Recurrent Neural Networks for NLP

Tim Metzler

Department of Computer Science

HBRS / ST.A.

What do we need them for? (I/II)

- General case:
  We need RNNs for representing a sequence of **variable length** as a single vector (encoder) OR generating a sequence of **variable length** from a single vector (decoder)

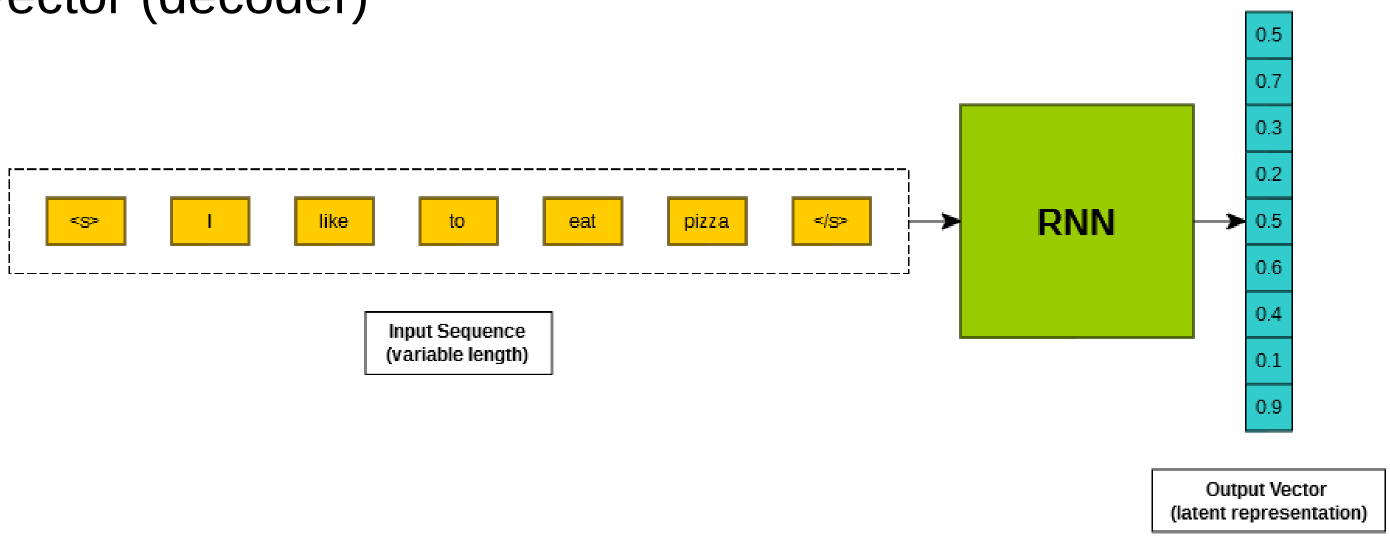


Fig. 1: High level representation of an encoder

What do we need them for? (II/II)

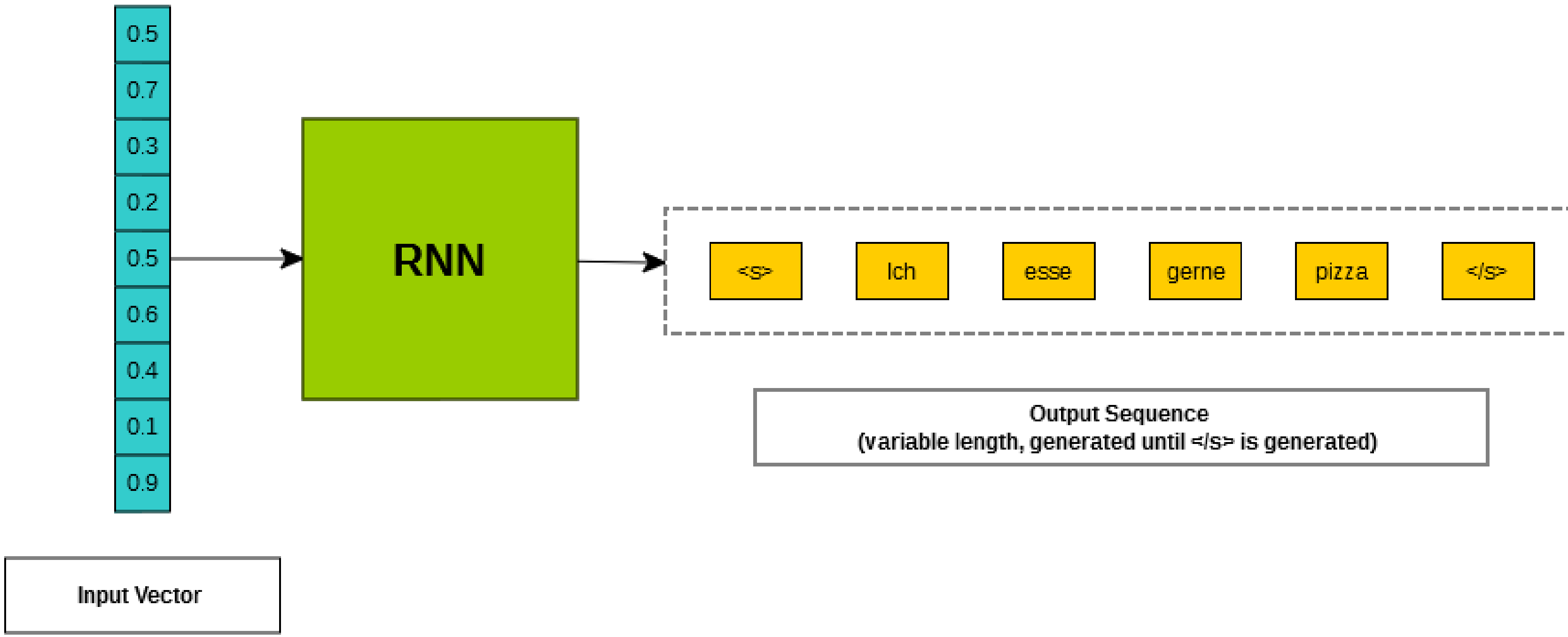


Fig. 2: High level representation of a decoder

Types of RNNs relevant to us

- Gated Recurrent Unit (GRU)
- Long Short Term Memory (LSTM)

Gated Recurrent Unit

Fig. 3.1: High level represenation of GRU cell

Gated Recurrent Unit

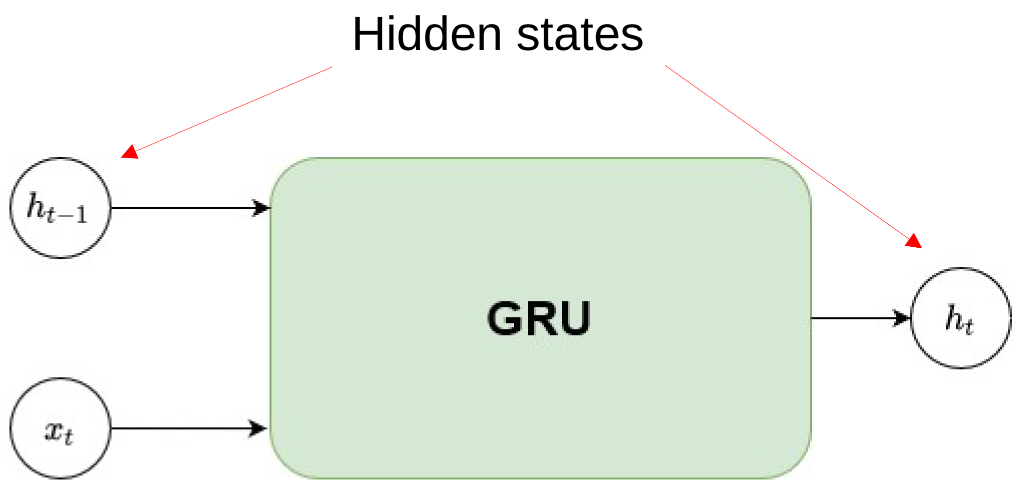


Fig. 3.2: High level represenation of GRU cell

Gated Recurrent Unit

Output of the previous time step. $h_{t=0} = [0, \ldots, 0]$

Fig. 3.3: High level represenation of GRU cell

Gated Recurrent Unit

Input for the current time step. Embedding for token t

Fig. 3.4: High level represenation of GRU cell

Gated Recurrent Unit

Output for the current time step.

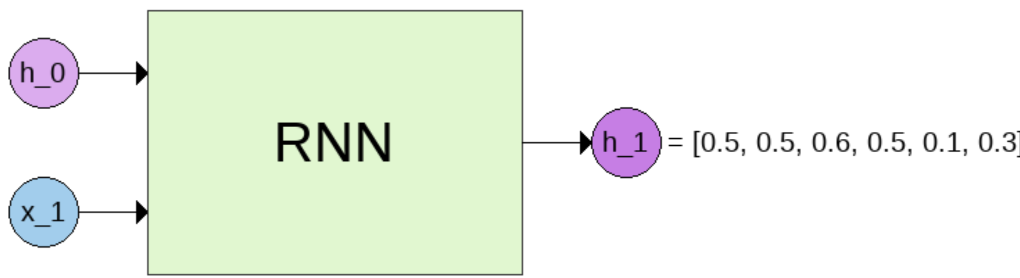
Fig. 3.5: High level represenation of GRU cell

Encoder Example

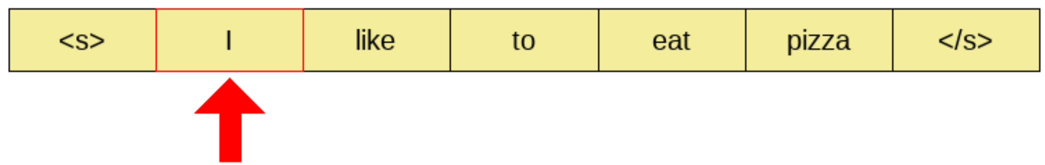
<s>	I	like	to	eat	pizza	</s>

Previous Output: h_0 = [0, 0, 0, 0, 0, 0]

Current Input: x_1 = Embedding (<s>) = [0.6, 0.7, 0.6, 0.5]

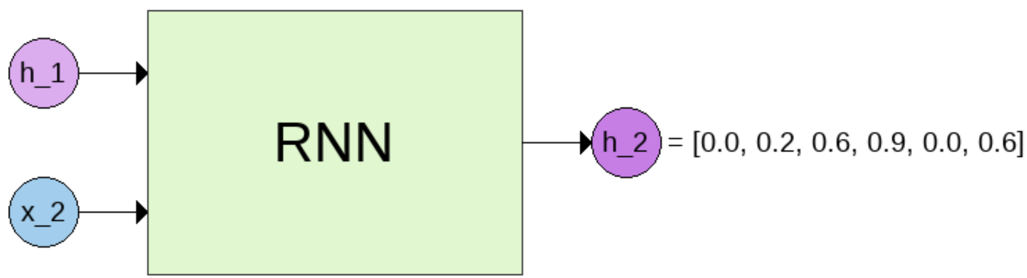


Encoder Example



Previous Output: h_1 = [0.5, 0.5, 0.6, 0.5, 0.1, 0.3]

Current Input: x_2 = Embedding (I) = [0.0, 0.5, 0.5, 0.2]

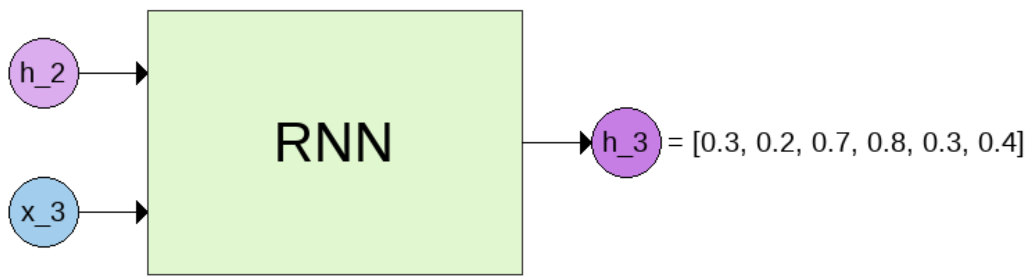


Encoder Example

<s>	I	like	to	eat	pizza	</s>

Previous Output: h_2 = [0.0, 0.2, 0.6, 0.9, 0.0, 0.6]

Current Input: x_3 = Embedding(like) = [0.0, 0.1, 0.1, 0.1]

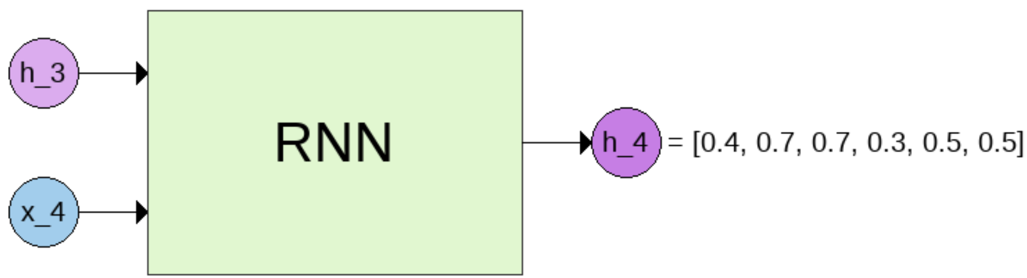


Encoder Example

<s>	I	like	to	eat	pizza	</s>

Previous Output: h_3 = [0.3, 0.2, 0.7, 0.8, 0.3, 0.4]

Current Input: x_4 = Embedding(to) = [0.2, 0.1, 0.2, 0.6]

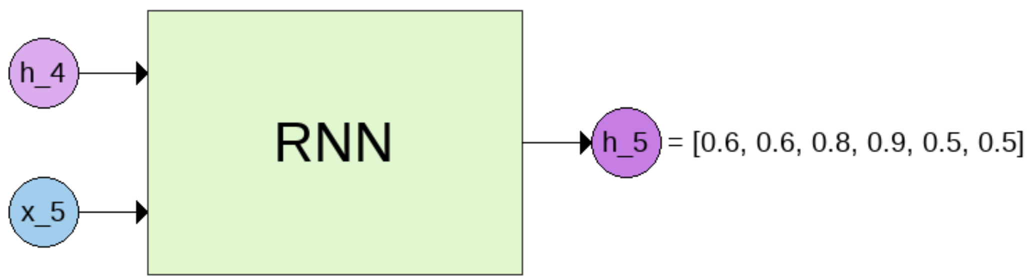


Encoder Example

<s>	I	like	to	eat	pizza	</s>

Previous Output: h_4 = [0.4, 0.7, 0.7, 0.3, 0.5, 0.5]

Current Input: x_5 = Embedding (eat) = [0.1, 0.4, 0.6, 0.5]

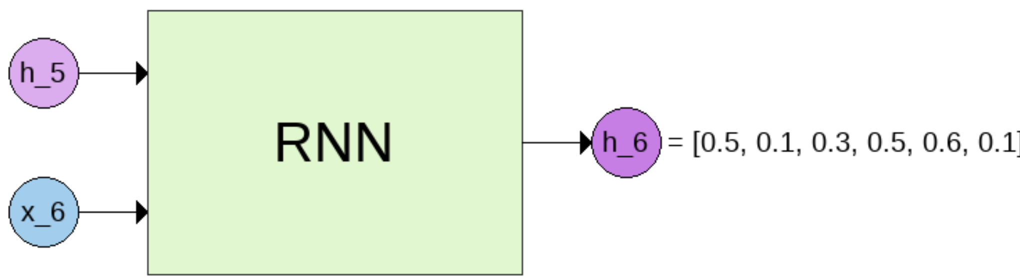


Encoder Example

<s>	I	like	to	eat	pizza	</s>

Previous Output: h_5 = [0.6, 0.6, 0.8, 0.9, 0.5, 0.5]

Current Input: x_6 = Embedding (pizza) = [0.9, 0.8, 0.2, 0.9]

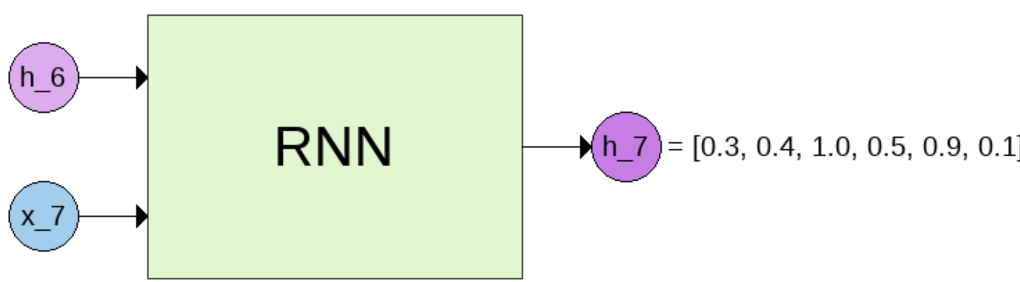


Encoder Example

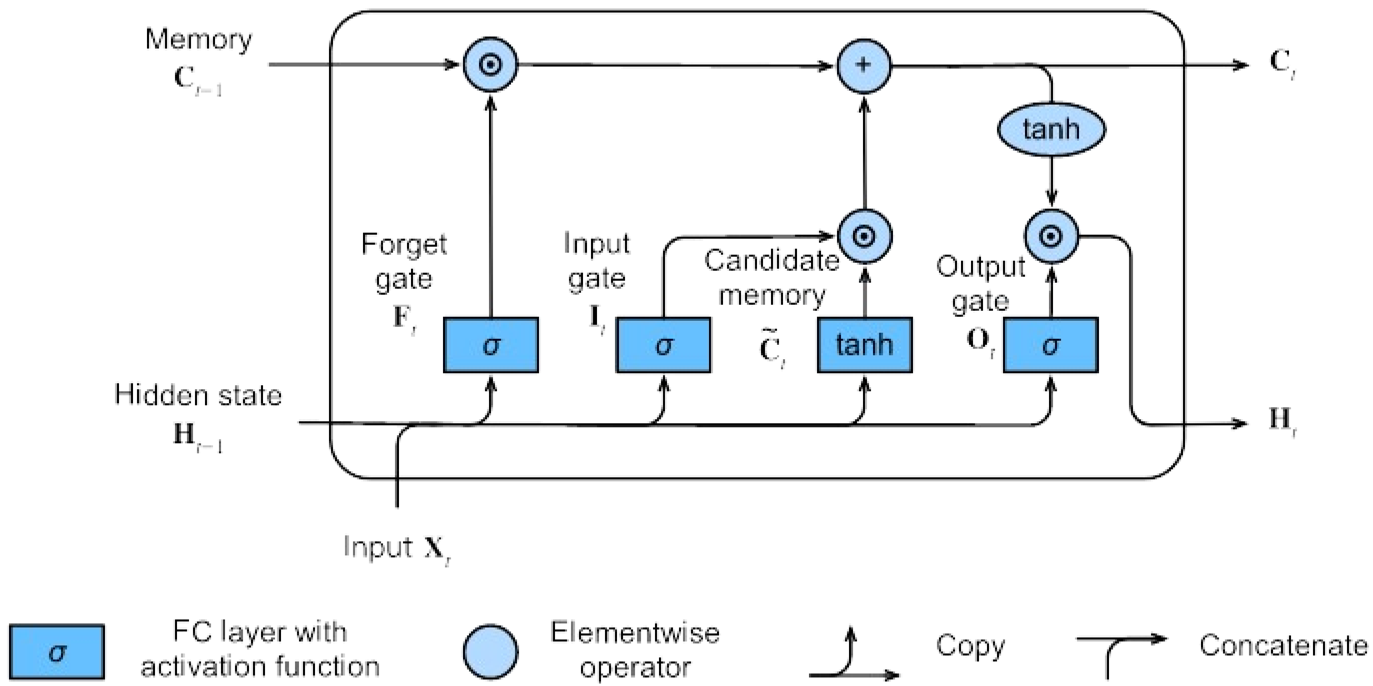
<s>	I	like	to	eat	pizza	</s>

Previous Output: h_6 = [0.5, 0.1, 0.3, 0.5, 0.6, 0.1]

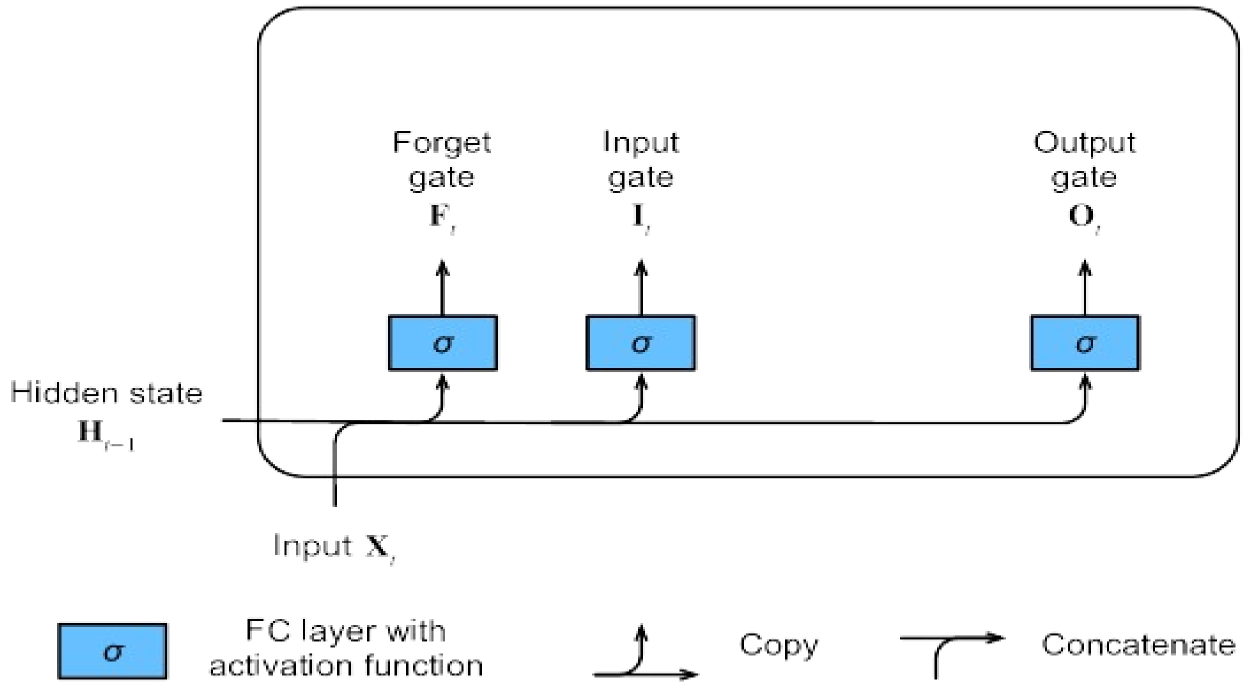
Current Input: x_7 = Embedding (</s>) = [0.0, 0.9, 0.8, 0.8]



Long Short Term Memory



Long Short Term Memory



Input Gate

$$\sigma\left(X_t \times W_{xi} + H_{t-1} \times W_{hi} + b_i \right) = I_t$$

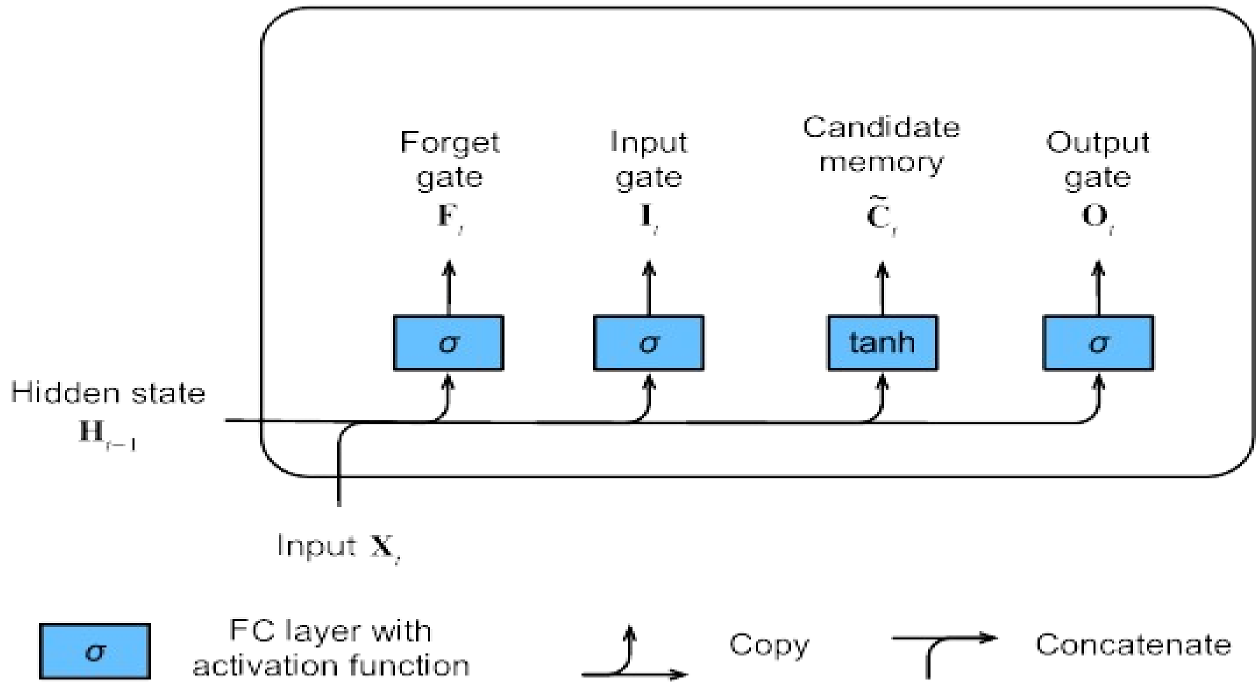
Forget Gate

$$\sigma\left(X_t \times W_{xf} + H_{t-1} \times W_{hf} + b_f \right) = F_t$$

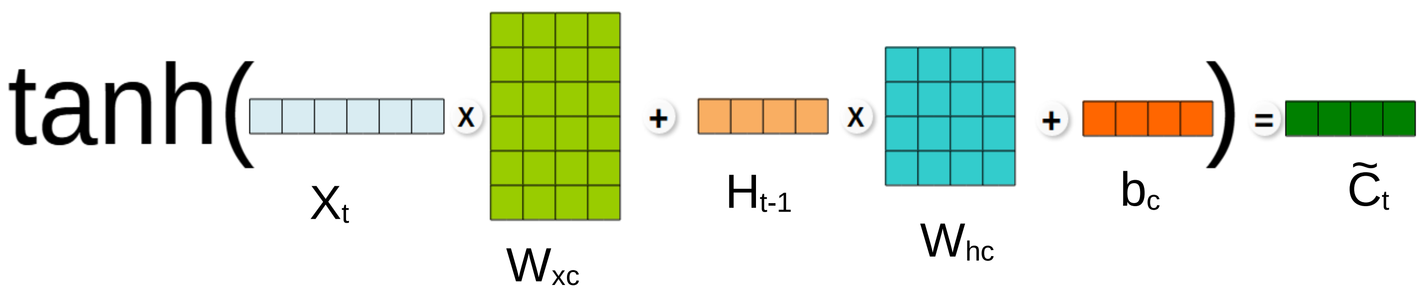
Output Gate

$$\sigma\left(X_t \times W_{xo} + H_{t-1} \times W_{ho} + b_o \right) = O_t$$

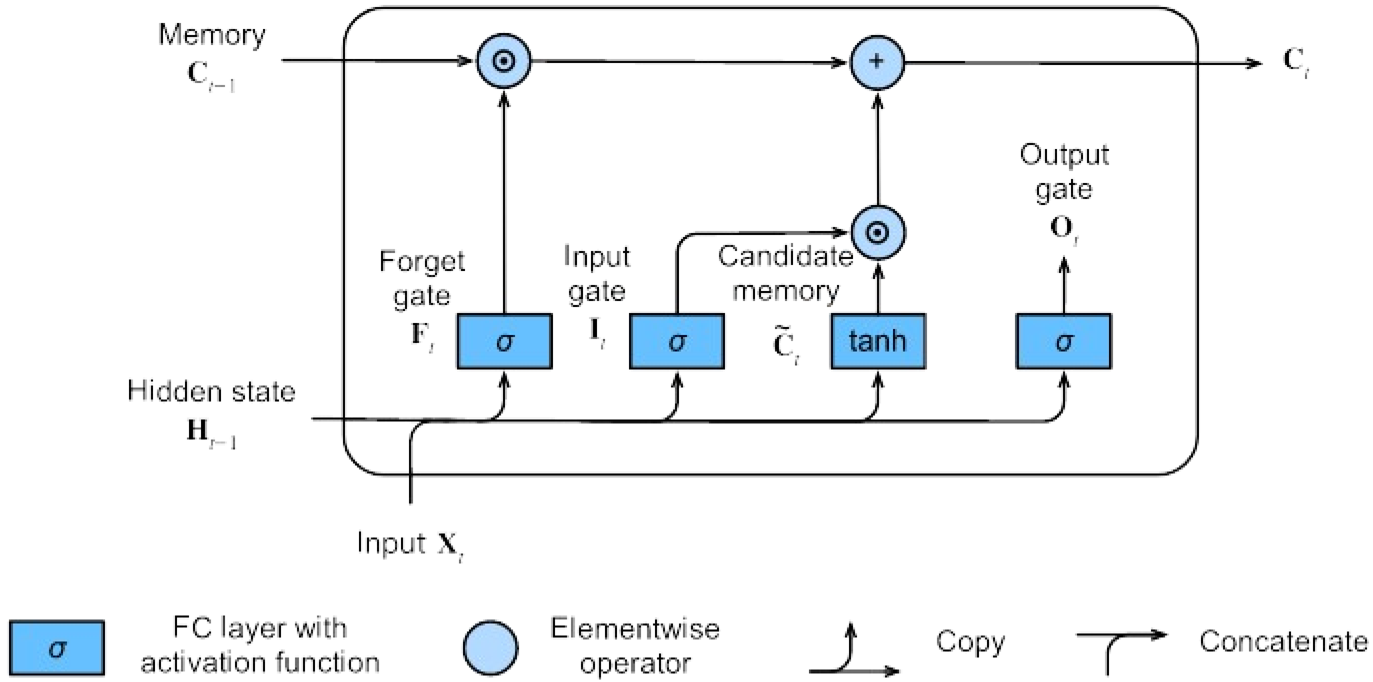
Long Short Term Memory



Long Short Term Memory



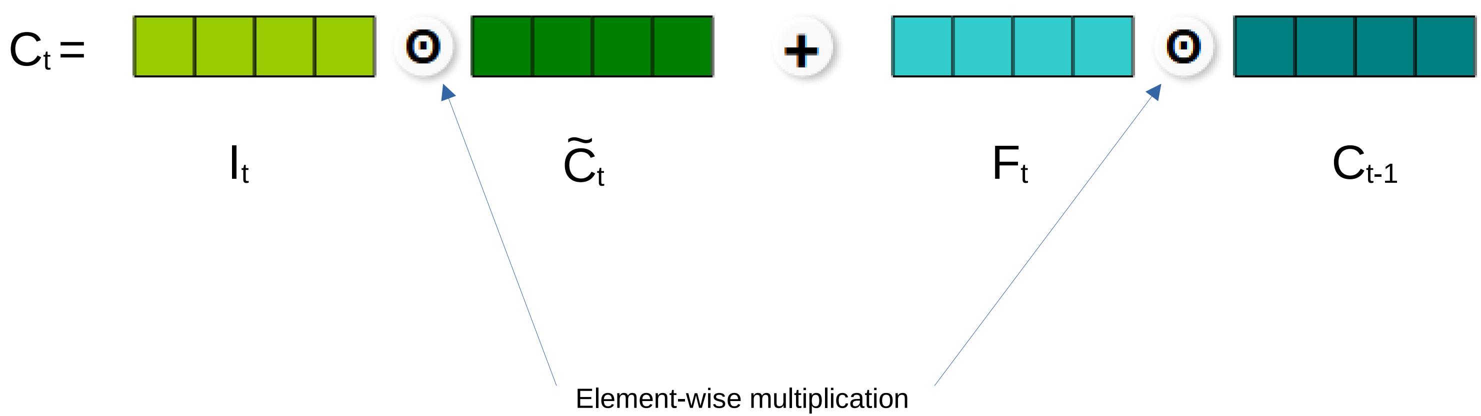
Long Short Term Memory



Long Short Term Memory

Update Cell state

$$C_t = I_t \odot \tilde{C}_t + F_t \odot C_{t-1}$$

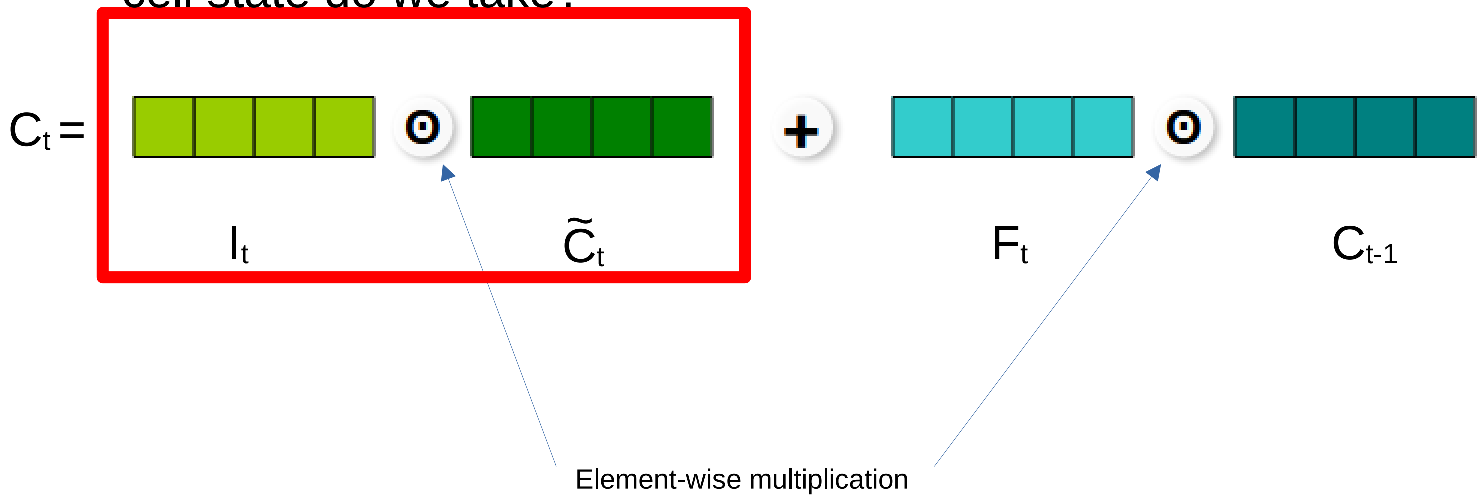


Long Short Term Memory

Update Cell state

$C_t = I_t \odot \tilde{C}_t + F_t \odot C_{t-1}$

How much of the candidate cell state do we take?

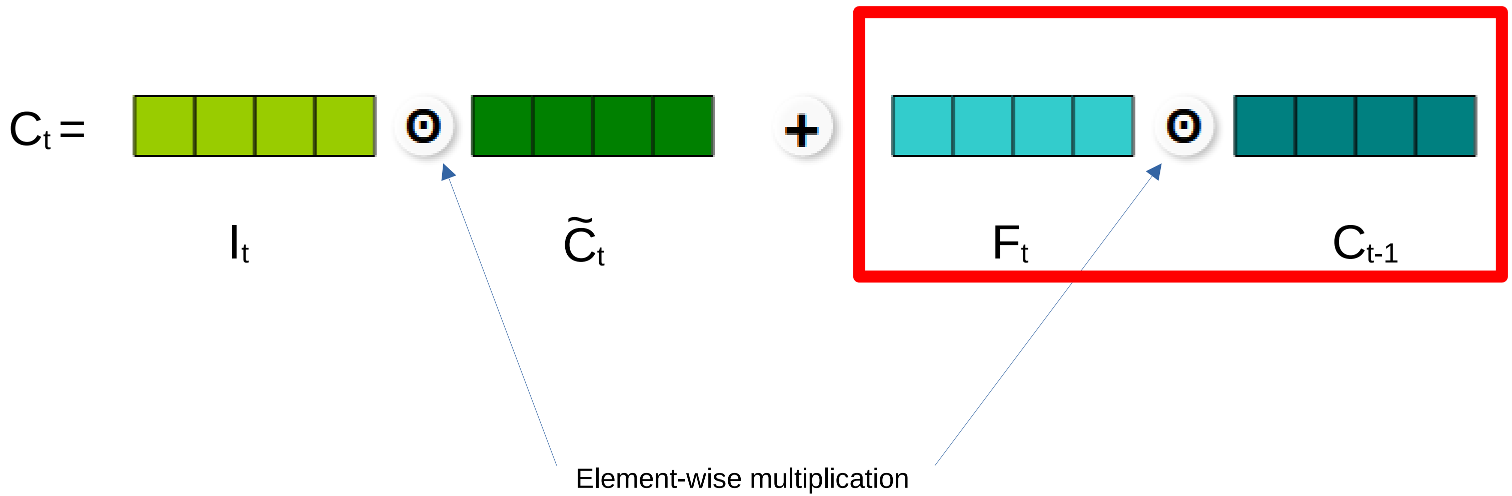


Long Short Term Memory

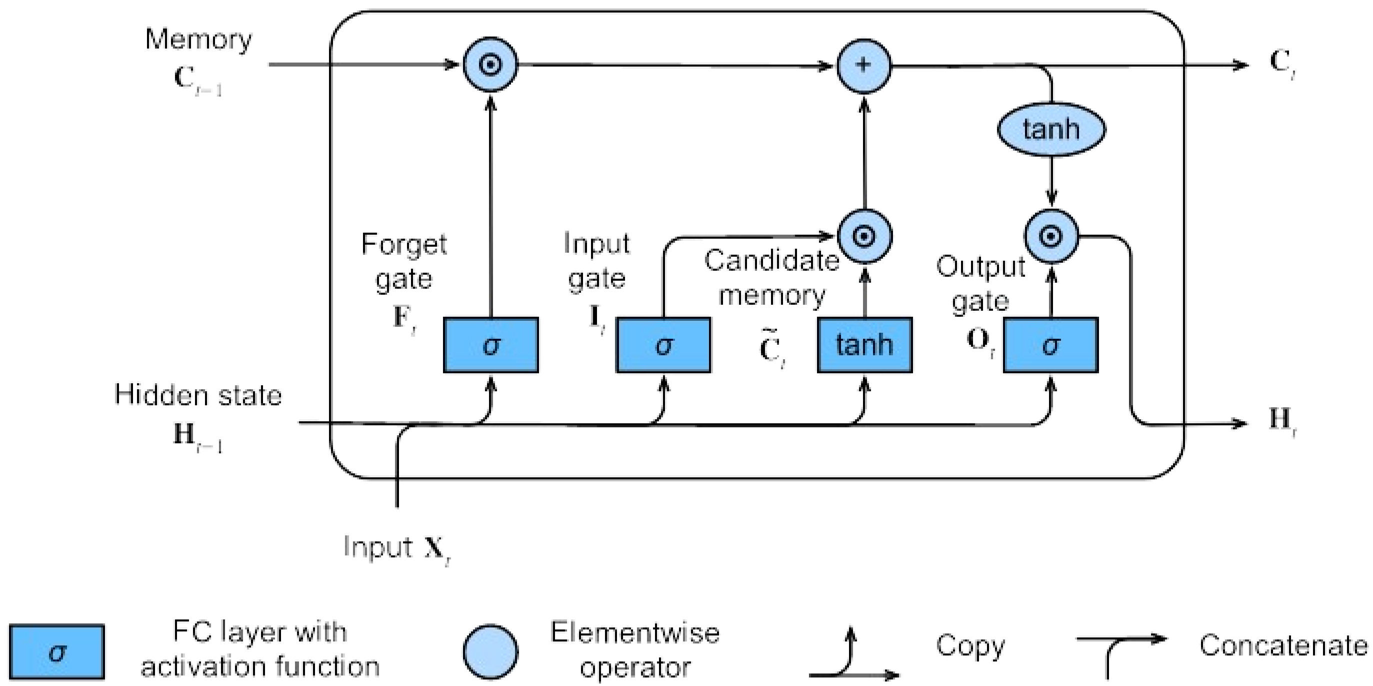
Update Cell state

$C_t = I_t \odot \tilde{C}_t + F_t \odot C_{t-1}$

How much of the previous cell state do we keep?



Long Short Term Memory



Long Short Term Memory

Compute hidden state (output)

$$H_t = \underset{O_t}{\boxed{\quad\quad\quad\quad}} \odot \tanh\left(\underset{C_t}{\boxed{\quad\quad\quad\quad}}\right)$$

LSTM - Complexity

- Number of parameters:
- \- Input dimensionality of d
  \- Output dimensionality of h
  \- 4 input weight matrices of (d x h)
  \- 4 hidden weight matrices of (h x h)
  \- 4 biases of (h x 1)
- → Number of parameters = 4*d*h + 4*h*h + 4*h = 4h*(d + h + 1)
- Example:
  \- Input size of 300
  \- Output size of 32
  → 42,624 parameters

RNNs in NLP

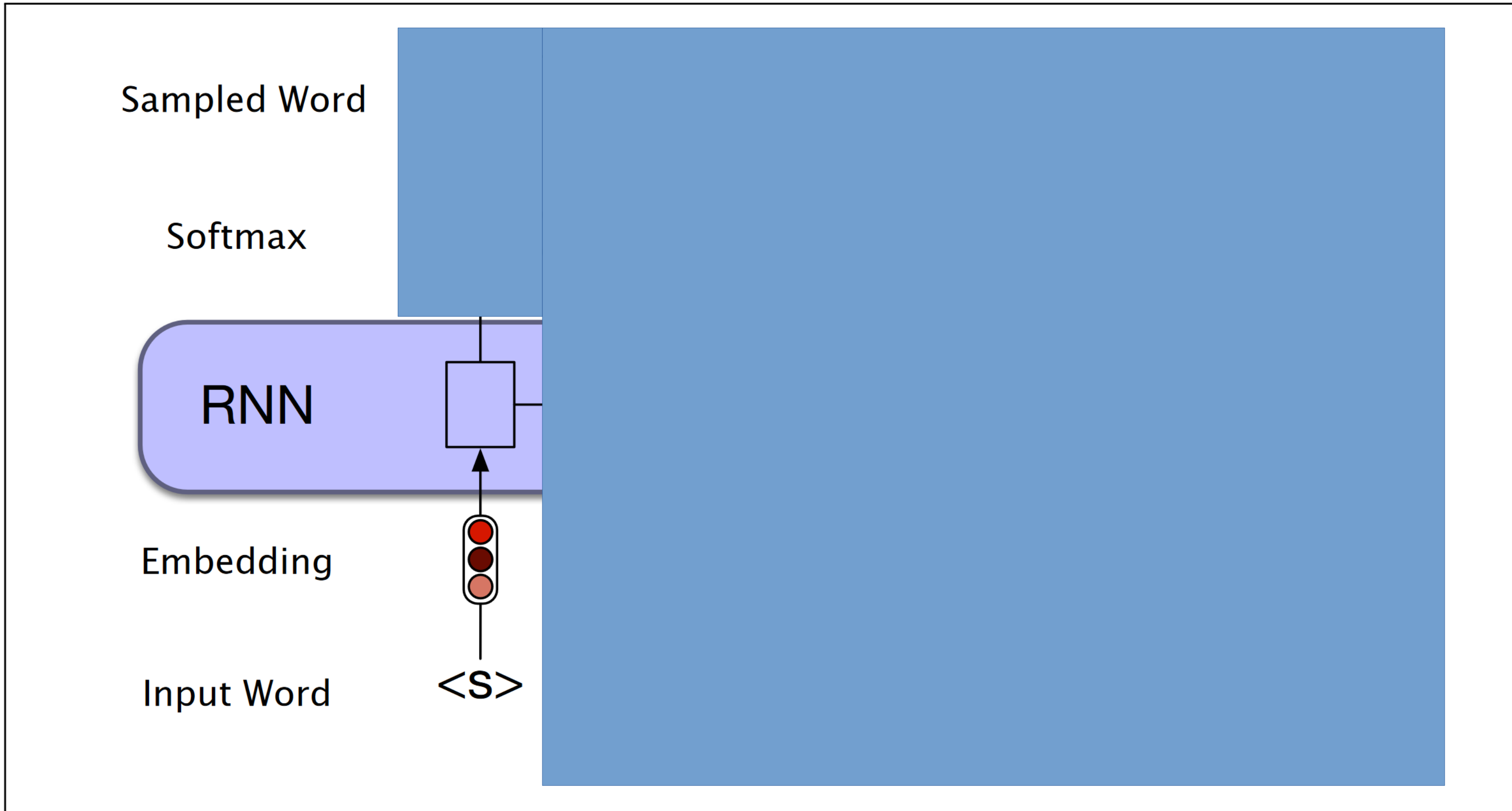


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

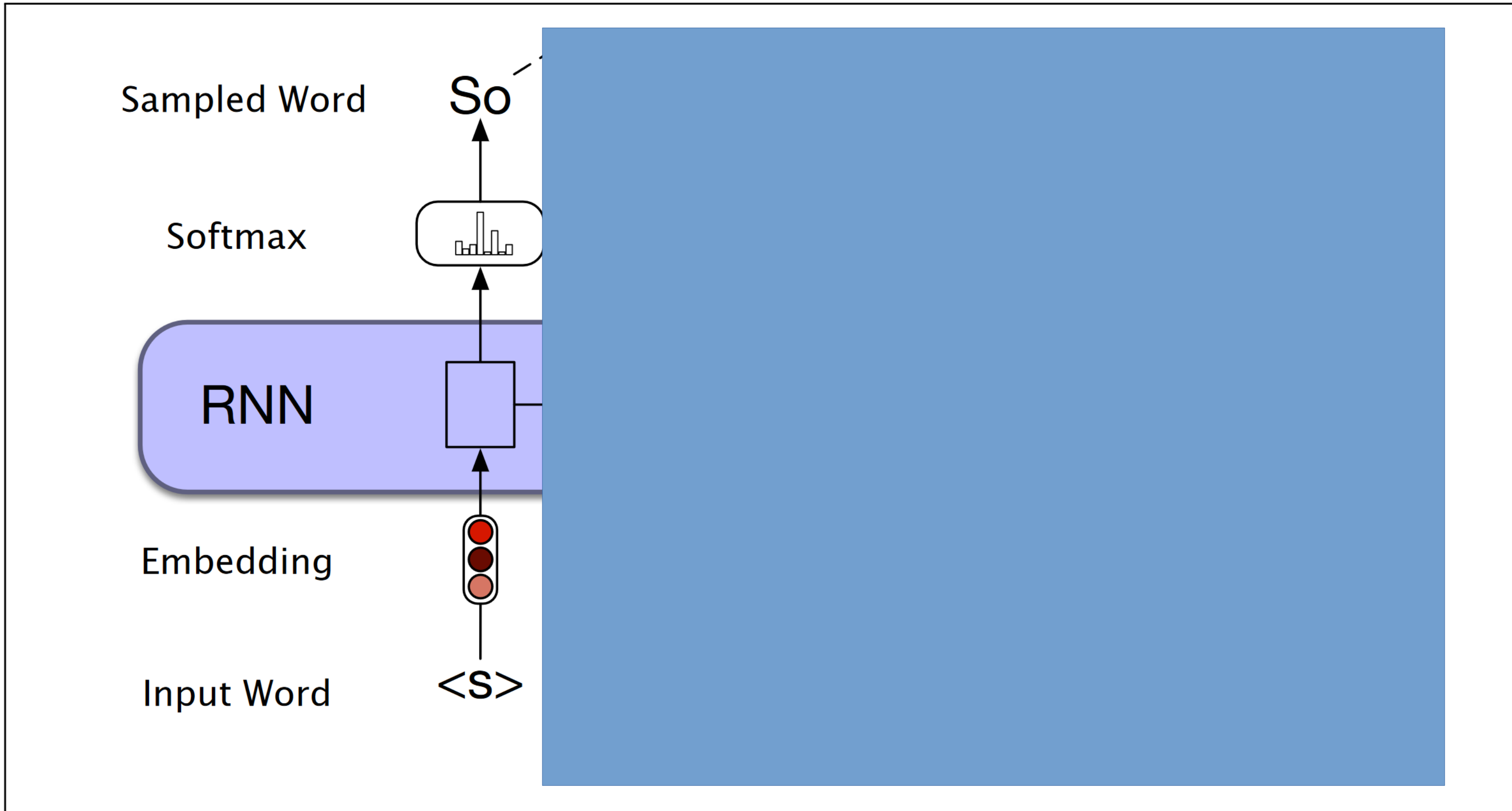


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

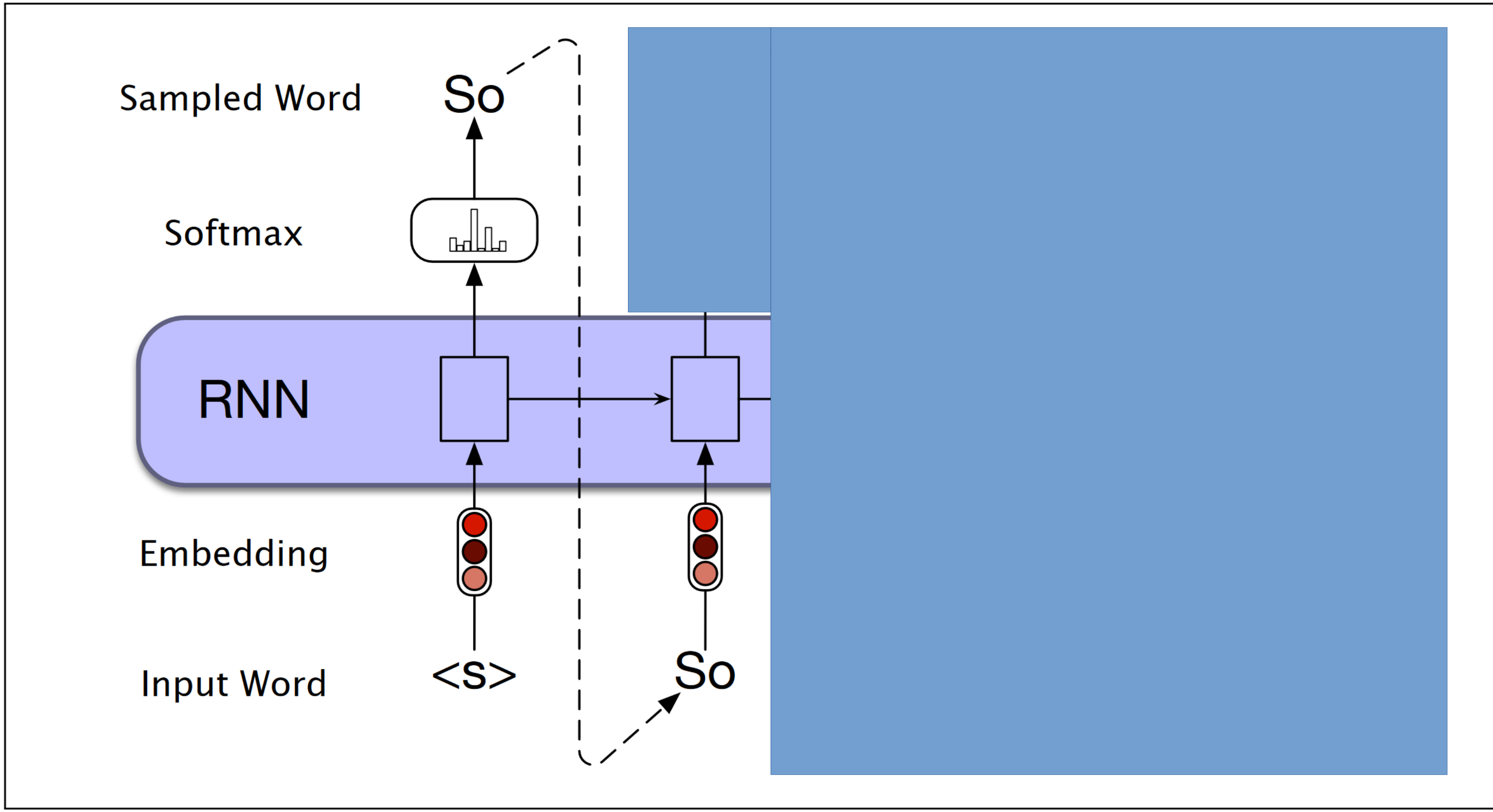


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

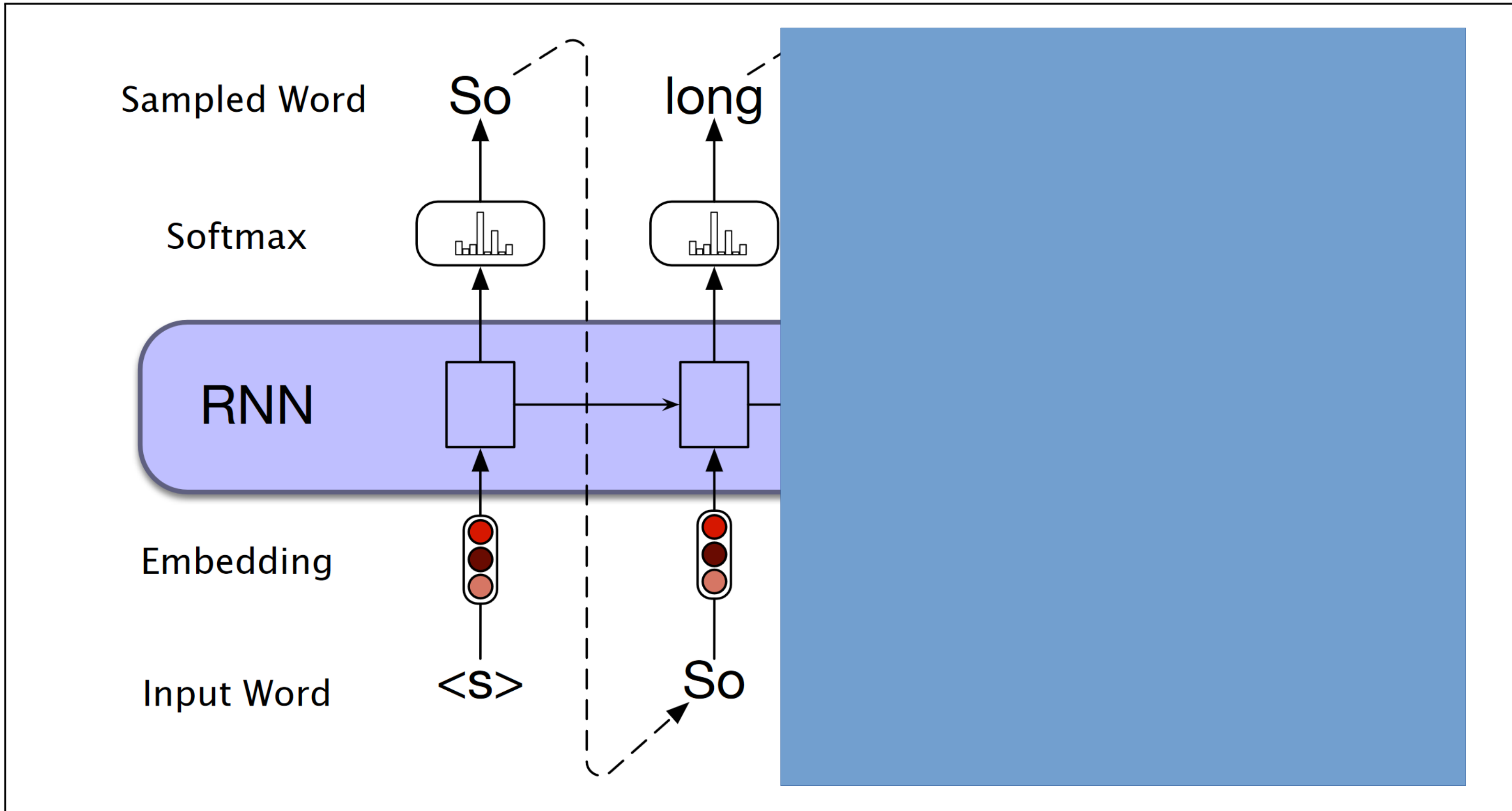


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

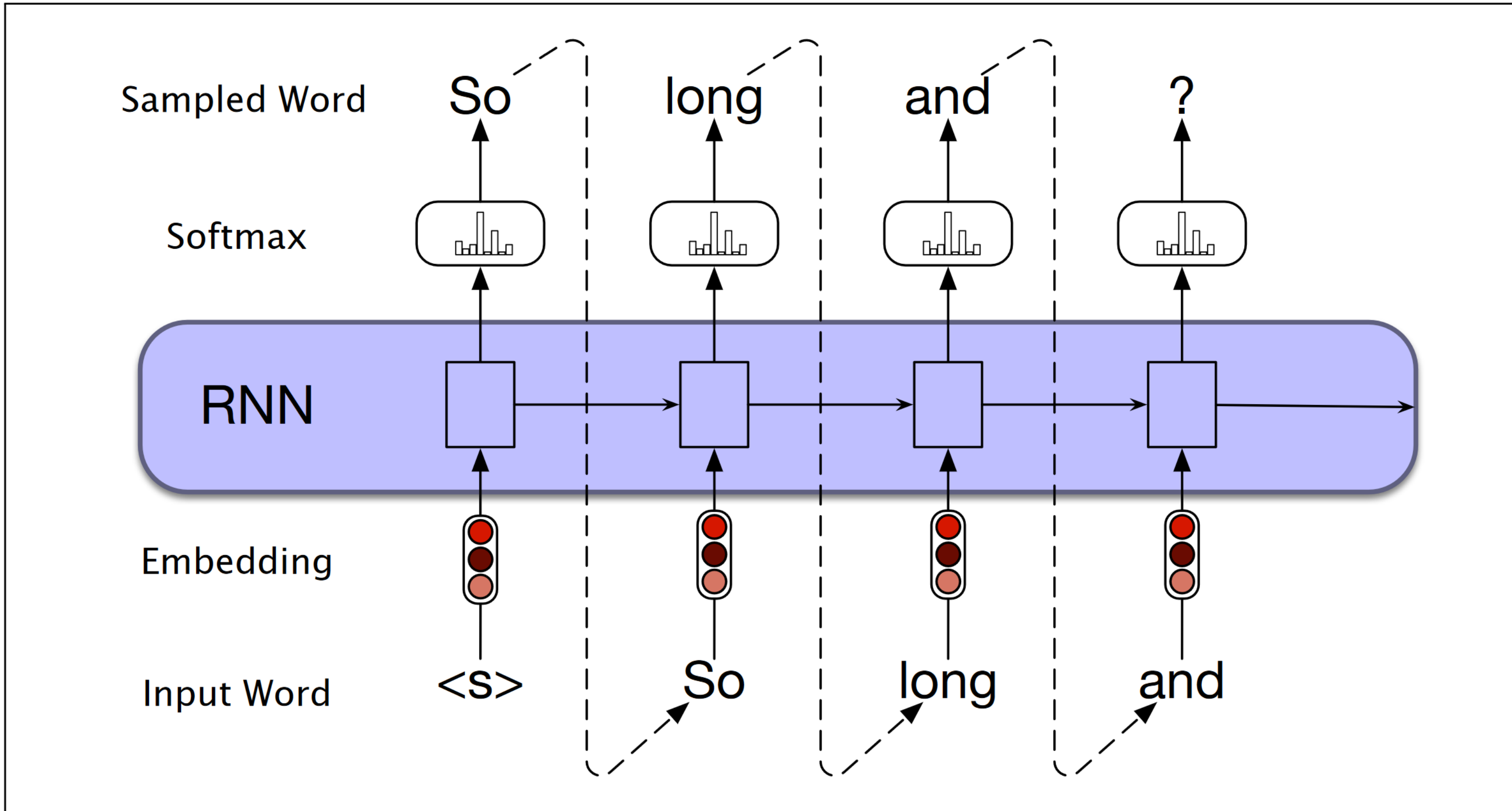


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

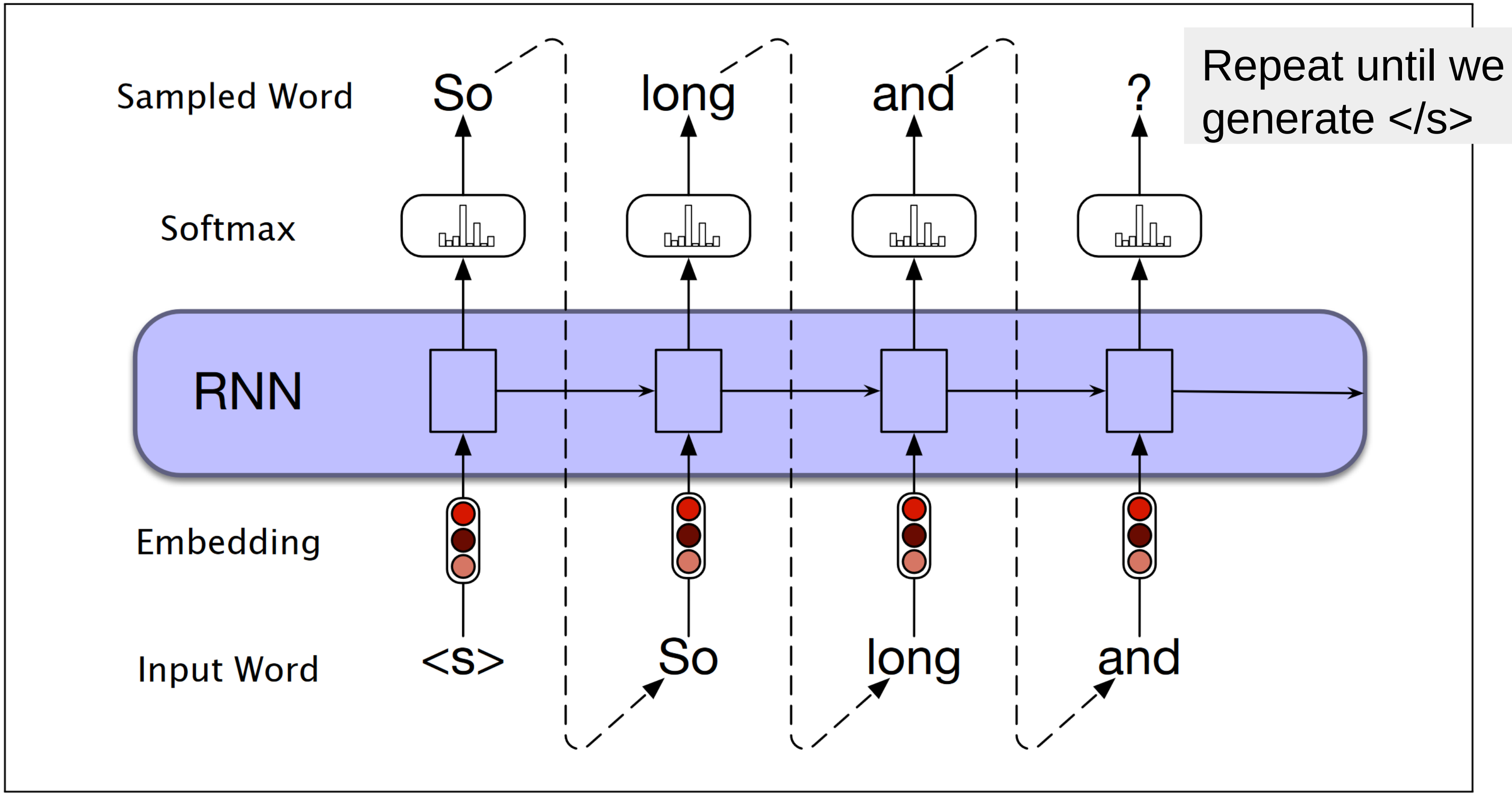


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

RNNs in NLP

How do we train this model?

RNNs in NLP

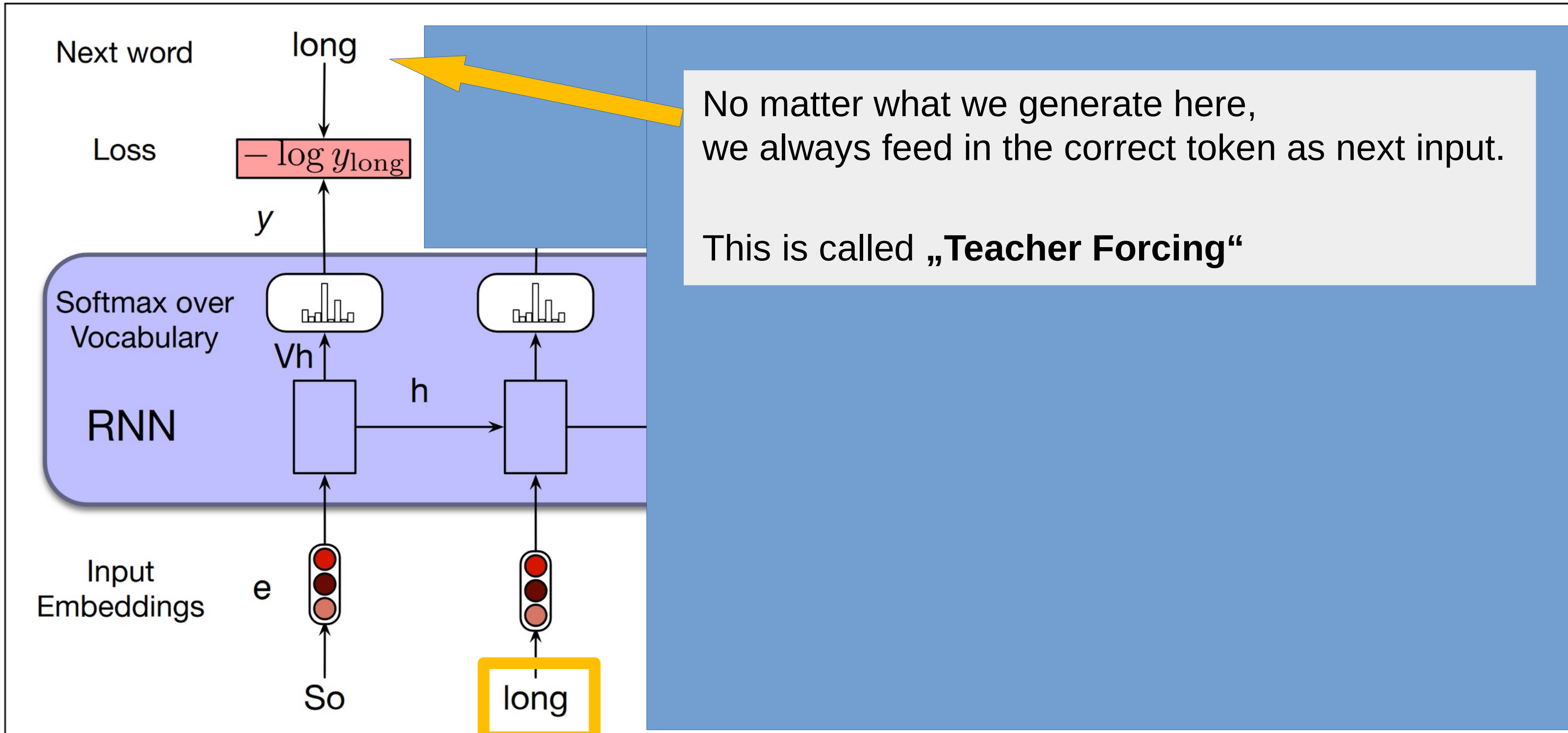
Target: So long and thanks for all the ...

Figure 9.6 Training RNNs as language models.

source: Jurafsky & Martin „Speech and Language Processing", Chapter 9, page 7

RNNs in NLP

Target: So long and thanks for all the ...



No matter what we generate here,
we always feed in the correct token as next input.

This is called **„Teacher Forcing“**

Figure 9.6 Training RNNs as language models.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 7

RNNs in NLP

Target: So long and thanks for all the ...

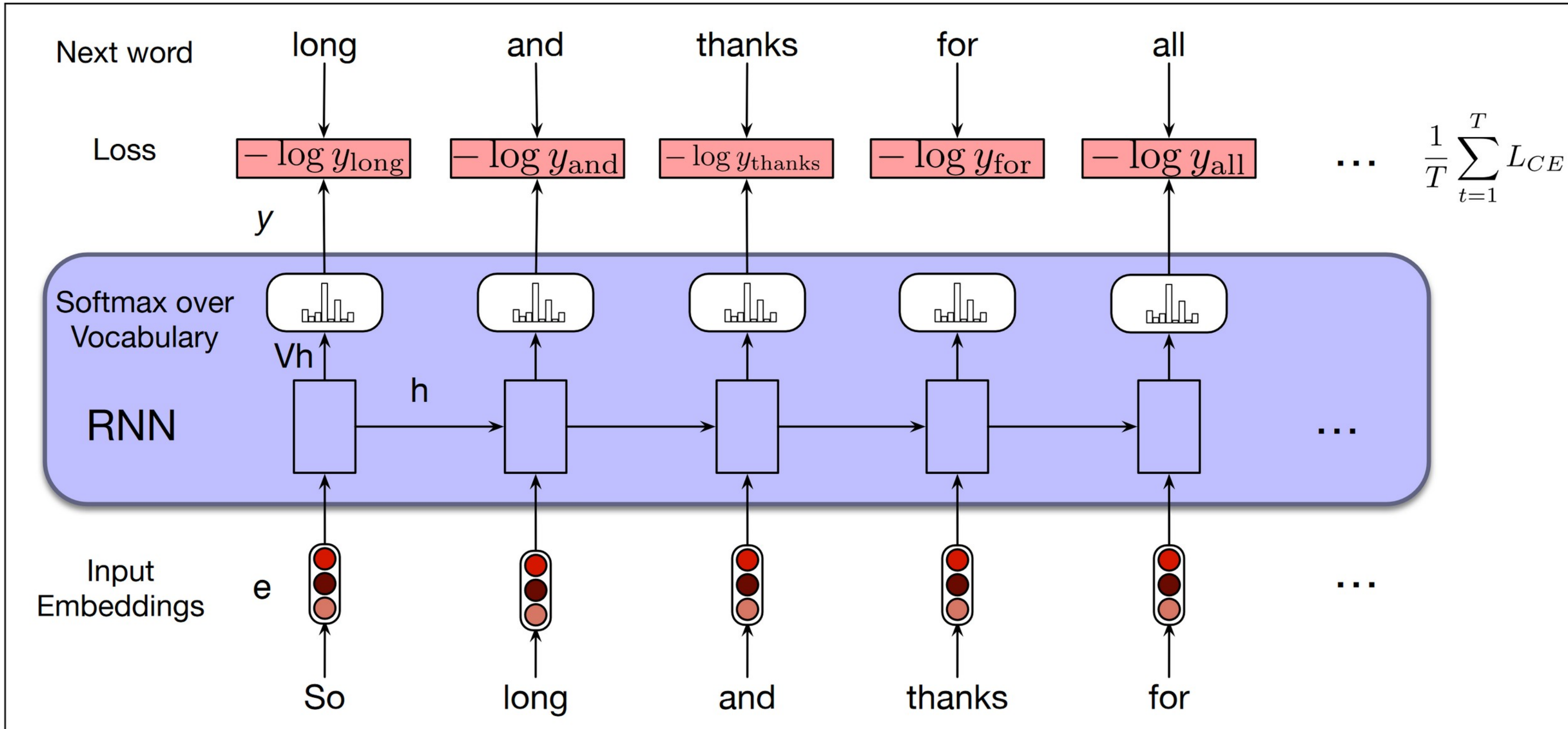


Figure 9.6 Training RNNs as language models.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 7

RNNs in NLP

Target: So long and thanks for all the ...

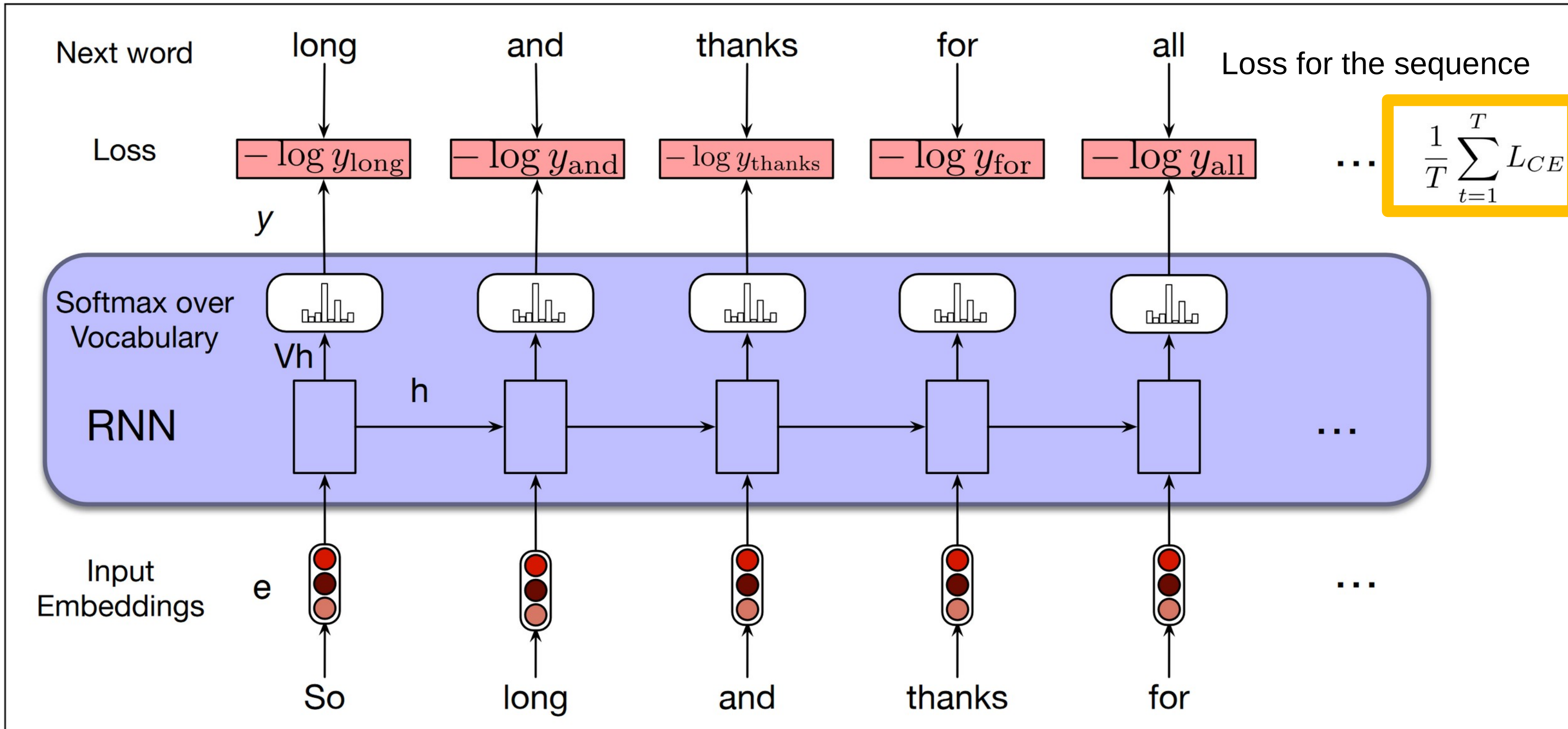


Figure 9.6 Training RNNs as language models.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 7

More applications

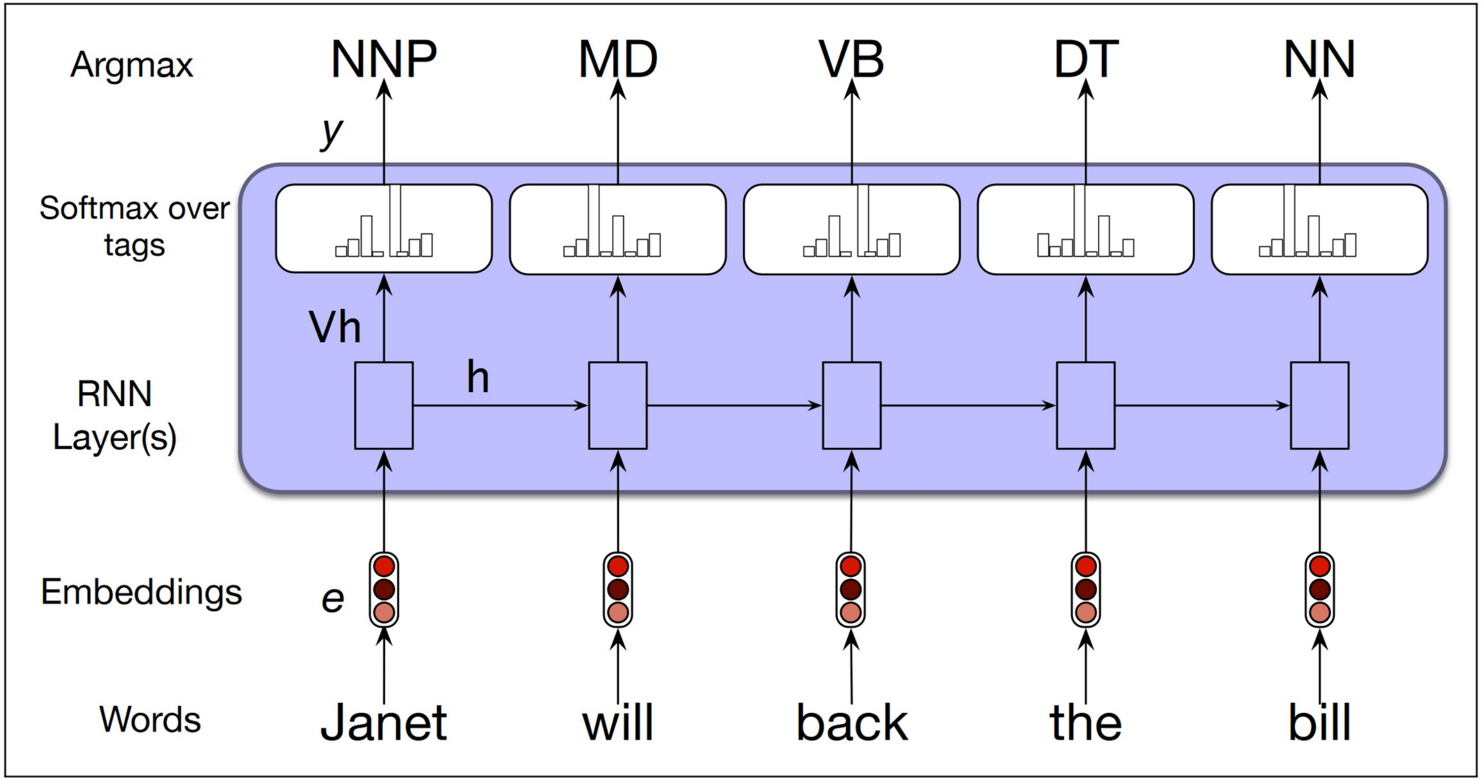


Figure 9.7 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 8

Machine Translation

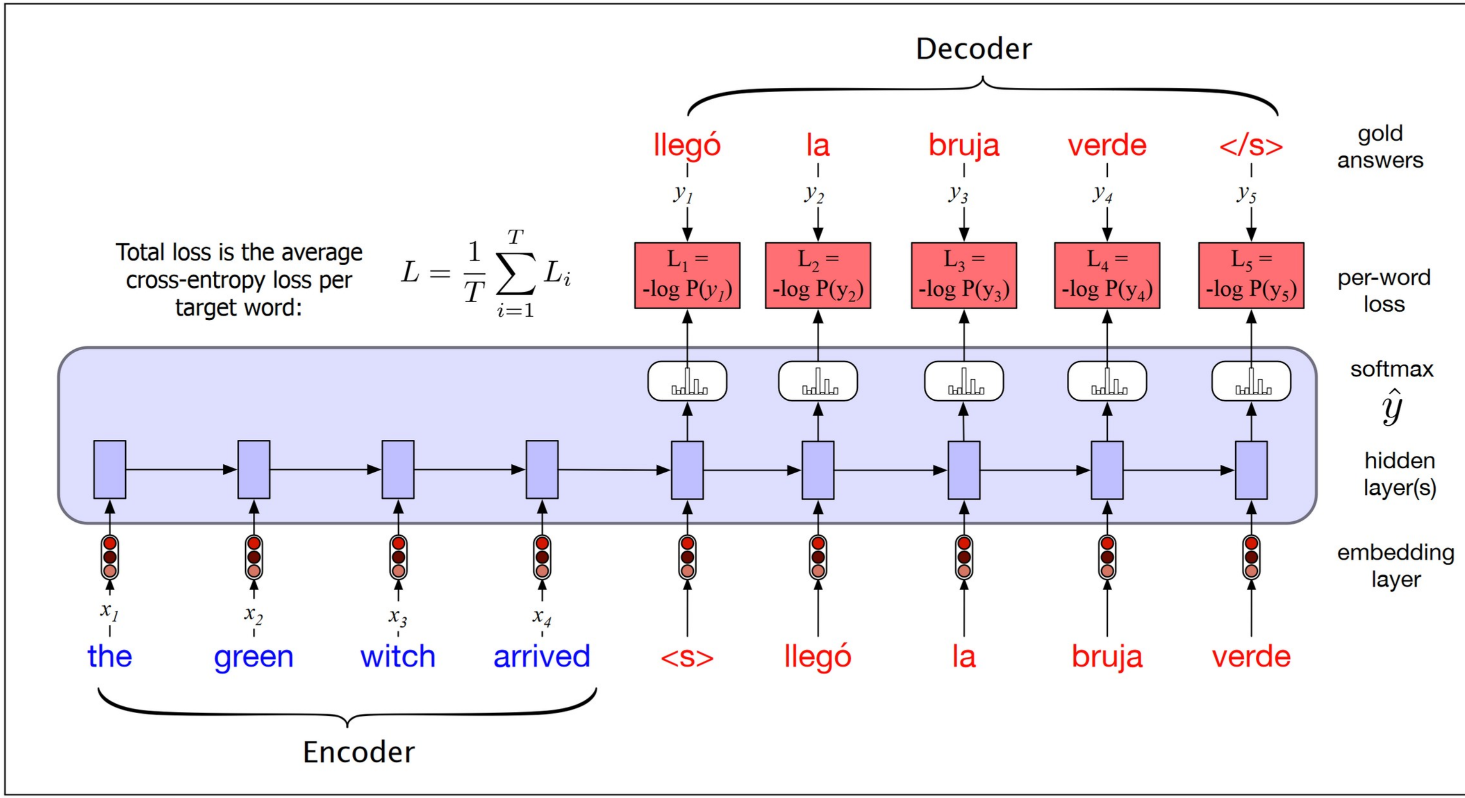


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution over $\hat{y}$ in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 19

Stacking RNN Layers

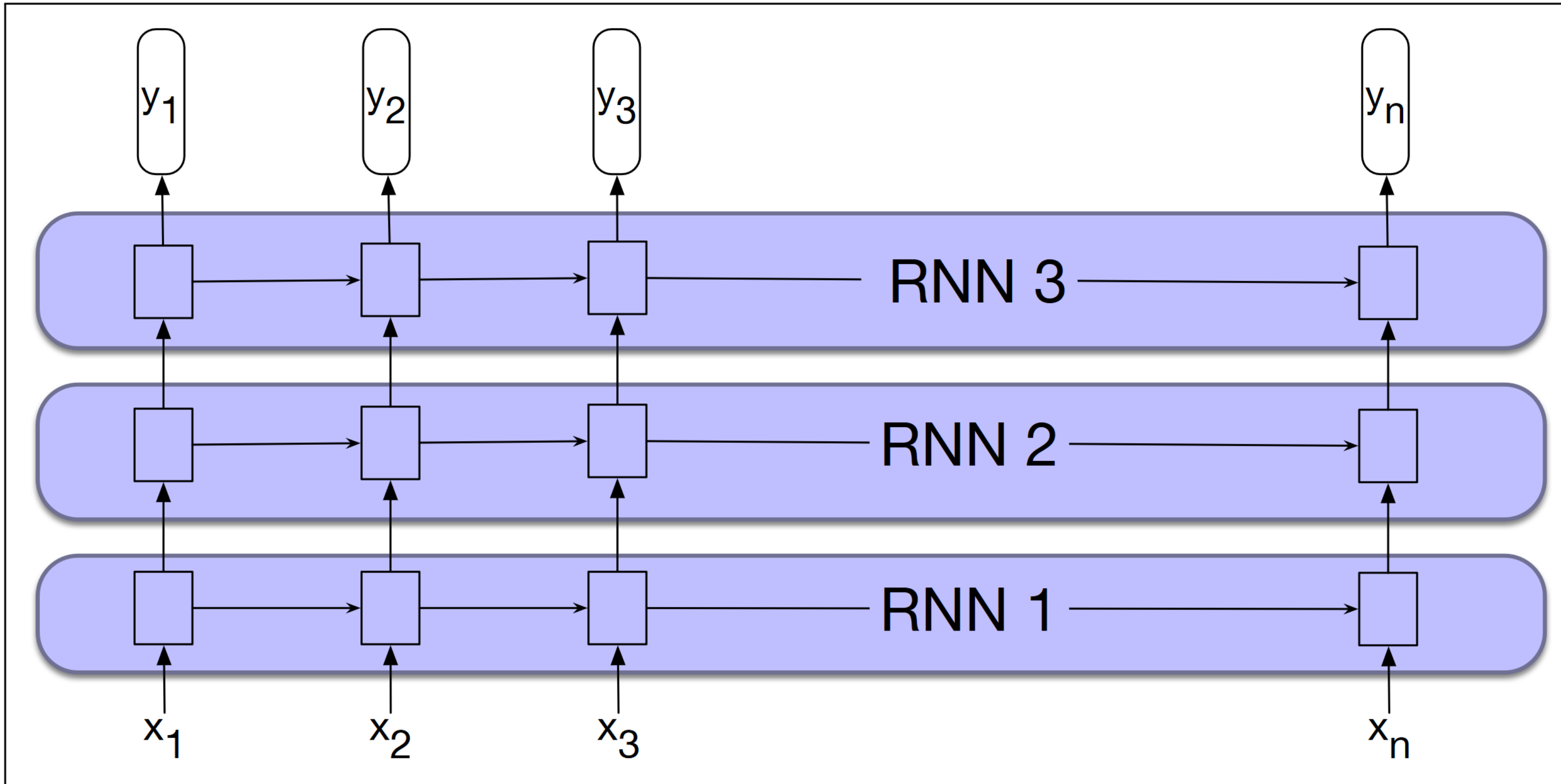


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

Bidirectional RNN

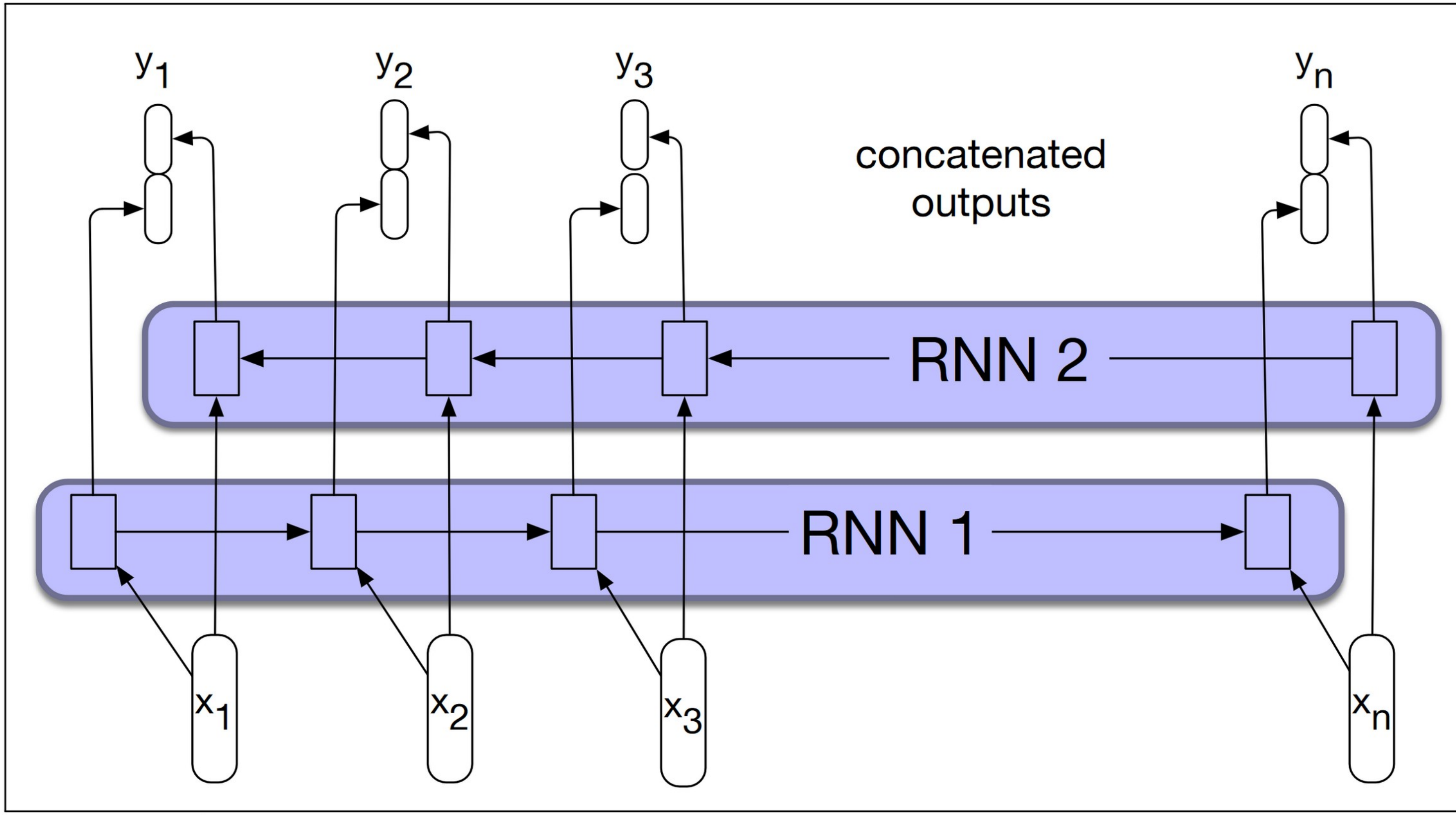


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing", Chapter 9, page 11

Bidirectional RNN

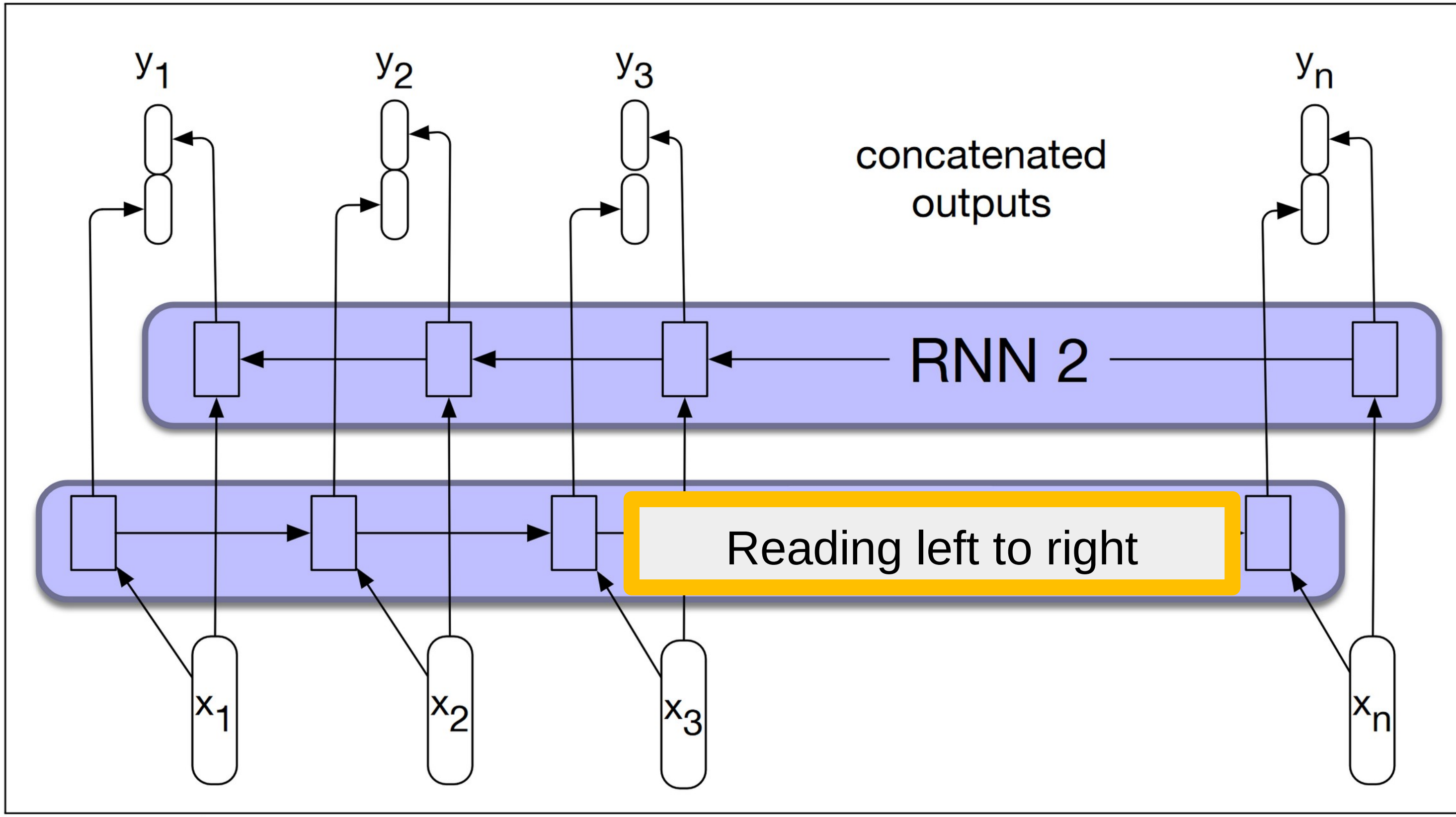


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

Bidirectional RNN

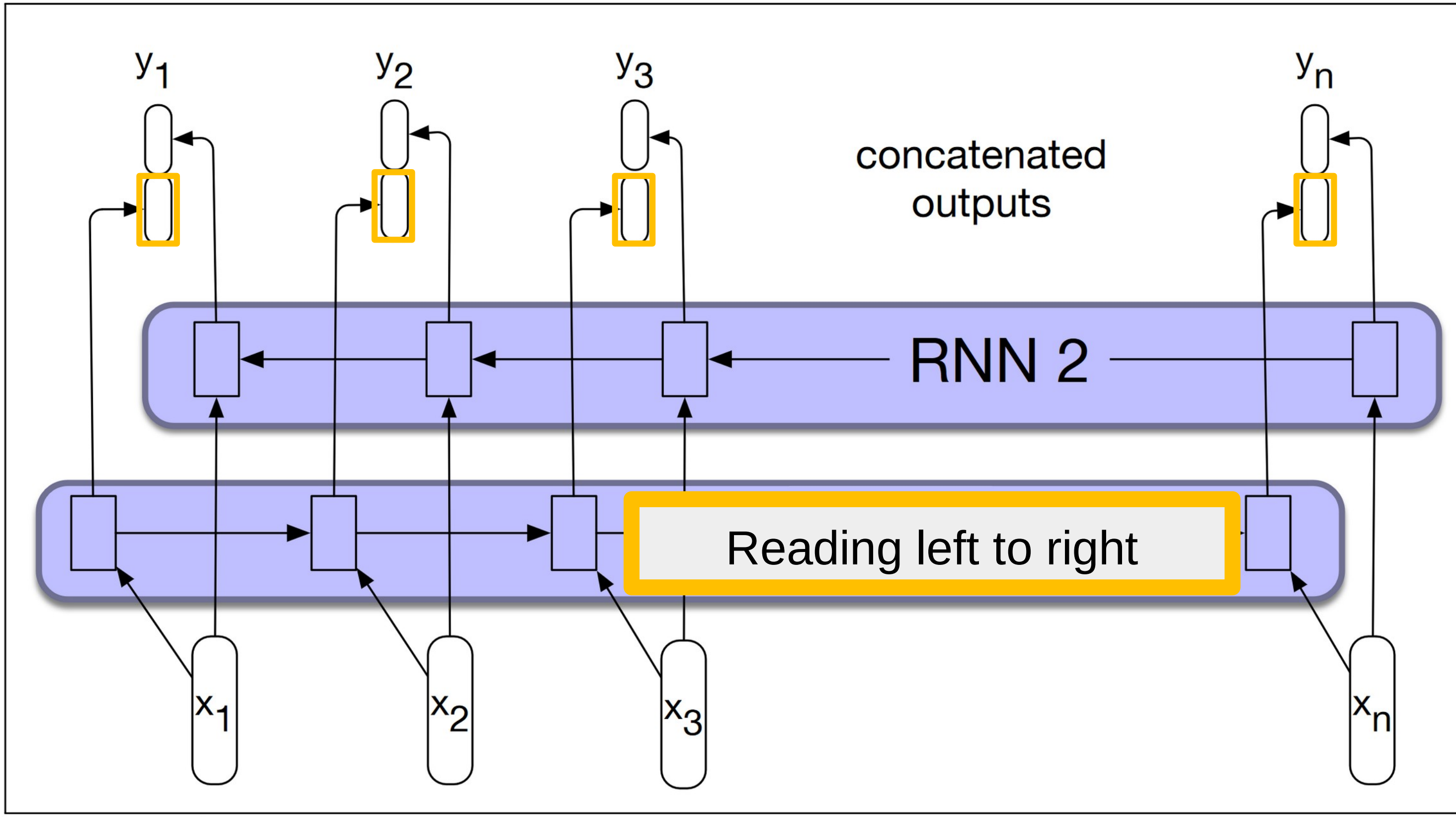


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

Bidirectional RNN

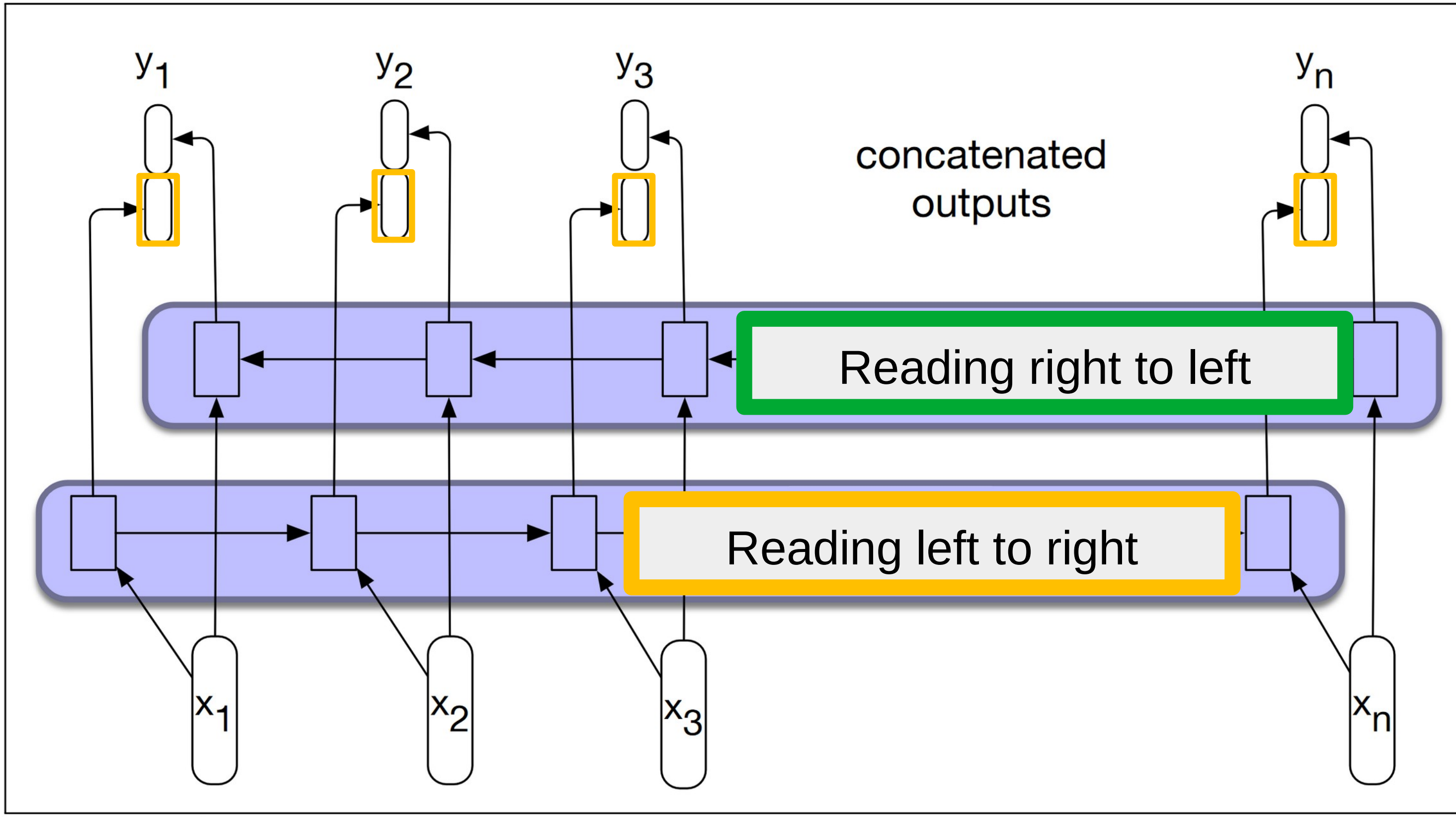


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

Bidirectional RNN

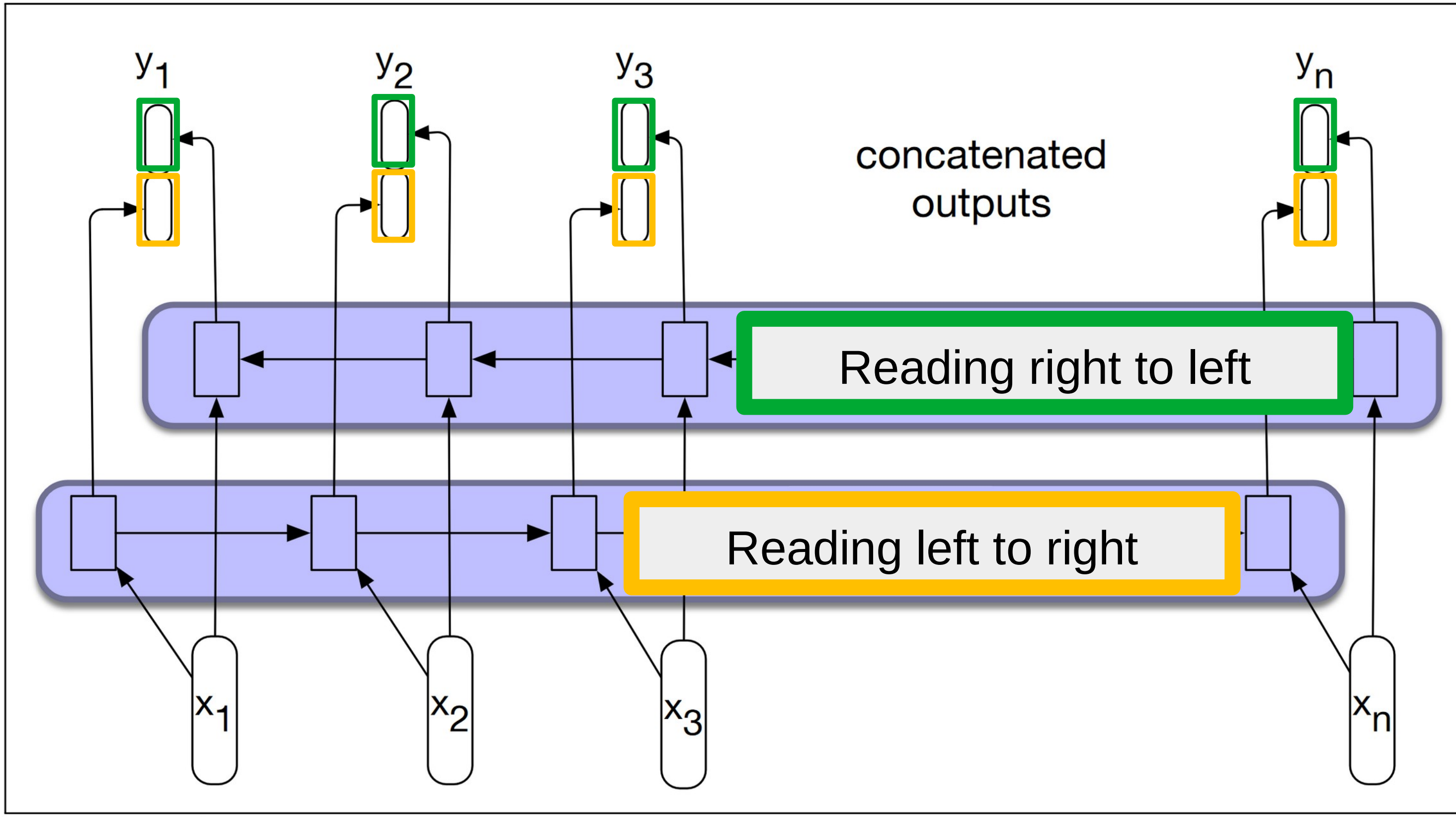


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11

Bidirectional RNN

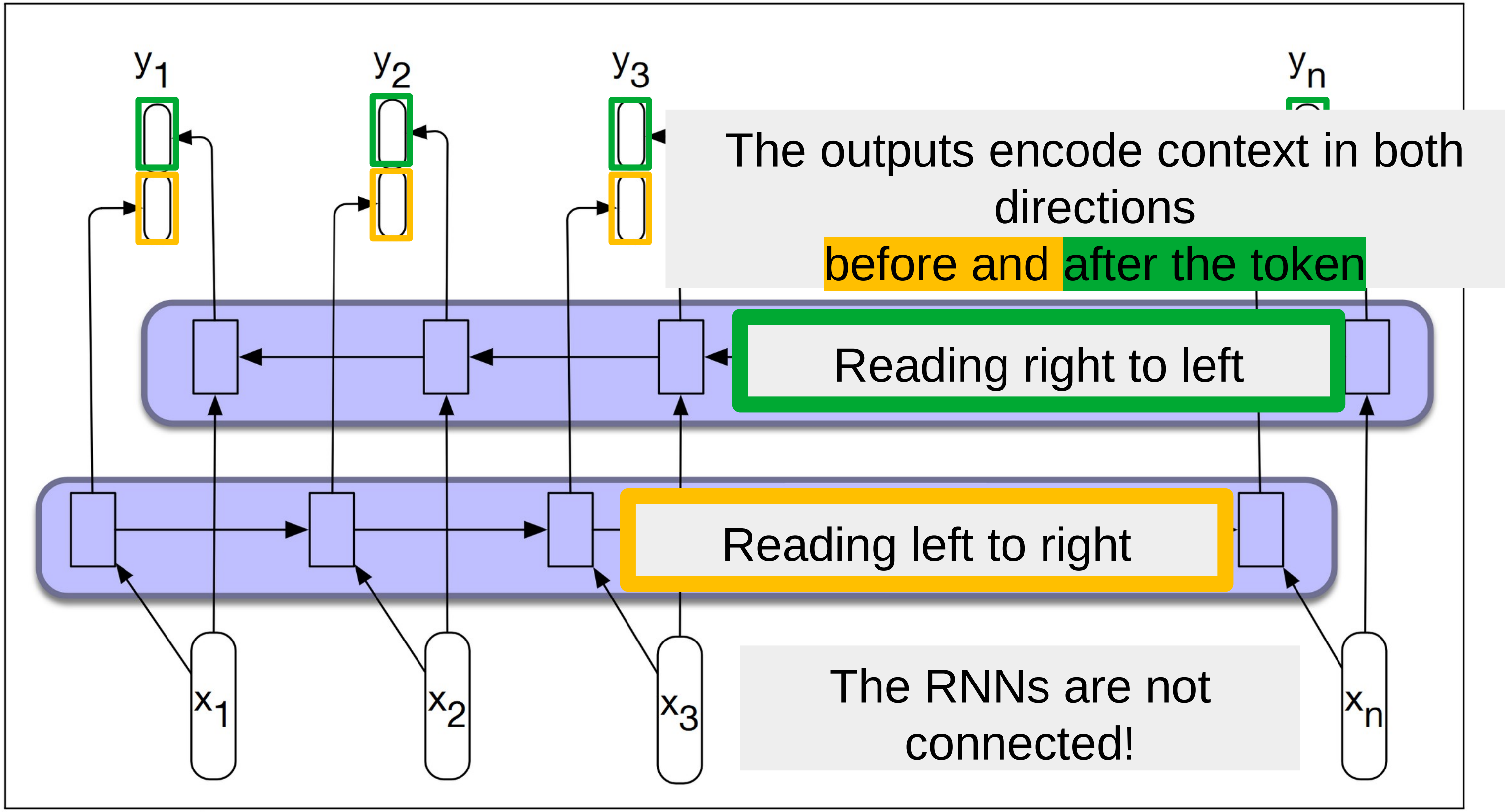


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

source: Jurafsky & Martin „Speech and Language Processing“, Chapter 9, page 11