THE OHIO STATE UNIVERSITY

CSE 5525: Foundations of Speech and Language Processing

Recurrent Neural Networks

Huan Sun (CSE@OSU)

Many thanks to Prof. Greg Durrett @ UT Austin for sharing his slides.

This Lecture

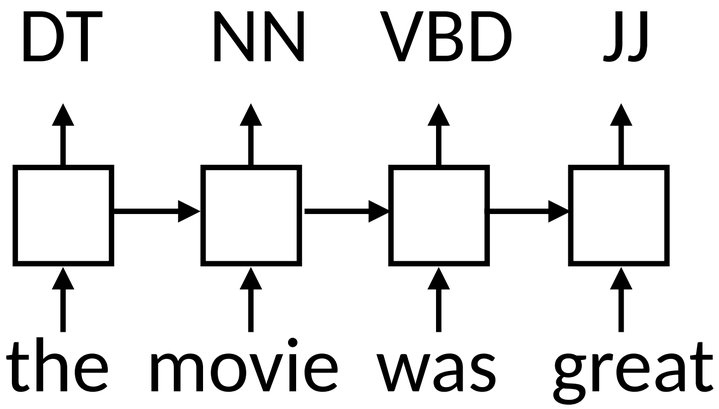
- Recurrent neural networks
- Vanishing gradient problem
- LSTMs / GRUs
- Applications / visualizations

RNN Basics

RNN Uses

Modeling sequential data

- Transducer: make some prediction for each element in a sequence

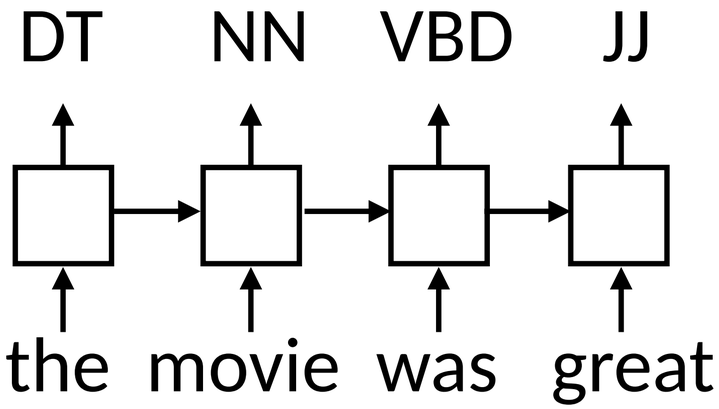


output **y** = score for each tag, then softmax

RNN Uses

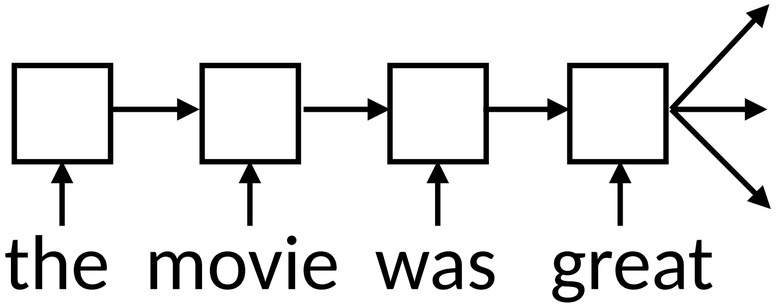
Modeling sequential data

- Transducer: make some prediction for each element in a sequence



output **y** = score for each tag, then softmax

- Encoder: encode a sequence into a fixed-sized vector and use that for some purpose, e.g.,

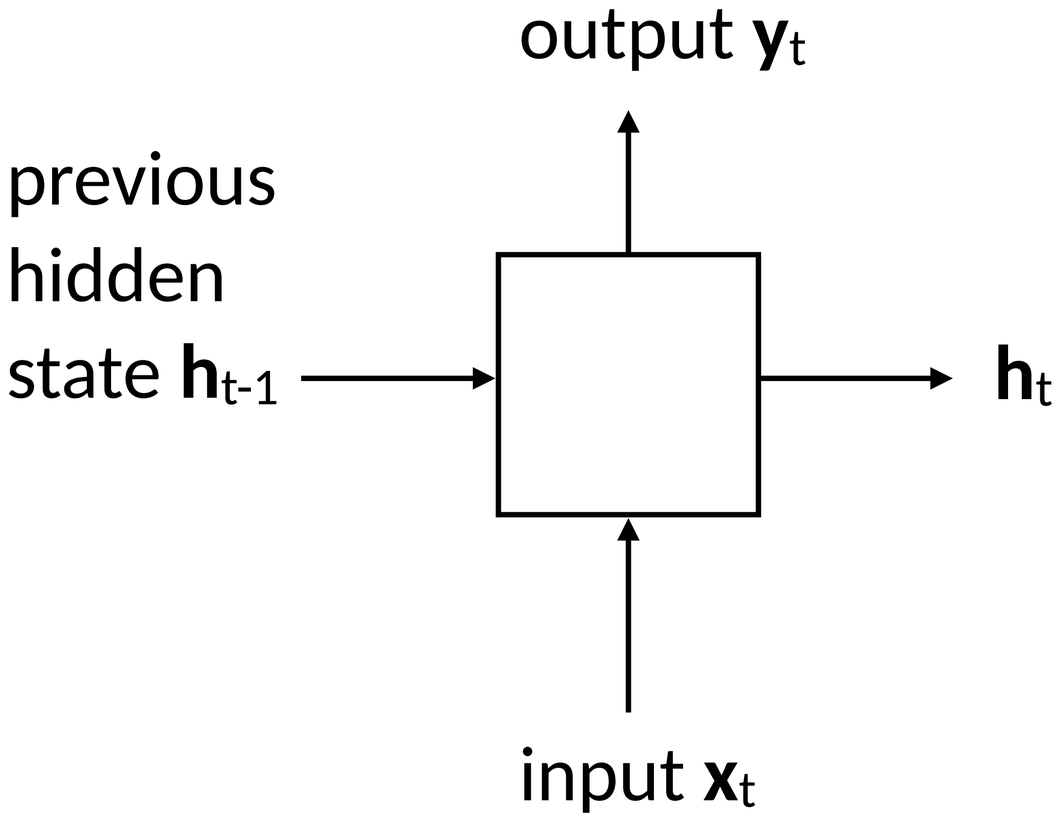


predict sentiment (matmul + softmax)

translate

paraphrase/compress

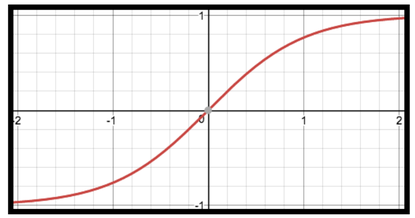
Elman Networks (Simple Recurrent Networks)



- 1. Updates hidden state based on input and previous hidden state

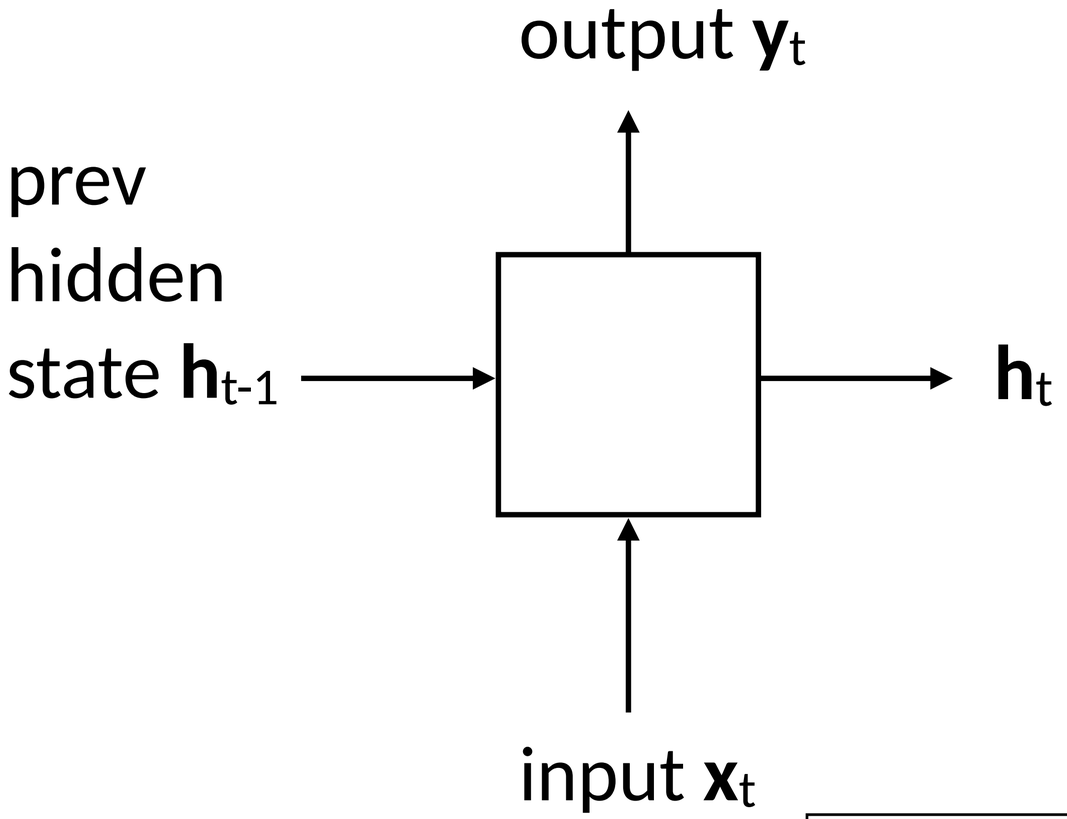
$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



- Long history! (invented in the late 1980s) Elman (1990)

Elman Networks



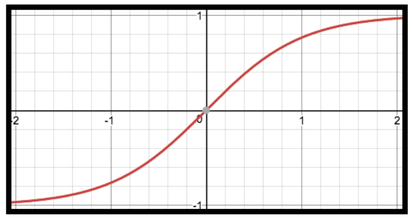
- 1. Updates hidden state based on input and previous hidden state

$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

- 2. Computes output from hidden state

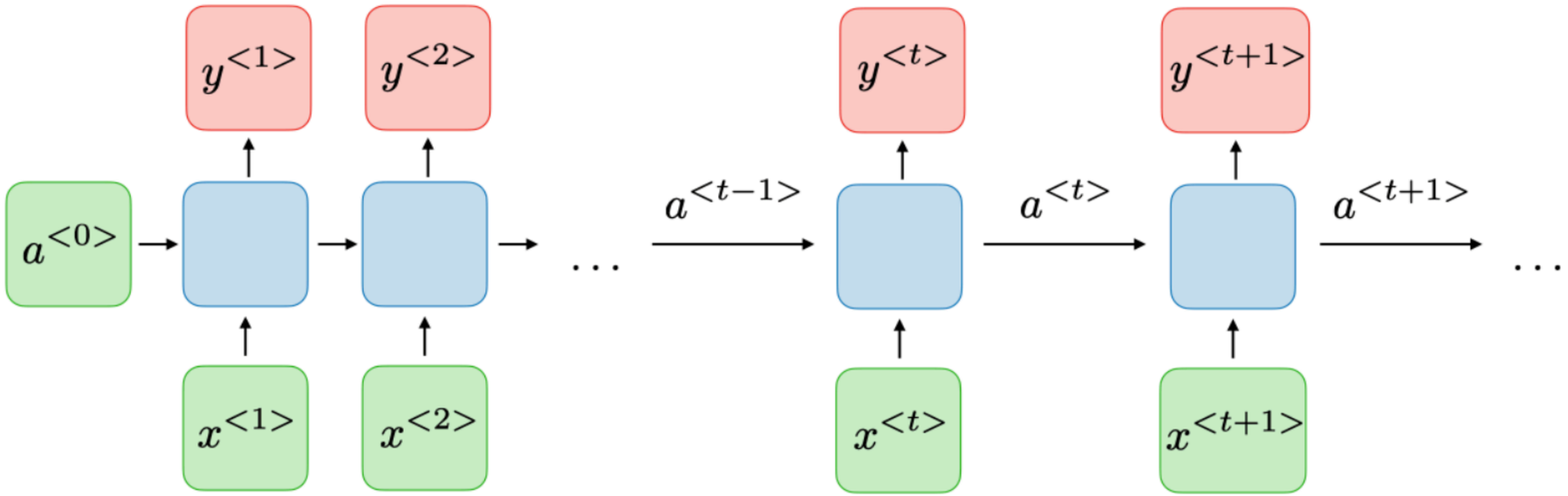
$$\mathbf{y}_t = \tanh(U\mathbf{h_t} + \mathbf{b}_y)$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



- Long history! (invented in the late 1980s) Elman (1990)

Elman Networks (Simple Recurrent Networks)



$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

$W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally

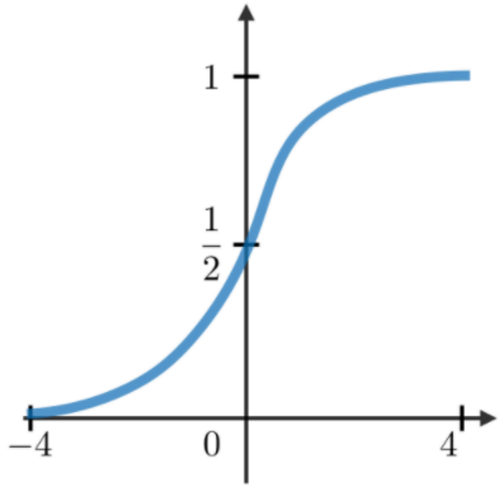
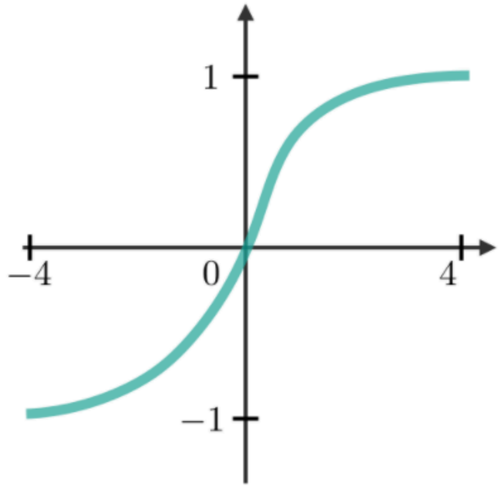
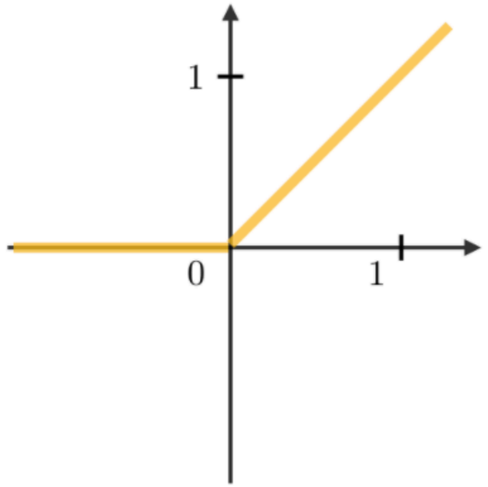
g_1, g_2 activation functions.

Same as previous slides, but we use a as hidden state

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

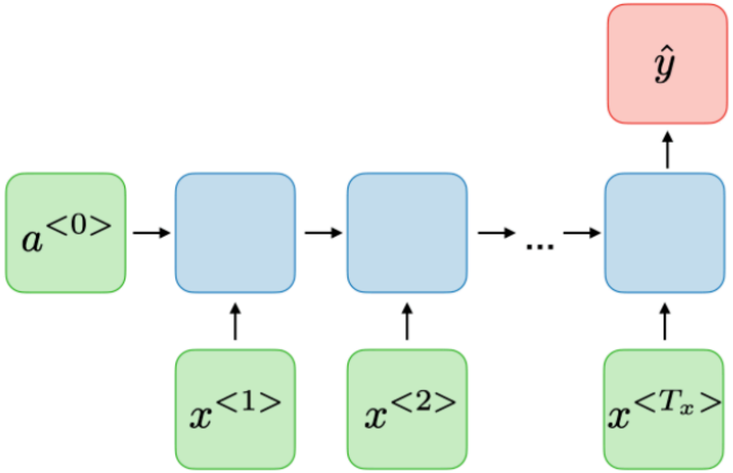
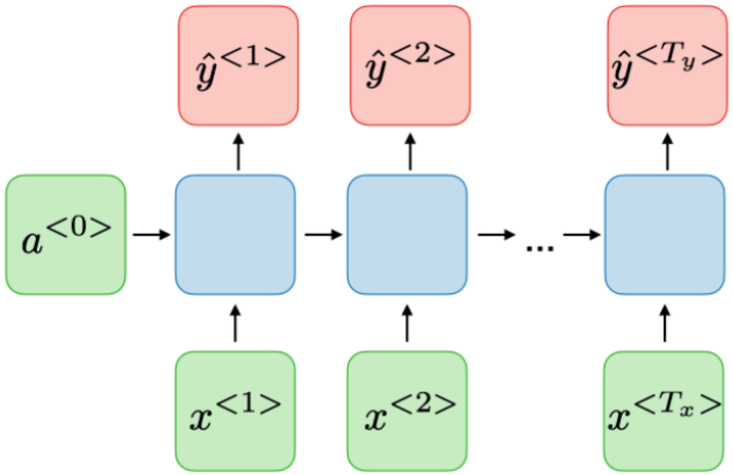
Elman Networks (Simple Recurrent Networks)

❒ **Commonly used activation functions** — The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

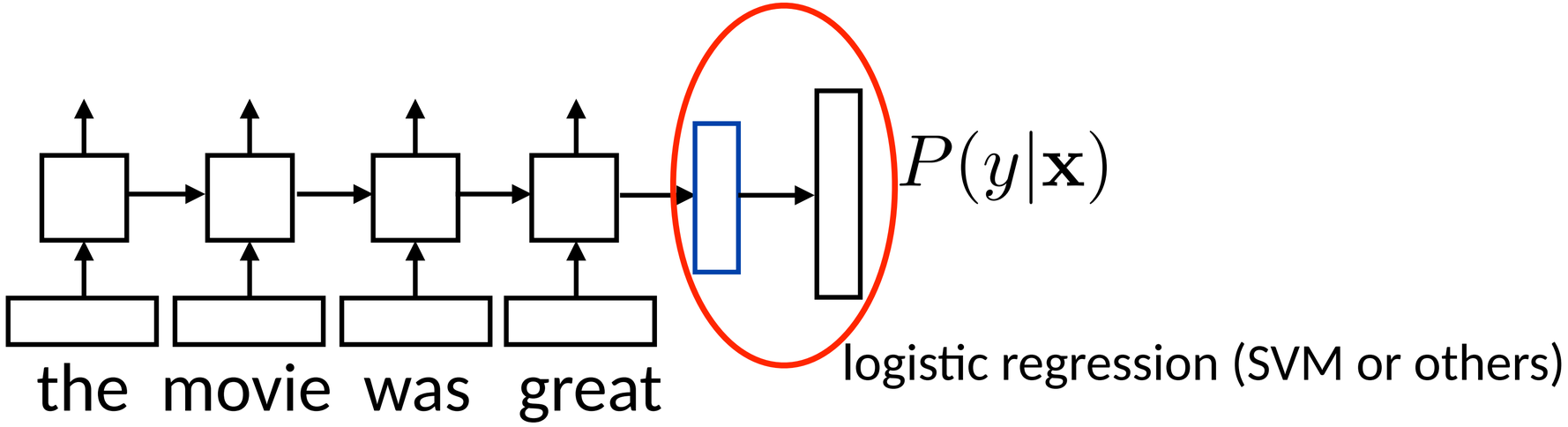
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

Applications

Type	Illustration	Example
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition

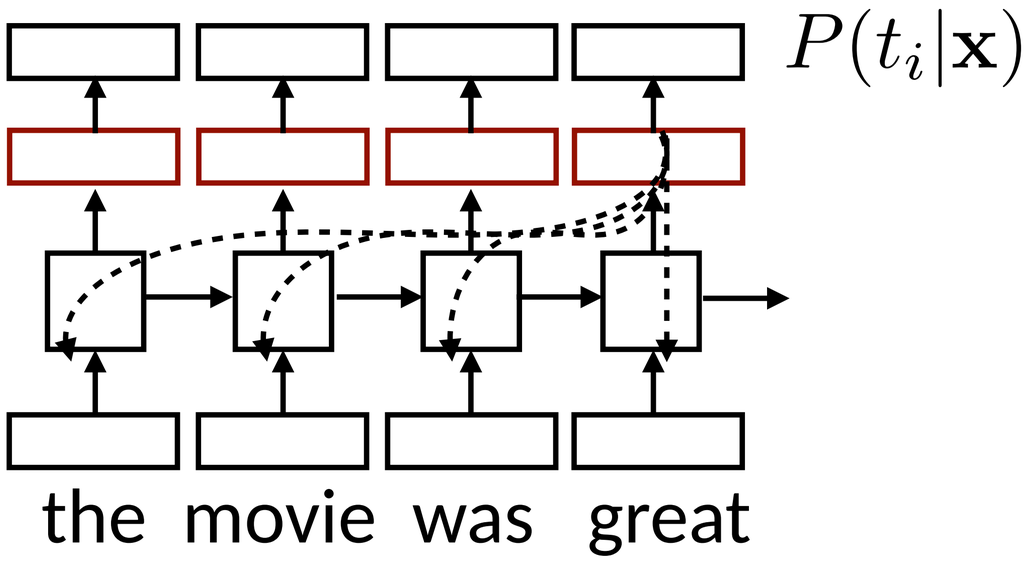
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

Training RNNs



- Loss = negative log likelihood of probability of gold label, if using logistic regression
- Backpropagate through entire network
- Example: sentiment analysis

Training RNNs



- Loss = negative log likelihood of probability of gold predictions, summed over the tags (i.e., all time steps)
- Loss terms backpropagate through network
- Example: language modeling (predict next word given context) or POS tagging

Training Simple Recurrent Networks

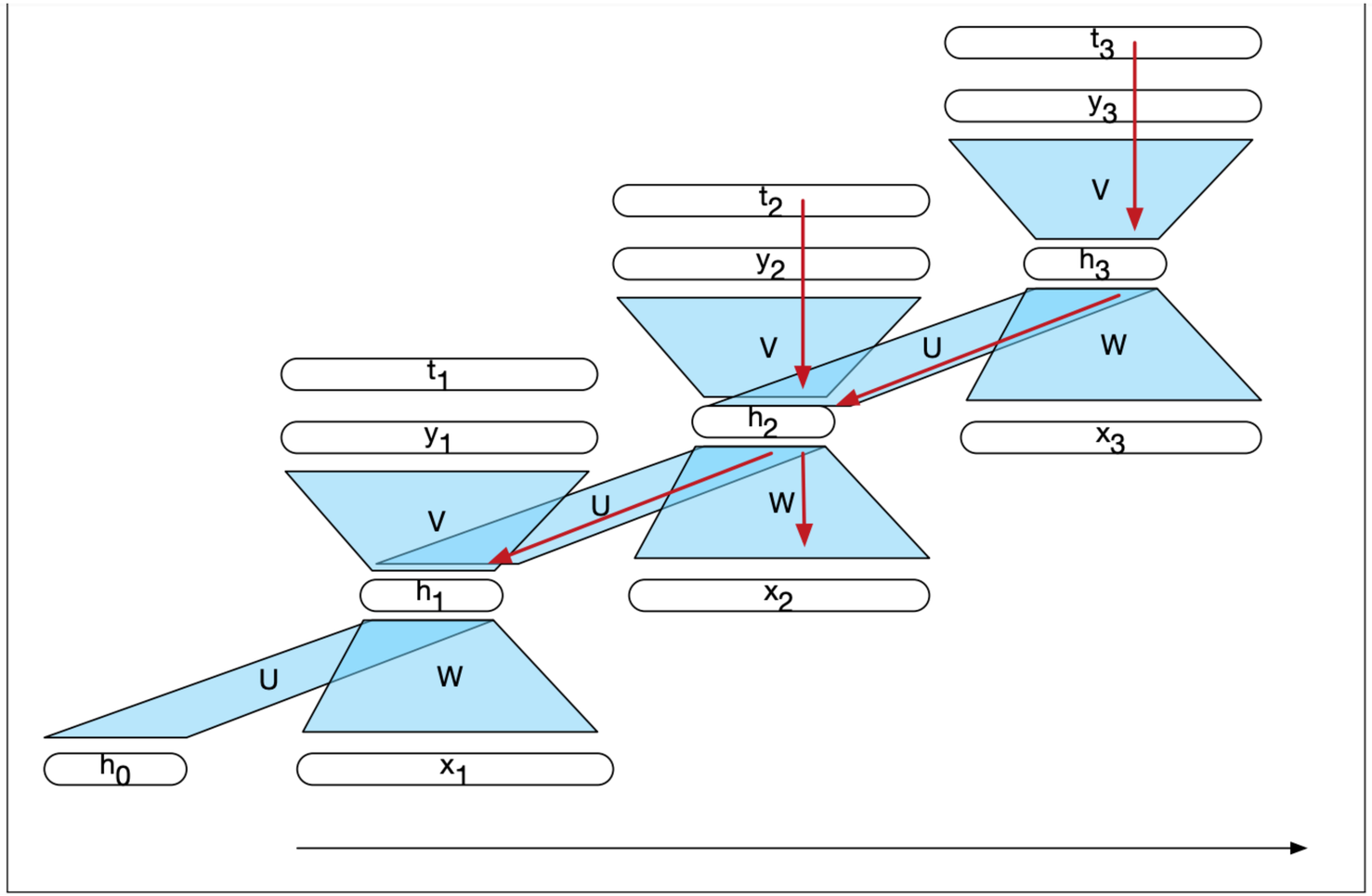


Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging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

Textbook JM, Chapter 9
https://web.stanford.edu/~jurafsky/slp3/9.pdf

t_i is the target and y_i is predicted at each step i

Training Simple Recurrent Networks

- "Backpropagation through time"

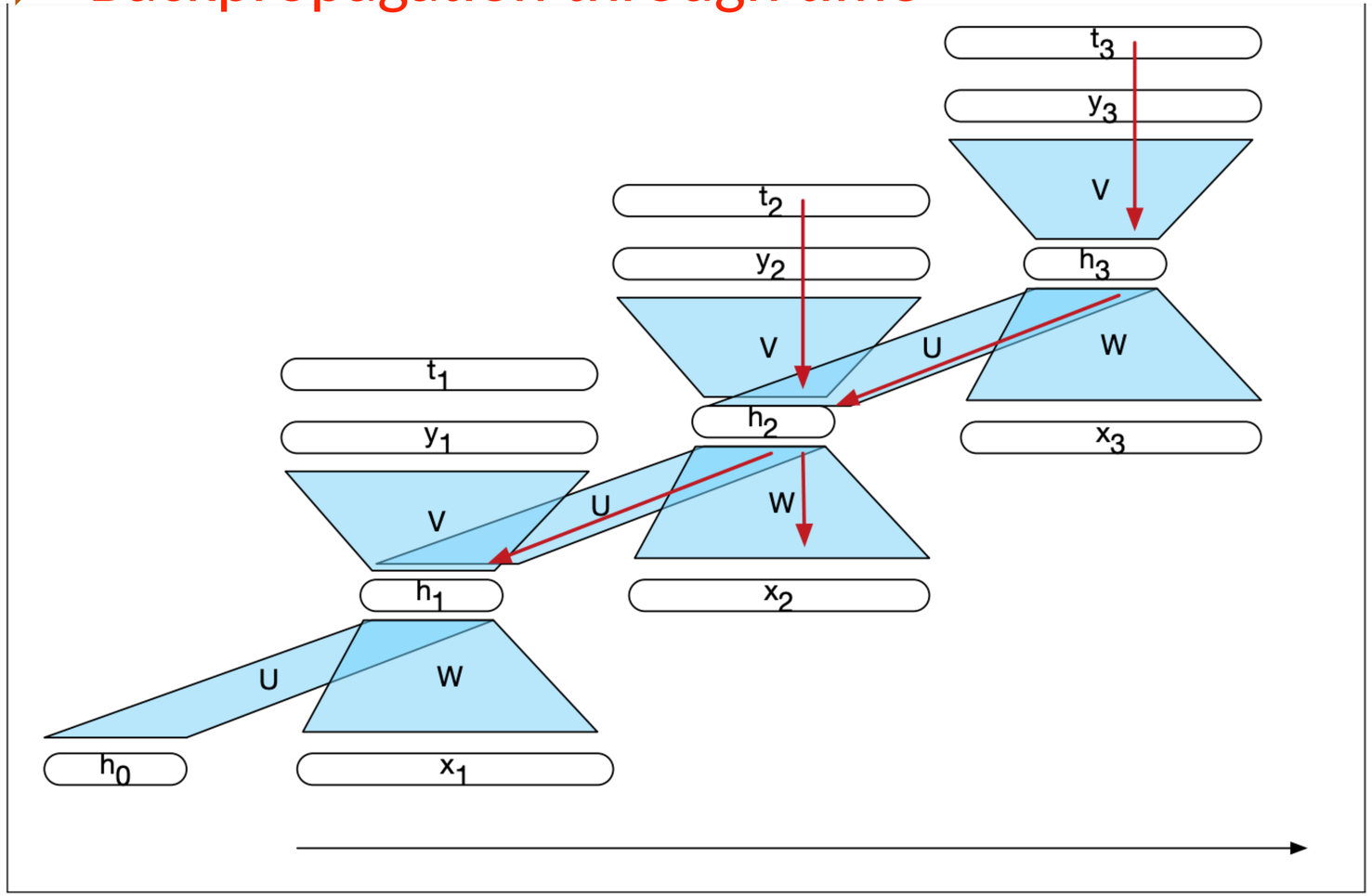
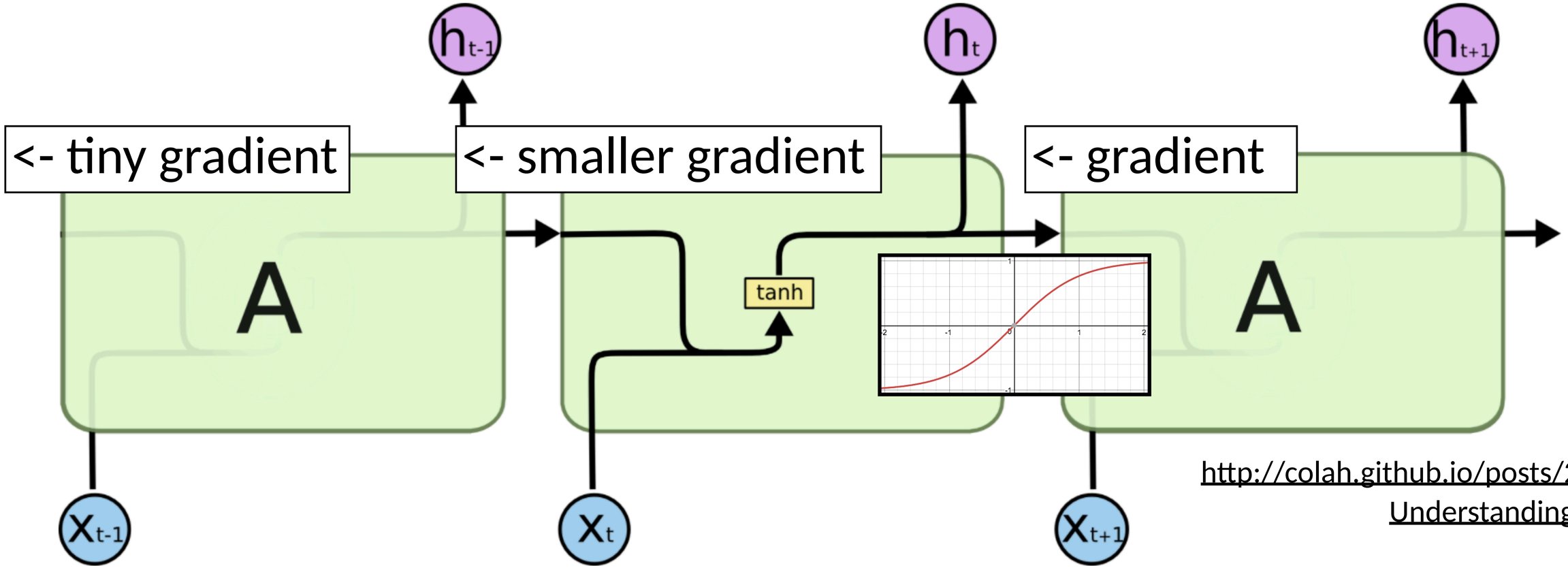


Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging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

Textbook JM, Chapter 9
https://web.stanford.edu/~jurafsky/slp3/9.pdf

t_i is the target and y_i is predicted at each step i

Vanishing Gradient



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

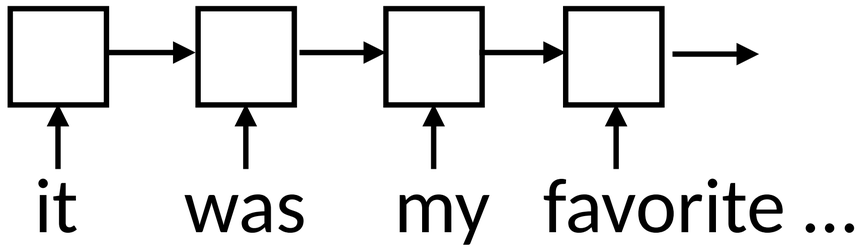
- Gradient diminishes during backpropagation
- Repeated multiplication by V causes problems in $\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

Training Elman Networks

“it was my favorite movie of 2016, though it wasn’t without problems” -> +

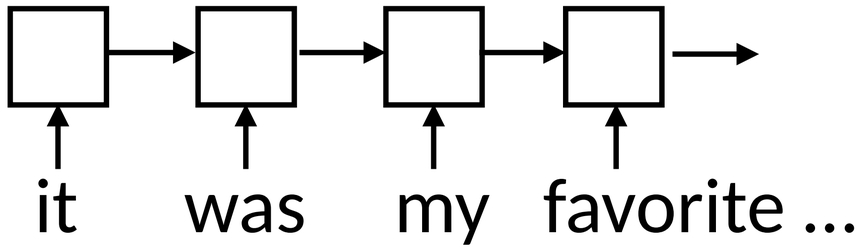
- RNN potentially needs to learn how to “remember” information for a long time!



Training Elman Networks

"it was my favorite movie of 2016, though it wasn't without problems" -> +

- RNN potentially needs to learn how to "remember" information for a long time!



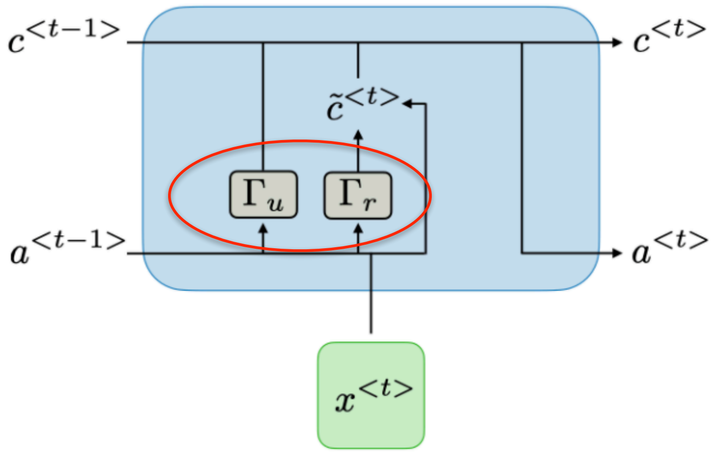
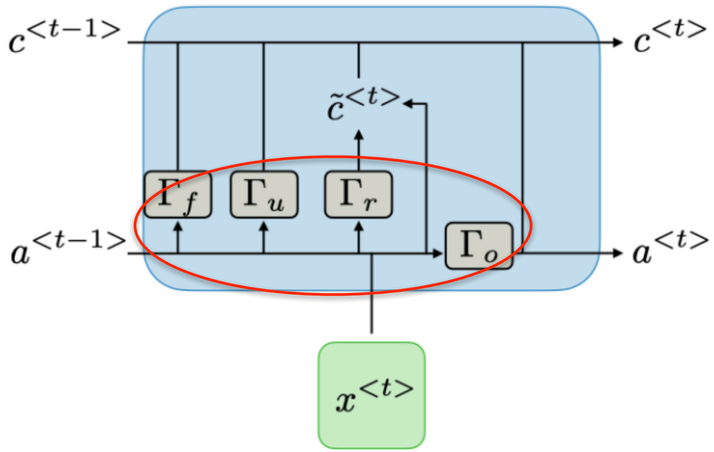
Challenging to handle such "long-term dependencies"

LSTMs/GRUs

- Designed to fix “vanishing gradient” problem using *gates*

LSTMs/GRUs

❒ **GRU/LSTM** — Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

Characterization	**Gated Recurrent Unit** (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Gates

Remark: the sign ⋆ denotes the element-wise multiplication between two vectors.

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

LSTMs/GRUs

❐ **Types of gates** — In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

$$\boxed{\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)}$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function. The main ones are summed up in the table below:

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

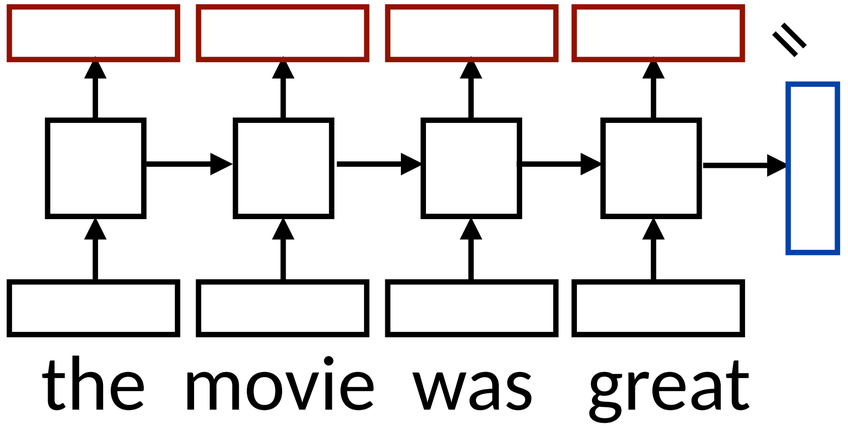
RNNs VS. LSTMs/GRUs

1. Why do RNNs suffer from vanishing and exploding gradients?

2. How do LSTMs keep the gradients from vanishing or explode?

See more:
https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a678497 1a577

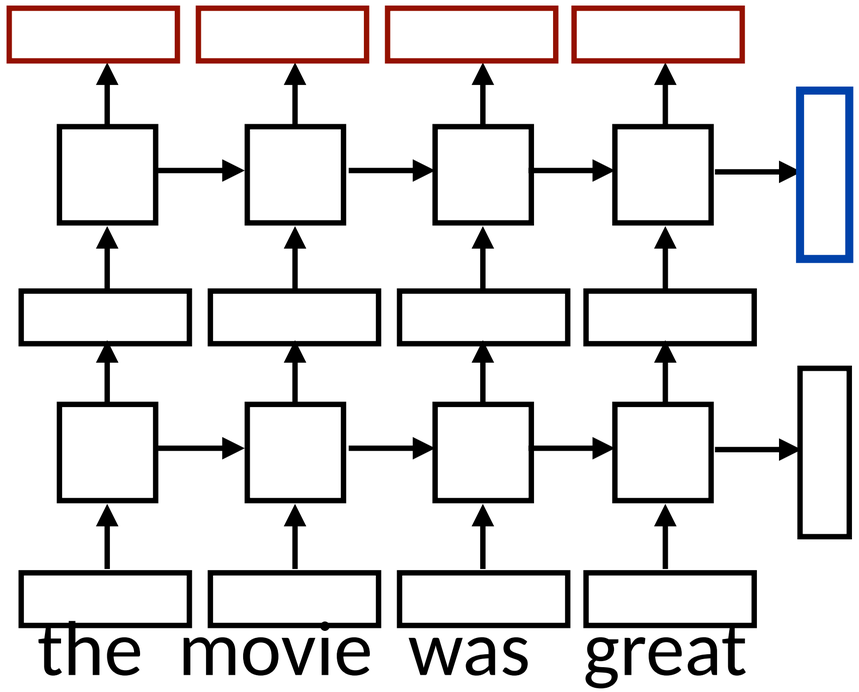
What do RNNs produce?



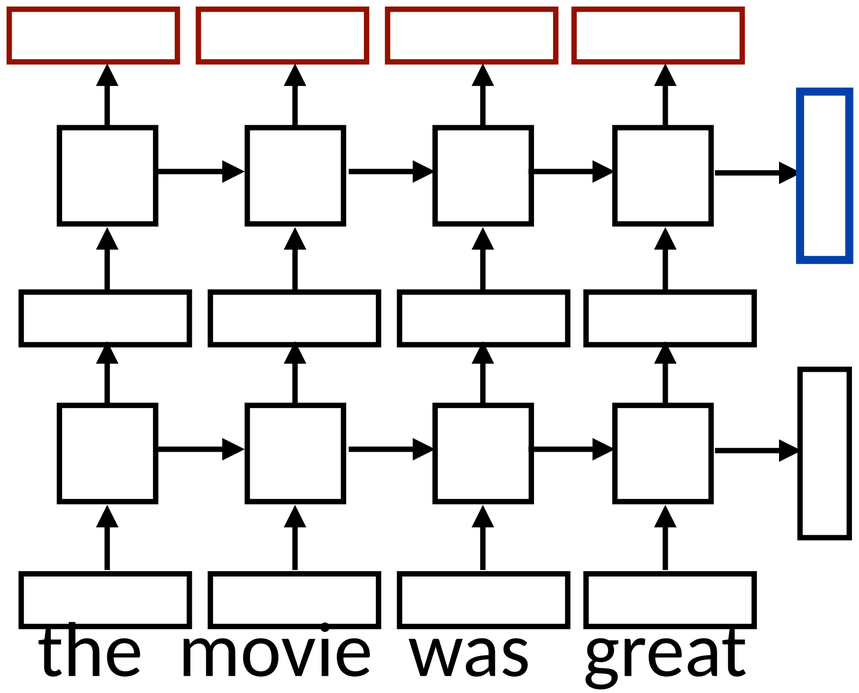
RNNs here include LSTMs /GRUs

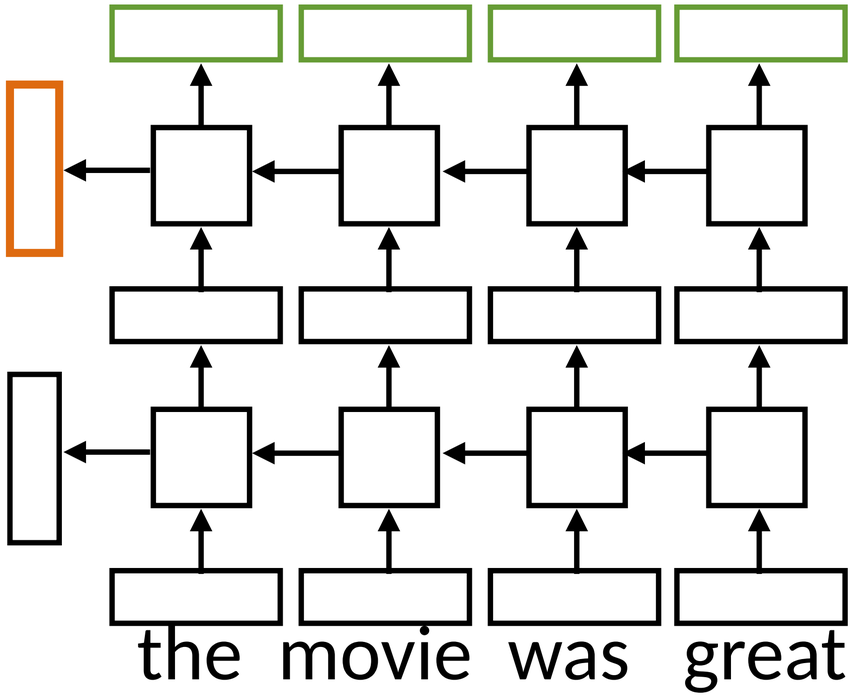
- Encoding of the sentence — can pass this to a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

Multilayer RNN



Multilayer Bidirectional RNN



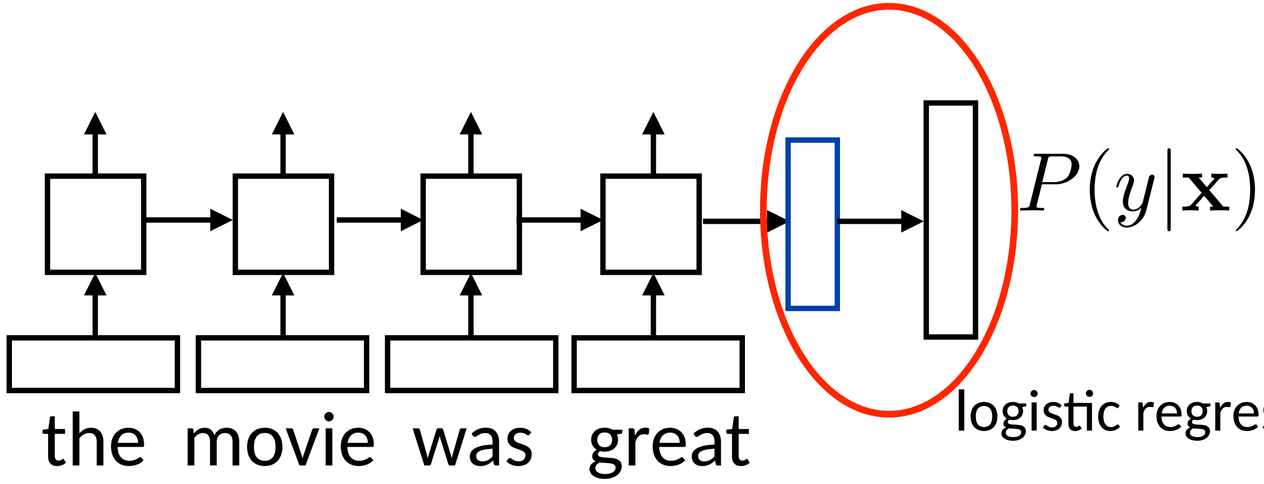


- Sentence classification based on concatenation of both final outputs

- Token classification based on concatenation of both directions' token representations

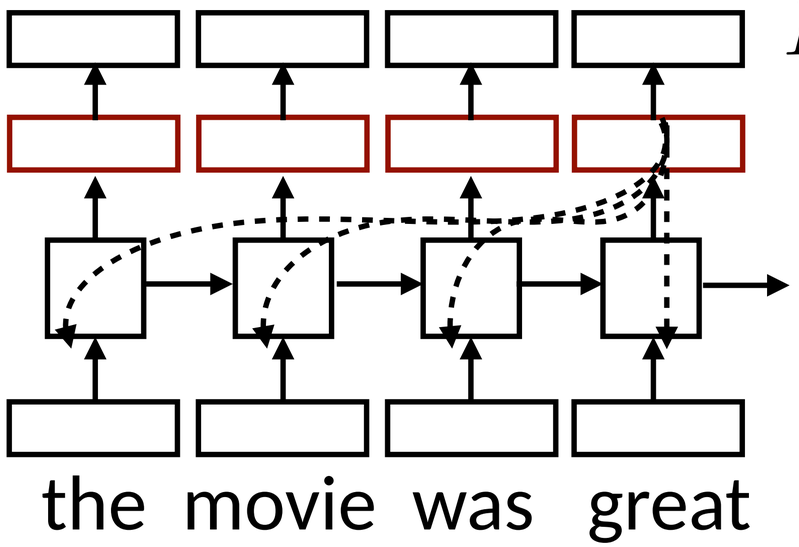
Training RNNs in general

Same as before



- Loss = negative log likelihood of probability of gold label, if using logistic regression
- Backpropagate through entire network
- Example: sentiment analysis

Training RNNs in general



$P(t_i|\mathbf{x})$

Same as before

- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network
- Example: language modeling (predict next word given context) or POS tagging

Applications

What can LSTMs model?

- Sentiment analysis (mentioned earlier)
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction (later in the course)
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)
- Textual entailment
  - Encode two sentences, predict

Natural Language Inference

Premise		Hypothesis
A boy plays in the snow	*entails*	A boy is outside
A man inspects the uniform of a figure	*contradicts*	The man is sleeping
An older and younger man smiling	*neutral*	Two men are smiling and laughing at cats playing

- Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)
- Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)

SNLI Dataset

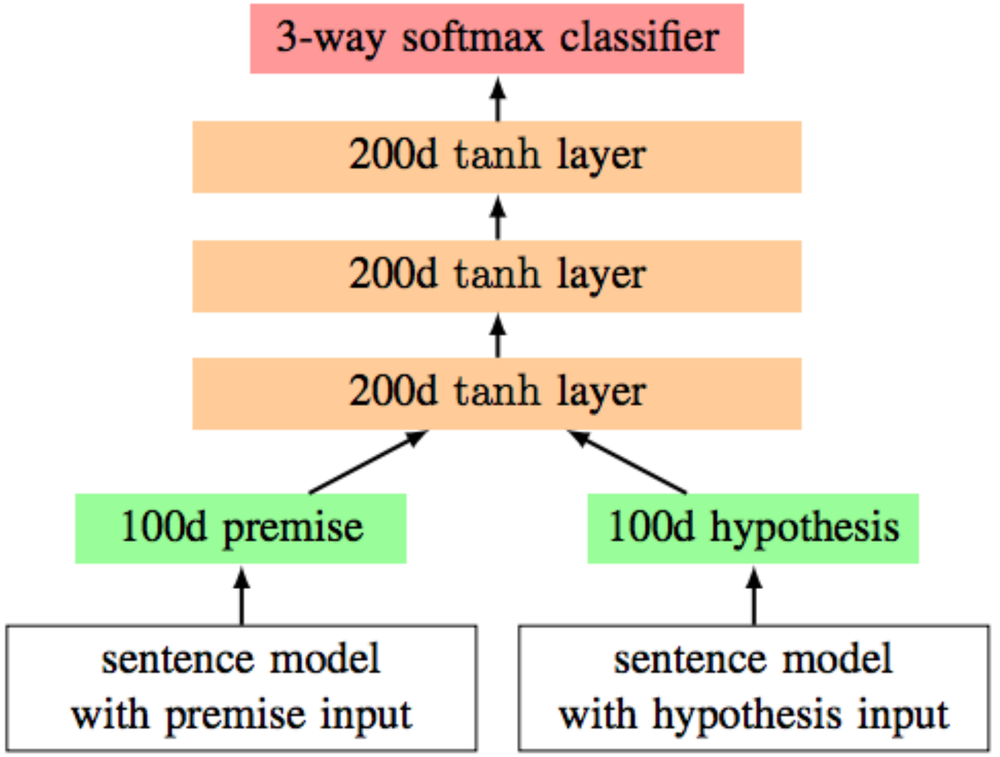
- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process

100D LSTM: 78% accuracy

300D LSTM: 80% accuracy
(Bowman et al., 2016)

300D BiLSTM: 83% accuracy
(Liu et al., 2016)

- Later: better models for this



Bowman et al. (2015)

Takeaways

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector

- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation, etc.