Language Models

Dan Klein
UC Berkeley

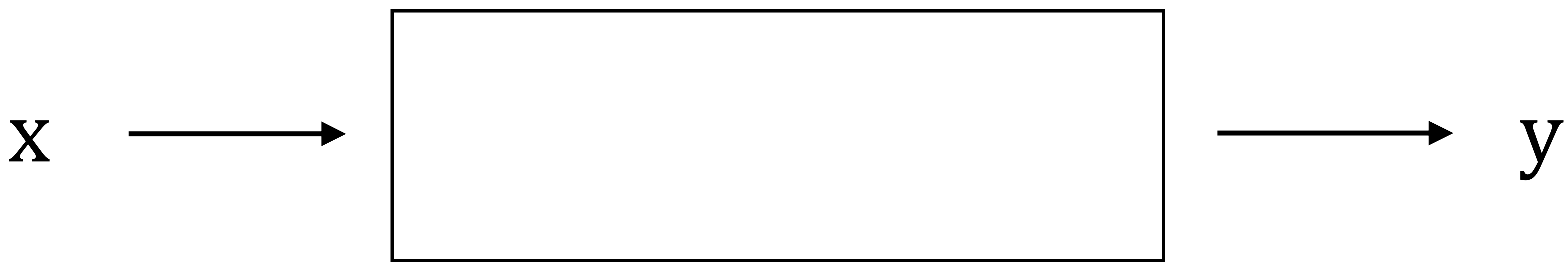
Neural Language Models

Bigram Models

$$w' \longrightarrow \boxed{\qquad\qquad} \longrightarrow P(w|w')$$

NOTE: This section will be done live,
slides are frames for live presentation

Bigram Models



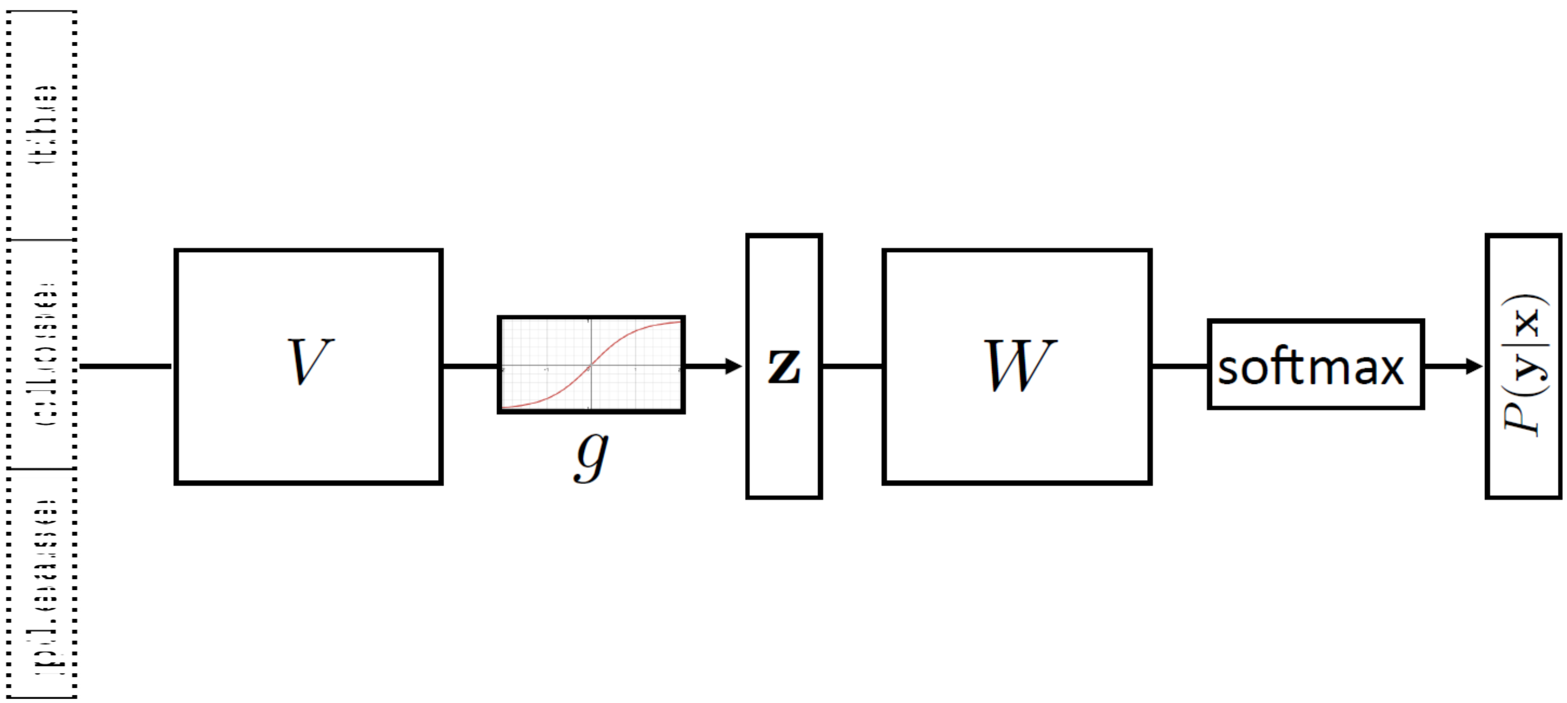
Learning a Model

Embeddings and Generalization

Some Efficiency Issues

Neural N-Gram Models

Neural N-Gram Models

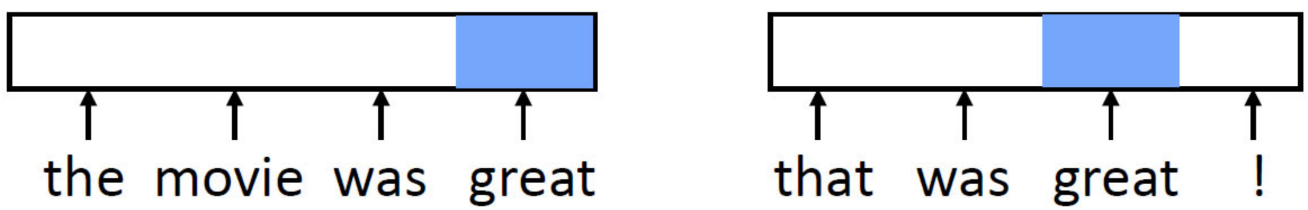


Recurrent NNs

Slides from Greg Durrett / UT Austin , Abigail See / Stanford

RNNs

- Feedforward NNs can’t handle variable length input: each position in the feature vector has fixed semantics



- These don’t look related (*great* is in two different orthogonal subspaces)

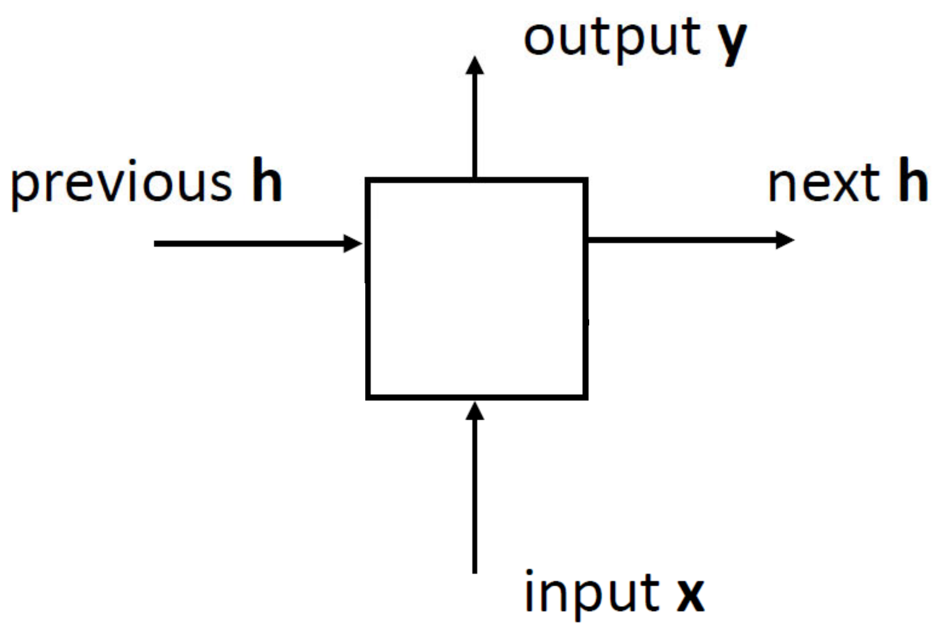
- Instead, we need to:

1) Process each word in a uniform way

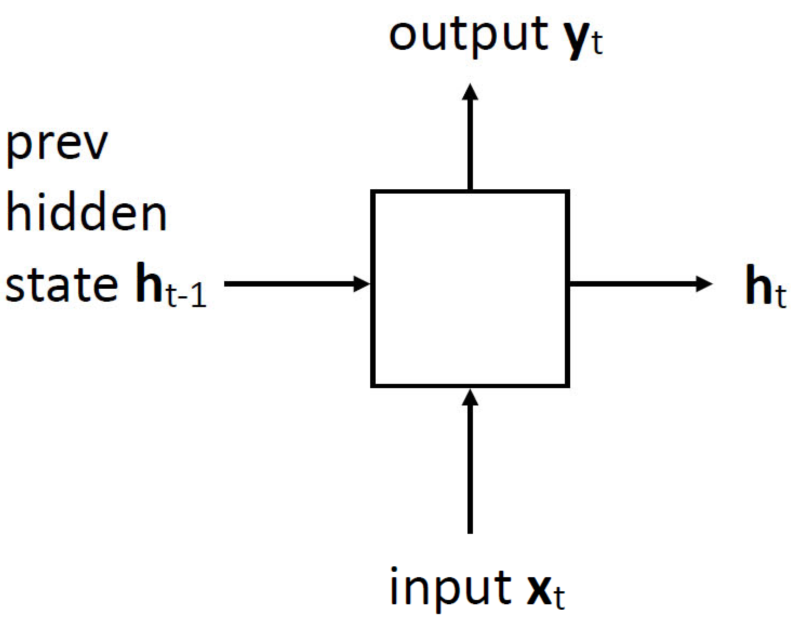
2) …while still exploiting the context that that token occurs in

General RNN Approach

- Cell that takes some input **x**, has some hidden state **h**, and updates that hidden state and produces output **y** (all vector-valued)



Basic RNNs



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

- Updates hidden state based on input and current hidden state

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

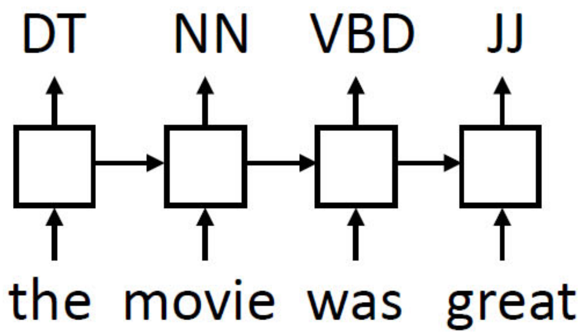
- Computes output from hidden state

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

Elman (1990)

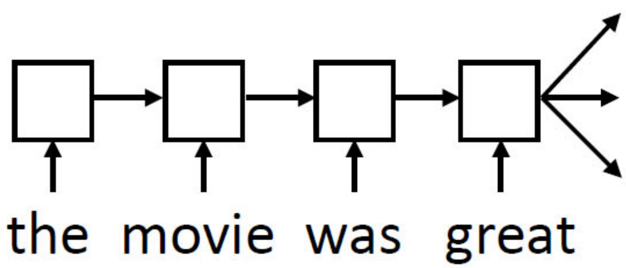
RNN Uses

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



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

- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

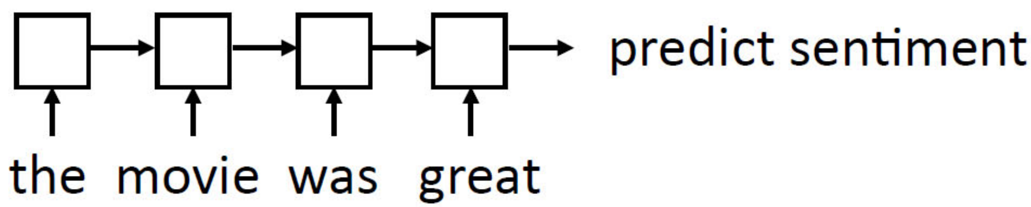


predict sentiment (matmul + softmax)

translate

paraphrase/compress

Training RNNs



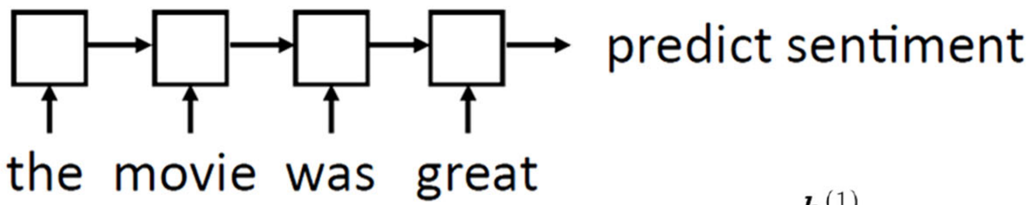
- “Backpropagation through time”: build the network as one big computation graph, some parameters are shared

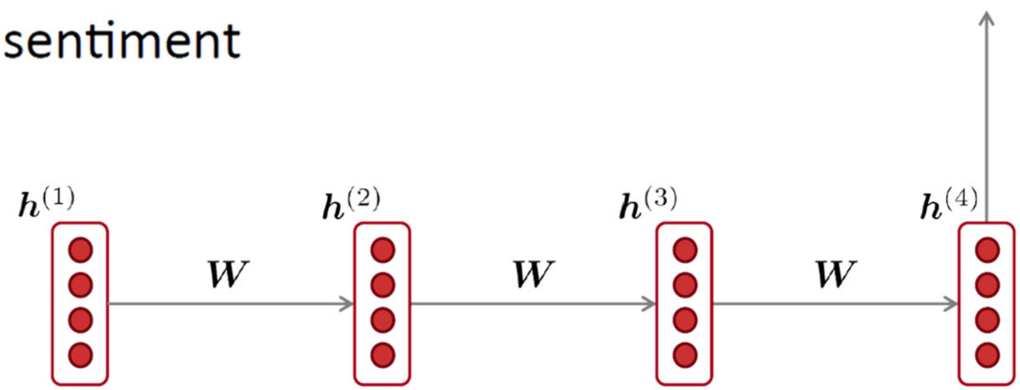
- RNN potentially needs to learn how to “remember” information for a long time!

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

- “Correct” parameter update is to do a better job of remembering the sentiment of *favorite*

Problem: Vanishing Gradients





- Contribution of earlier inputs decreases if matrices are contractive (first eigenvalue < 1), non-linearities are squashing, etc
- Gradients can be viewed as a measure of the effect of the past on the future
- That's a problem for optimization but also means that information naturally decays quickly, so model will tend to capture local information

Next slides adapted from Abigail See / Stanford

Core Issue: Information Decay

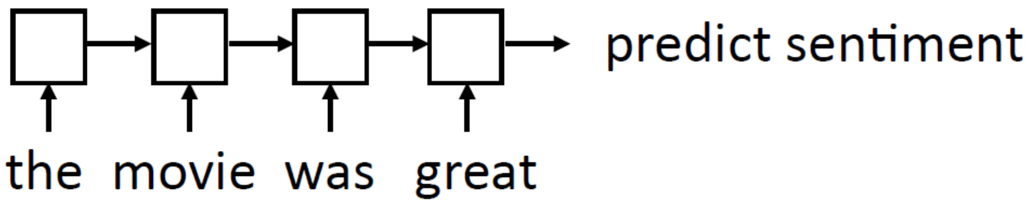
- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.

- In a vanilla RNN, the hidden state is constantly being rewritten

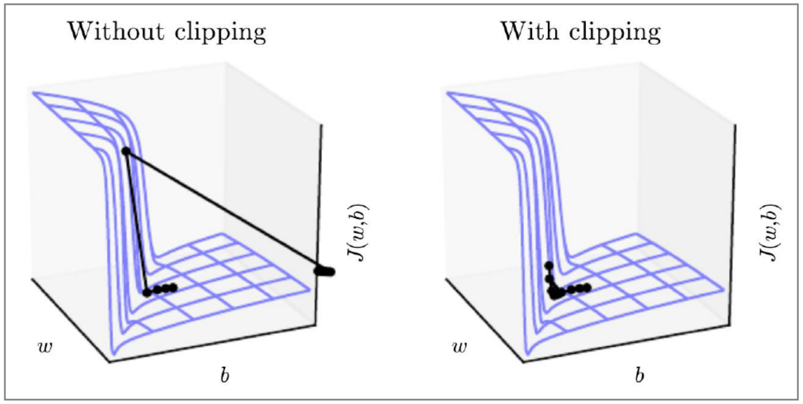
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- How about a RNN with separate memory?

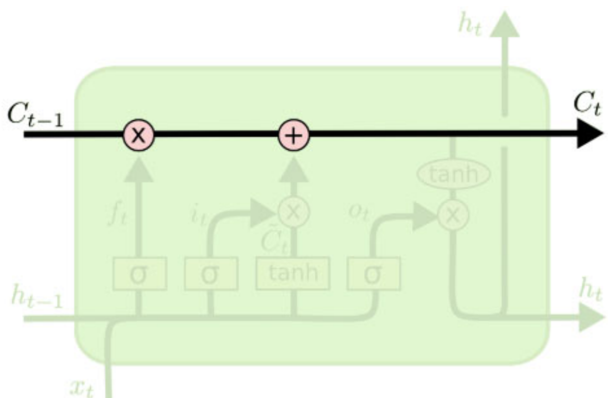
Problem: Exploding Gradients



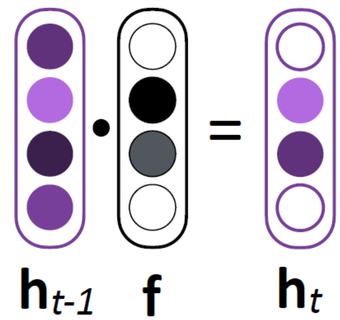
- Gradients can also be too large
  - Leads to overshooting / jumping around the parameter space
  - Common solution: gradient clipping



Key Idea: Propagated State



Cell State

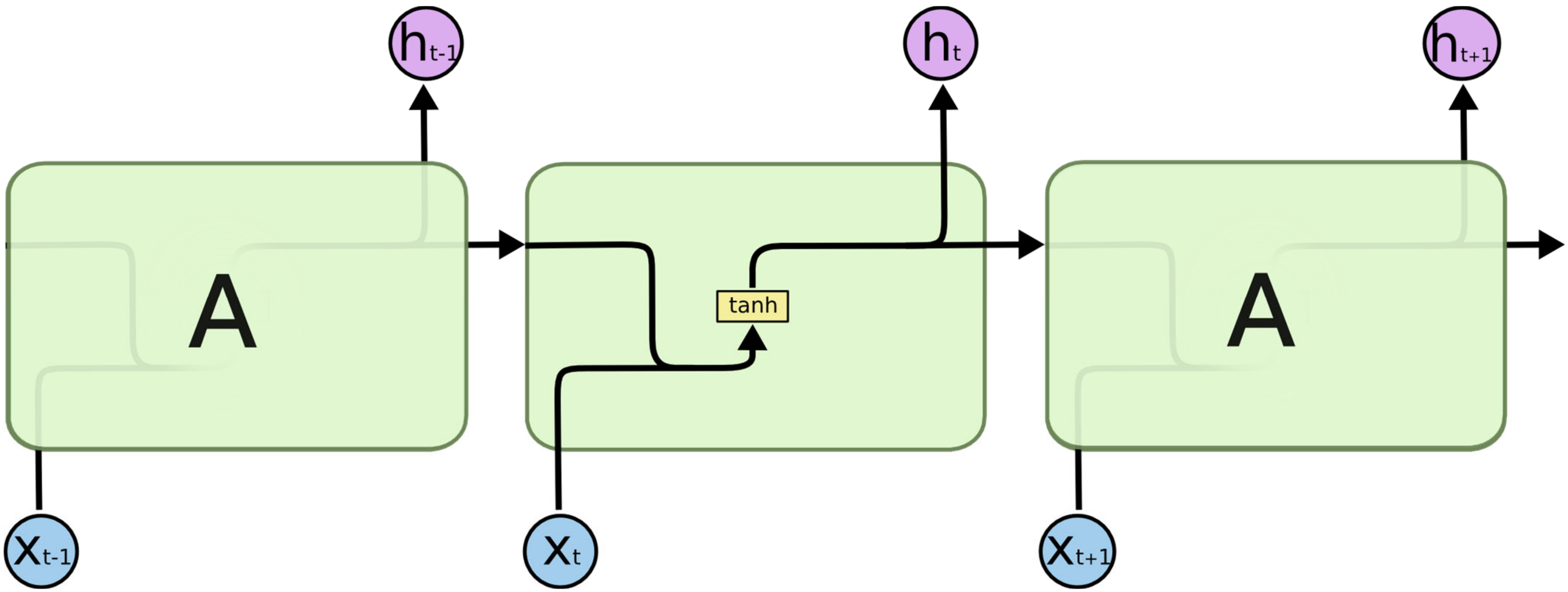


Gating

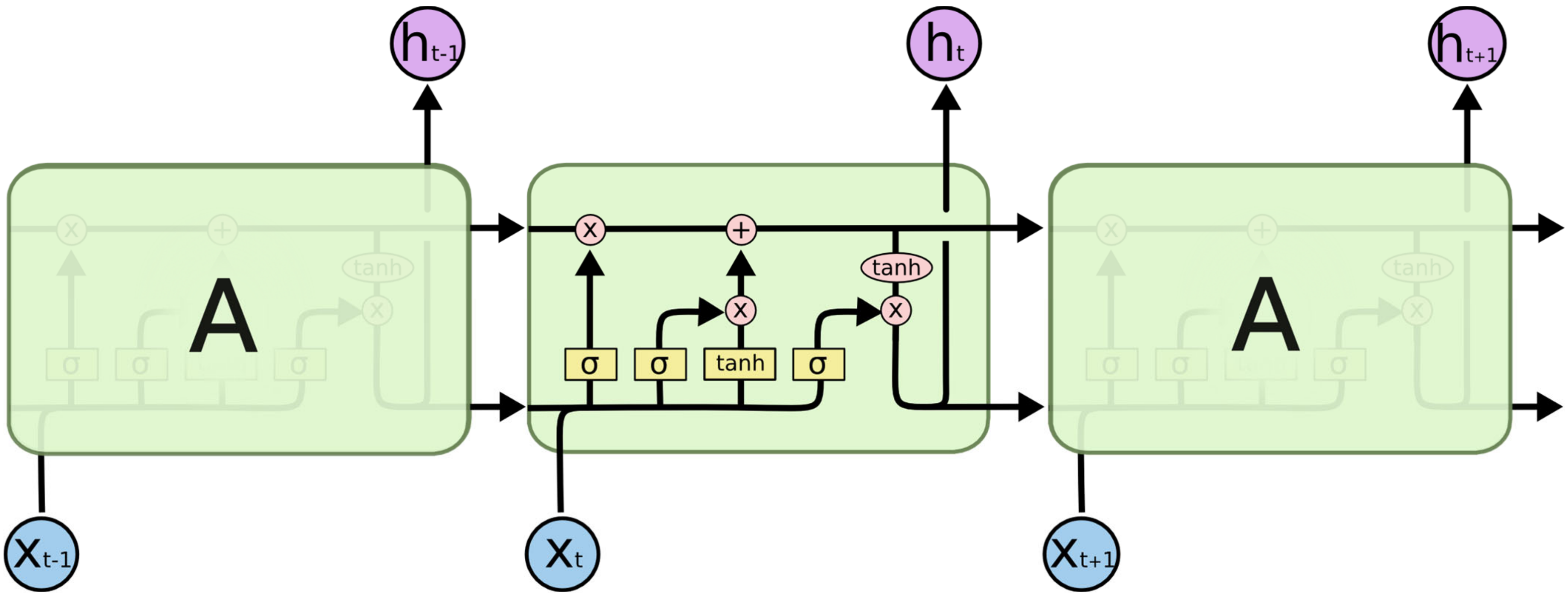
- Information decays in RNNs because it gets multiplied each time step
- Idea: have a channel called the *cell state* that by default just gets propagated (the “conveyer belt”)
- Gates make explicit decisions about what to add / forget from this channel

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

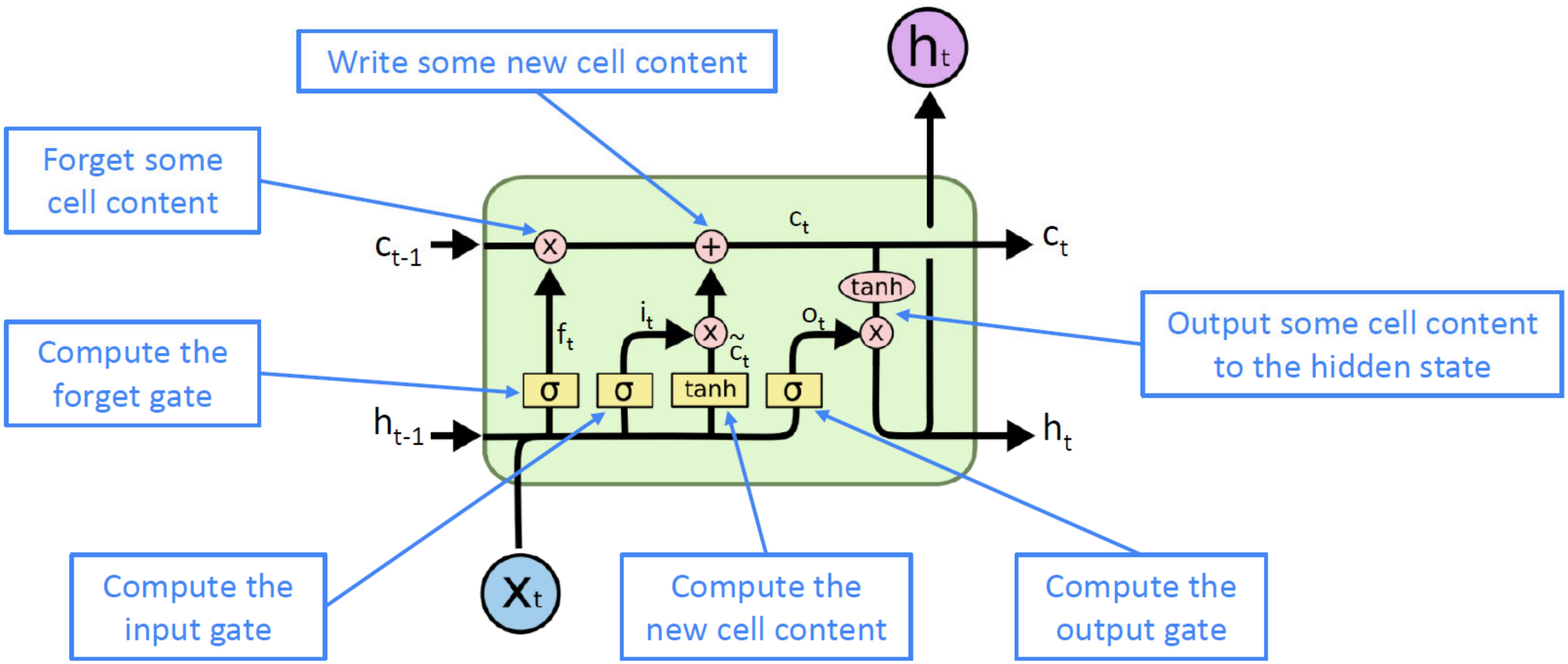
RNNs



LSTMs

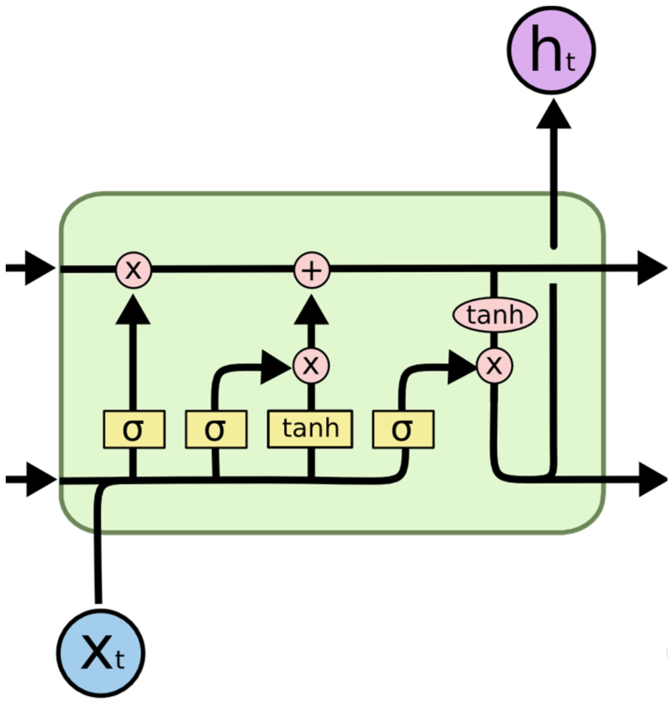


LSTMs



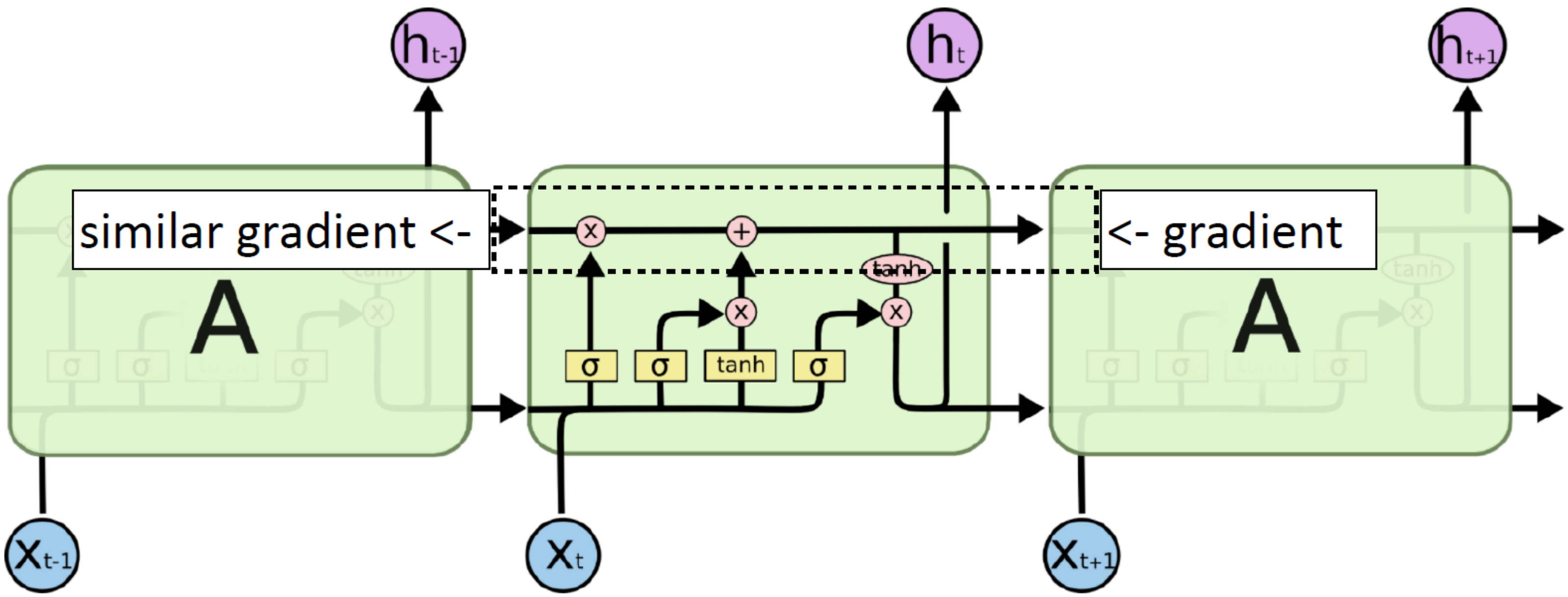
Write some new cell content
Forget some cell content
Compute the forget gate
Compute the input gate
Compute the new cell content
Compute the output gate
Output some cell content to the hidden state
h_t
c_{t-1}
c_t
c_t
f_t
i_t
$\tilde{c}_t$
o_t
σ
σ
tanh
σ
tanh
h_{t-1}
h_t
x_t

LSTMs



- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

What about the Gradients?



- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

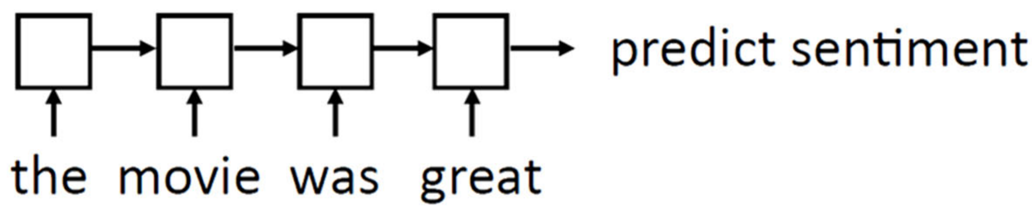
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Uses of RNNs

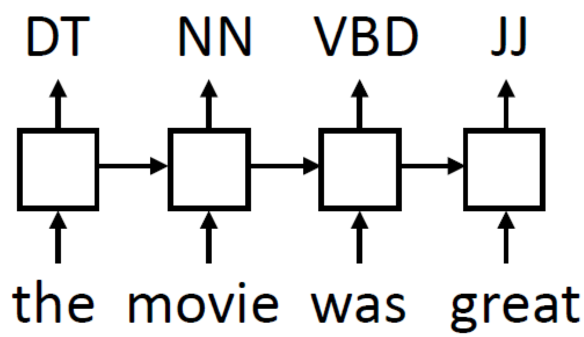
Slides from Greg Durrett / UT Austin

Reminder: Tasks for RNNs

- Sentence Classification (eg Sentiment Analysis)

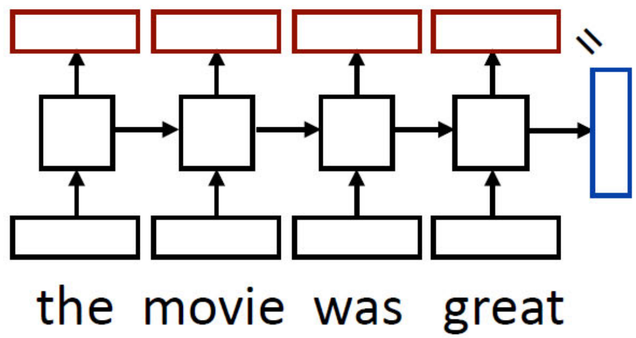


- Transduction (eg Part-of-Speech Tagging, NER)



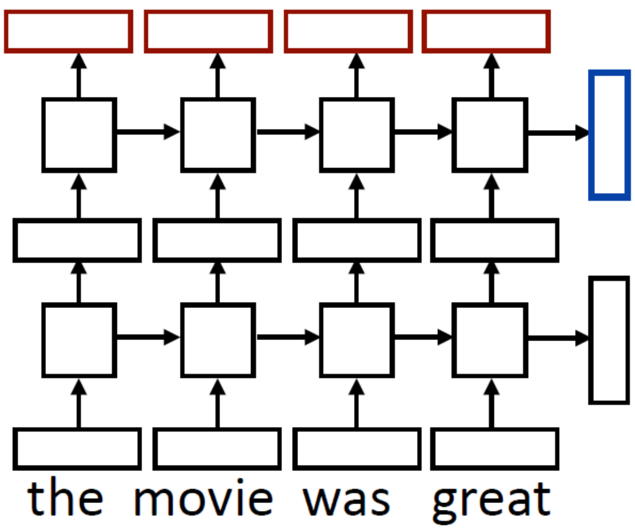
- Encoder/Decoder (eg Machine Translation)

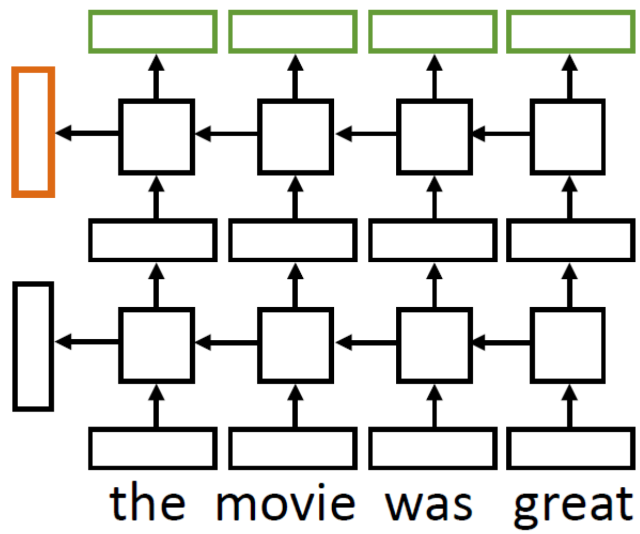
Encoder / Decoder Preview



- Encoding of the sentence — can pass this 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 and Bidirectional RNNs

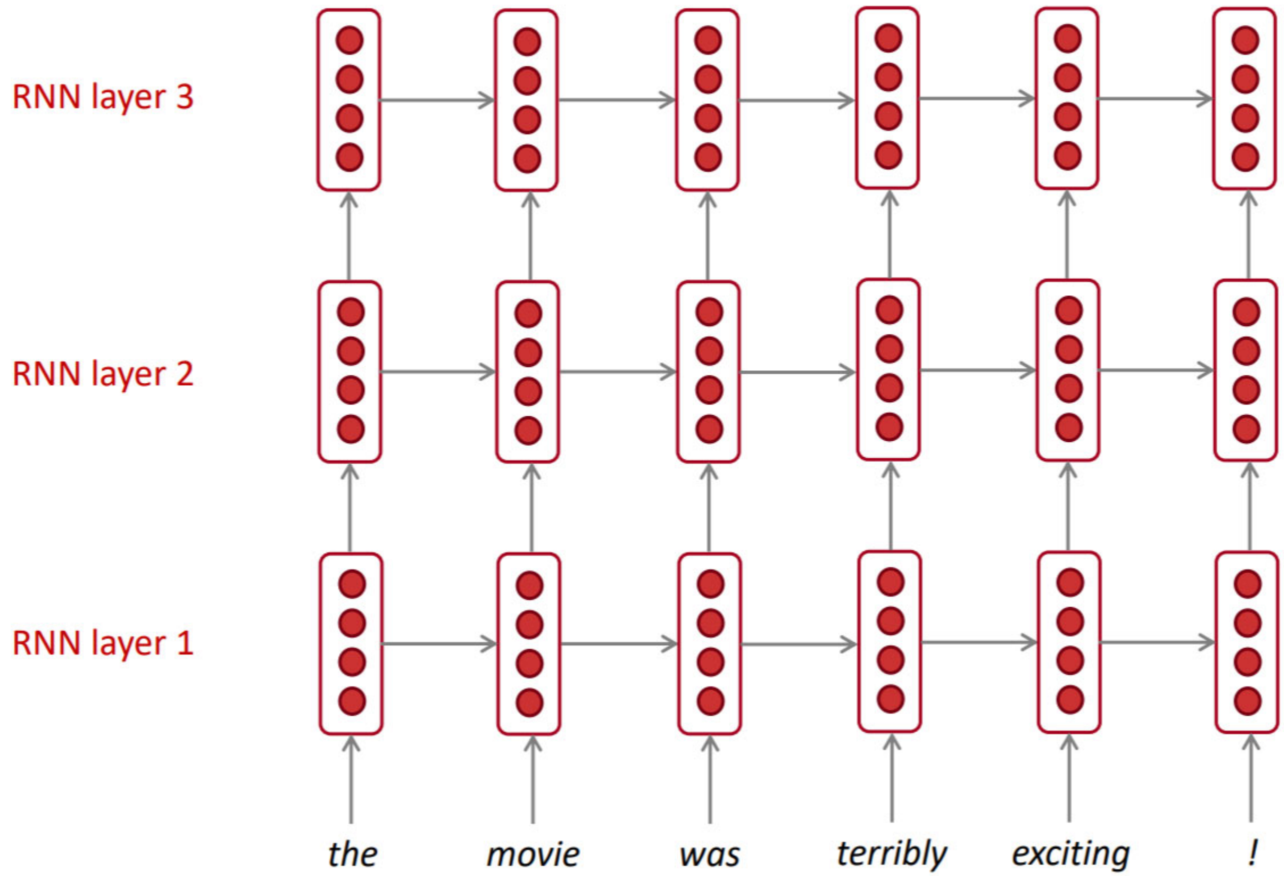




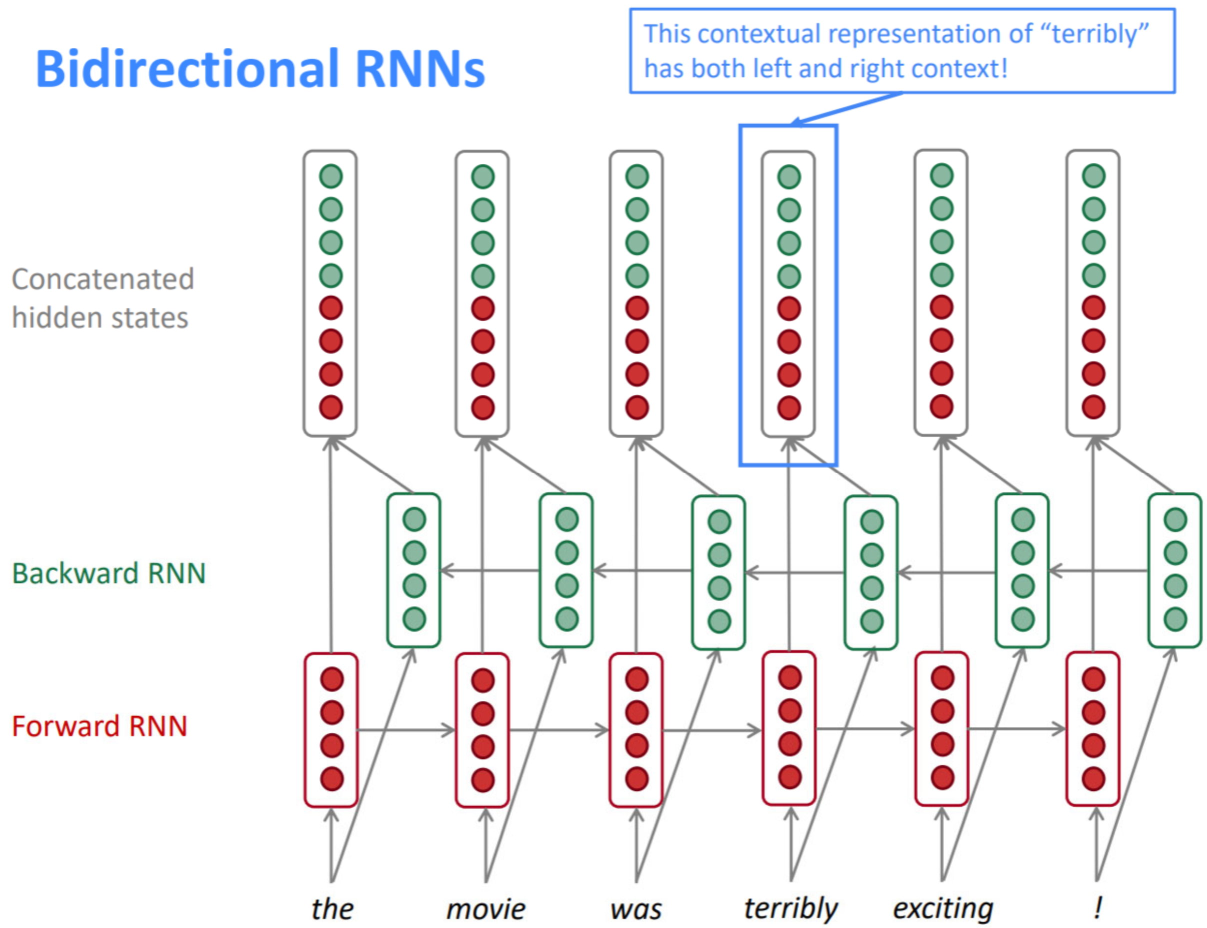
- Sentence classification based on concatenation of both final outputs

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

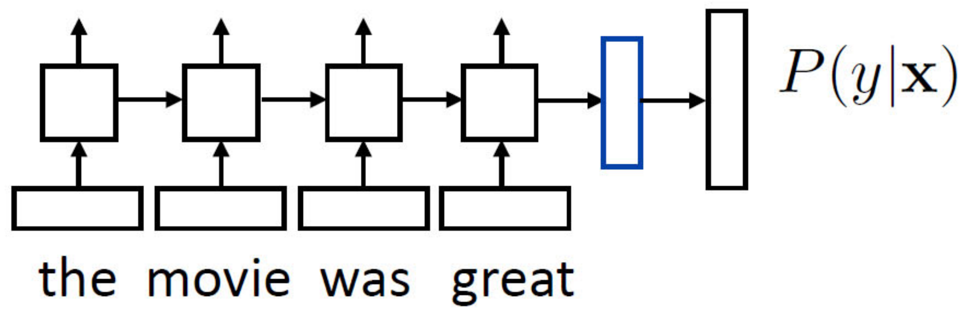
Multi-Layer RNNs



Bi-Directional RNNs

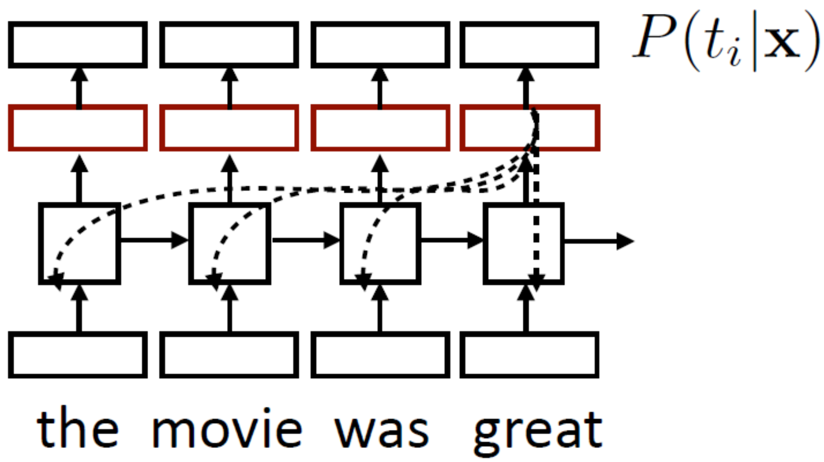


Training for Sentential Tasks



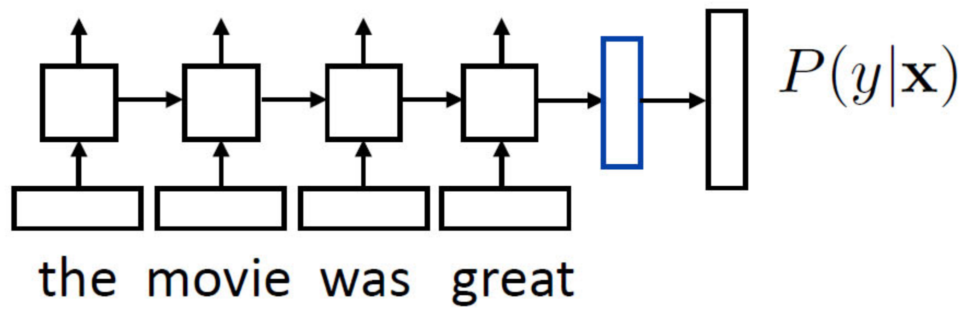
- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis

Training for Transduction Tasks



- 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)

Training for Sentential Tasks



- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis

Example Sentential Task: NL 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

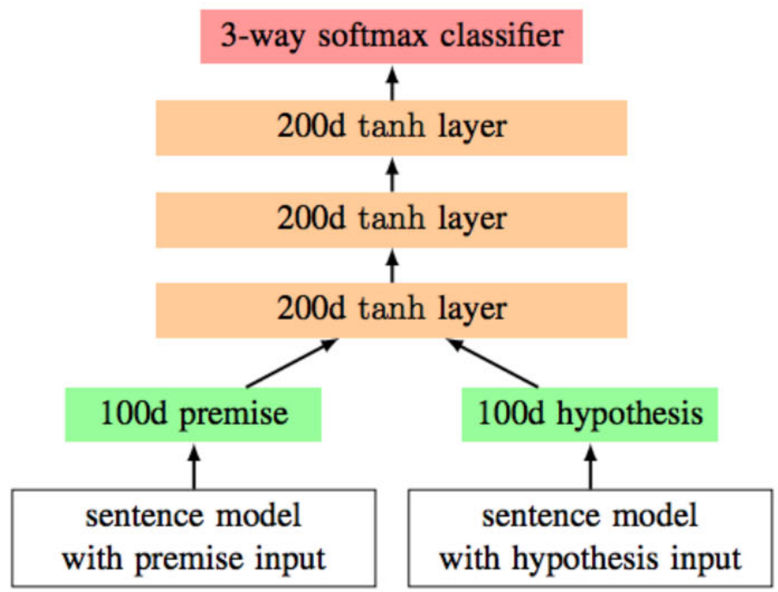
- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process

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



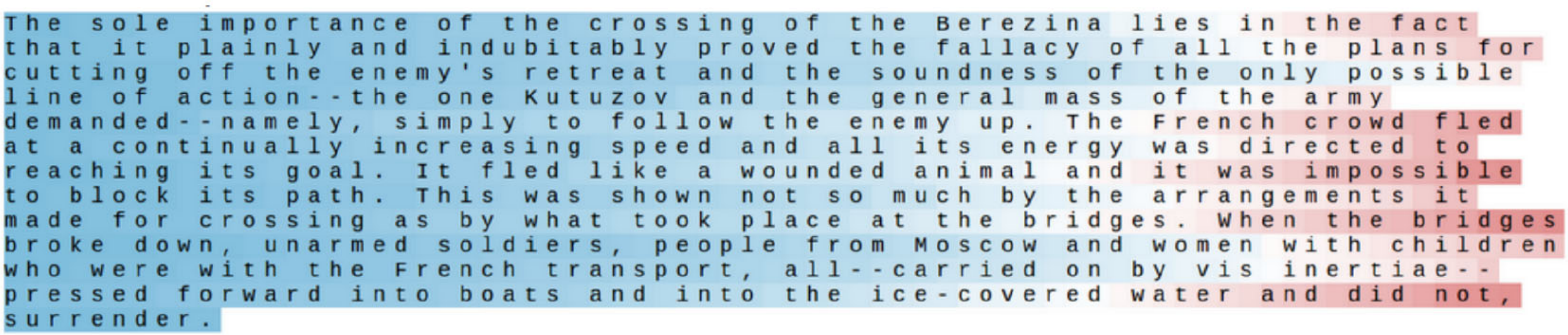
Bowman et al. (2015)

Visualizing RNNs

Slides from Greg Durrett / UT Austin

LSTMs Can Model Length

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n



Karpathy et al. (2015)

LSTMs Can Model Long-Term Bits

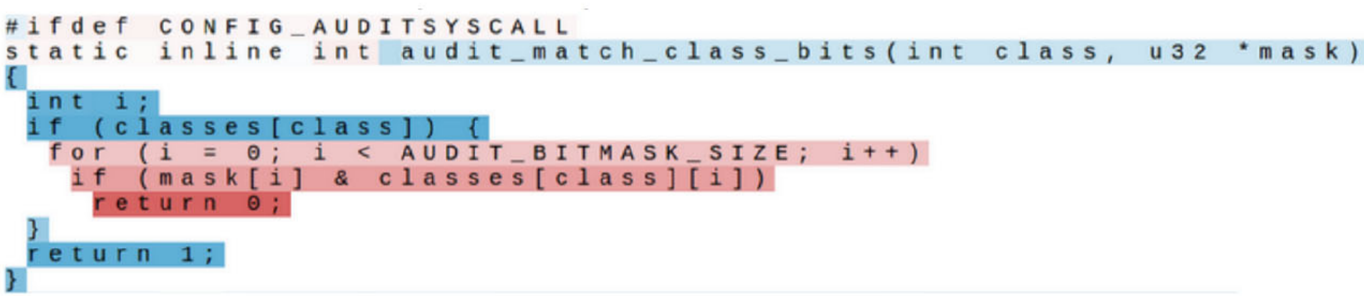
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: tells us if we're in a quote or not

> "You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.
>
> Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Karpathy et al. (2015)

LSTMs Can Model Stack Depth

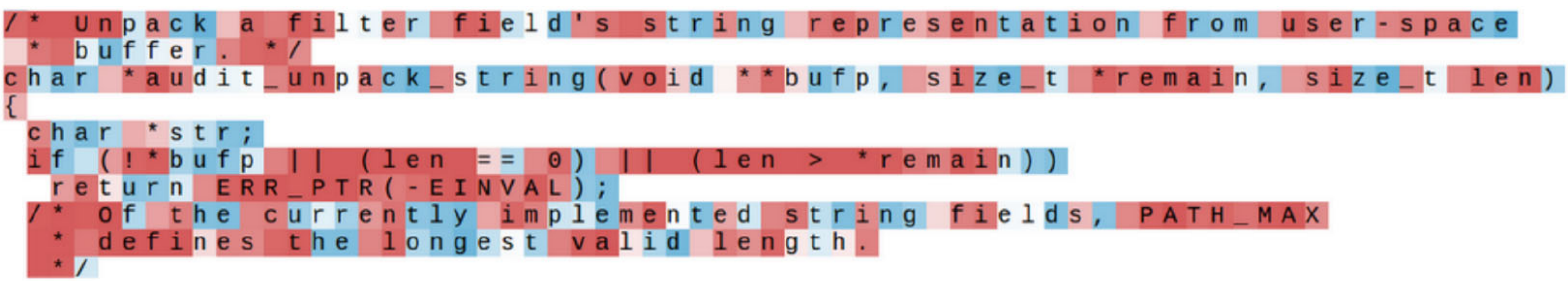
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation



Karpathy et al. (2015)

LSTMs Can Be Completely Inscrutable

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Uninterpretable: probably doing double-duty, or only makes sense in the context of another activation



Karpathy et al. (2015)