IN5550: Neural Methods in Natural Language Processing
Sub-lecture 6.2
Enter Recurrent Neural Networks

Andrey Kutuzov

University of Oslo

24 February 2022

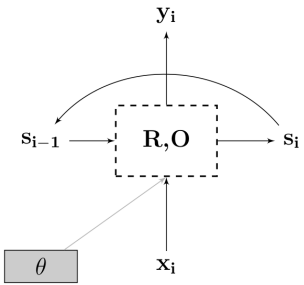


Contents



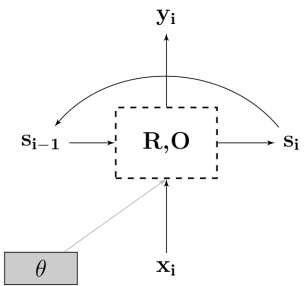
1 Enter recurrent neural networks

Recurrent Neural Networks in the Abstract



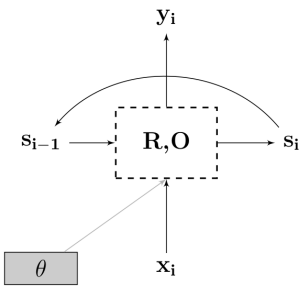
- Recurrent Neural Networks (RNNs) take variable-length sequences as input
- map input sequence $\boldsymbol{x_{1:n}}$ to output $\boldsymbol{y_{1:n}}$

Recurrent Neural Networks in the Abstract



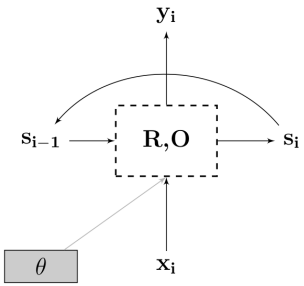
- Recurrent Neural Networks (RNNs) take variable-length sequences as input
- map input sequence $\boldsymbol{x_{1:n}}$ to output $\boldsymbol{y_{1:n}}$
- unlike feed-forward architectures, RNNs contain cycles
- are highly sensitive to linear order; need not make any Markov assumptions

Recurrent Neural Networks in the Abstract



- Recurrent Neural Networks (RNNs) take variable-length sequences as input
- map input sequence $\boldsymbol{x}_{1:n}$ to output $\boldsymbol{y}_{1:n}$
- unlike feed-forward architectures, RNNs contain cycles
- are highly sensitive to linear order; need not make any Markov assumptions
- internal state sequence $\boldsymbol{s}_{1:n}$ as 'history'

Recurrent Neural Networks in the Abstract

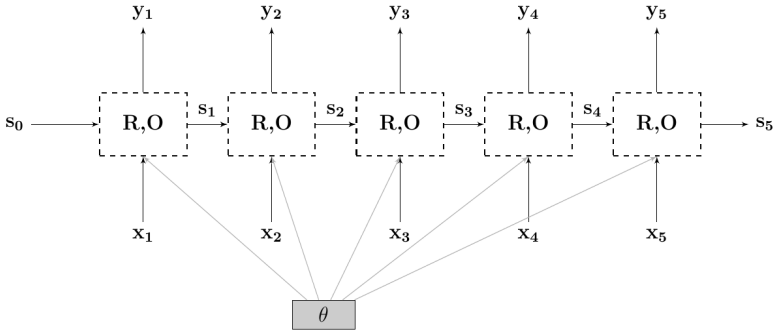


- Recurrent Neural Networks (RNNs) take variable-length sequences as input
- map input sequence $\boldsymbol{x_{1:n}}$ to output $\boldsymbol{y_{1:n}}$
- unlike feed-forward architectures, RNNs contain cycles
- are highly sensitive to linear order; need not make any Markov assumptions
- internal state sequence $\boldsymbol{s_{1:n}}$ as 'history'

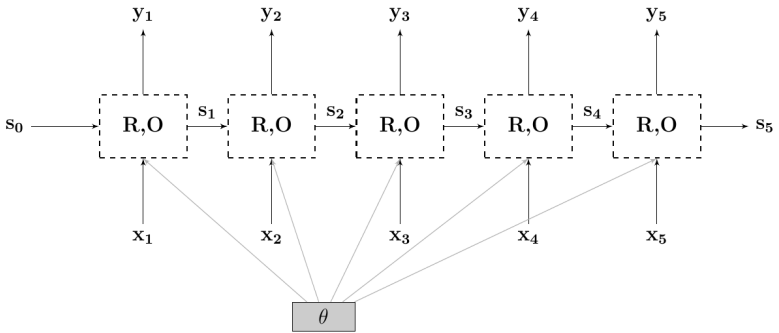
$$\begin{aligned} \text{RNN}(\boldsymbol{x_{1:n}}, \boldsymbol{s_0}) &= \boldsymbol{y_{1:n}} \\ \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) \end{aligned}$$

$$\boldsymbol{x_i} \in \mathbb{R}^{d_x};\ \boldsymbol{y_i} \in \mathbb{R}^{d_y};\ \boldsymbol{s_i} \in \mathbb{R}^{f(d_y)}$$

Still High-Level: The RNN Abstraction Unrolled



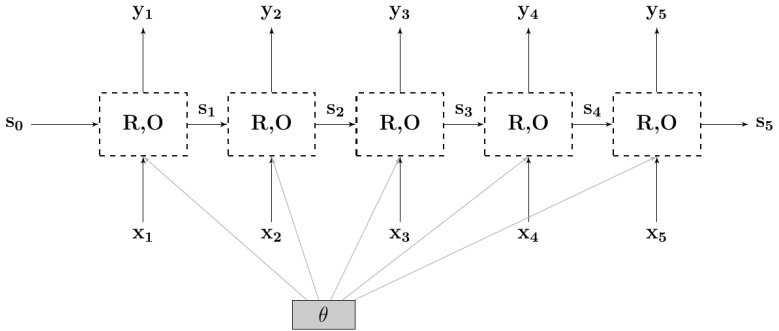
Still High-Level: The RNN Abstraction Unrolled



- This is how it looks like in a *PyTorch* or *TensorFlow* computational graph.
- Each state s_i and output y_i depend on the full previous context, e.g.

$$s_4 = R(R(R(R(x_1, s_o), x_2), x_3)x_4)$$

Still High-Level: The RNN Abstraction Unrolled



- This is how it looks like in a *PyTorch* or *TensorFlow* computational graph.
- Each state s_i and output y_i depend on the full previous context, e.g.

$$s_4 = R(R(R(R(x_1, s_o), x_2), x_3)x_4)$$

- Functions $R(\cdot)$ and $O(\cdot)$ shared across time points; fewer parameters

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Arguably the Most Basic RNN Implementation

$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i} \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Arguably the Most Basic RNN Implementation

$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i} \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

- Does this maybe look familiar?

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Arguably the Most Basic RNN Implementation

$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i} \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

- Does this maybe look familiar? Merely a continuous bag of words

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Arguably the Most Basic RNN Implementation

$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i} \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

- Does this maybe look familiar? Merely a continuous bag of words
- order-insensitive: *Google acquired YouTube* $\equiv$ *YouTube acquired Google*

Implementing the RNN Abstraction



- We have yet to define the nature of the $R(\cdot)$ and $O(\cdot)$ functions
- RNNs are actually a family of architectures; much variation for $R(\cdot)$

Arguably the Most Basic RNN Implementation

$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = \boldsymbol{s_{i-1}} + \boldsymbol{x_i} \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

- Does this maybe look familiar? Merely a continuous bag of words
- order-insensitive: *Google acquired YouTube* $\equiv$ *YouTube acquired Google*
- actually has no parameters of it own: $\theta = \{\}$; thus, no learning ability

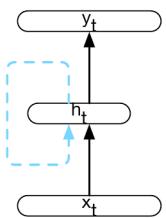
The ‘Simple’ RNN



- We want to learn the dependencies between elements of the sequence
- nature of the $R(\cdot)$ function needs to be determined during training

The 'Simple' RNN

- We want to learn the dependencies between elements of the sequence
- nature of the $R(\cdot)$ function needs to be determined during training



The Elman RNN: most basic type

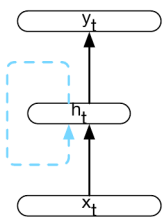
$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g(\boldsymbol{s_{i-1}}\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b}) \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

$$\boldsymbol{x_i} \in \mathbb{R}^{d_x};\ \boldsymbol{s_1}, \boldsymbol{y_i} \in \mathbb{R}^{d_y};\ \boldsymbol{W^x} \in \mathbb{R}^{d_x \times d_s};\ \boldsymbol{W^s} \in \mathbb{R}^{d_s \times d_s};\ \boldsymbol{b} \in \mathbb{R}^{d_s}$$

[Elman, 1990]

The 'Simple' RNN

- We want to learn the dependencies between elements of the sequence
- nature of the $R(\cdot)$ function needs to be determined during training



The Elman RNN: most basic type

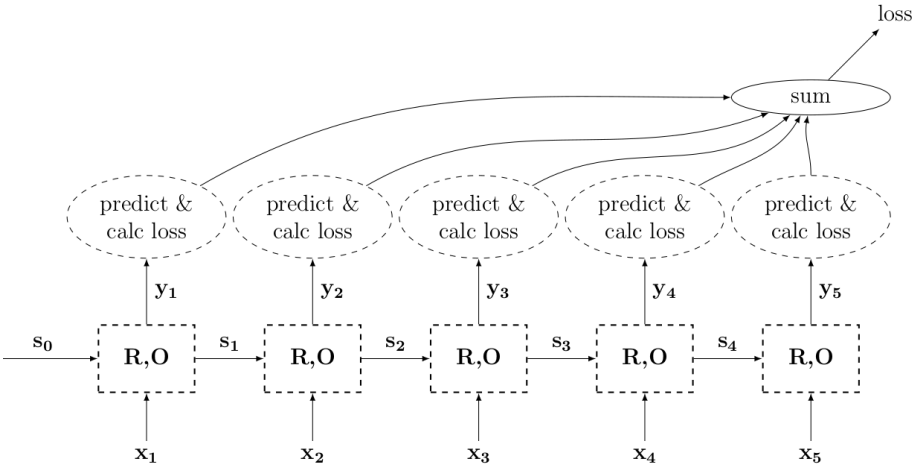
$$\begin{aligned} \boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g(\boldsymbol{s_{i-1}}\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b}) \\ \boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{s_i} \end{aligned}$$

$$\boldsymbol{x_i} \in \mathbb{R}^{d_x};\ \boldsymbol{s_1}, \boldsymbol{y_i} \in \mathbb{R}^{d_y};\ \boldsymbol{W^x} \in \mathbb{R}^{d_x \times d_s};\ \boldsymbol{W^s} \in \mathbb{R}^{d_s \times d_s};\ \boldsymbol{b} \in \mathbb{R}^{d_s}$$

[Elman, 1990]

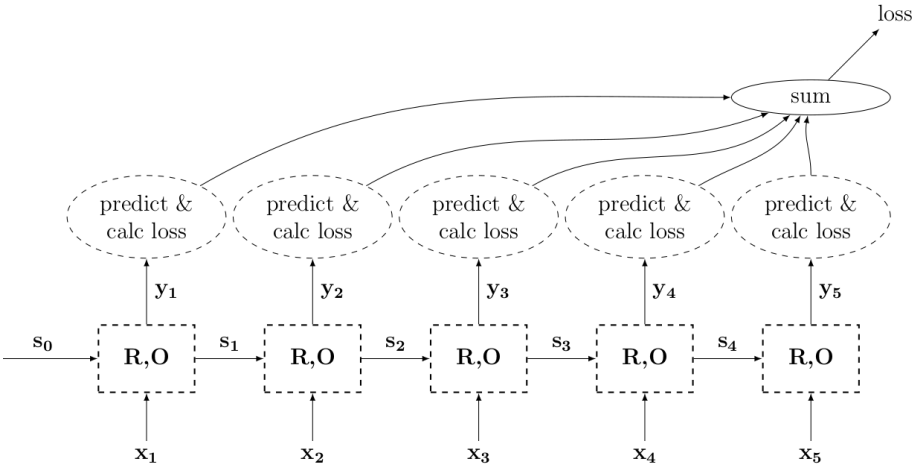
- Linear transformations of states and inputs; non-linear activation
- alternative, equivalent definition of $R(\cdot)$: $\boldsymbol{s_i} = g([\boldsymbol{s_{i-1}}; \boldsymbol{x_i}]\boldsymbol{W} + \boldsymbol{b})$

Training Recurrent Neural Networks end-to-end



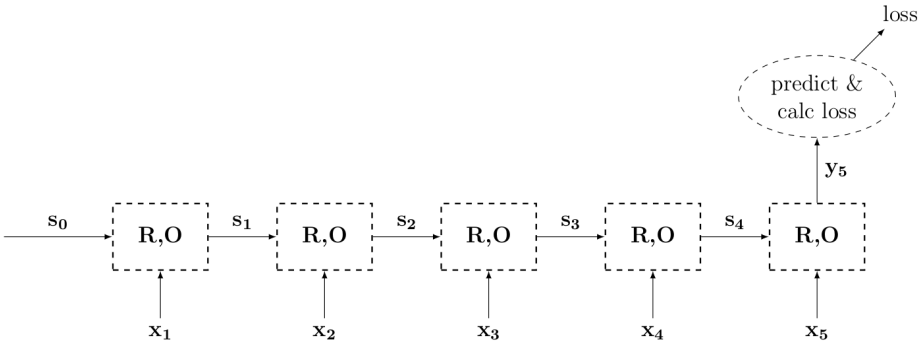
- Elman RNN in a sequence labeling task, e.g. classification from output states y_i

Training Recurrent Neural Networks end-to-end

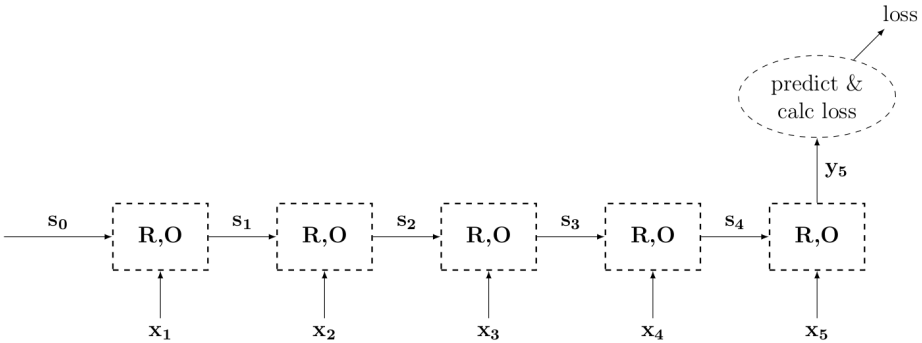


- Elman RNN in a sequence labeling task, e.g. classification from output states y_i
- standard loss functions, optimizers and Back-propagation Through Time (BPTT)

An Alternate Training Regime: sequence classification

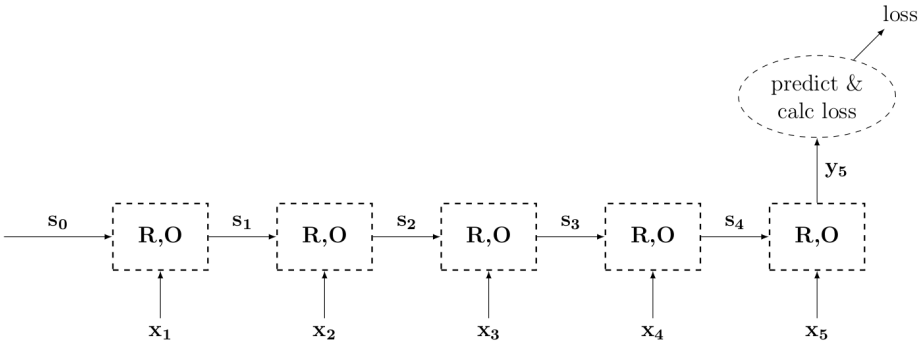


An Alternate Training Regime: sequence classification



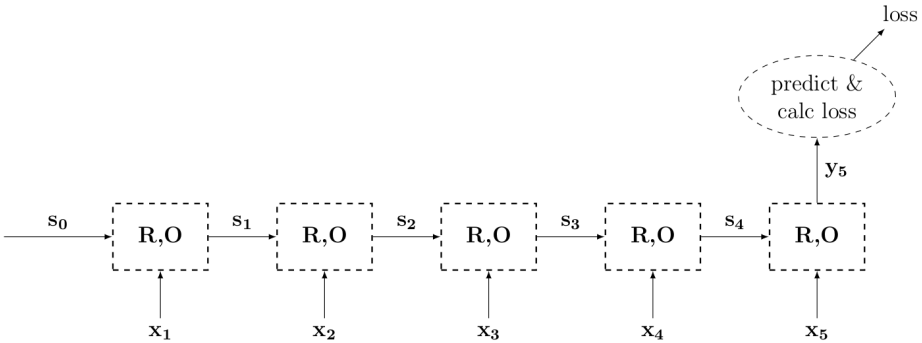
- Focus on final output state: y_n as encoding of full sequence $x_{1:n}$
- Looking familiar?

An Alternate Training Regime: sequence classification



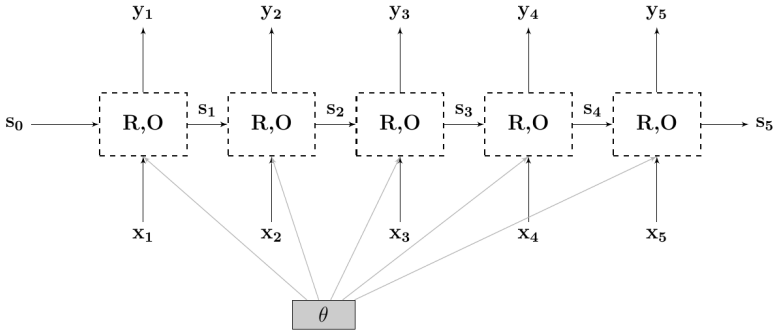
- Focus on final output state: y_n as encoding of full sequence $x_{1:n}$
- Looking familiar? Mapping variable-length sequence to a fixed-size vector

An Alternate Training Regime: sequence classification

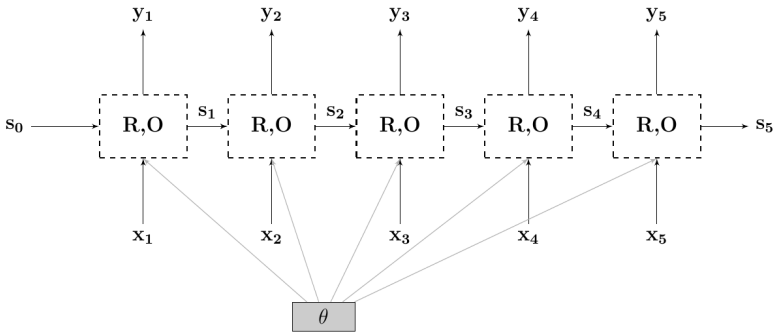


- Focus on final output state: y_n as encoding of full sequence $x_{1:n}$
- Looking familiar? Mapping variable-length sequence to a fixed-size vector
- sentence-level classification; language modeling; conditioned generation
  - aka sequence–to–sequence models; e.g. translation or summarization

Unrolled RNNs, in a Sense, are very Deep MLPs

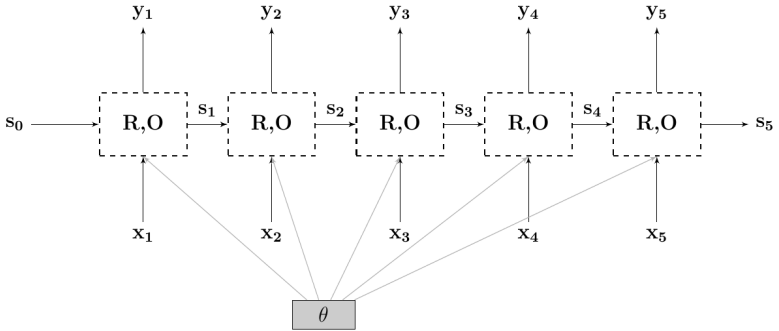


Unrolled RNNs, in a Sense, are very Deep MLPs



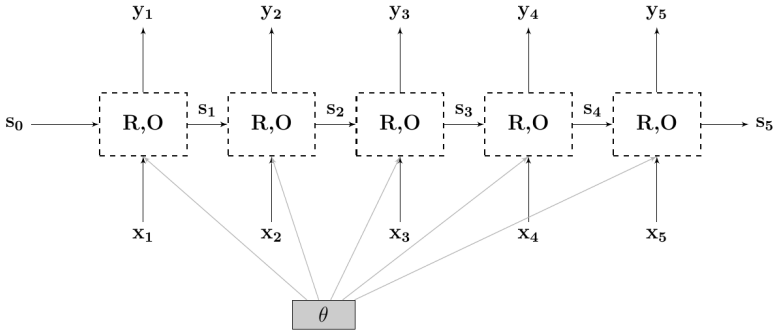
$$\boldsymbol{s_i} \quad = \quad R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g(\boldsymbol{s_{i-1}}\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b})$$

Unrolled RNNs, in a Sense, are very Deep MLPs



$$
\begin{aligned}
\boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g(\boldsymbol{s_{i-1}}\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b}) \\
&= g(g(\boldsymbol{s_{i-2}}\boldsymbol{W^s} + \boldsymbol{x_{i-1}}\boldsymbol{W^x} + \boldsymbol{b})\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b})
\end{aligned}
$$

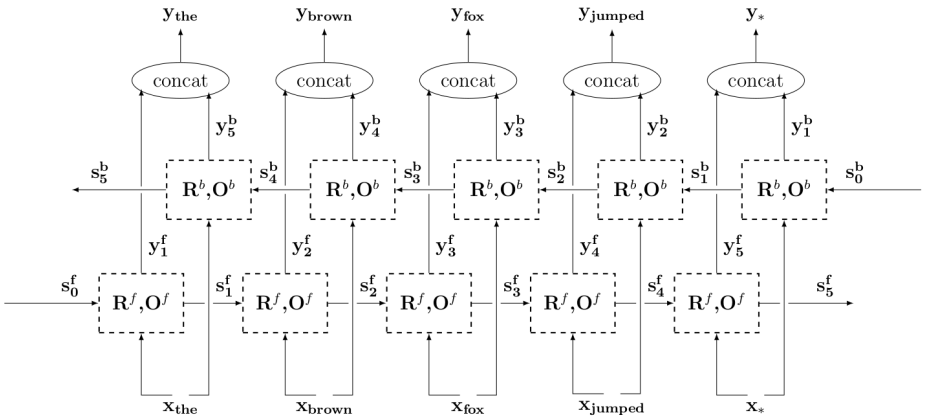
Unrolled RNNs, in a Sense, are very Deep MLPs



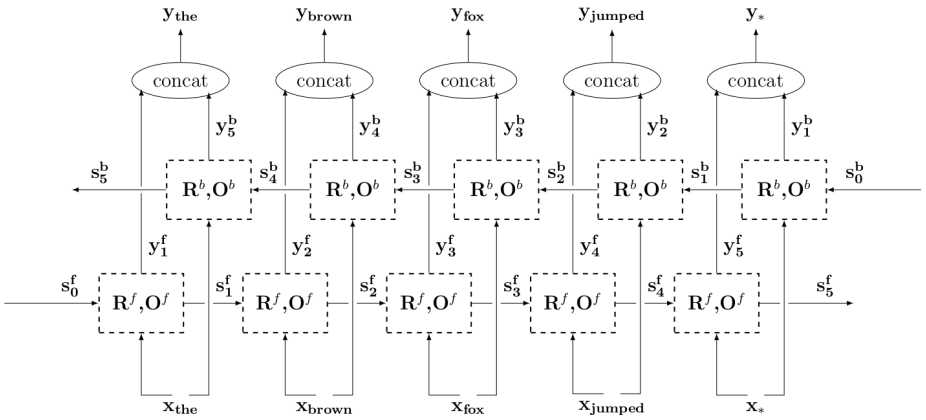
$$
\begin{aligned}
\boldsymbol{s_i} &= R(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}) = g(\boldsymbol{s_{i-1}}\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b}) \\
&= g(g(\boldsymbol{s_{i-2}}\boldsymbol{W^s} + \boldsymbol{x_{i-1}}\boldsymbol{W^x} + \boldsymbol{b})\boldsymbol{W^s} + \boldsymbol{x_i}\boldsymbol{W^x} + \boldsymbol{b})
\end{aligned}
$$

- $\boldsymbol{W^s}$, $\boldsymbol{W^x}$ shared across all layers $\rightarrow$ exploding or vanishing gradients (later about this)

Variants: Bi-Directional Recurrent Networks

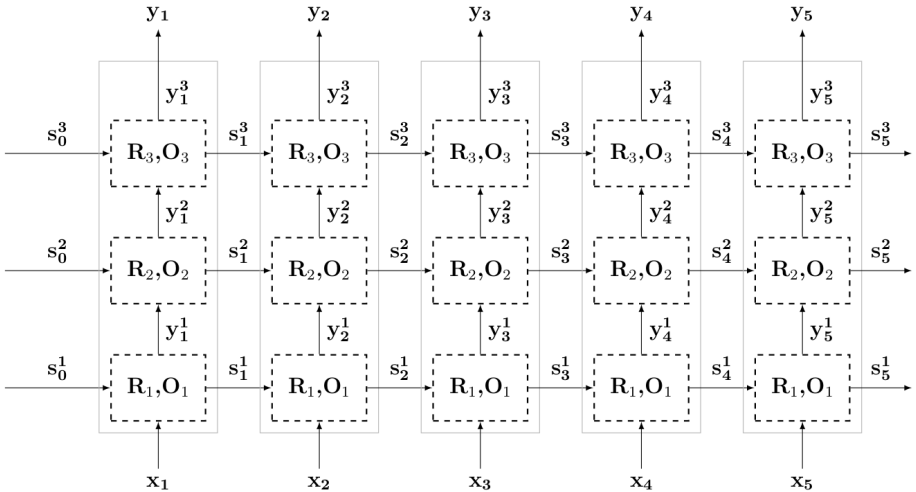


Variants: Bi-Directional Recurrent Networks



- Capture full left and right context: 'history' and 'future' for each $\boldsymbol{x_i}$
- moderate increase in parameters (double); still linear-time computation

Variants: 'Deep' (Stacked) Recurrent Networks



How to use these architectures? See the next sub-lecture 6.3.

References I



Elman, J. L. (1990).
Finding structure in time.
Cognitive science, 14(2):179–211.