CS 4248
Natural Language Processing

Professor NG Hwee Tou

Department of Computer Science
School of Computing
National University of Singapore
nght@comp.nus.edu.sg

Materials

- NNM4NLP Chapter 14, 15, 16

Characteristics of Natural Language

- Sequence data
  - Word (a sequence of characters)
  - Sentence (a sequence of words)
  - Document (a sequence of sentences)
- Convolution neural networks
  - Modeling of word order restricted to mostly local patterns (ngrams)

Recurrent Neural Networks (RNN)

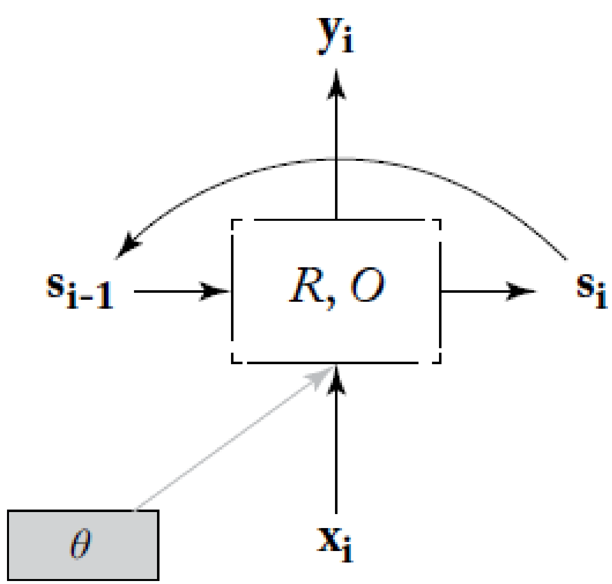
- Jeffrey L. Elman, Finding structure in time. *Cognitive Science*, March 1990
- Capture subtle patterns and regularities in sequences
- Model non-Markovian dependencies
- Consider infinite window and zoom in on informative sequential patterns in the window

Recurrent Neural Networks

- Represent an arbitrarily sized sequence of inputs in a fixed size vector
- Output can be fed into a larger network downstream

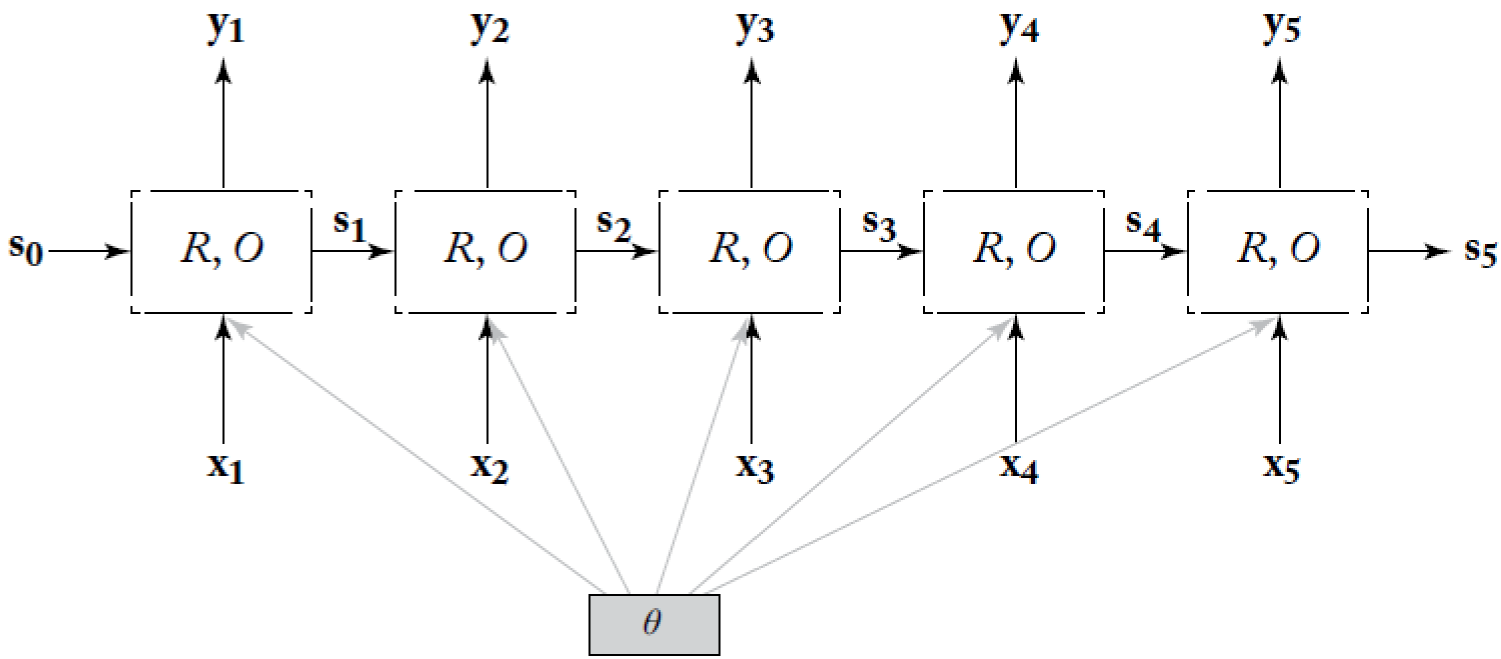
Graphical Representation of an RNN

Recursive view:



Graphical Representation of an RNN

Unrolled view:



- Unrolled view makes clear that RNN is a deep neural network with many layers
- The same parameters θ are shared across all time steps

The RNN Abstraction

$$\text{RNN}^*(\boldsymbol{x_{1:n}}; \boldsymbol{s_0}) = \boldsymbol{y_{1:n}}$$
$$\boldsymbol{s_i} = R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i})$$
$$\boldsymbol{y_i} = O(\boldsymbol{s_i})$$
$$\boldsymbol{x_i} \in \mathbb{R}^{d_{in}},\ \boldsymbol{y_i} \in \mathbb{R}^{d_{out}}, \boldsymbol{s_i} \in \mathbb{R}^{f(d_{out})}$$

- The functions R and O are the same across the sequence positions
- The last $\boldsymbol{y_n}$ encodes the entire input sequence and can be used for further prediction

The RNN Abstraction

$$
\begin{aligned}
\boldsymbol{s_4} &= R(\boldsymbol{s_3}, \boldsymbol{x_4}) \\
&= R(\overbrace{R(\boldsymbol{s_2}, \boldsymbol{x_3})}^{\boldsymbol{s_3}}, \boldsymbol{x_4}) \\
&= R(R(\overbrace{R(\boldsymbol{s_1}, \boldsymbol{x_2})}^{\boldsymbol{s_2}}, \boldsymbol{x_3}), \boldsymbol{x_4}) \\
&= R(R(R(\overbrace{R(\boldsymbol{s_0}, \boldsymbol{x_1})}^{\boldsymbol{s_1}}, \boldsymbol{x_2}), \boldsymbol{x_3}), \boldsymbol{x_4})
\end{aligned}
$$

RNN

- Sensitive to the order of words in a sequence

$$\boldsymbol{s_i} = R_{\mathrm{RNN}}(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g([\boldsymbol{s_{i-1}}; \boldsymbol{x_i}]\boldsymbol{W} + \boldsymbol{b})$$
$$\boldsymbol{y_i} = O_{\mathrm{RNN}}(\boldsymbol{s_i}) = \boldsymbol{s_i}$$
$$\boldsymbol{x_i} \in \mathbb{R}^{d_x},\ \boldsymbol{s_i}, \boldsymbol{y_i} \in \mathbb{R}^{d_s}, \boldsymbol{b} \in \mathbb{R}^{d_s}$$
$$\boldsymbol{W} \in \mathbb{R}^{(d_s + d_x) \times d_s}$$

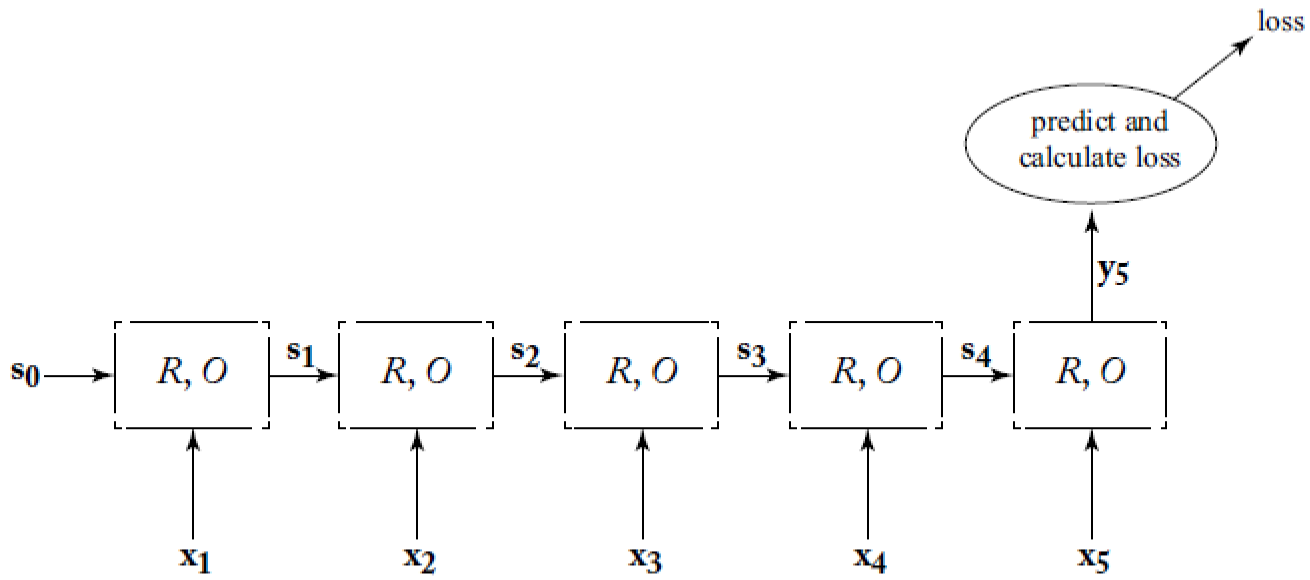
The RNN Abstraction

- E.g., a model for predicting the conditional probability of an event e given the sequence $\boldsymbol{x_{1:n}}$:
- $p(e = j|\boldsymbol{x_{1:n}}) = \mathrm{softmax}(\boldsymbol{y_n} \cdot \boldsymbol{W} + \boldsymbol{b})_{[j]}$
- RNN serves as a trainable component in a larger network
- RNN allows conditioning on the entire history $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ without making the Markov assumption

RNN Training

- Backpropagation through time (BPTT)
  - Create the unrolled computation graph for a given input sequence
  - Add a loss node
  - Use backpropagation to compute the gradients w.r.t. the loss function

RNN as Acceptor



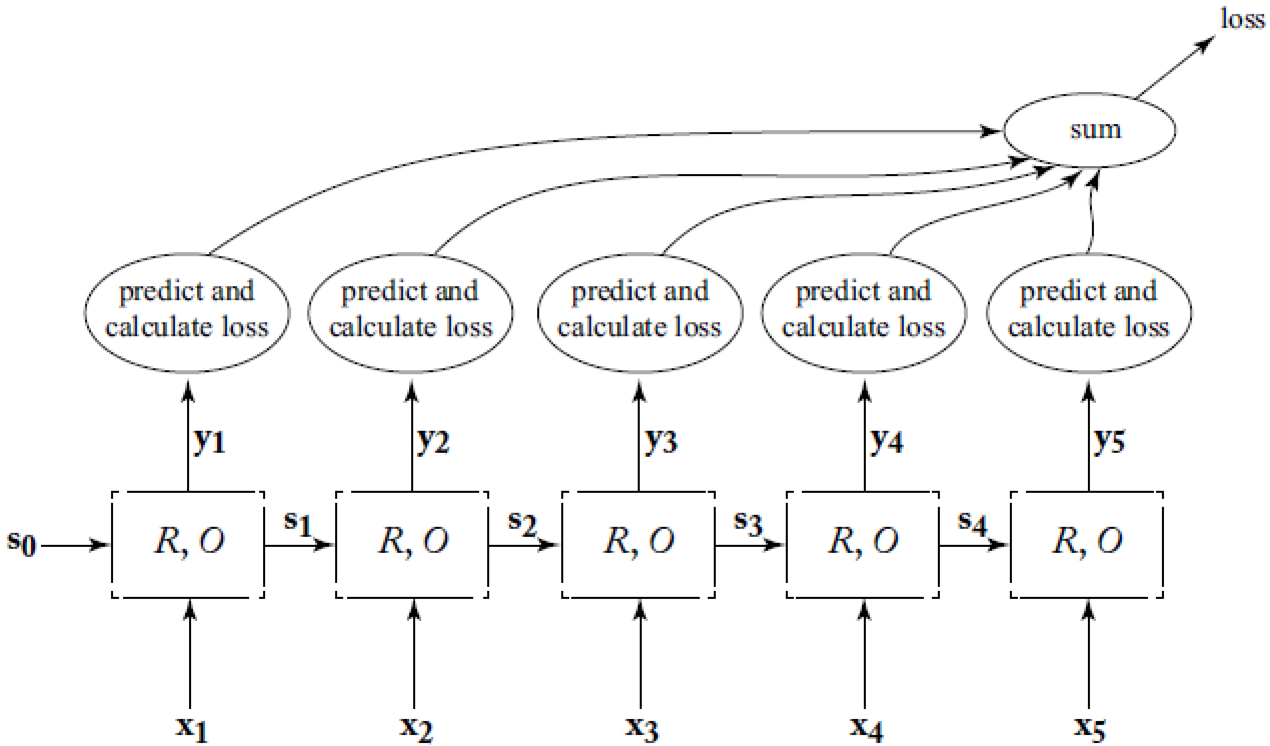
RNN as Acceptor

- Based on the final state $\boldsymbol{y_n}$, make a prediction
- E.g., read the words of a sentence s and predict whether s conveys a positive or negative sentiment

RNN as Encoder

- $\boldsymbol{y_n}$ is treated as an encoding of $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$, and used together with other features
- E.g., an extractive document summarizer first uses an RNN to encode an input document with $\boldsymbol{y_n}$, then uses $\boldsymbol{y_n}$ together with other features to select the sentences to include in the summary

RNN as Transducer

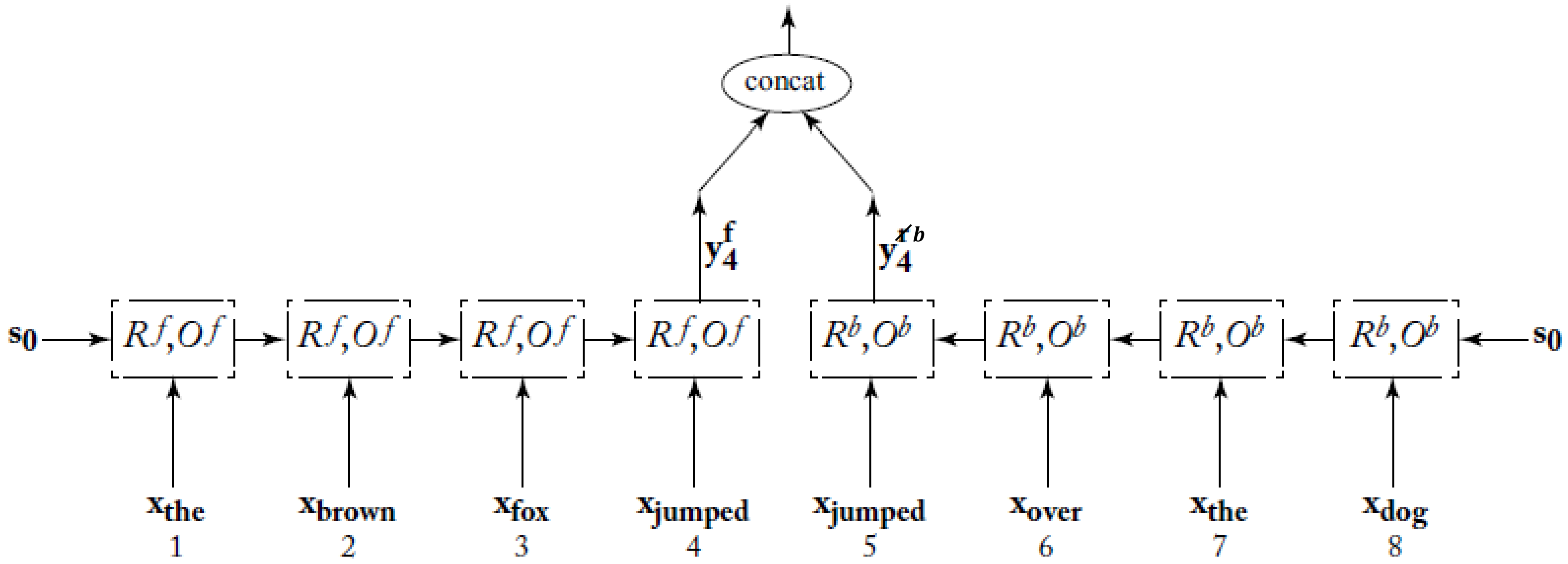


RNN as Transducer

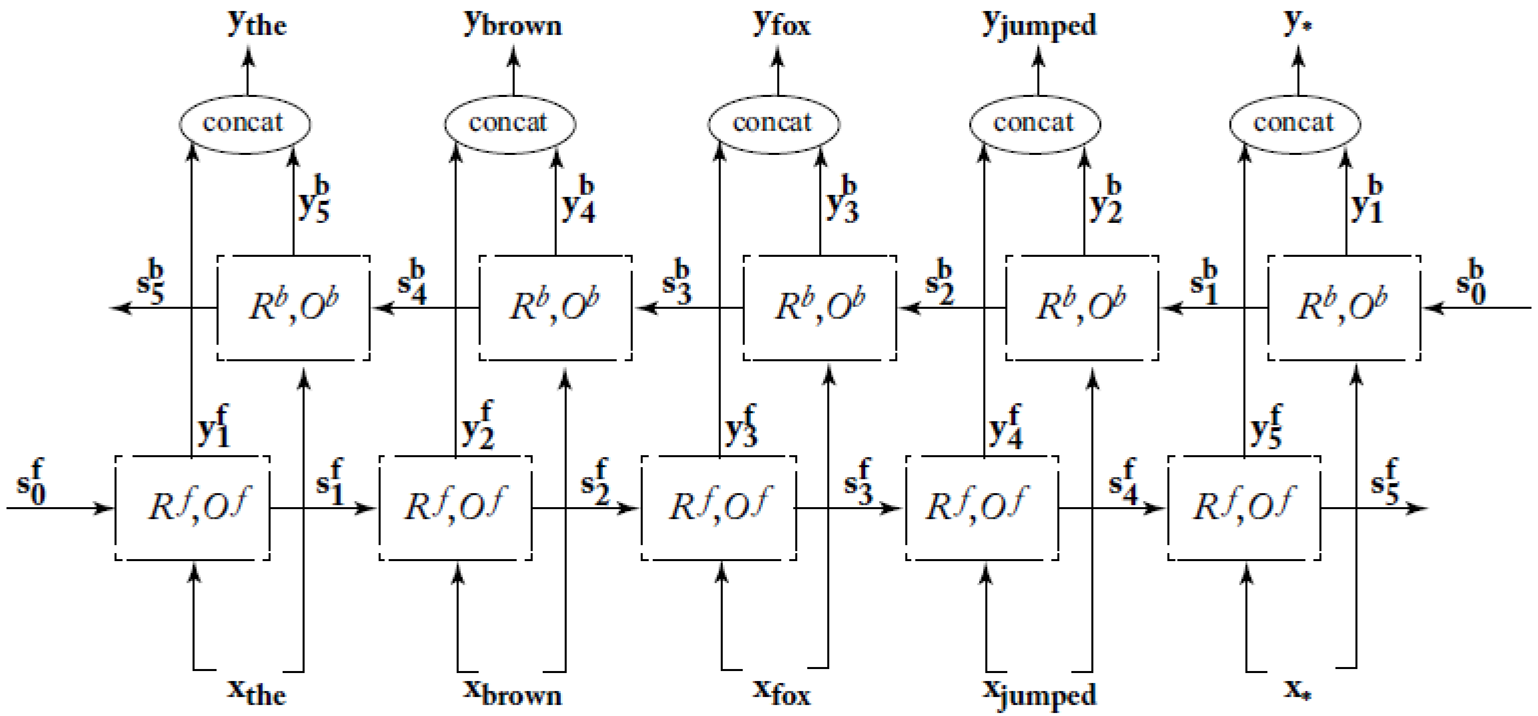
- Produce an output $\hat{\boldsymbol{t}}_{\boldsymbol{i}}$ after each input $\boldsymbol{x}_{\boldsymbol{i}}$ is read
- Local loss $L_{\text{local}}(\hat{\boldsymbol{t}}_{\boldsymbol{i}}, \boldsymbol{t}_{\boldsymbol{i}})$
- $L\left(\hat{\boldsymbol{t}}_{\boldsymbol{1};\boldsymbol{n}}, \boldsymbol{t}_{\boldsymbol{1}:\boldsymbol{n}}\right) = \sum_{i=1}^{n} L_{\text{local}}(\hat{\boldsymbol{t}}_{\boldsymbol{i}}, \boldsymbol{t}_{\boldsymbol{i}})$
- E.g., a sequence tagger that predicts the tag of each word
- E.g., language modeling predicts the next word based on $\boldsymbol{x}_{\boldsymbol{1}}, \dots, \boldsymbol{x}_{\boldsymbol{i}}$

Bidirectional RNN

- Bidirectional RNN (biRNN): use both the previous and following words (context)



Bidirectional RNN



Bidirectional RNN

- Input sequence: $\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}$
- 2 separate states for each position i: $\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{f}}$ and $\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{b}}$
- $\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{f}}$: based on $\boldsymbol{x}_{\boldsymbol{1}}, \boldsymbol{x}_{\boldsymbol{2}}, \ldots, \boldsymbol{x}_{\boldsymbol{i}}$
- $\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{b}}$: based on $\boldsymbol{x}_{\boldsymbol{n}}, \boldsymbol{x}_{\boldsymbol{n-1}}, \ldots, \boldsymbol{x}_{\boldsymbol{i}}$

Output vector at position i:

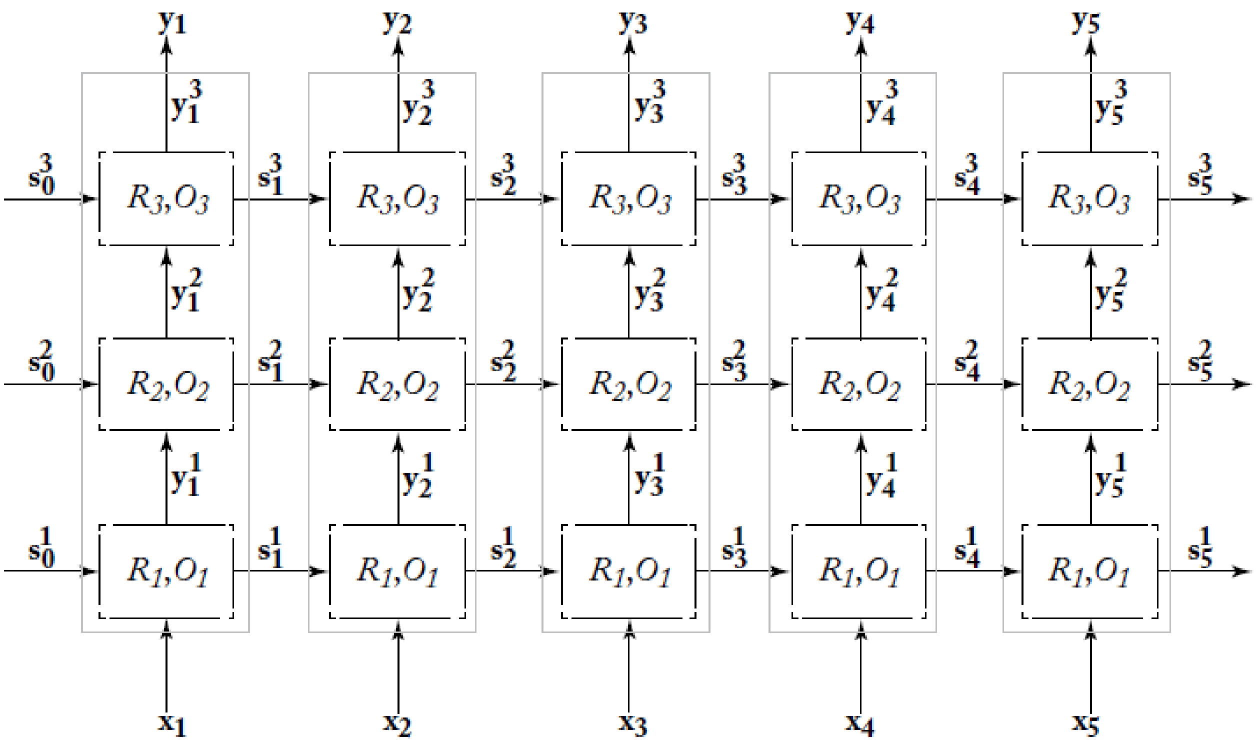
- $\mathrm{biRNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}, i) = \boldsymbol{y}_{\boldsymbol{i}} = \left[\mathrm{RNN}^f(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{i}}); \mathrm{RNN}^b(\boldsymbol{x}_{\boldsymbol{n}:\boldsymbol{i}})\right] = \left[\boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{f}}; \boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{b}}\right] = \left[O^f\left(\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{f}}\right); O^b\left(\boldsymbol{s}_{\boldsymbol{i}}^{\boldsymbol{b}}\right)\right]$

Bidirectional RNN

- $\mathrm{biRNN}^*(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}) = \boldsymbol{y}_{\boldsymbol{1}:\boldsymbol{n}} = \mathrm{biRNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}, 1), \ldots, \mathrm{biRNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}, n)$
- biRNN is very effective for sequence tagging tasks
- biRNN is used as a trainable feature extractor

Multi-Layer (Stacked) RNN

Deep RNN



Multi-Layer (Stacked) RNN

- k RNNs, the jth RNN has states $\boldsymbol{s}_{\boldsymbol{1:n}}^{\boldsymbol{j}}$ and outputs $\boldsymbol{y}_{\boldsymbol{1:n}}^{\boldsymbol{j}}$
- Input for the first RNN: $\boldsymbol{x}_{\boldsymbol{1:n}}$
- Input for the jth RNN ($j \geq 2$) = Output of the RNN below it, $\boldsymbol{y}_{\boldsymbol{1:n}}^{\boldsymbol{j-1}}$
- Output of the entire deep RNN = Output of the last RNN $\boldsymbol{y}_{\boldsymbol{1:n}}^{\boldsymbol{k}}$
- Deep RNNs outperform shallower ones on some tasks (e.g., neural machine translation)

CBOW

- CBOW can be considered as an RNN without taking into account order of words

$$\boldsymbol{s_i} = R_{\text{CBOW}}(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i}$$
$$\boldsymbol{y_i} = O_{\text{CBOW}}(\boldsymbol{s_i}) = \boldsymbol{s_i}$$
$$\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{s_i} \in \mathbb{R}^{d_s}$$

- It follows that $\boldsymbol{s_n} = \sum_{i=1}^{n} \boldsymbol{x_i}$ (assume $\boldsymbol{s_0} = \boldsymbol{0}$)

Modeling with RNNs

- RNN as feature extractor
- RNN as acceptor

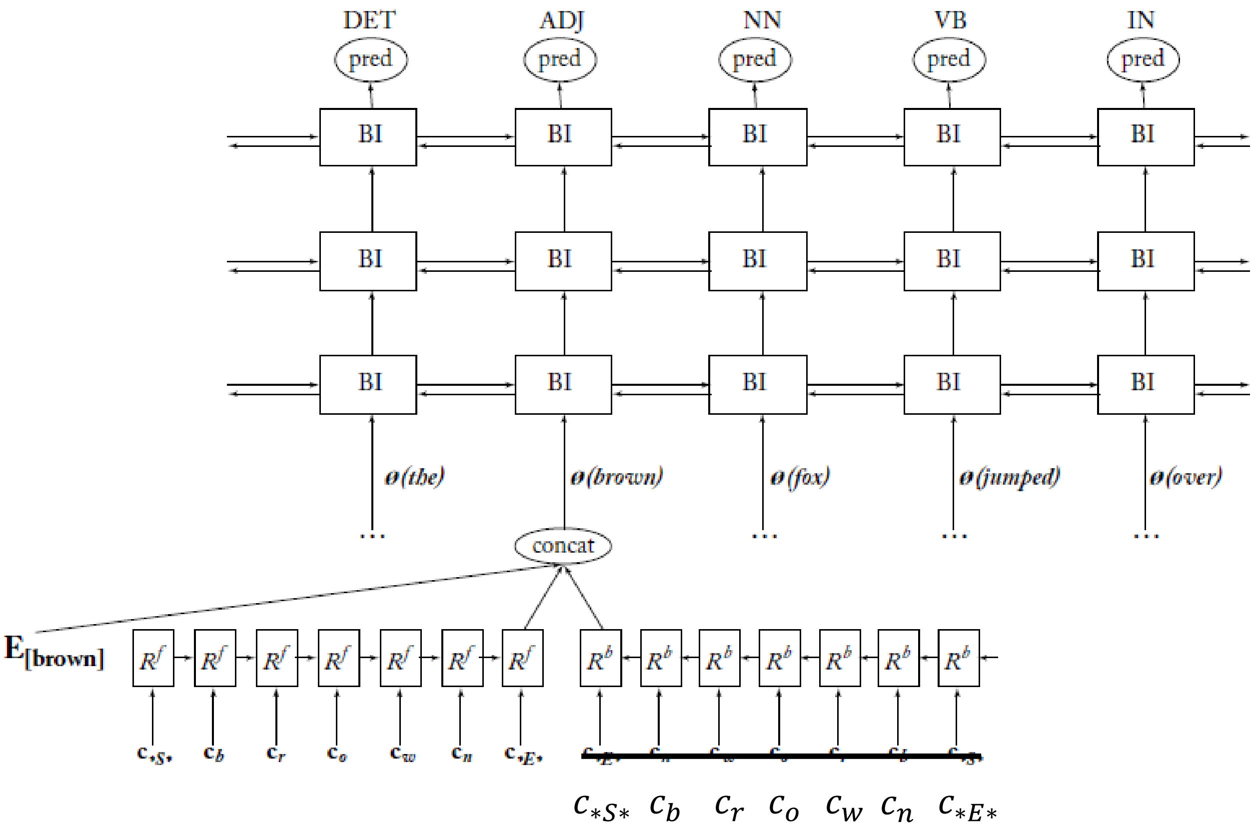
RNN as Feature Extractor

- RNN is used as a trainable feature extractor, to replace parts of the more traditional feature extractor
- E.g., POS tagging

POS Tagging

- A sequence tagging task: assign a tag to each of the n input words
- biRNN

POS Tagging



DET
ADJ
NN
VB
IN
pred
BI
ø(the)
ø(brown)
ø(fox)
ø(jumped)
ø(over)
concat
E[brown]
R^f
R^b
c_{*S*} c_b c_r c_o c_w c_n c_{*E*}
c_{*S*} c_b c_r c_o c_w c_n c_{*E*}

POS Tagging

- s: input sentence
- $w_i = c_1, \ldots, c_l$
- $\boldsymbol{c_i}$: character embedding vector for c_i
- $\boldsymbol{x_i} = \phi(s, i) = \left[\boldsymbol{E}_{[w_i]}; \mathrm{RNN}^f(\boldsymbol{c_{1:l}}); \mathrm{RNN}^b(\boldsymbol{c_{l:1}})\right]$
- $p(t_i = j | w_1, \ldots, w_n) =$ $\mathrm{softmax}(\mathrm{MLP}(\mathrm{biRNN}(\boldsymbol{x_{1:n}}, i)))_{[j]}$
- Trained with cross-entropy loss function

POS Tagging

- Condition on previous POS tags predicted
- Condition on k previous POS tags predicted using tag embeddings $\boldsymbol{E}_{[t]}$
- $p(t_i = j | w_1, \ldots, w_n, t_{i-1}, \ldots, t_{i-k}) = \mathrm{softmax}(\mathrm{MLP}([\mathrm{biRNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}, i); \boldsymbol{E}_{[t_{i-1}]}; \ldots; \boldsymbol{E}_{[t_{i-k}]}]))_{[j]}$
- Condition on the entire sequence of POS tags predicted
- $p(t_i = j | w_1, \ldots, w_n, t_{1:i-1}) = \mathrm{softmax}(\mathrm{MLP}([\mathrm{biRNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}}, i); \mathrm{RNN}^t(\boldsymbol{t}_{\boldsymbol{1}:\boldsymbol{i-1}})]))_{[j]}$

POS Tagging

- Character embeddings take into account suffix, prefix, and orthographic (capitalization, hyphens, digits) cues important for POS tagging
- POS tag prediction is conditioned on the entire input sentence
- biRNN is used as feature extractor, with no manually defined features

RNN as Acceptor

- Read an input sequence and produce a binary or multi-class answer
- E.g., sentiment classification

Sentence-Level Sentiment Classification

- Given a sentence s, classify s as positive or negative
- Examples:
  - Positive: *It's not life-affirming – it's vulgar and mean, but I liked it.*
  - Negative: *It's disappointing that it only manages to be decent instead of dead brilliant.*
- Need to deal with negation, sarcasm, sentence structure, etc.

Sentence-Level Sentiment Classification

- $p(\text{label} = k | w_{1:n}) = \hat{\boldsymbol{y}}_{[k]} \quad k = 1,2$
- $\hat{\boldsymbol{y}} = \text{softmax}(\text{MLP}(\text{RNN}(\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}})))$
- $\boldsymbol{x}_{\boldsymbol{1}:\boldsymbol{n}} = \boldsymbol{E}_{[w_1]}, \ldots, \boldsymbol{E}_{[w_n]}$
- Trained with cross-entropy loss function
- Word embedding matrix $\boldsymbol{E}$ is initialized using pre-trained word embeddings (e.g., from Word2Vec)

Sentence-Level Sentiment Classification

- Bidirectional RNN
- $p(\text{label} = k | w_{1:n}) = \widehat{\boldsymbol{y}}_{[k]} \quad k = 1,2$
- $\widehat{\boldsymbol{y}} = \text{softmax}(\text{MLP}([\text{RNN}^f(\boldsymbol{x}_{\mathbf{1}:\boldsymbol{n}}); \text{RNN}^b(\boldsymbol{x}_{\boldsymbol{n}:\mathbf{1}})]))$
- $\boldsymbol{x}_{\mathbf{1}:\boldsymbol{n}} = \boldsymbol{E}_{[w_1]}, \dots, \boldsymbol{E}_{[w_n]}$