Lecture 11
Recurrent Neural Networks for NLP

CS 6320

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Outline

- Sequential Data in NLP
- Recurrent Neural Networks
- Simple Recurrent Neural Networks
- Training
- Simple Recurrent Neural Networks Problems
- Gated Architectures
- LSTM
- Bidirectional RNN

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Sequential Data in NLP

- Language is constructed from sequential data.
  - Words are sequences of letters.
  - Sentences are sequences of words.
  - Documents are sequences of sentences.
- Long distance dependencies in language.
  - Agreement in number, gender, -

    He started to tell the rest of the students his experience.
  - Selectional preference -

    The engine noise on the left side of the car came from the water pump.
  - Co-reference across sentences.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Recurrent Neural Networks

- RNN allow processing of arbitrarily sized sequential inputs, while taking into consideration structural properties of the inputs.
- The conditioning of the next word is on the entire sentence history (not like in Markov assumption).
- We will look at
  - Simple Recurrent Networks
  - Bidirectional RNN
  - LSTM (Long Short-Term Memory)

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Simple Recurrent Neural Networks (SRNN)

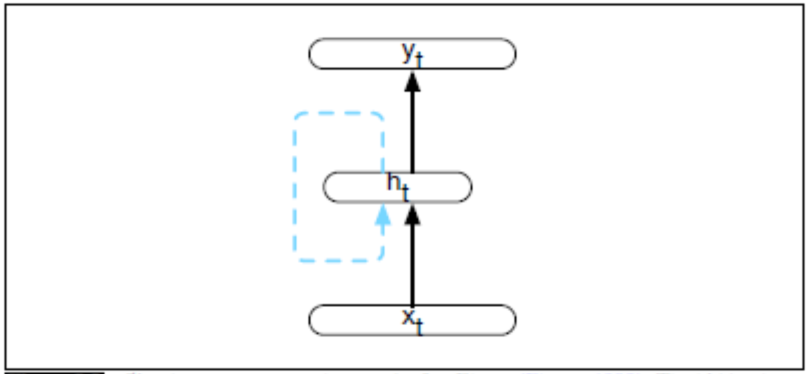


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

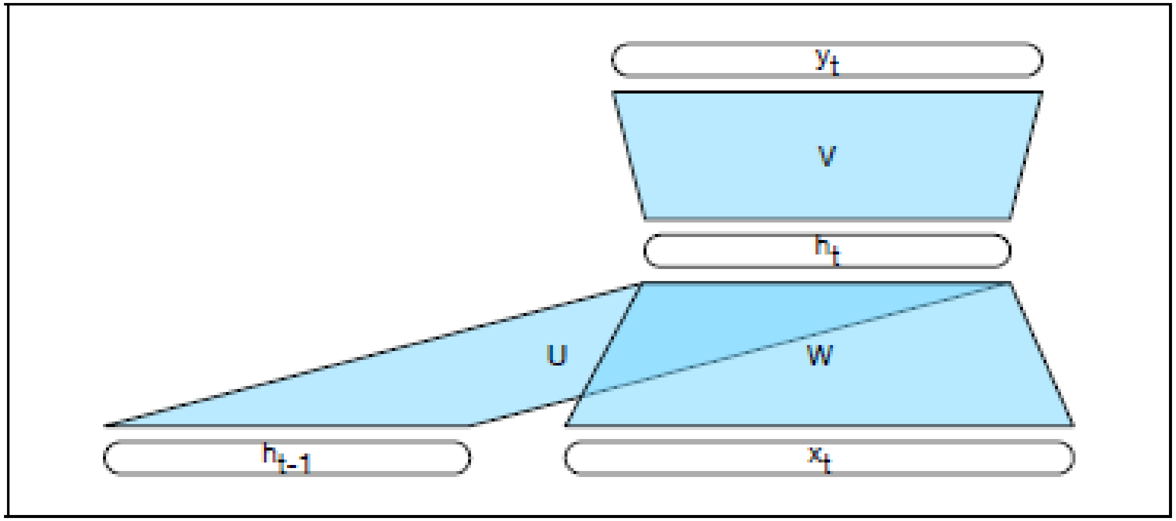


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

SRNN

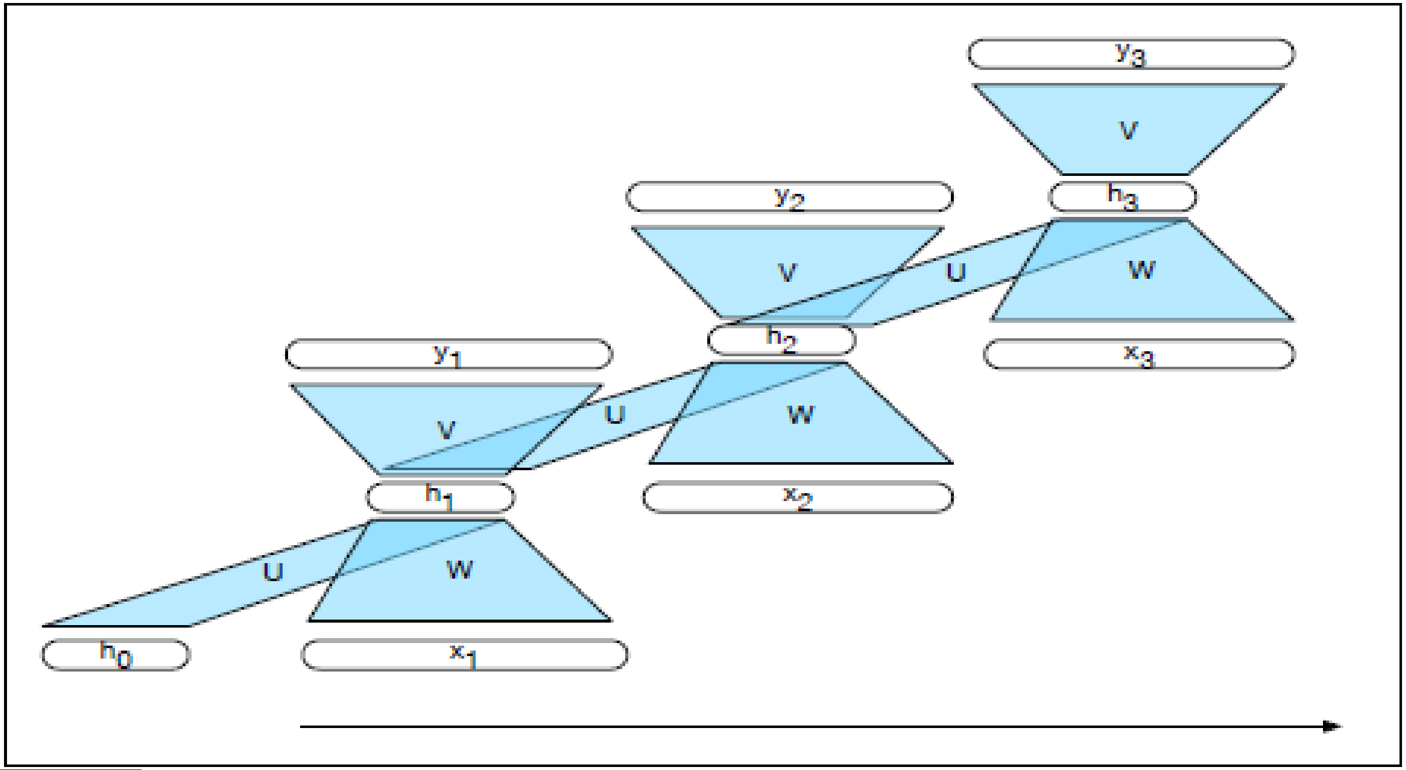


Figure 9.4 A simple recurrent neural network shown unrolled in time. Network layers are copied for each timestep, while the weights U, V and W are shared in common across all timesteps.

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Training

- Use a training set, a loss function and backpropagation.
- Adjust the weights for *W, U, V*.

$$z^{[1]} = Wx$$
$$a^{[1]} = g\left(z^{[1]}\right)$$
$$z^{[2]} = Ua^{[1]}$$
$$a^{[2]} = g\left(z^{[2]}\right)$$
$$y = a^{[2]}$$

- Two complications here:
  - Computation of loss function at time t needs the hidden layer from time t - *1*.
  - The hidden layer at time t influences both the output at time t , and the hidden layer at t+1, thus the output and loss at time t +1.

    Error on h_t impacts current output and the next one.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Training

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial V}$$

Define δ_{out}

$$\delta_{out} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}$$

$$\delta_{out} = L' g'(z)$$

The final gradient we need to update matrix V.

$$\frac{\partial L}{\partial V} = \delta_{out} h_t$$

Error term δ_h is the sum of error term from current output and its error from the next step.

$$\delta_h = g'(z)\, V\, \delta_{out} + \delta_{next}$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Training

- Compute gradients for the weights *U* and *W*.

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{da}\frac{da}{dW}$$

$$\frac{dL}{dU} = \frac{dL}{dz}\frac{dz}{da}\frac{da}{dU}$$

$$\frac{\partial L}{\partial W} = \delta_h x_t$$

$$\frac{\partial L}{\partial U} = \delta_h h_{t-1}$$

- Need to assign proportional blame back to the previous hidden layer h_{t-1}.
- Backpropagate the error δ_h to h_{t-1} proportionally based on the weights of *U*.

$$\delta_{next} = g'(z)U\,\delta_h$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Training

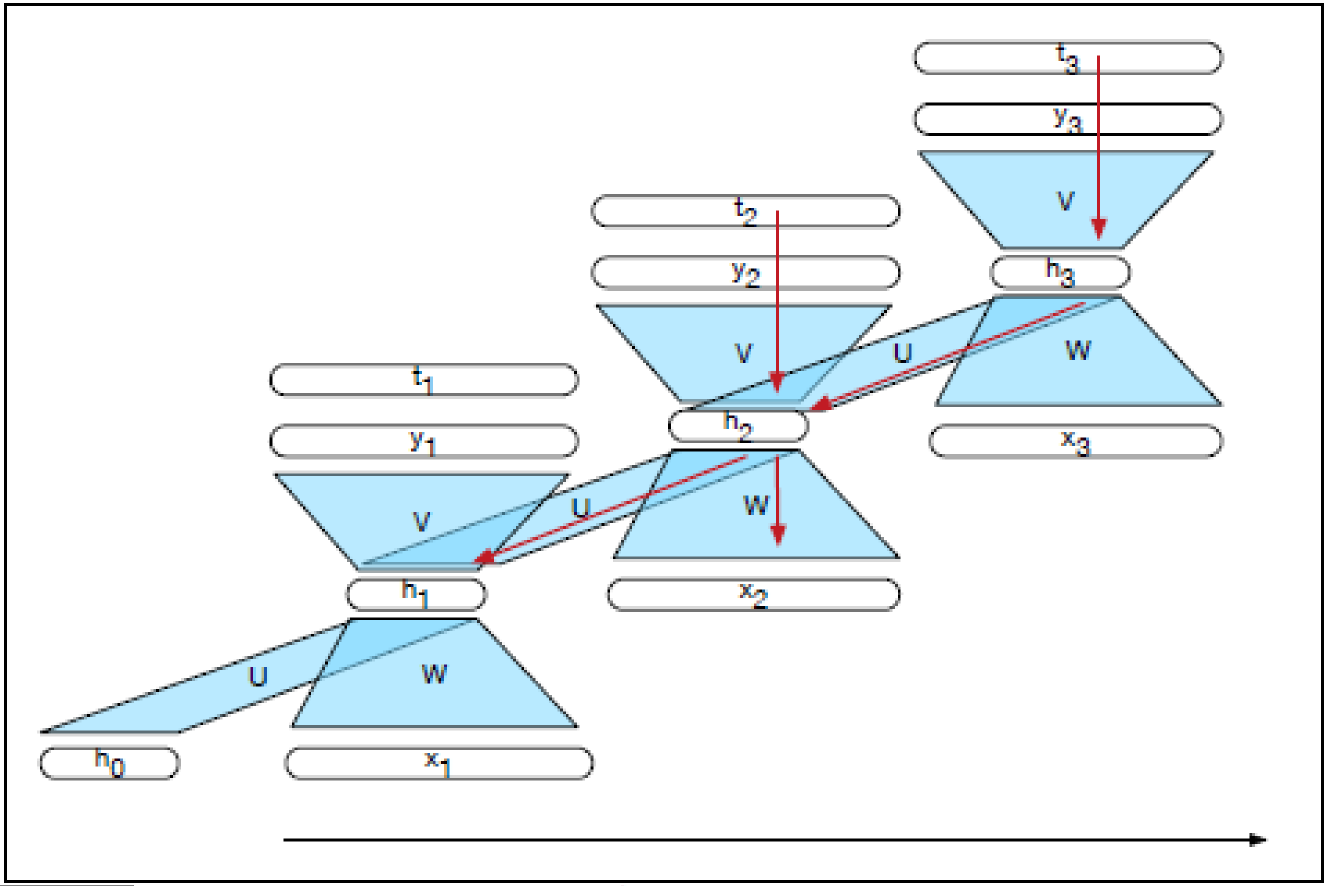


Figure 9.6 The backpropagation of errors in an SRN. The t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the updates for U, V and W at time 2. The two incoming arrows converging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

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Training

- Two-pass algorithm

  - First pass; perform forward inference computing h_t, y_t and a loss at each step in time. Save the value of hidden layer and each step for use at next time step.

  - Second pass; process the sequence in reverse; compute error term gradients saving error term for use in the hidden layer for each step backward.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

SRNN Problems

- SRNN are hard to train due to the vanishing gradient problem.
- Gradients decrease as they get pushed back due to non-linearities.

$$\xrightarrow{h_0} \boxed{RNN} \xrightarrow{h_1} \boxed{RNN} \xrightarrow{h_2} \boxed{RNN} \xrightarrow{h_3} \boxed{RNN} \xrightarrow[L]{}$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ x_1 & x_2 & x_3 \end{array}$$

$$\begin{array}{cccc} \dfrac{dL}{dh_0} & \dfrac{dL}{dh_1} & \dfrac{dL}{dh_2} & \dfrac{dL}{dh_3} \\ tiny & small & medium & large \end{array}$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Gated Architectures

- Consider an abstract model for RNN

  $$s_i = R_{SRNN}(x_i, s_{i-1})$$
  $$y_i = O_{SRNN}(s_i) = s_i$$

- Each application of R function reads the current memory state s_{i-1} operates on them based on x_i and writes the result back into memory as s_i
- Problem is that memory access is not controlled, the entire memory state is read and re-written.
- Idea: Use a gate vector to control access to n-dimensional vectors using Hadamard product

  $$x \odot g \quad \text{with} \qquad g \in \{0, 1\}^n$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Gated Architectures

- This way

$$s' \leftarrow g \odot x + (1 - g) \odot \mathrm{s}$$

- Example

$$\underset{\mathbf{s}'}{\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix}} \leftarrow \underset{\mathbf{g}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}} + \underset{(\mathbf{1}-\mathbf{g})}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \odot \underset{\mathbf{s}}{\begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}}$$

- But gate vectors are not differentiable.
- Solution: use a sigmoid gate

$$\sigma(g) \odot x \quad \text{with} \quad g \in \mathbb{R}^n$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Long Short-Term Memory (LSTM)

- LSTM splits the state vector s_t into two halves:
  - c_t - memory component
  - h_t- hidden state component, or working memory

There are three gates

i- controlling input

f- forget

o- output

Gate values are computed based on linear combination of input x_t and previous h_{t-1} passed through a sigmoid activation function.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

LSTM

- Mathematically, the LSTM architecture is defined as

$$g_t = \tanh\left(U_g h_{t-1} + W_g x_t\right)$$
$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$
$$f_t = \sigma\left(U_f h_{t-1} + W_f x_t\right)$$
$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

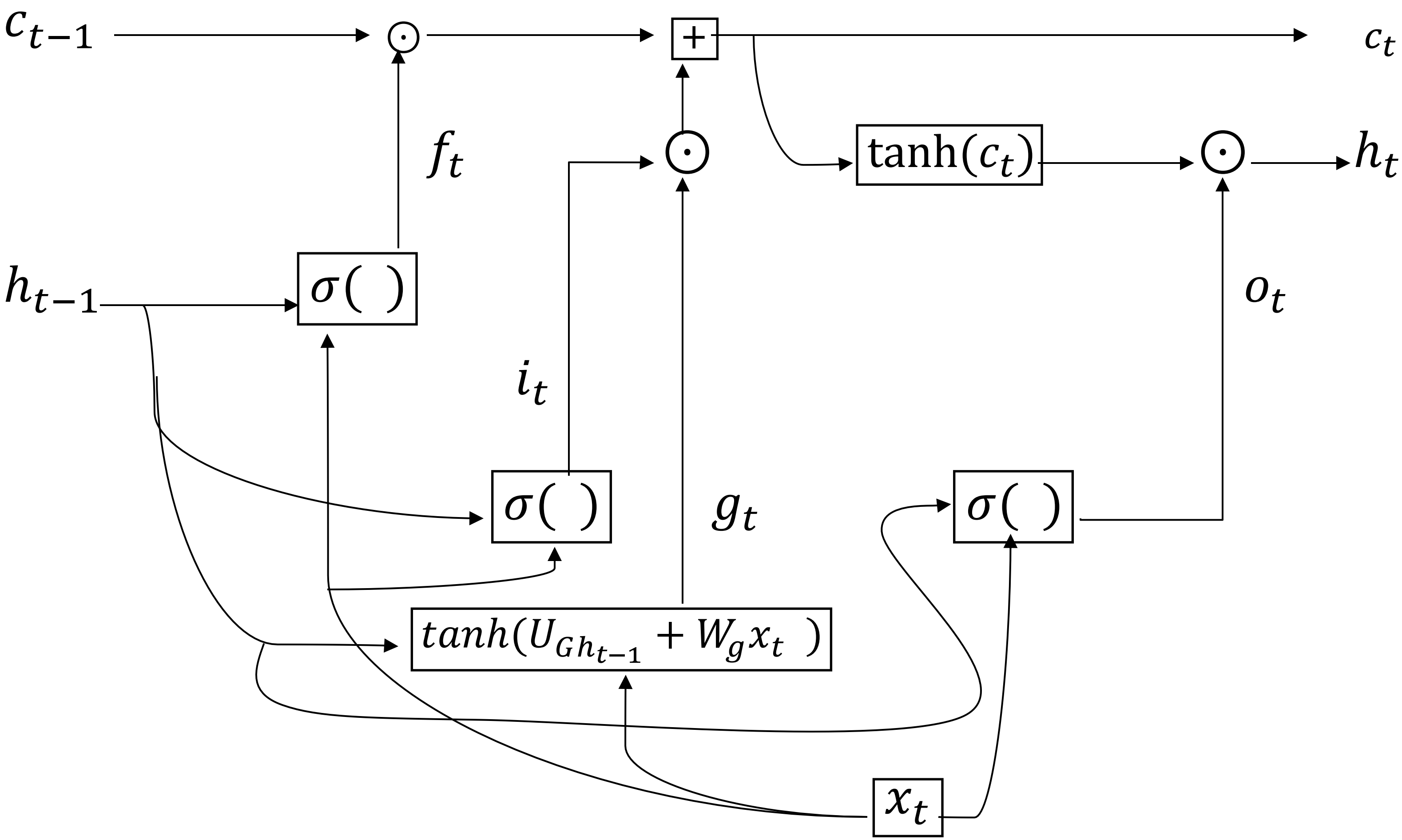
LSTM

- An update candidate g_t is computed and passed through a *tanh* activation function.
- Forget gate controls how much of the previous memory to keep $f \odot c_{t-1}$
- Input gate controls how much of the proposed update to keep $i \odot g$
- Hidden value h_t is determined based on the content of memory c_t passed through a *tanh.*
- Output gate controls h_t

- This architecture allows for gradients related to the memory part c_t to stay high across long sequences.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

LSTM Cell Architecture



© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Bidirectional RNN

- Consider an input sequence

  $x_1, x_2 \dots x_t \dots x_n$

  We form two sequences

  Forward sequence $x_1, x_2, \dots, x_i$

  Backward sequence $x_n, x_{n-1}, \dots, x_i$

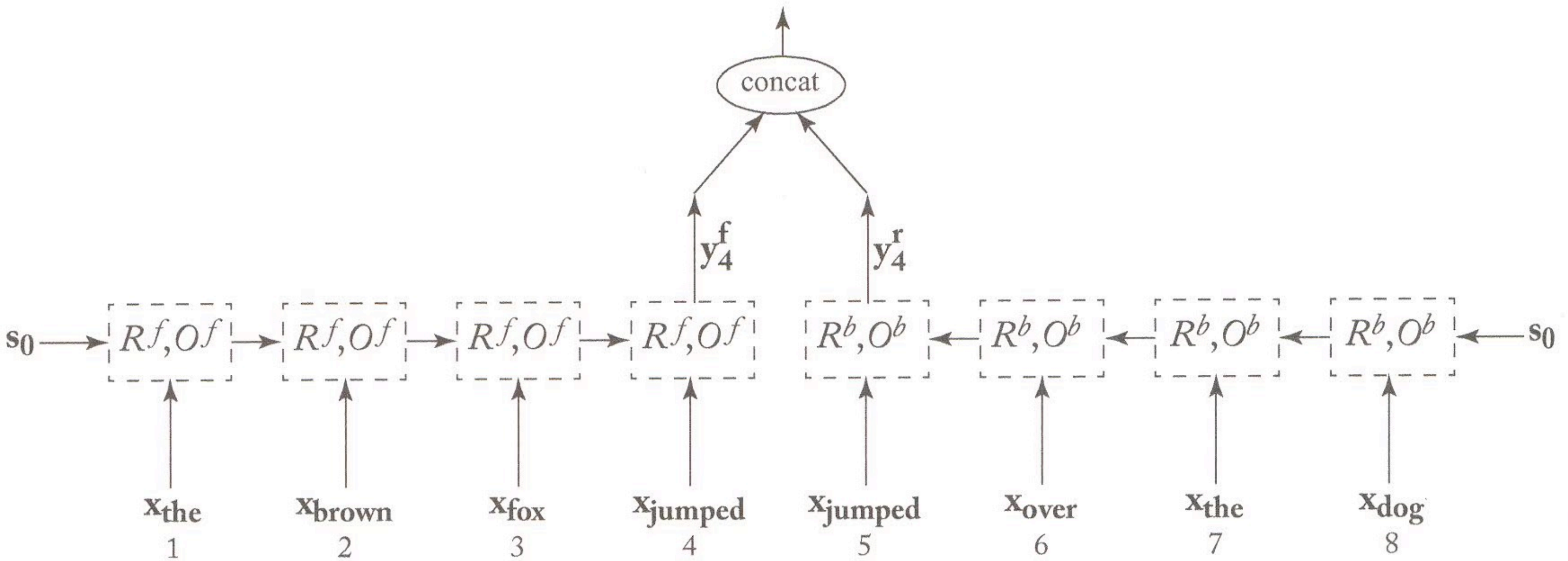
The forward and backward states are generated by two different $RNNs$.

$$biRNN(x_{1:n}, i) = y_i = [RNN^f(x_{1:i}); RNN^b(x_{n:i})]$$

- Output y_i can be used for prediction and is conditioned on previous inputs as well as the inputs that follow.

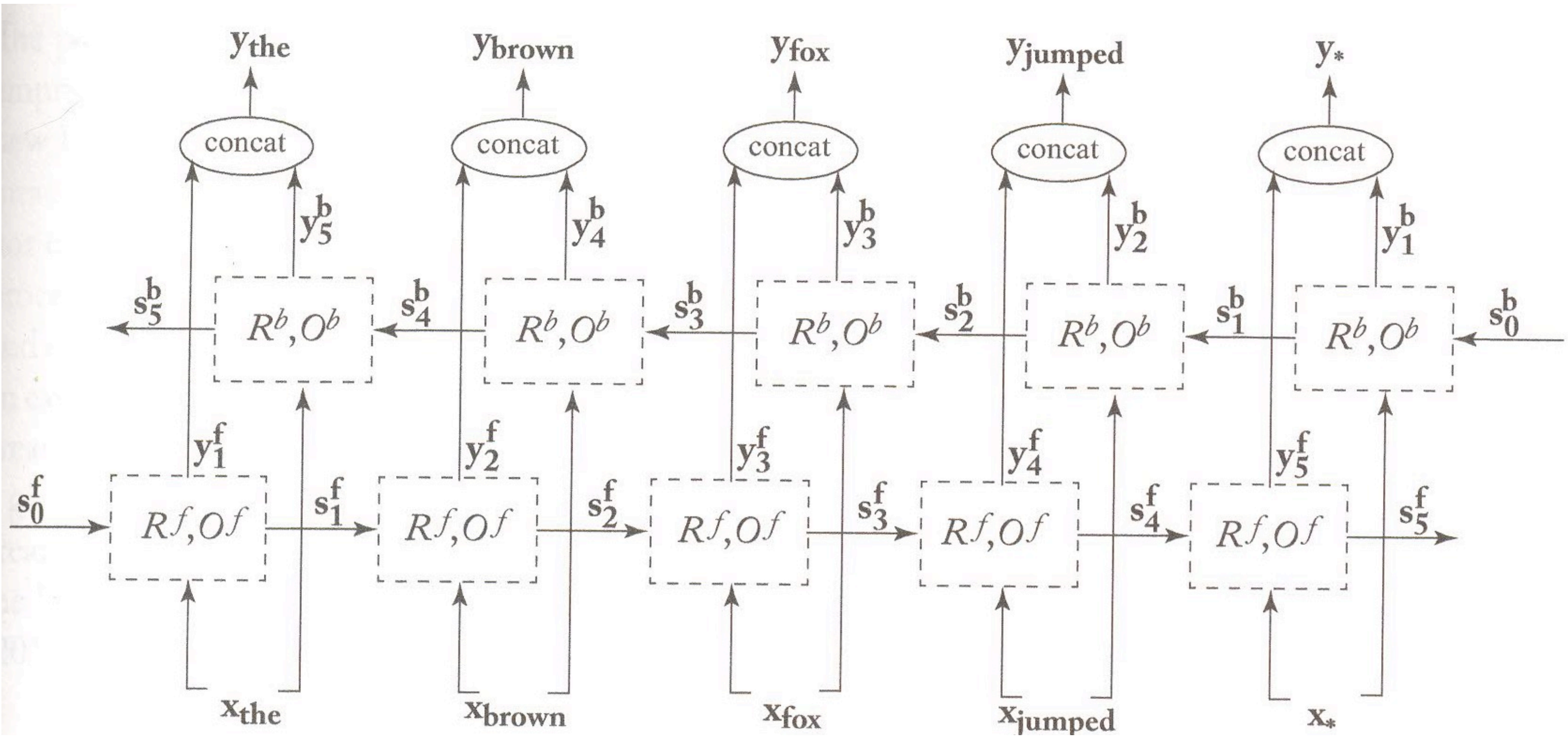
© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Bidirectional RNN



© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas

Bidirectional RNN



- Bidirectional RNN sequence of vectors $y_{1:n}$ are:

$$biRNN(x_{1:n}) = y_{1:n} = biRNN(x_{1:n}, 1), \ldots, biRNN(x_{1:n}, n)$$

Run the forward and backward RNN, then concatenate the relevant outputs.

© 2020 Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas