Deep Learning

741.

Recurrent Neural Networks

(RNN)

Sequence representations

- Language is represented as sequences
  - Discourse is made of sentences
  - Sentences are made of words
  - Words are made of characters
- Language modeling
  - Too many parameters
  - HMM: ignore earlier history
- Long-distance dependencies
- Example 1
  - The girl ate the apple because she was hungry.
  - The boy ate the apple because he was hungry.
- Example 2
  - That night was fairly lonely.
  - That knight was fairly lonely.

http://www.singularis.ltd.uk/bifroest/misc/homophones-list.htm

Winograd Schema

- Classic example
  - The **city councilmen** refused **the demonstrators** a permit because **they [feared/advocated]** violence.

    Who refused the permit?
  - **The city councilmen** refused the demonstrators a permit because **they feared** violence.
  - The city councilmen refused **the demonstrators** a permit because **they advocated** violence.
- One more example
  - The trophy would not fit in the brown suitcase because it was too big (*small*).

    What was too big (small)?

http://commonsensereasoning.org/winograd.html

Recurrent representations

- How can we represent entire sentences of variable length?
- Can we do this using a fixed size vector?
- Answer
  - Recurrent Neural Networks (RNN) [Elman 1990]

Recurrent Neural Networks

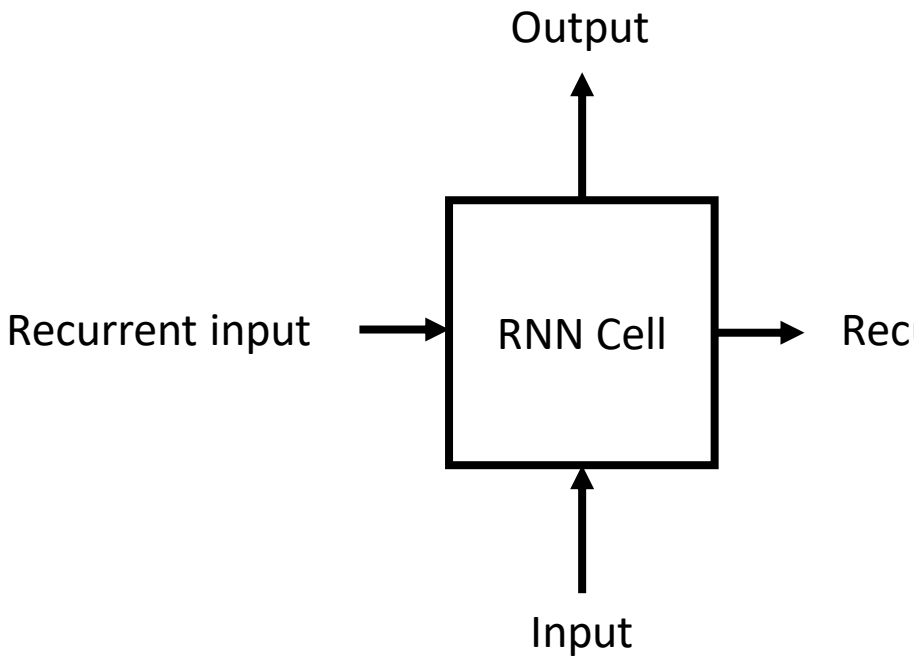
COGNITIVE SCIENCE **14**, 179–211 (1990)

Finding Structure in Time

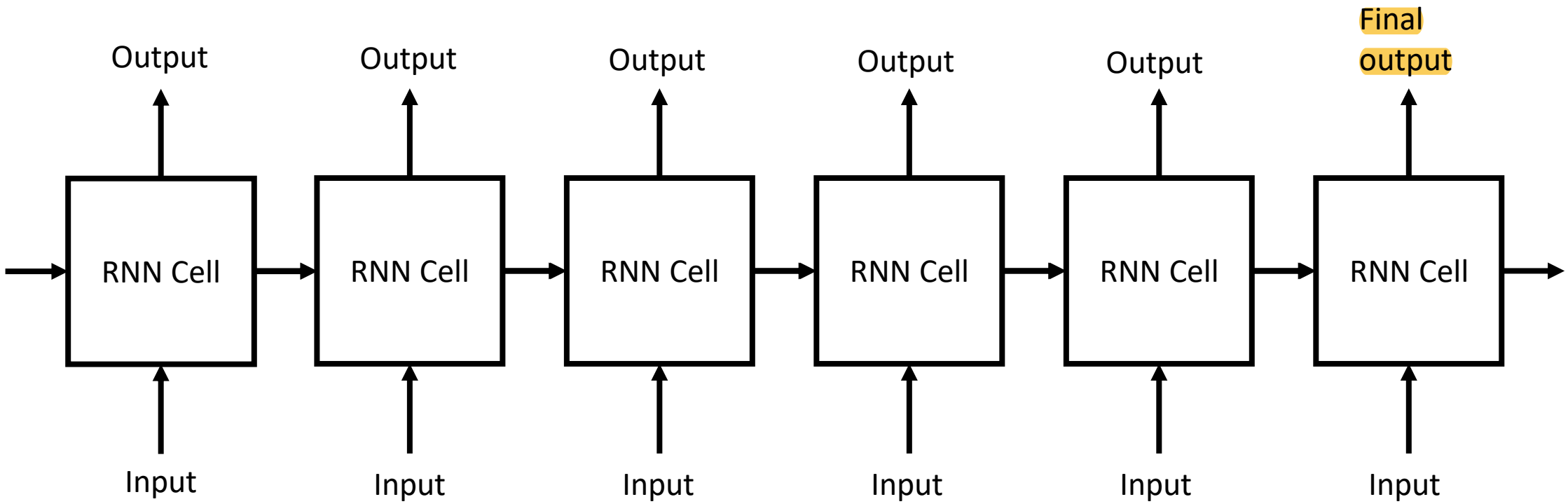
JEFFREY L. ELMAN
University of California, San Diego

Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal along these lines first described by Jordan (1986) which involves the use of recurrent links in order to provide networks with a dynamic memory. In this approach, hidden unit patterns are fed back to themselves; the internal representations which develop thus reflect task demands in the context of prior internal states. A set of simulations is reported which range from relatively simple problems (temporal version of XOR) to discovering syntactic/semantic features for words. The networks are able to learn interesting internal representations which incorporate task demands with memory demands; indeed, in this approach the notion of memory is inextricably bound up with task processing. These representations reveal a rich structure, which allows them to be highly context-dependent, while also expressing generalizations across classes of items. These representations suggest a method for representing lexical categories and the type/token distinction.

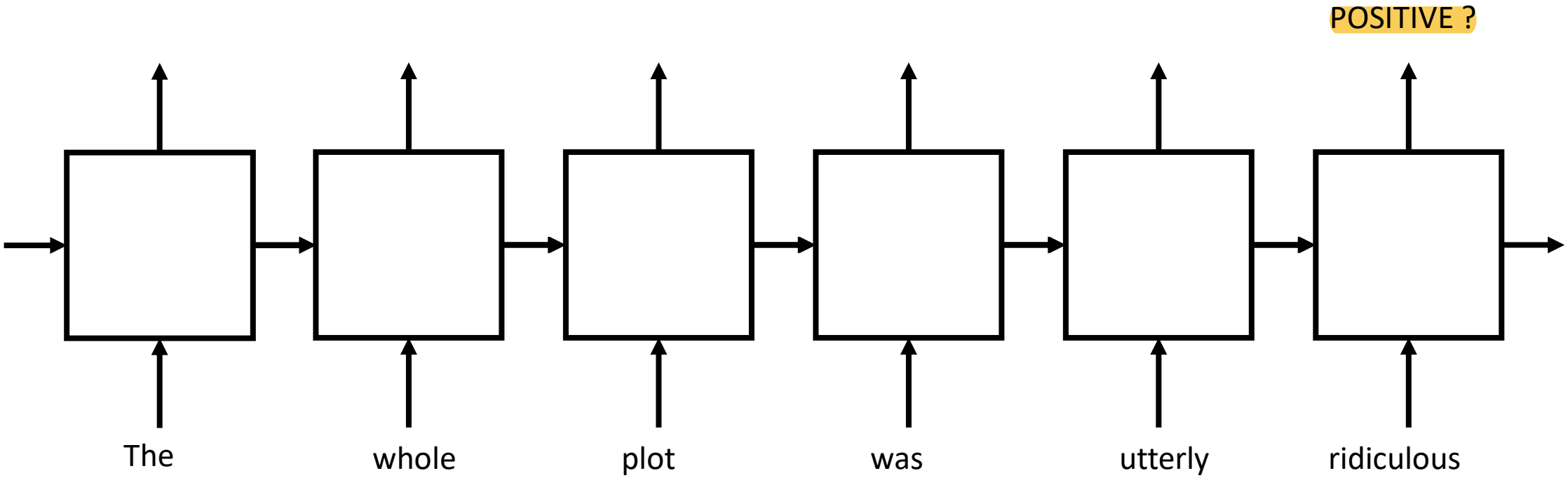
RNN Cell

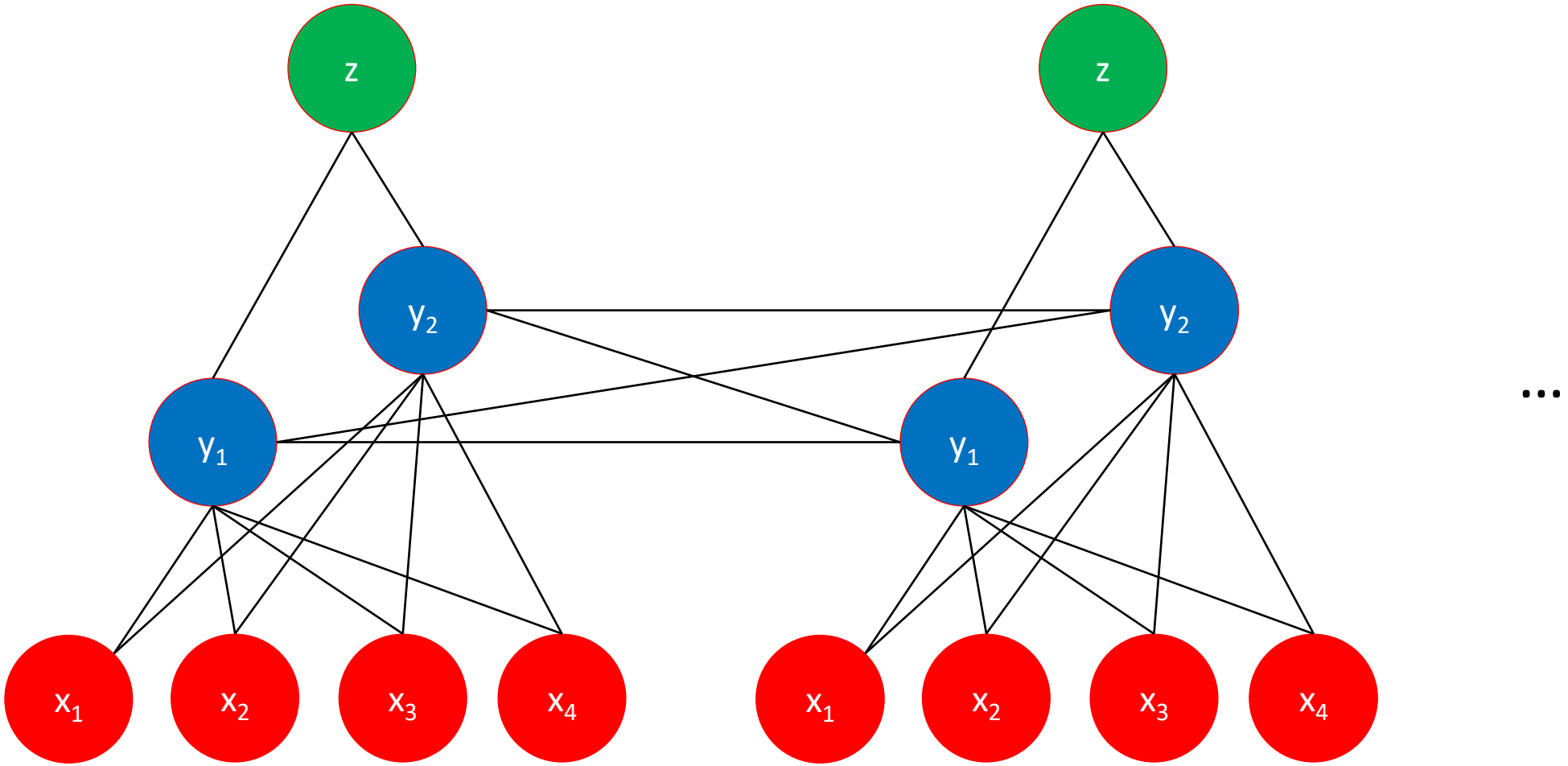


RNN Sequence



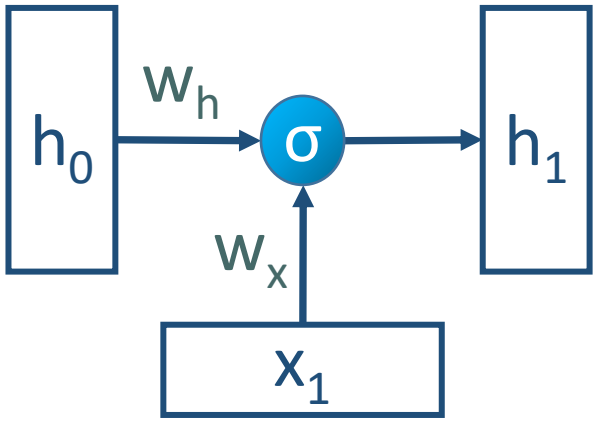
Example



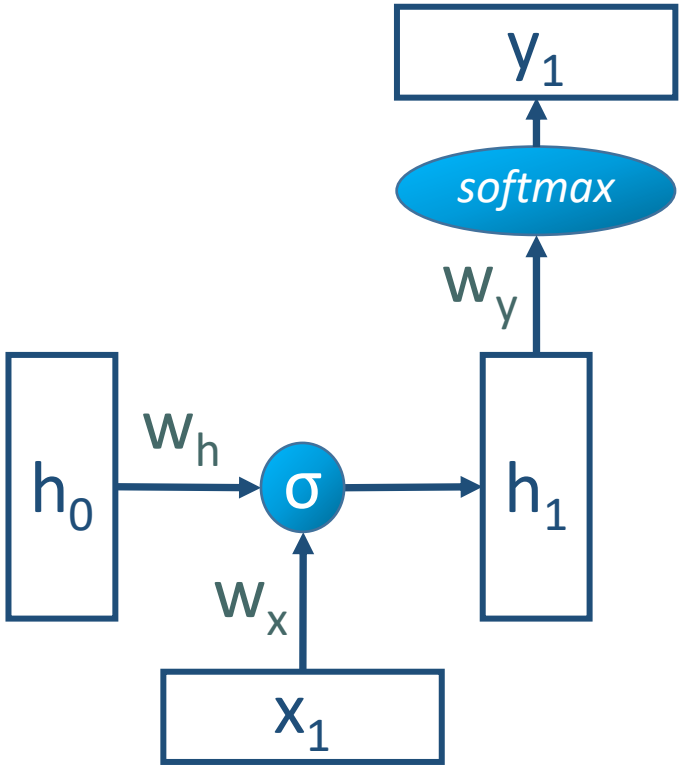

z
z
y2
y2
y1
y1
x1
x2
x3
x4
x1
x2
x3
x4
...
the
cat

Recurrent Neural Networks

$$h_t = \sigma(W_h h_{t-1} + W_x x_t)$$

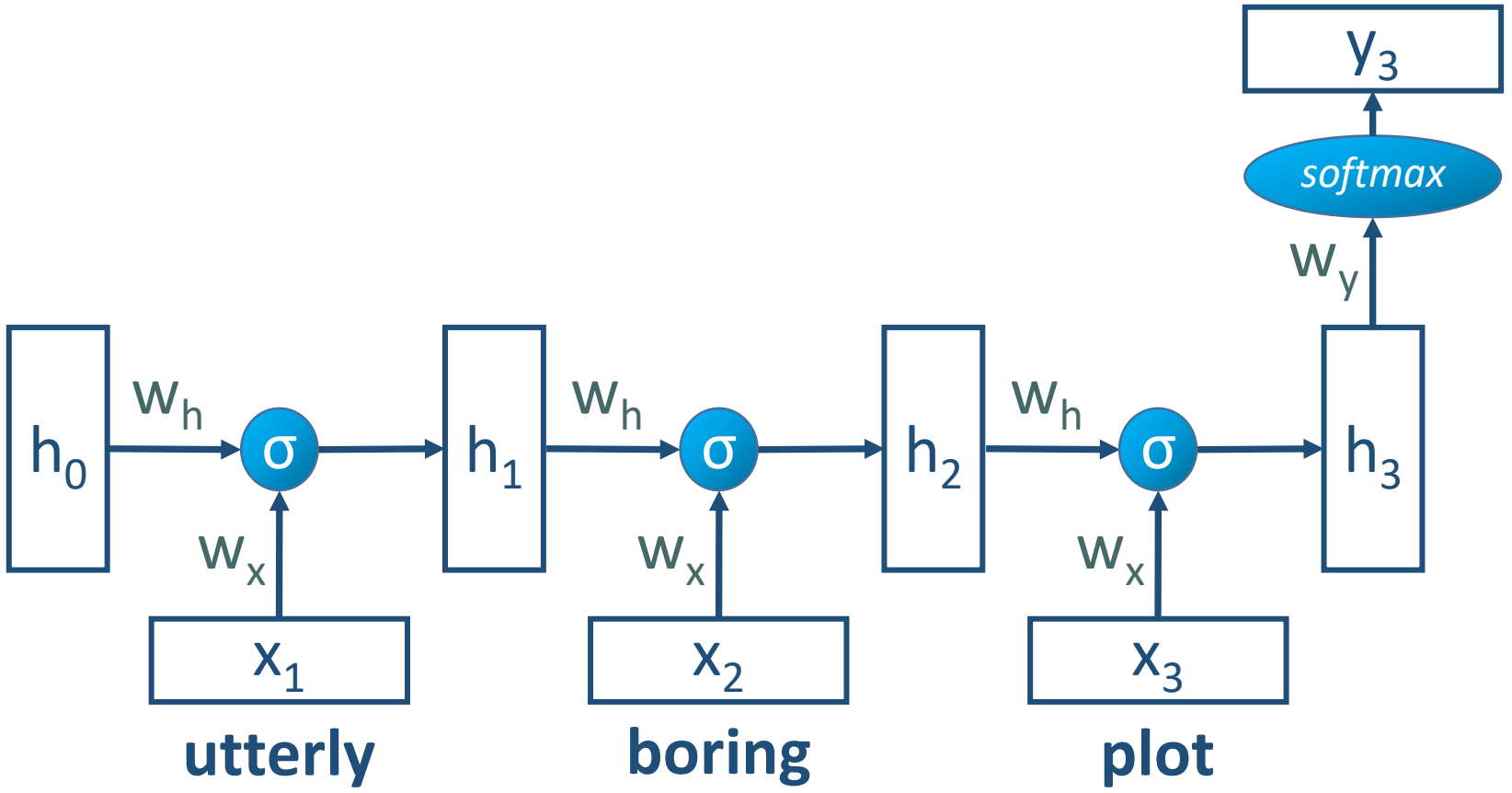


RNN



$$h_t = \sigma(W_h h_{t-1} + W_x x_t)$$
$$y_t = softmax(W_y h_t)$$

RNN


y_3
softmax
w_y
h_0
w_h
σ
h_1
w_h
σ
h_2
w_h
σ
h_3
w_x
w_x
w_x
x_1
x_2
x_3
utterly
boring
plot

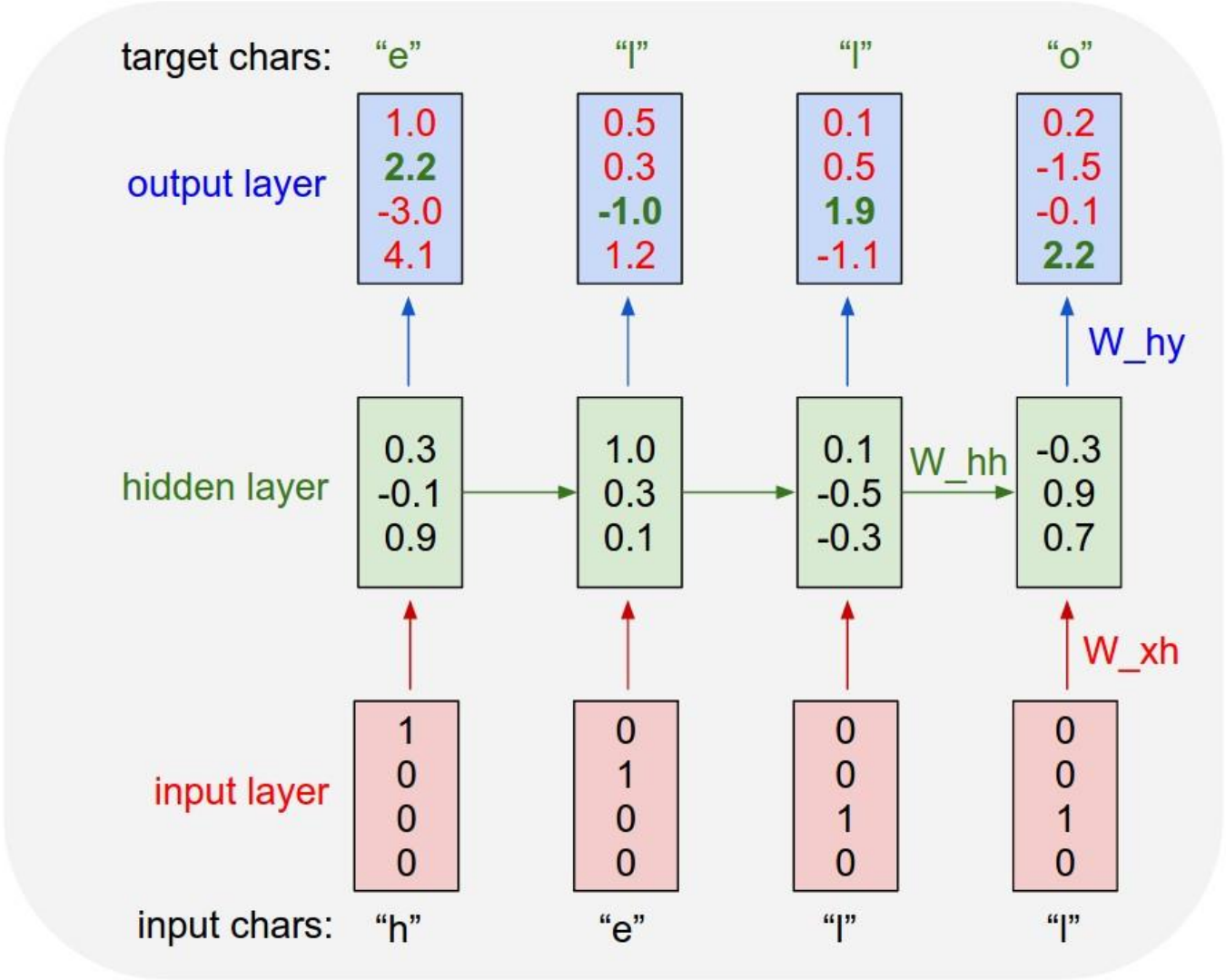
Forward Inference

function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
  $y_i \leftarrow f(V\ h_i)$
return y

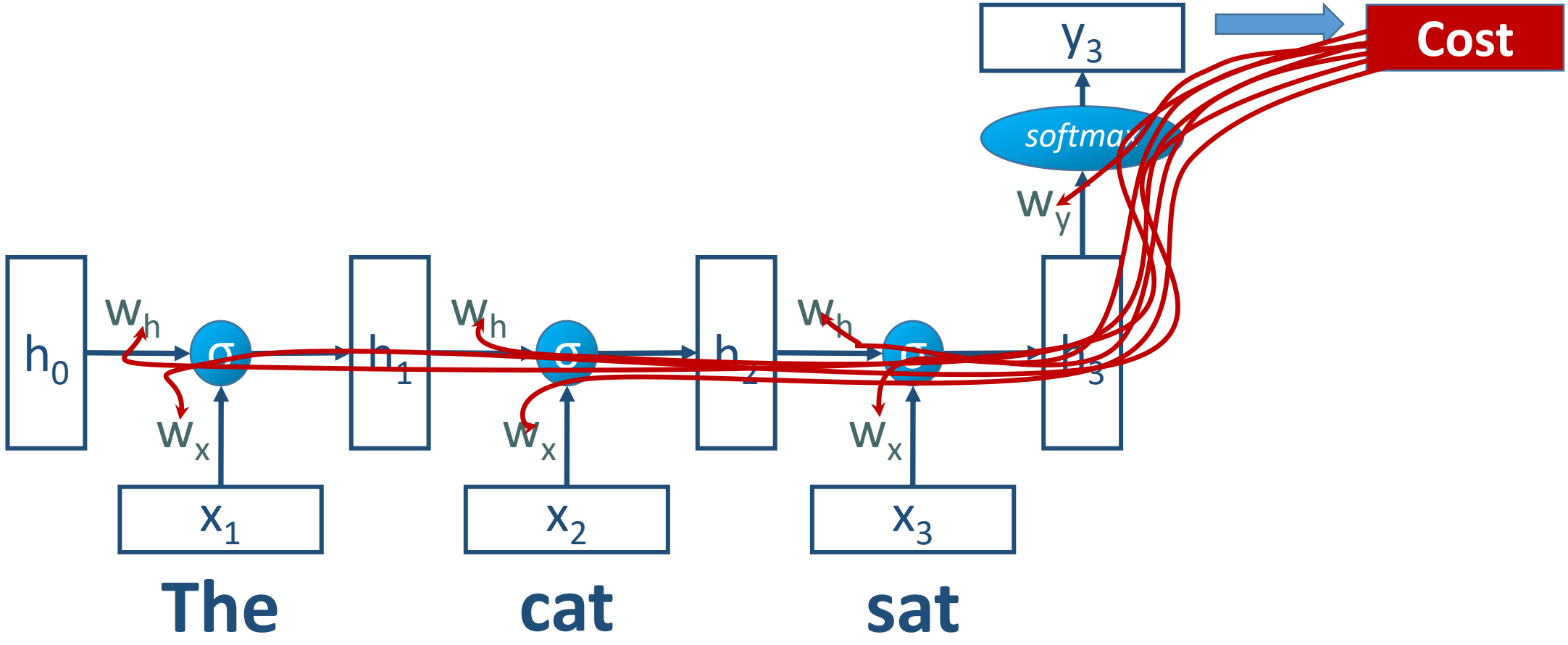
Figure 9.4 Forward inference in a simple recurrent network. The matrices U, V and W are shared across time, while new values for h and y are calculated with each time step.

Character-based RNN



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Updating Parameters of an RNN



Backpropagation through time

- The parameters are shared
- The derivatives are accumulated

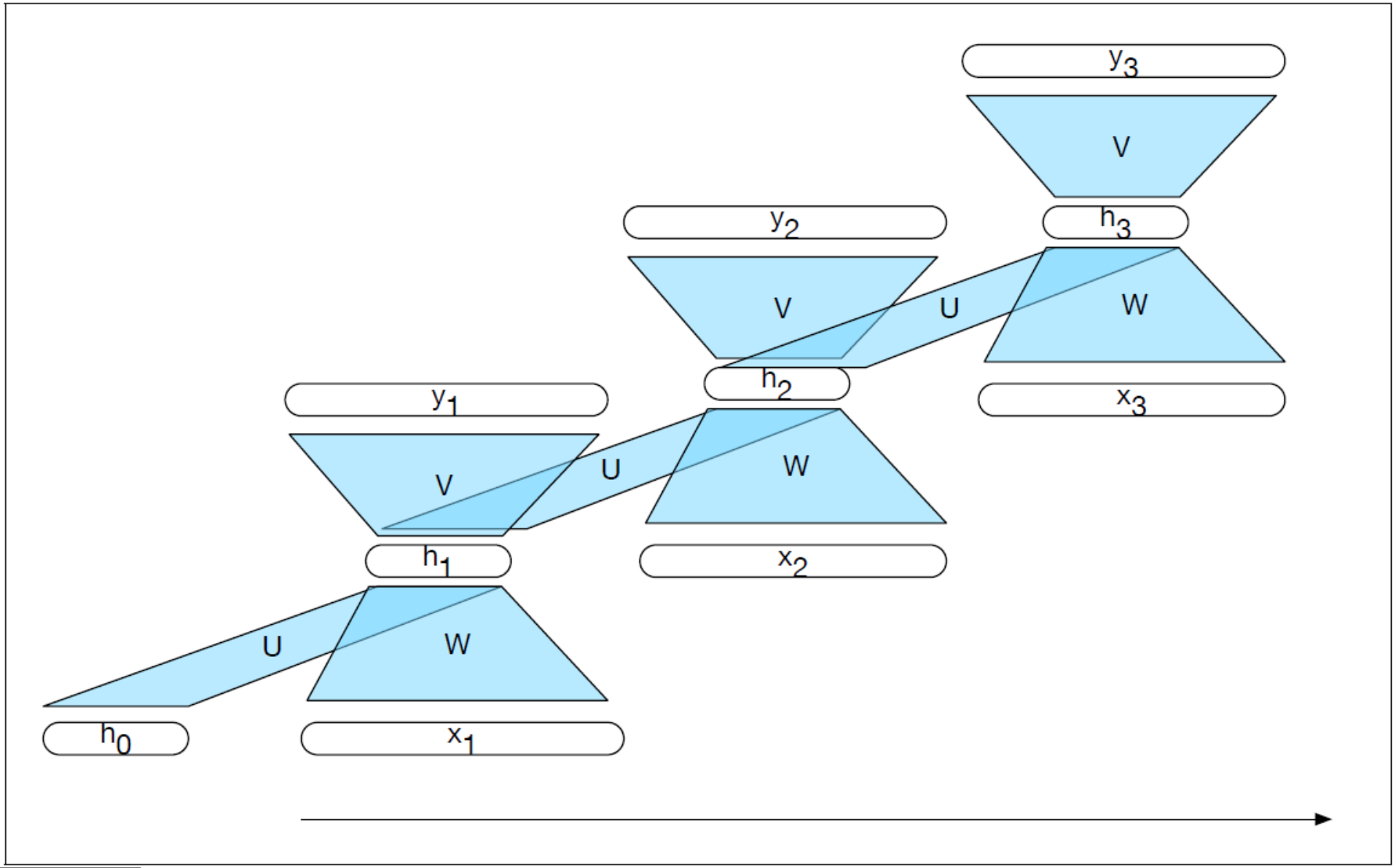


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

Backpropagation through time

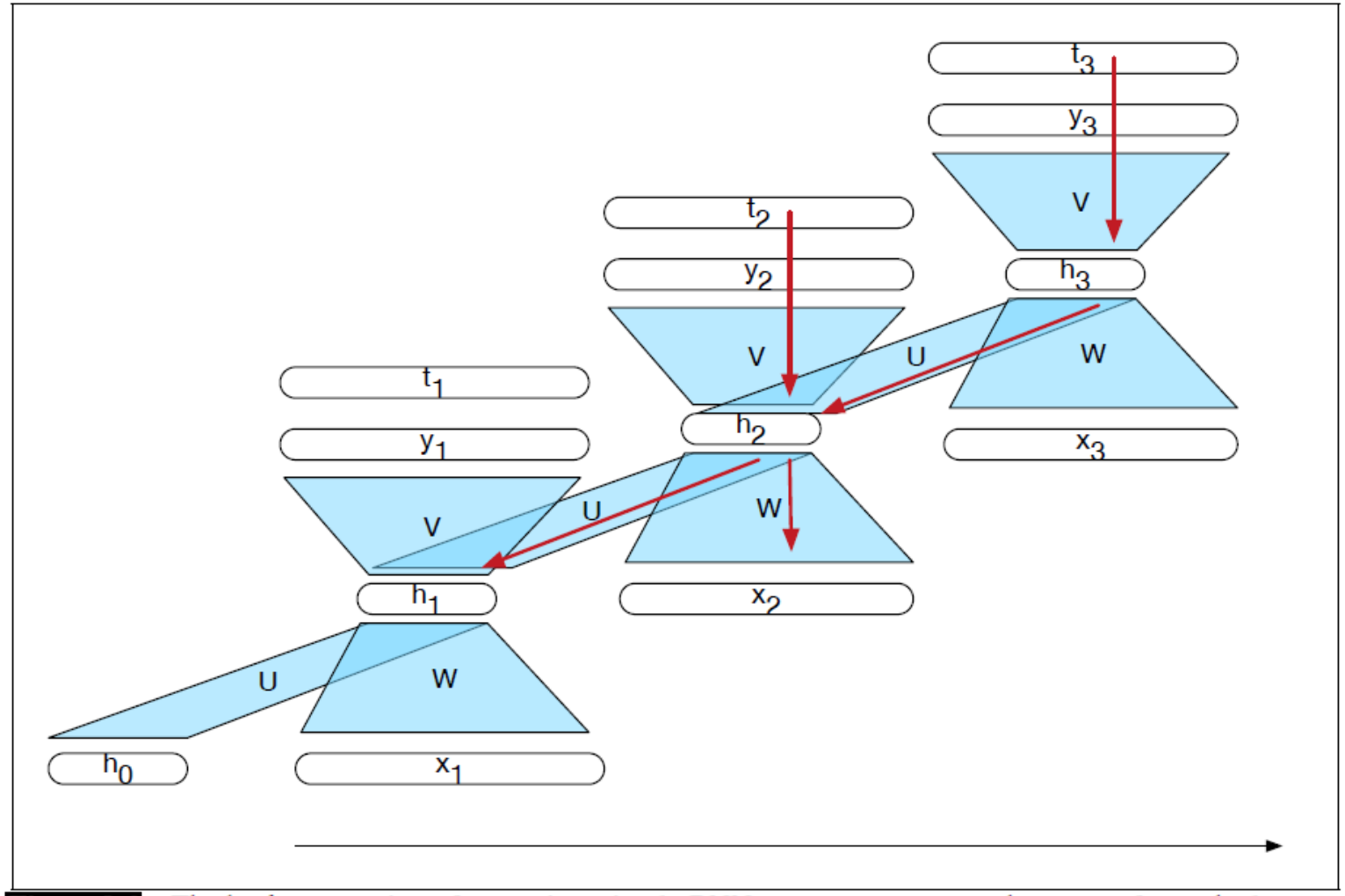


Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

Generation with a neural language model

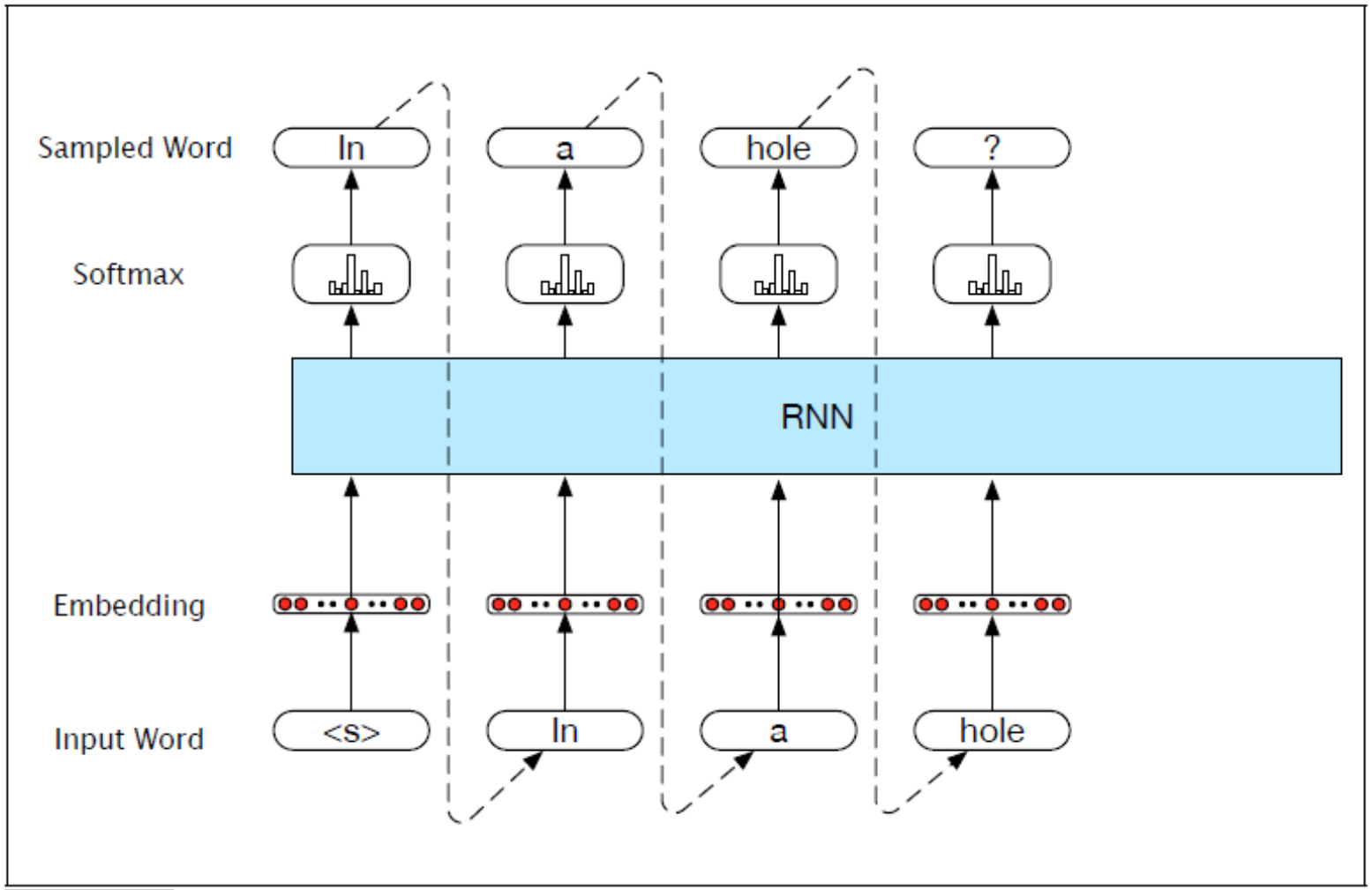


Figure 9.7 Autoregressive generation with an RNN-based neural language model.

Part of Speech Tagging

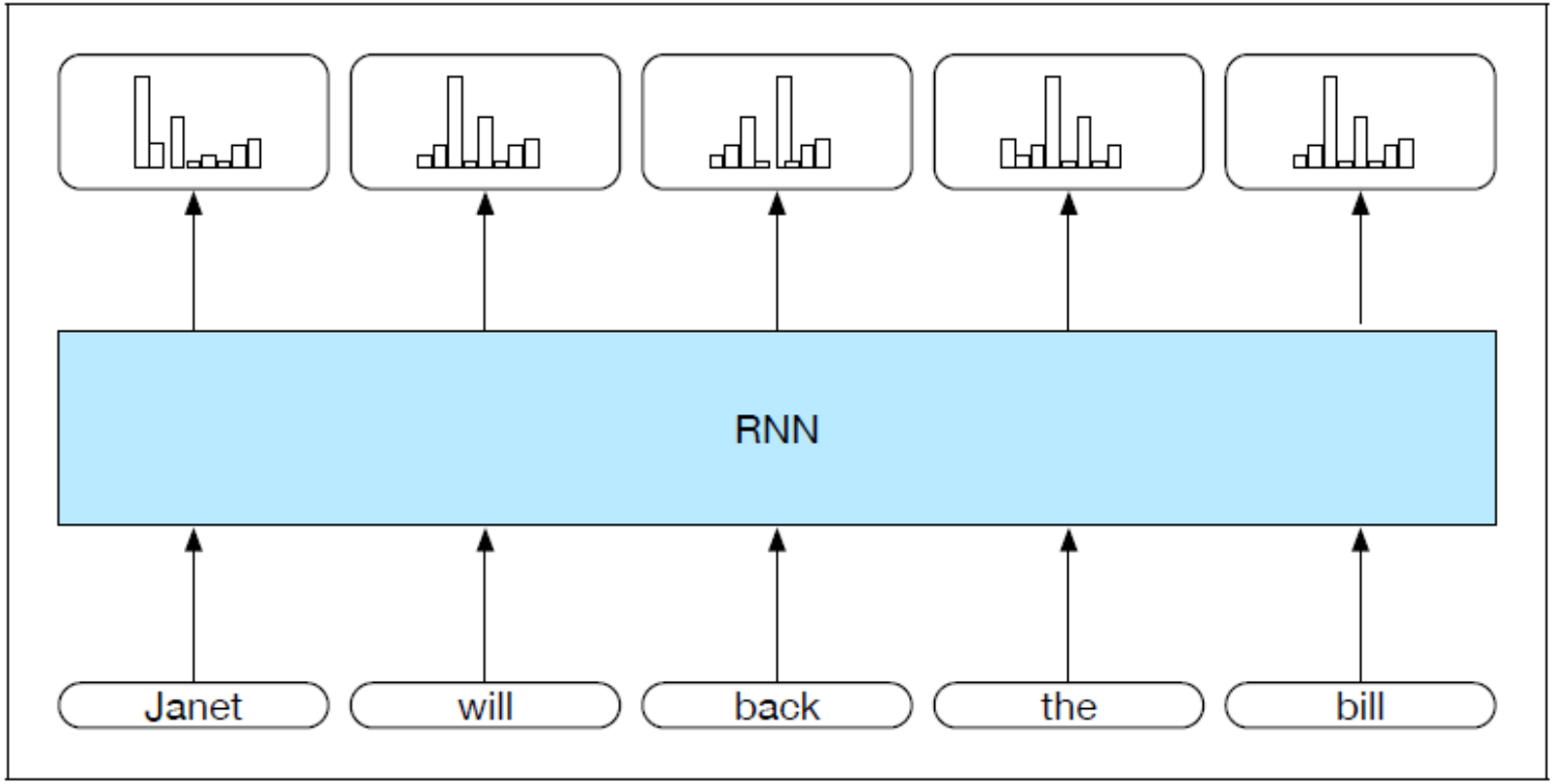


Figure 9.8 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

Sequence Classification

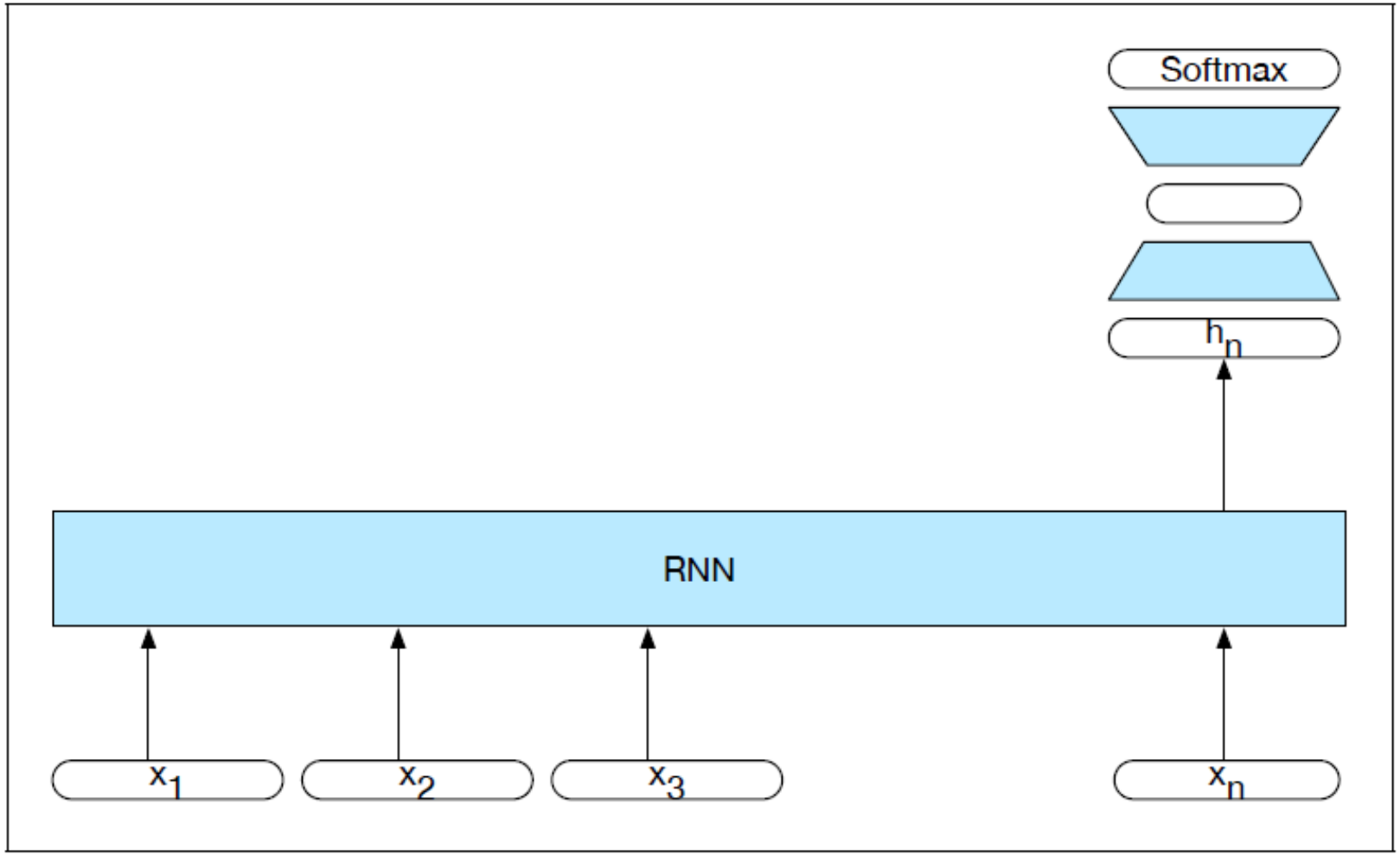


Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Uses of RNN

- RNNs are used to keep “memory”, just like finite-state automata
- They can be used as generators, acceptors, transducers (e.g., for machine translation), etc.
- Key application – language modeling: predicting the next word in the sequence

Other Applications

- Text generation
- Semantic role labeling:
  - http://www.aclweb.org/anthology/P15-1109
- Dependency parsing:
  - http://www.aclweb.org/anthology/K/K15/K15-1015.pdf

Stacked RNN

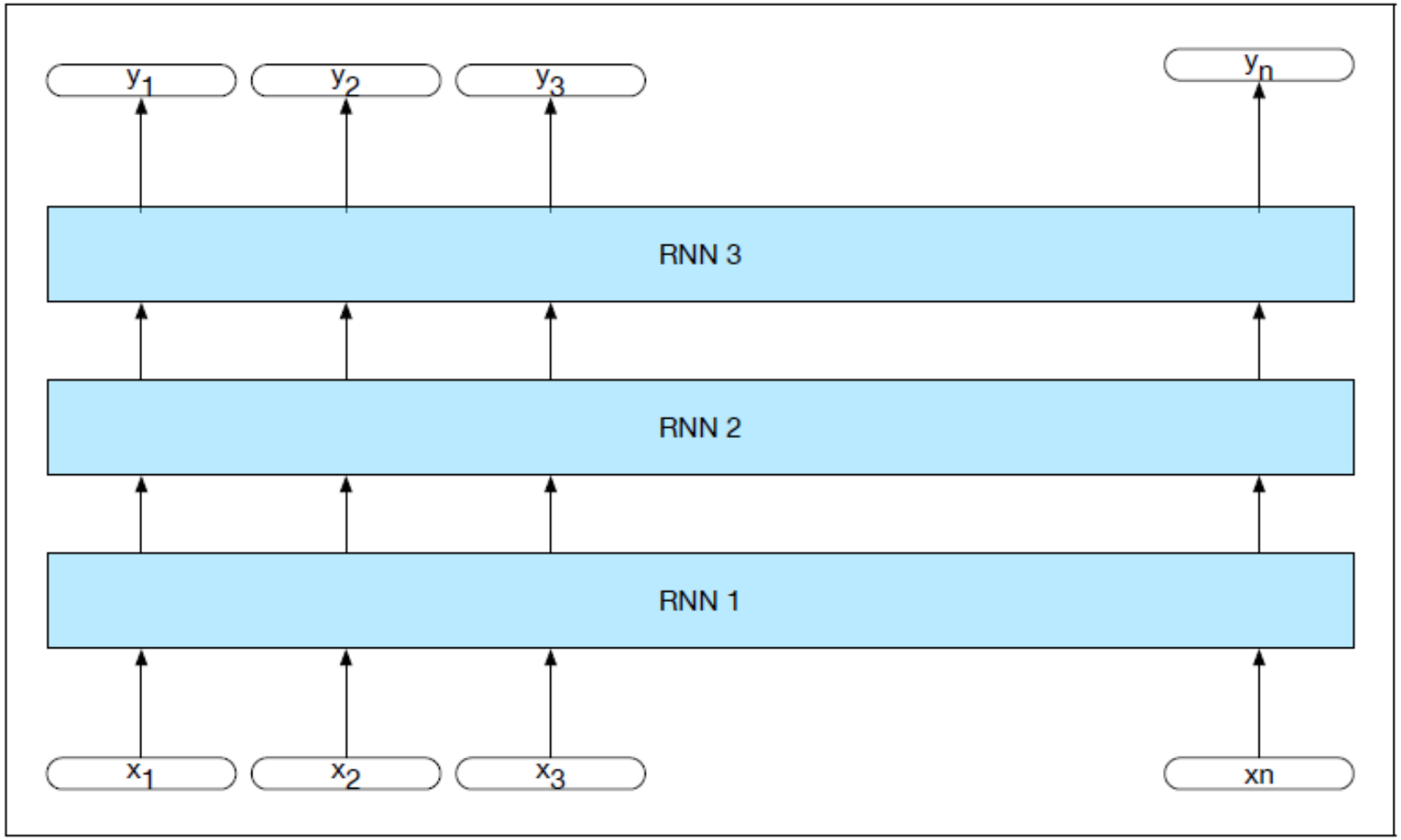


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Bidirectional RNN

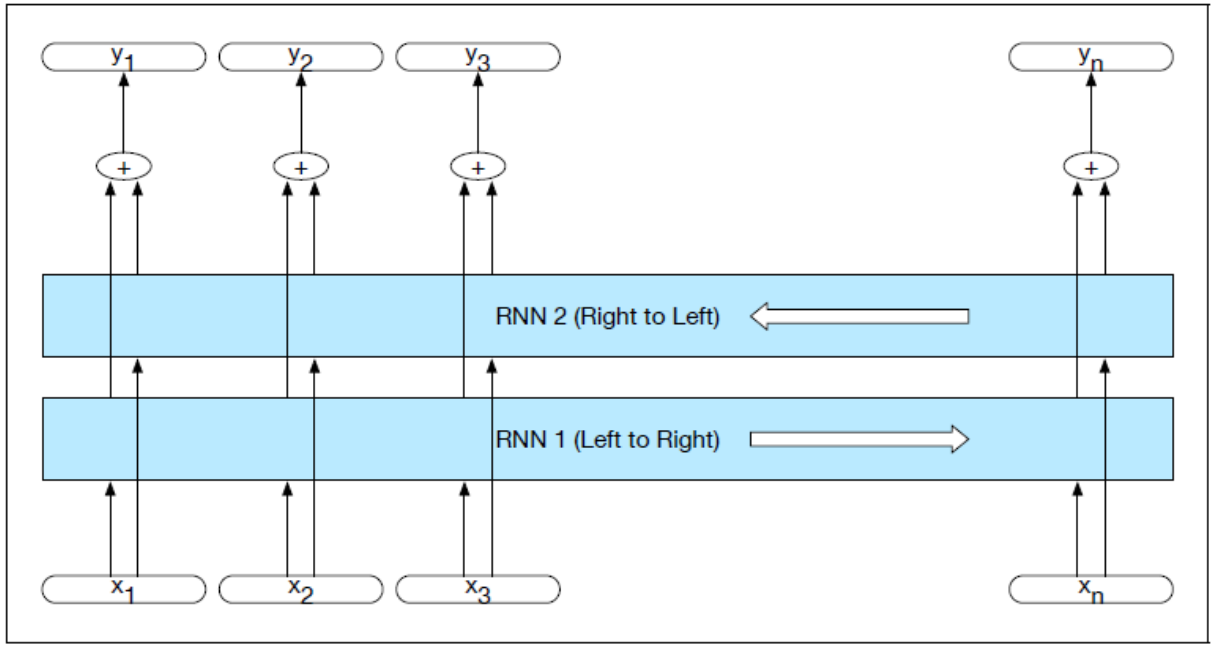


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions with the output of each model at each time point concatenated to represent the state of affairs at that point in time. The box wrapped around the forward and backward network emphasizes the modular nature of this architecture.

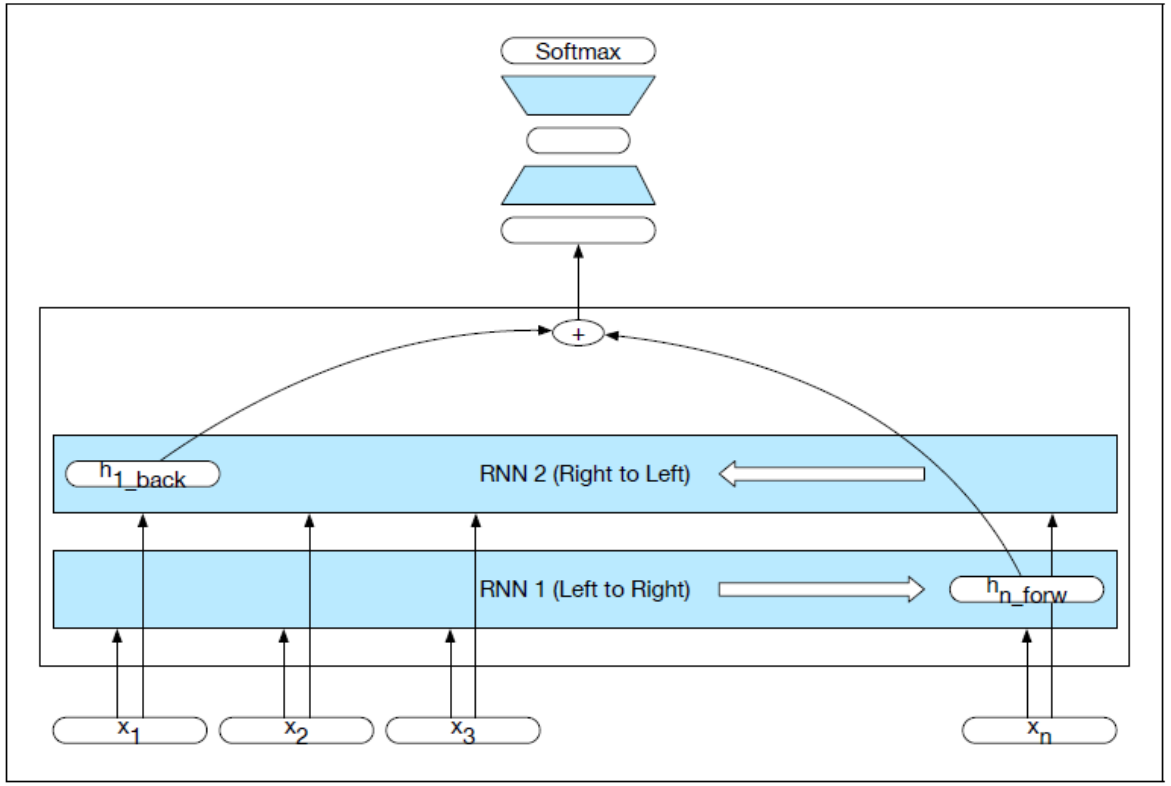


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

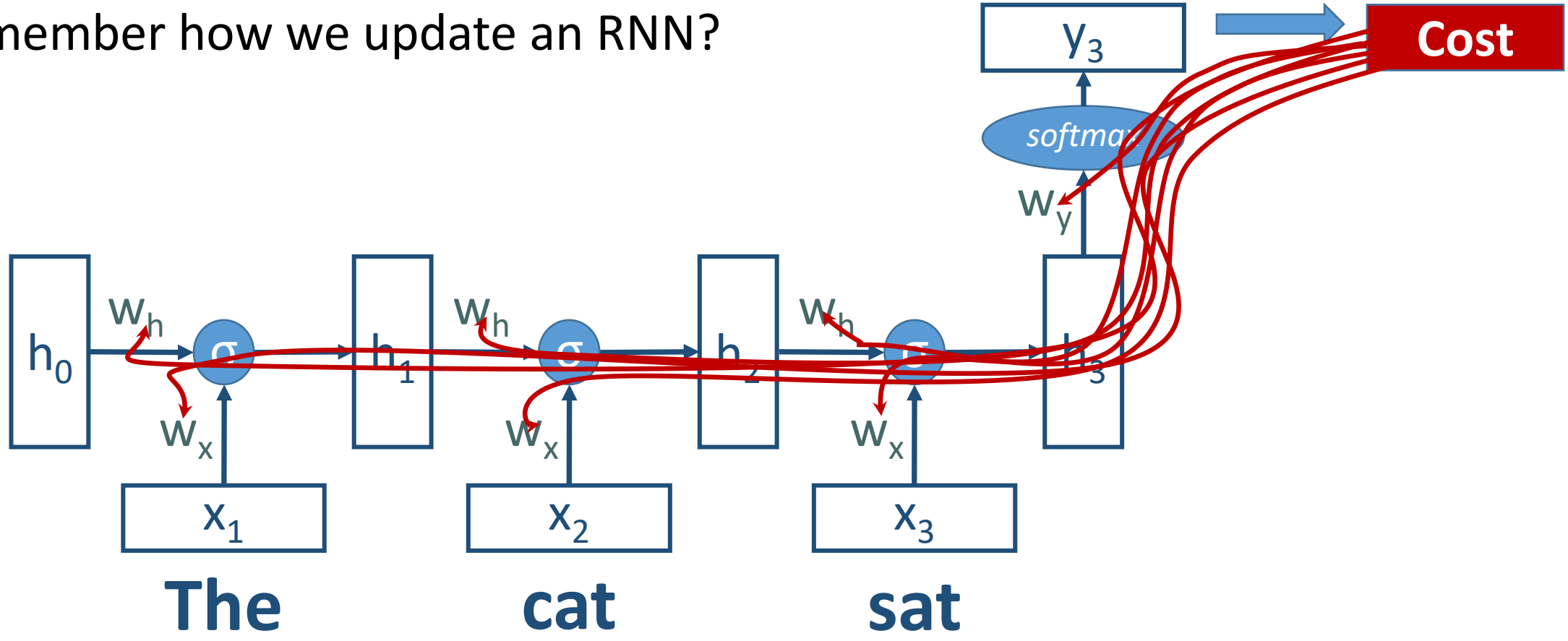
Deep Learning

742.

Long Short-Term Memory Networks LSTM

LSTM Motivation

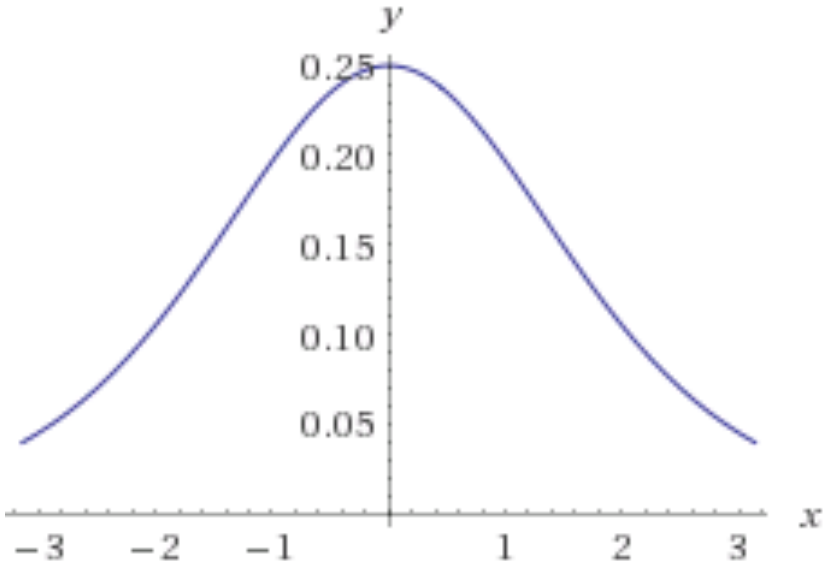
Remember how we update an RNN?



[slides from Catherine Finegan-Dollak]

The Vanishing Gradient Problem

- Deep neural networks use backpropagation.
- Back propagation uses the chain rule.
- The chain rule multiplies derivatives.
- Often these derivatives between 0 and 1.
- As the chain gets longer, products get smaller
- until they disappear.



Wolfram|Alpha

Derivative of sigmoid function

Or do they explode?

- With gradients larger than 1,
- you encounter the opposite problem
- with products becoming larger and larger
- as the chain becomes longer and longer,
- causing overlarge updates to parameters.
- This is the exploding gradient problem.

Vanishing/Exploding Gradients Are Bad

- If we cannot backpropagate very far through the network, the network cannot learn long-term dependencies.

  - My dog [chase/chases] squirrels. 

  vs.

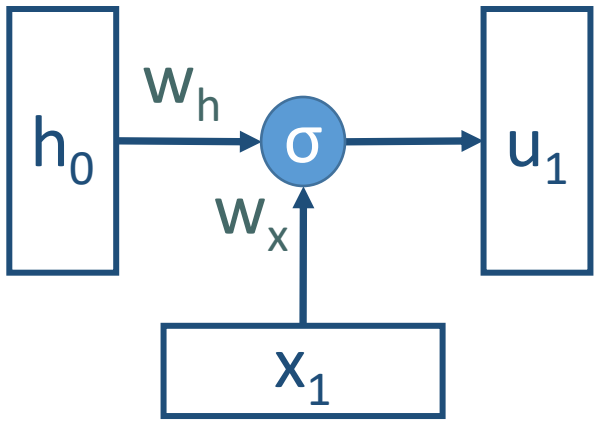
  - My dog, whom I adopted in 2009, [chase/chases] squirrels. 

LSTM Solution

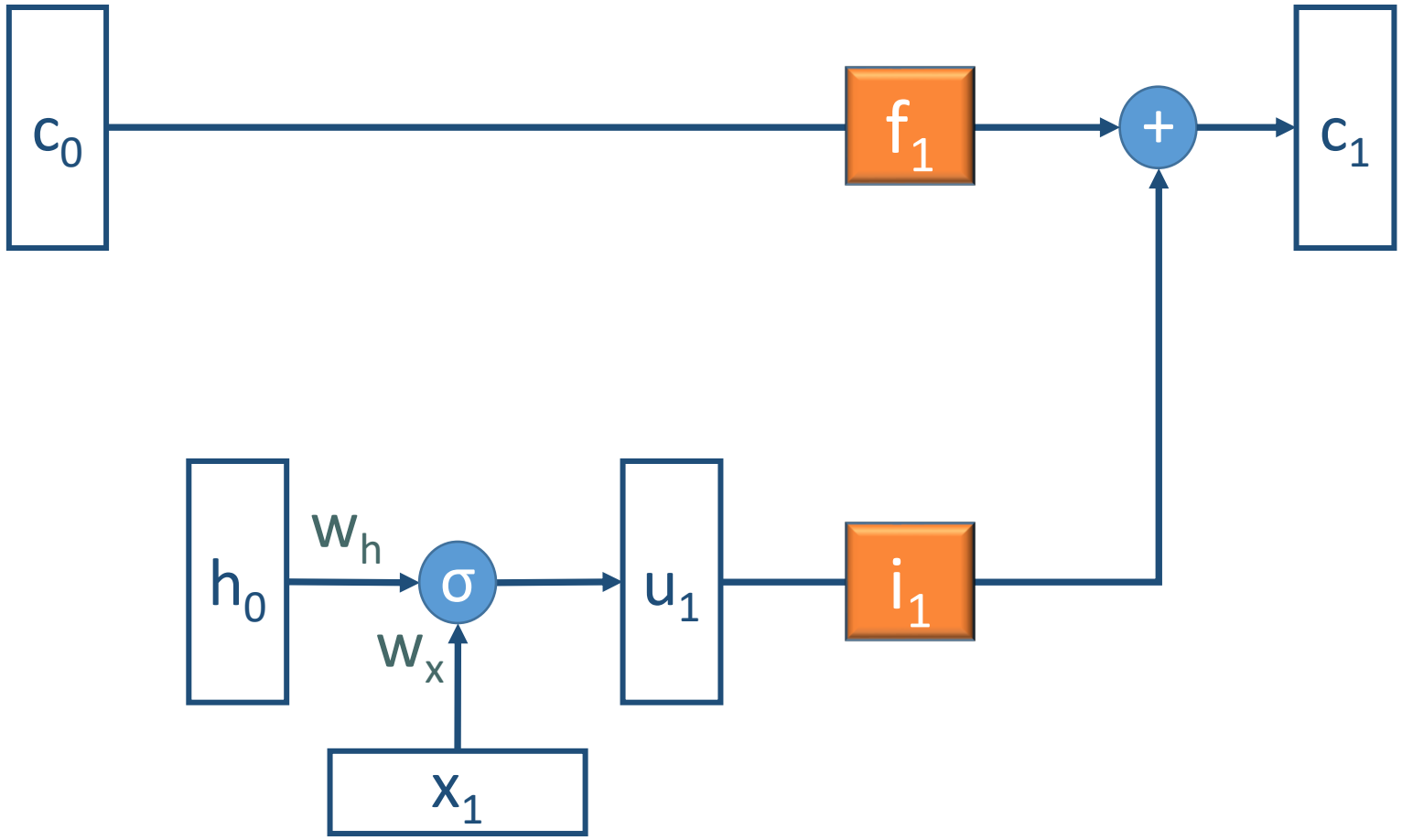
- Use memory cell to store information at each time step.
- Use “gates” to control the flow of information through the network.
  - Input gate: protect the current step from irrelevant inputs
  - Output gate: prevent the current step from passing irrelevant outputs to later steps
  - Forget gate: limit information passed from one cell to the next

Transforming RNN to LSTM

$$u_t = \sigma(W_h h_{t-1} + W_x x_t)$$



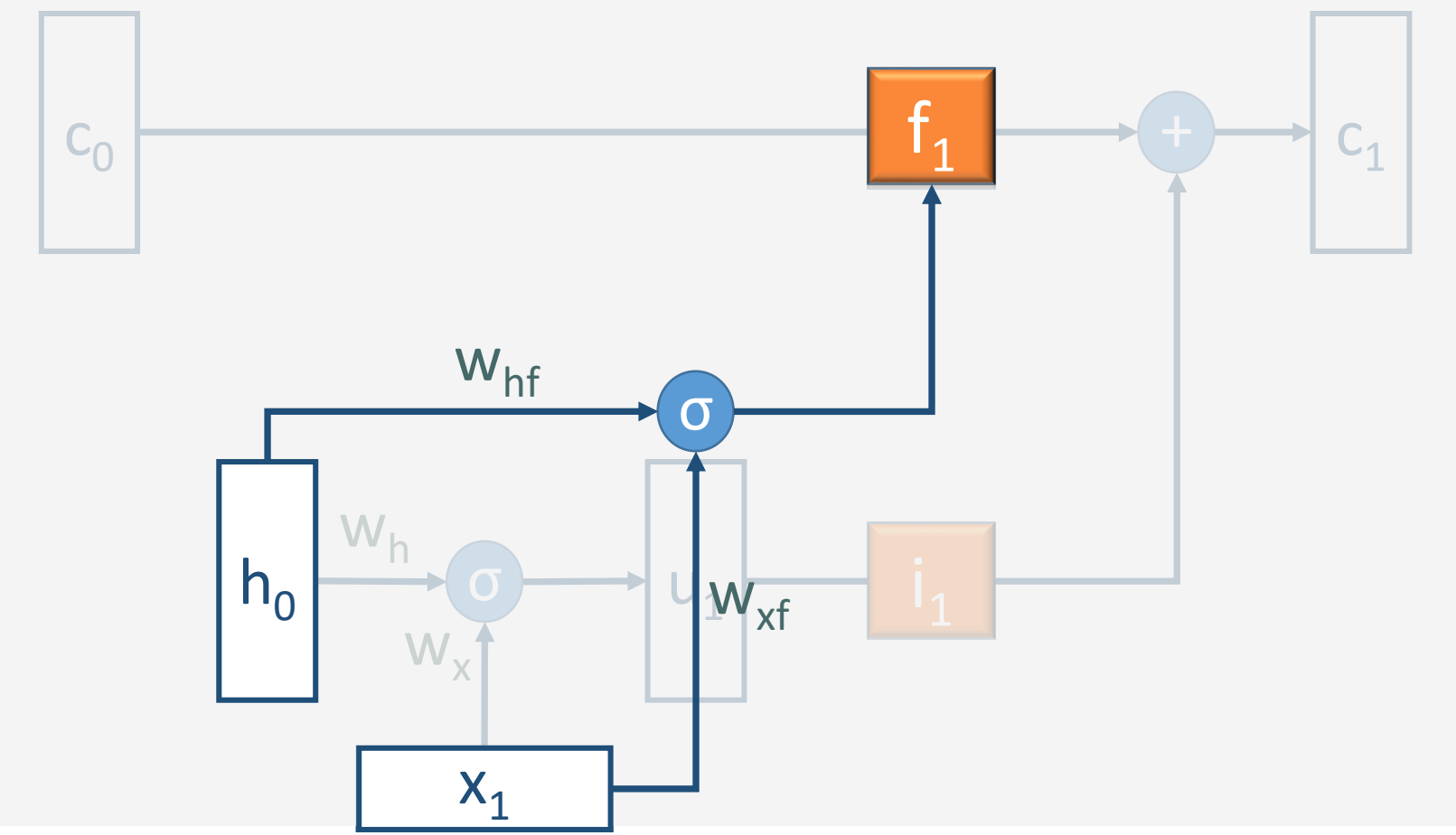
Transforming RNN to LSTM



$$c_t = f_t \odot c_{t-1} + i_t \odot u_t$$

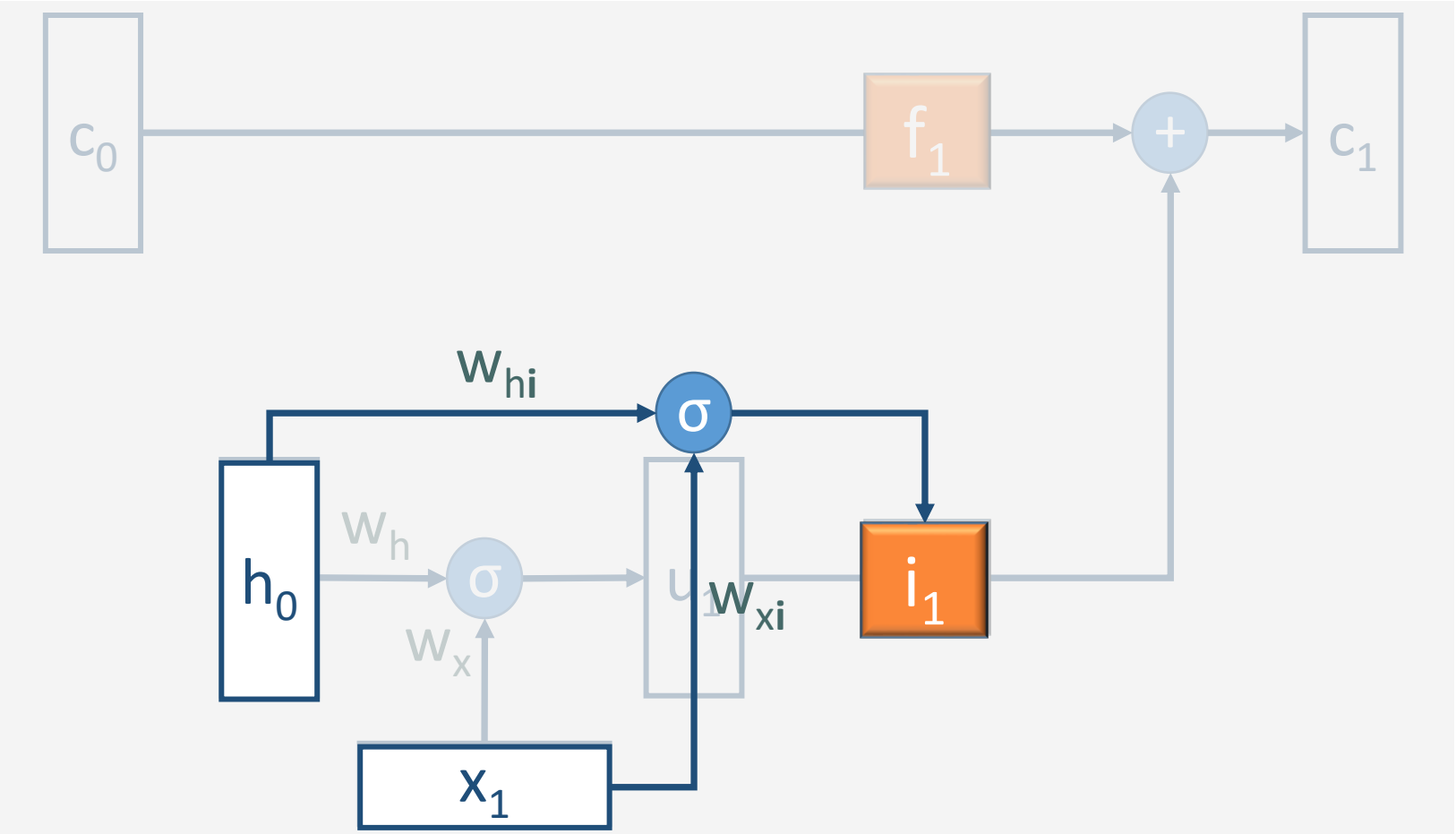
Elementwise (Hadamard) matrix product
(array product "*" in Python)

Transforming RNN to LSTM



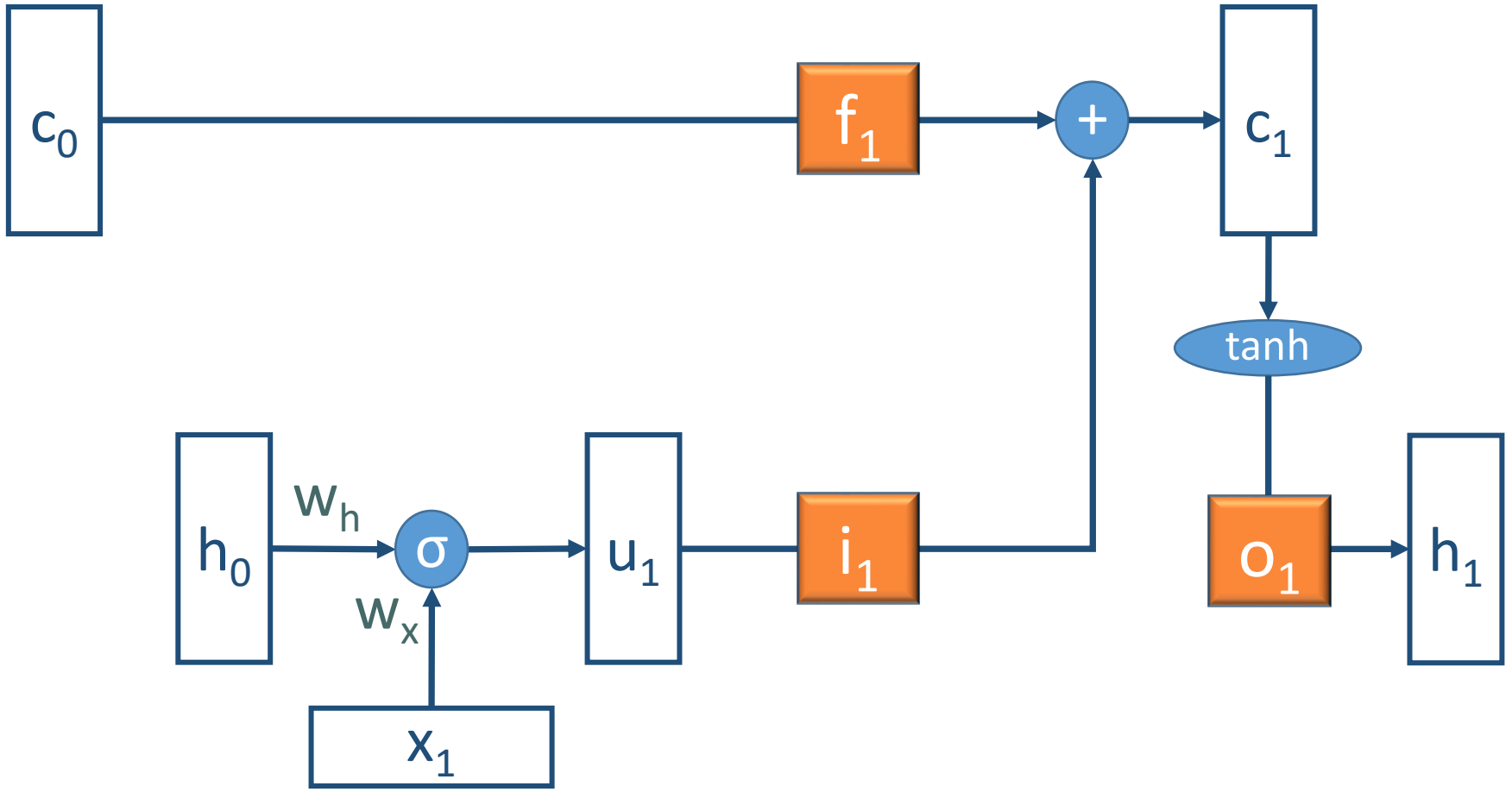
$$f_t = \sigma(W_{hf} h_{t-1} + W_{xf} x_t)$$

Transforming RNN to LSTM



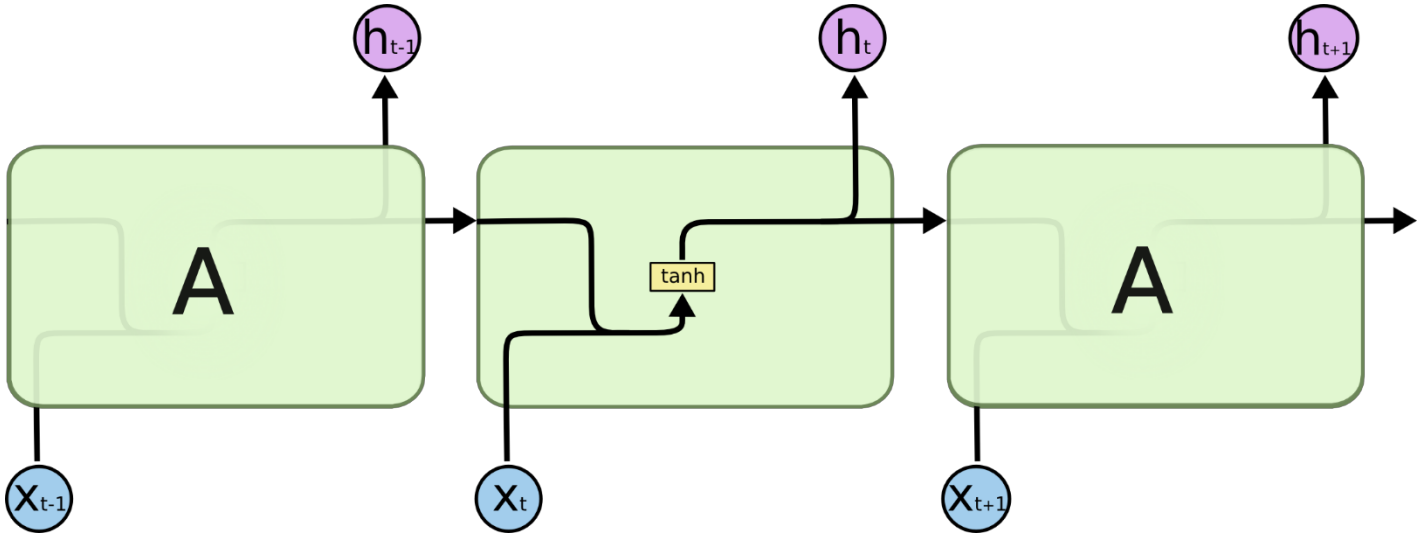
$$i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t)$$

Transforming RNN to LSTM

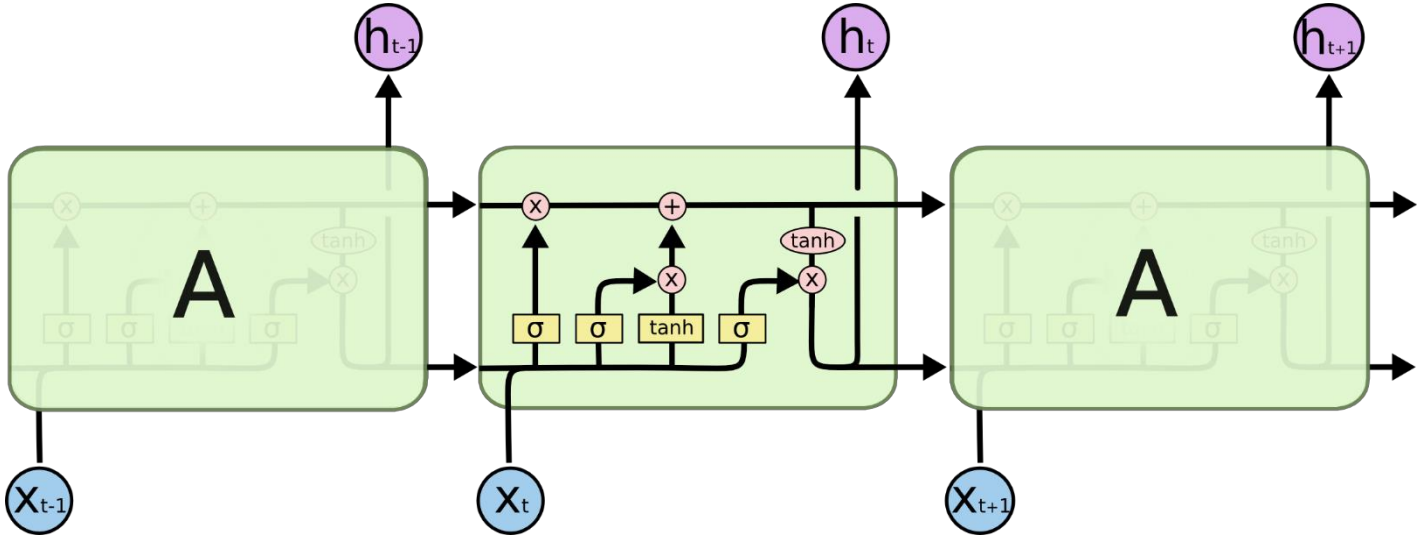


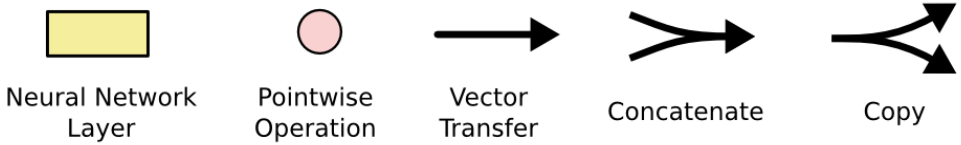
$$h_t = o_t \odot \tanh c_t$$

RNN

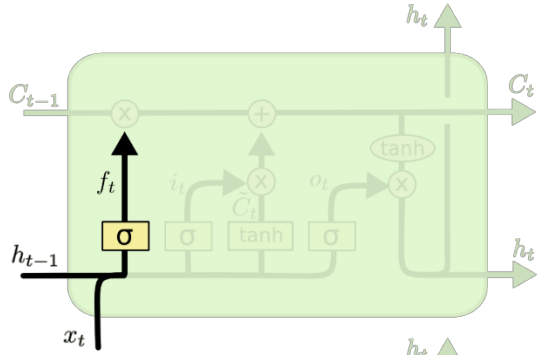


LSTM

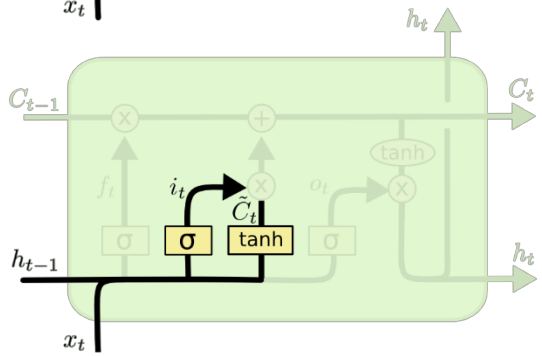




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

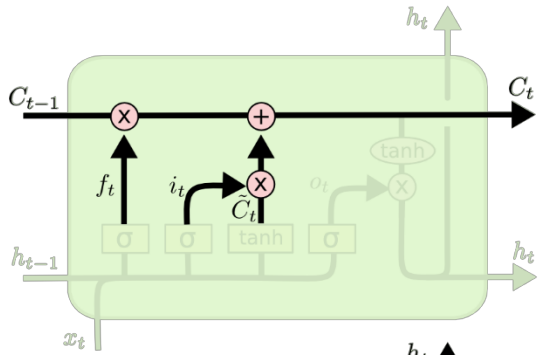


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

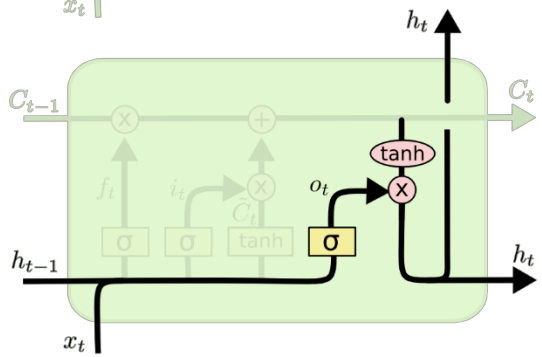


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



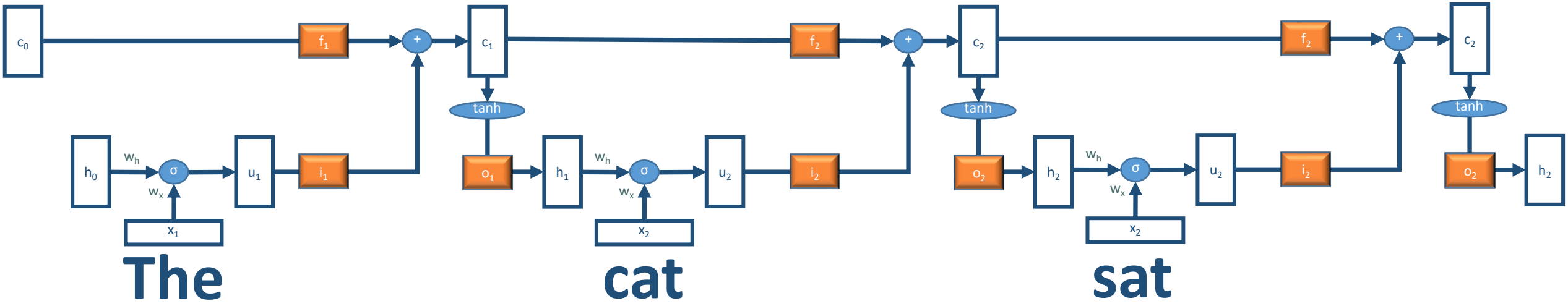
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM for Sequences



LSTM Applications

- Language identification (Gonzalez-Dominguez et al., 2014)
- Paraphrase detection (Cheng & Kartsaklis, 2015)
- Speech recognition (Graves, Abdel-Rahman, & Hinton, 2013)
- Handwriting recognition (Graves & Schmidhuber, 2009)
- Music composition (Eck & Schmidhuber, 2002) and lyric generation (Potash, Romanov, & Rumshisky, 2015)
- Robot control (Mayer et al., 2008)
- Natural language generation (Wen et al. 2015) (best paper at EMNLP)
- Named entity recognition (Hammerton, 2003)

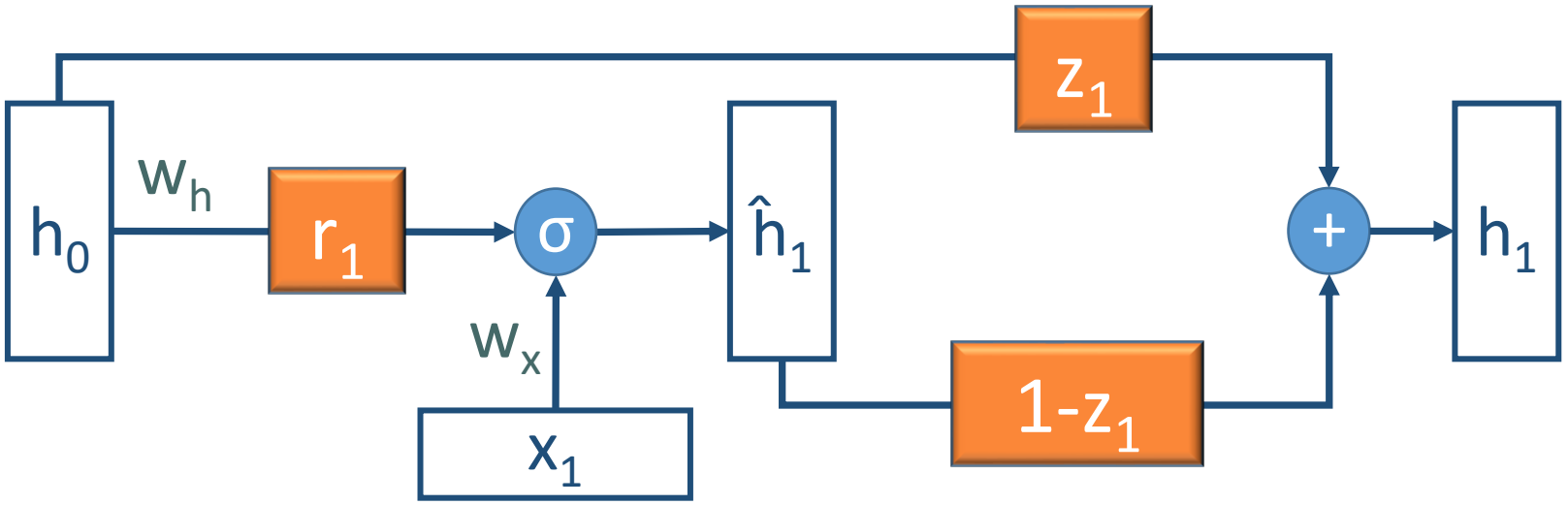
Handwriting generation

http://www.cs.toronto.edu/~graves/handwriting.html

Other Architectures

- Bidirectional LSTM
  - Concatenate two one-directional LSTMs
- Stacked LSTM

Related Architectures: GRU



Chung et al. (2014) reports comparable performance to LSTM

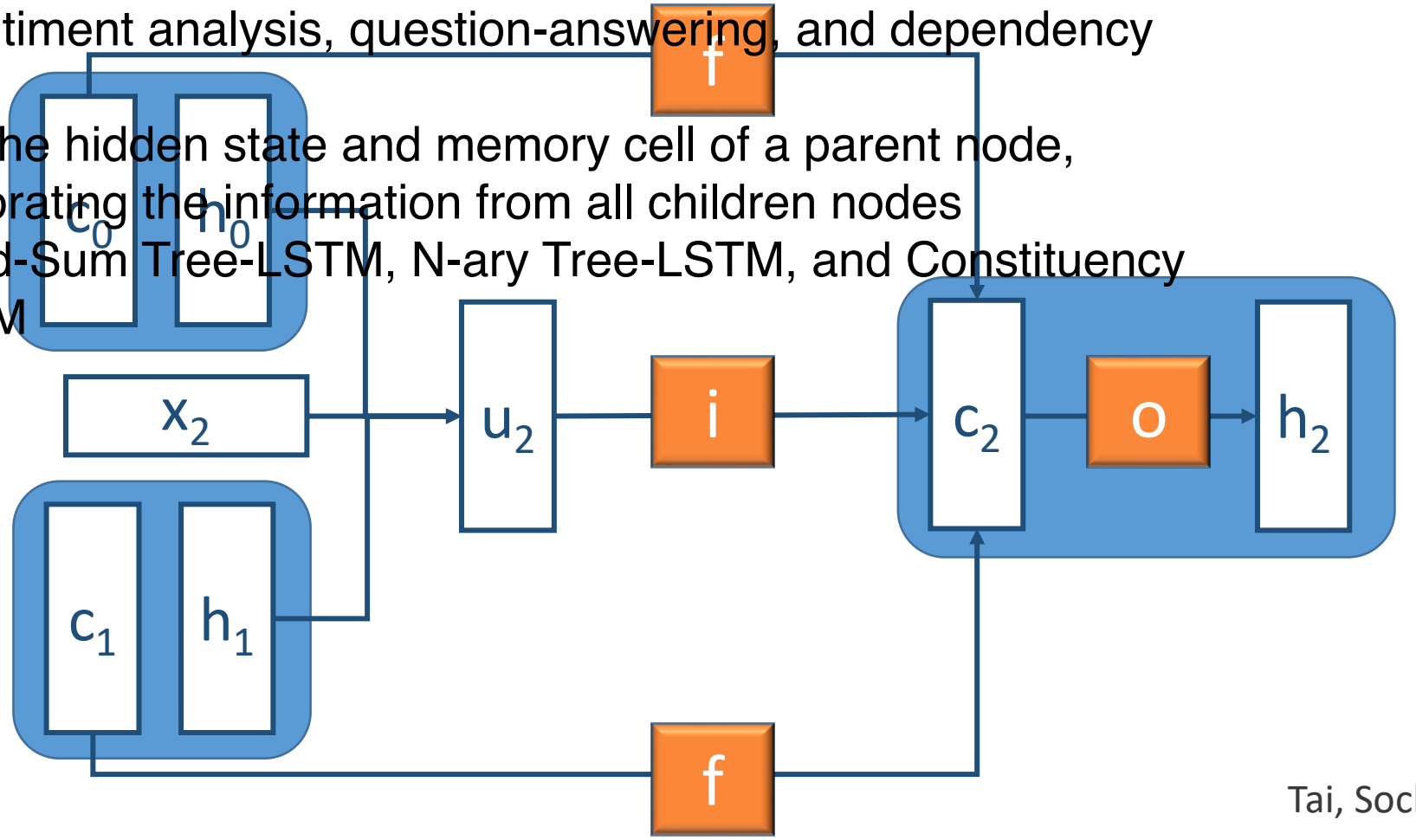
Related Architectures: Tree LSTMs

for tree-structure data, sentences can be represented as parse trees

tasks: sentiment analysis, question-answering, and dependency parsing.

compute the hidden state and memory cell of a parent node, by incorporating the information from all children nodes

type: Child-Sum Tree-LSTM, N-ary Tree-LSTM, and Constituency Tree-LSTM



Tai, Socher, Manning 2015

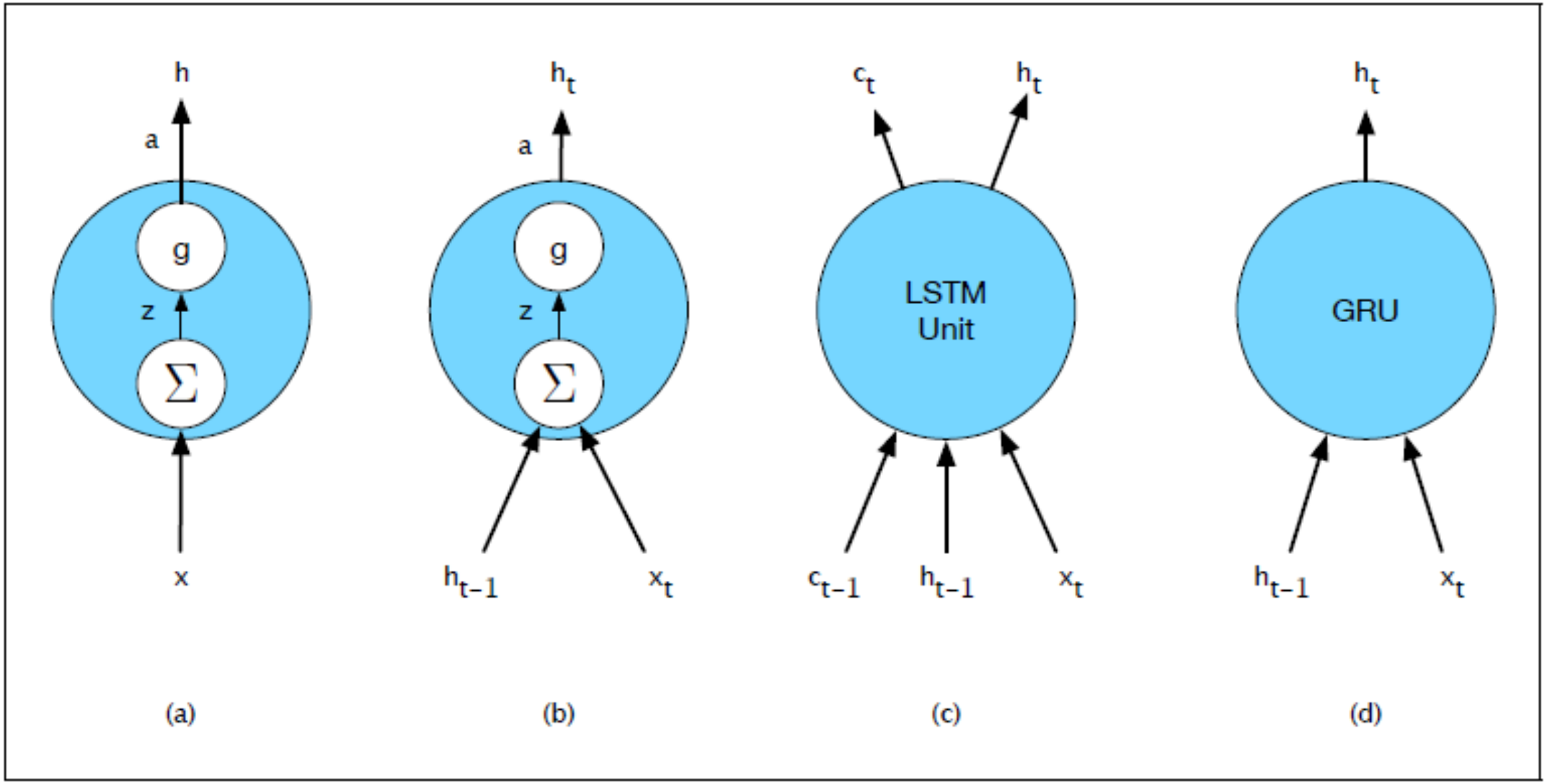


Figure 9.14 Basic neural units used in feedforward, simple recurrent networks (SRN), long short-term memory (LSTM) and gate recurrent units. g: activation function

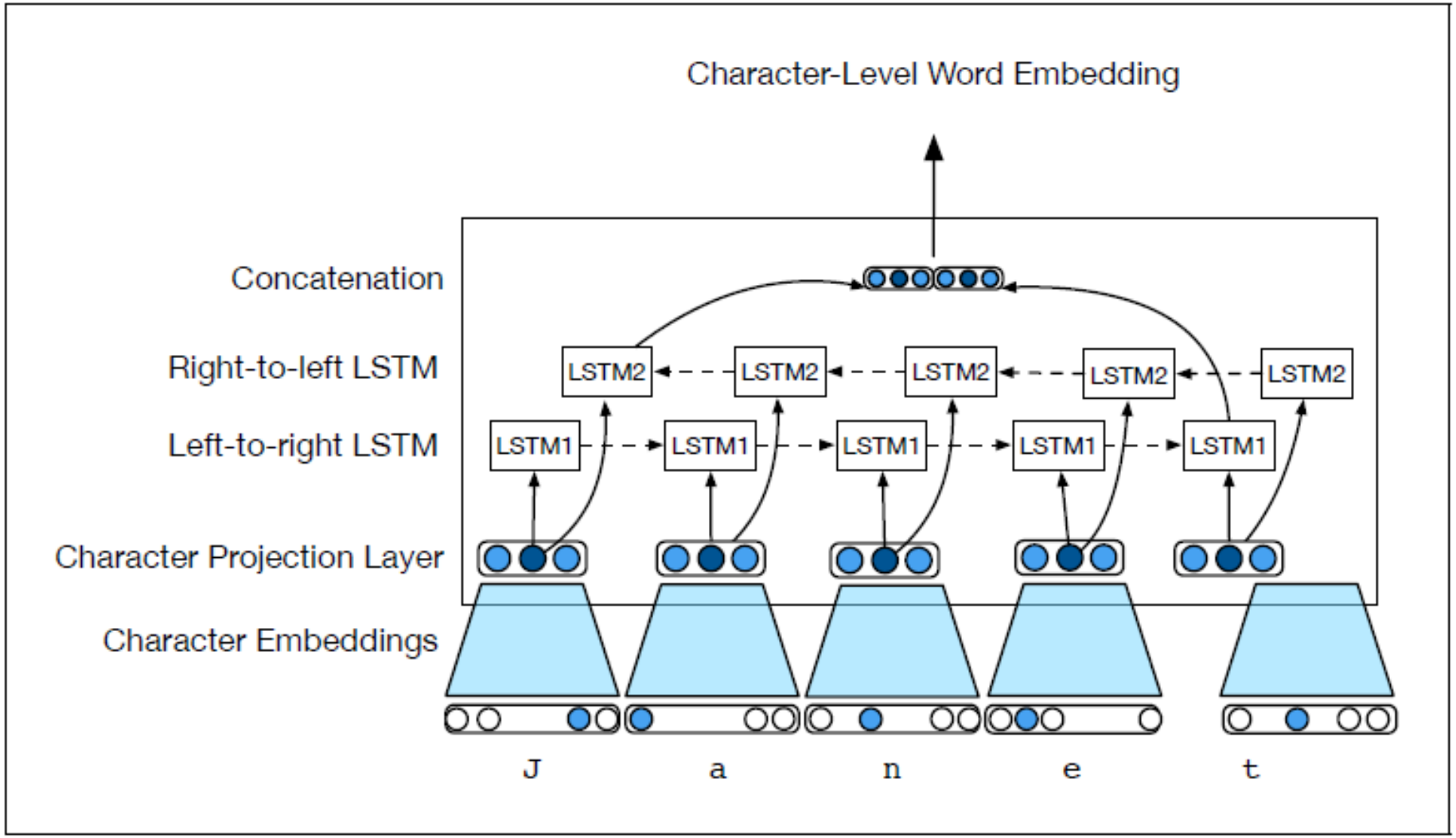


Figure 9.16 Bi-RNN accepts word character sequences and emits embeddings derived from a forward and backward pass over the sequence. The network itself is trained in the context of a larger end-application where the loss is propagated all the way through to the character vector embeddings.

External Link

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Next Steps

- Moving away from RNN/LSTM to other architectures, such as transformers
  - Later lecture
- (Chris Manning) the WMT 2016 final report has 44 instances of RNN whereas the WMT 2018 report has RNN 9 times and Transformer 63 times.