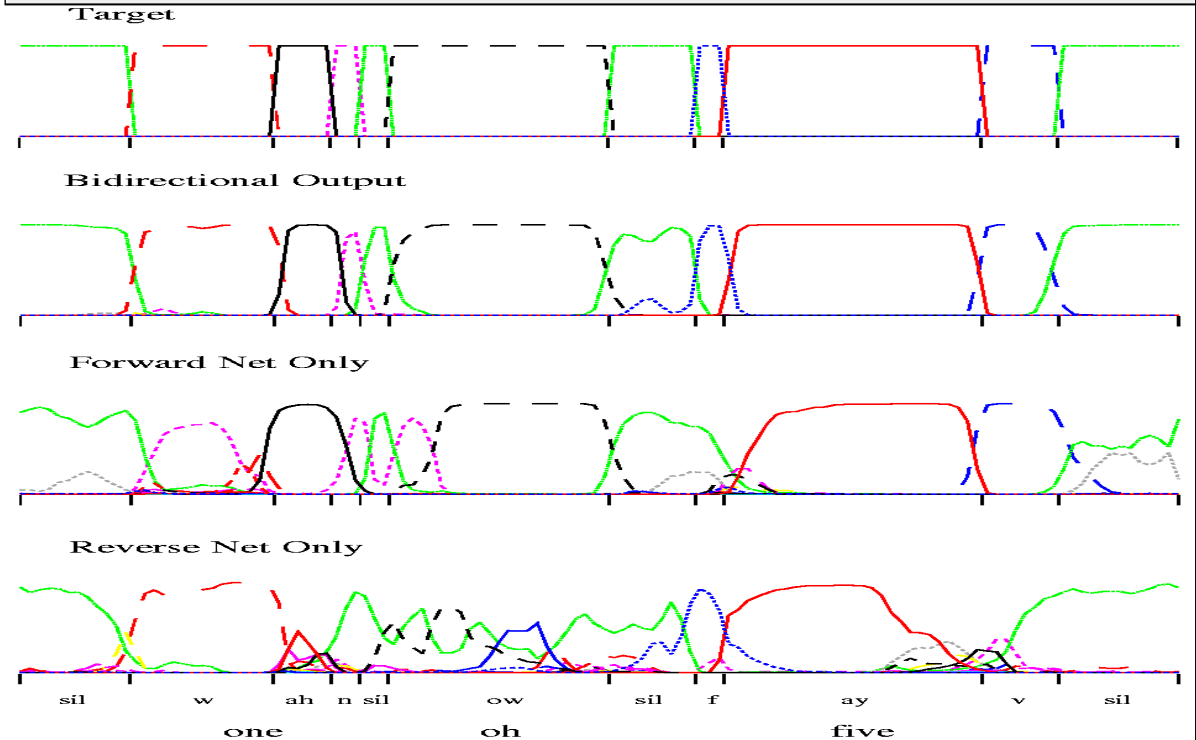
Past and future context often important for sequence learning tasks:

Protein structure prediction

Speech recognition

BRNNs have forward and reverse subnets: future and past on an equal footing

Speech 4: BLSTM classifying phones in "one oh five"

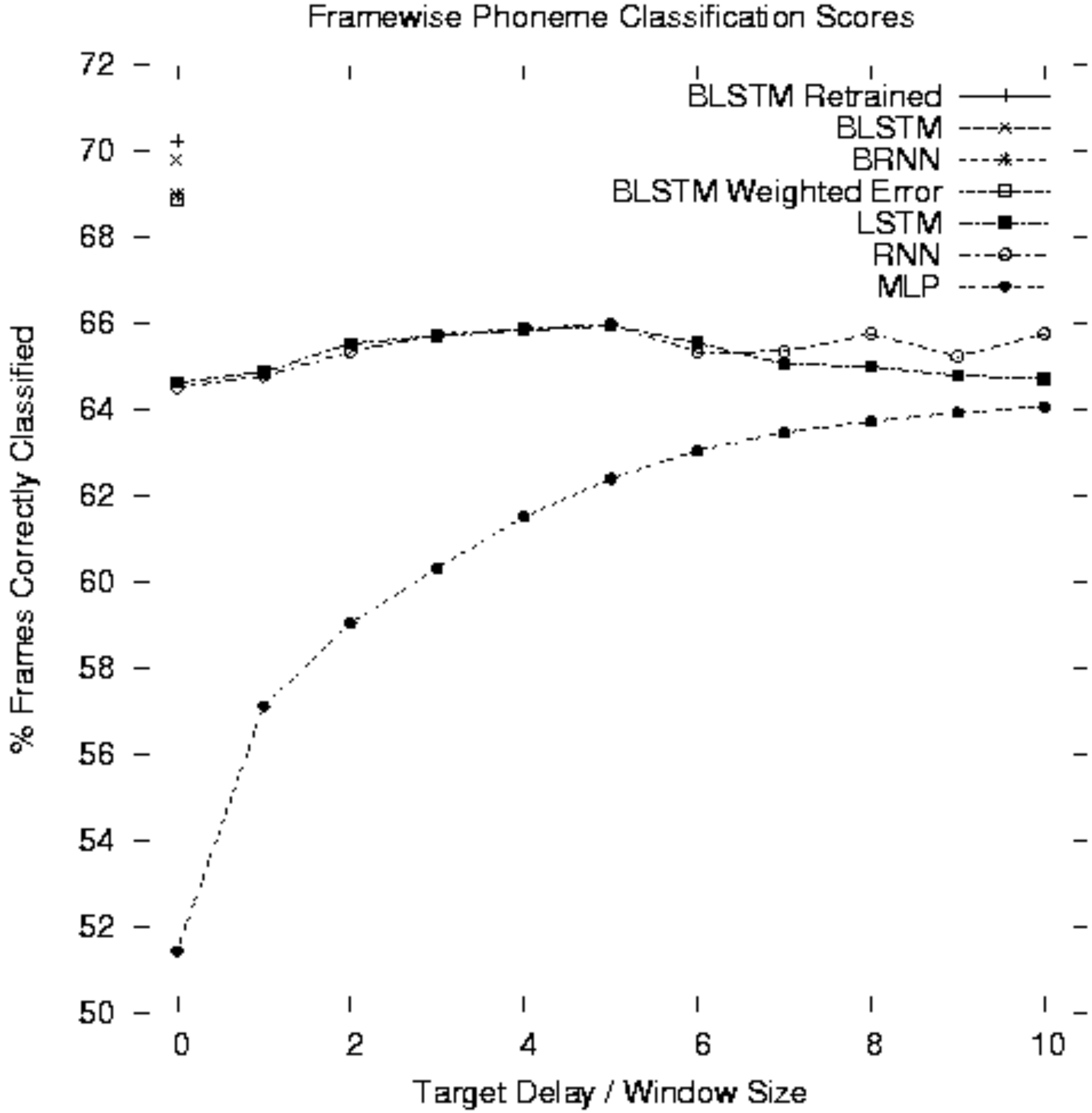


Output similar to targets => good classification

Forward net more accurate But its errors corrected by reverse net:

substitutions ('w') insertions (start of 'ow') deletions ('f')

Reverse net finds starts of phones, forward net finds ends ('ay')



Graves: Framewise phoneme classification: bidirectional LSTM vs others

- Bi-nets imrove on uni
- LSTM outperforms standard RNNs
- LSTM faster to train
- Retraining raises score