Recurrent Neural Networks

Wei Xu

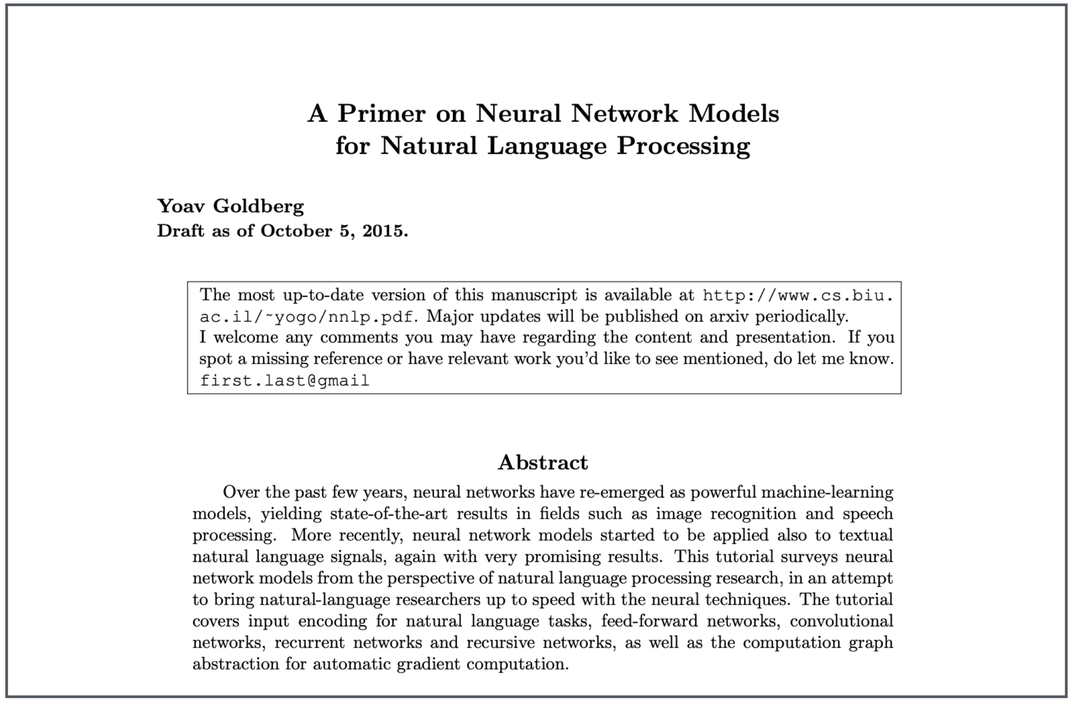
(many slides from Greg Durrett)

This Lecture

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

Administrivia

- Reading: RNNs
  - Eisenstein 7.6
  - Jurafsky and Martin, Chapter 9
  - Goldberg 10, 11 https://u.cs.biu.ac.il/~yogo/nnlp.pdf



A Primer on Neural Network Models for Natural Language Processing

Yoav Goldberg
Draft as of October 5, 2015.

The most up-to-date version of this manuscript is available at `http://www.cs.biu.ac.il/~yogo/nnlp.pdf`. Major updates will be published on arxiv periodically.
I welcome any comments you may have regarding the content and presentation. If you spot a missing reference or have relevant work you'd like to see mentioned, do let me know.
`first.last@gmail`

Abstract

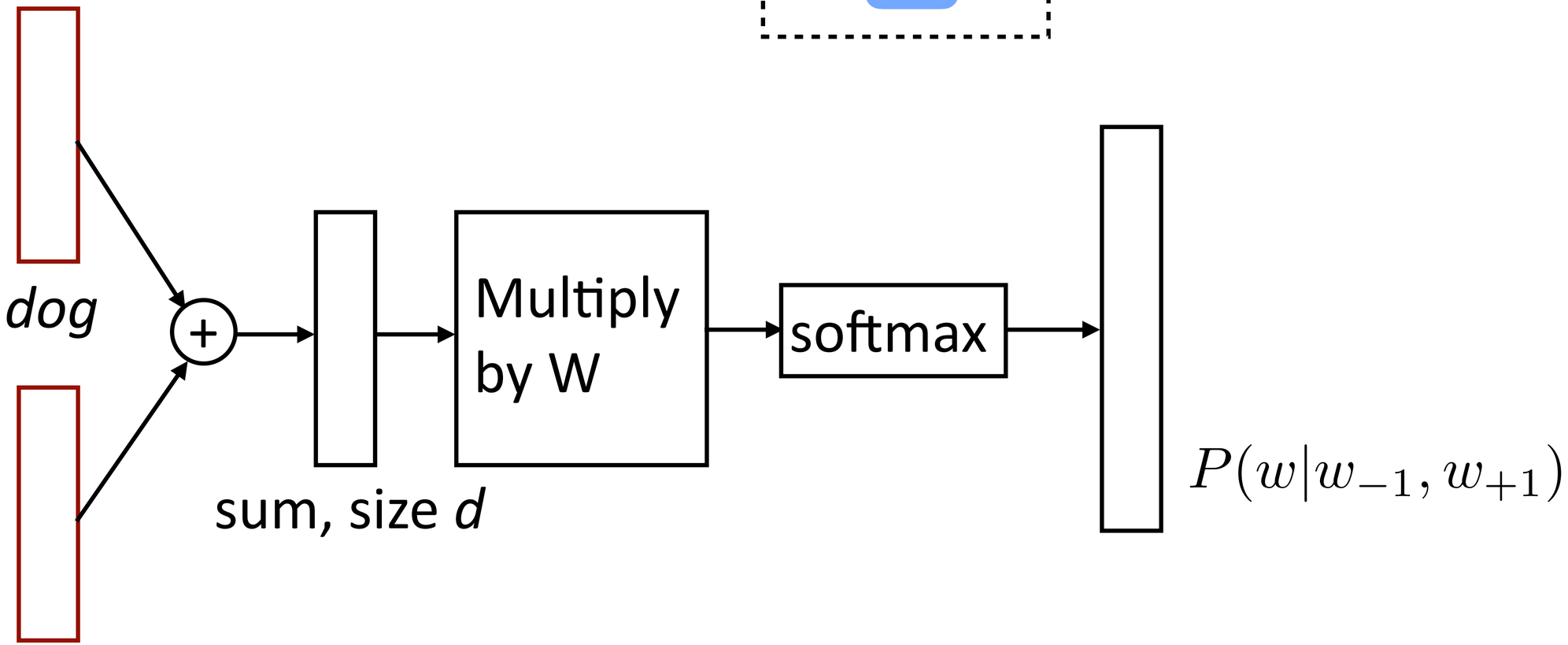
Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textual natural language signals, again with very promising results. This tutorial surveys neural network models from the perspective of natural language processing research, in an attempt to bring natural-language researchers up to speed with the neural techniques. The tutorial covers input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.

Recall: Word2vec - Continuous Bag-of-Words

- Predict word from context

the *dog* ***bit*** *the* *man*

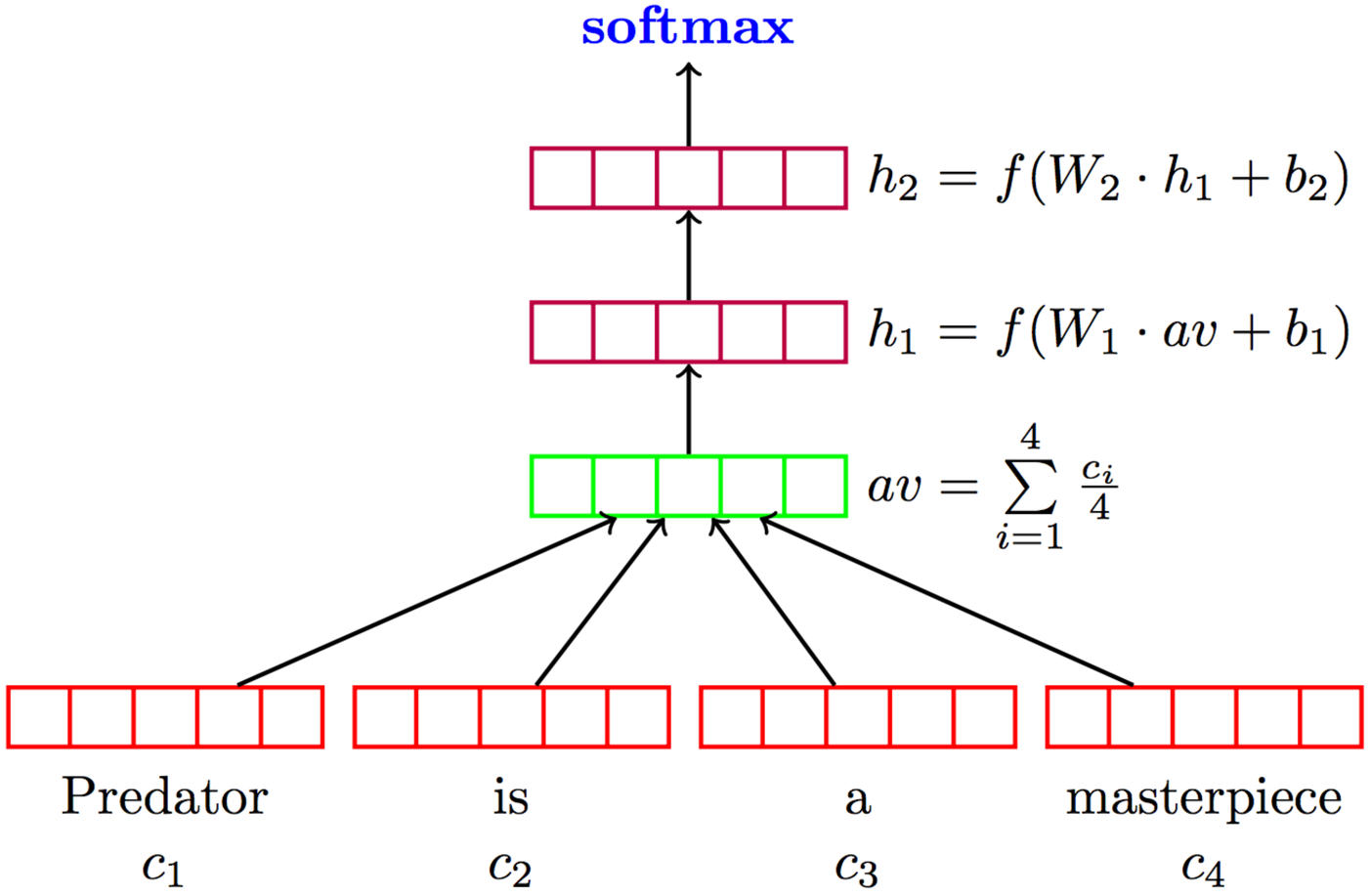
Mikolov et al. (2013)



- Matrix factorization approaches useful for learning vectors from really large data

Recall: Neural Bag-of-Words

- feedforward neural network on average of word embeddings from input



Iyyer et al. (2015)

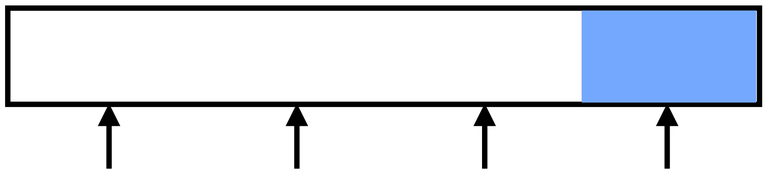
Compositional Semantics

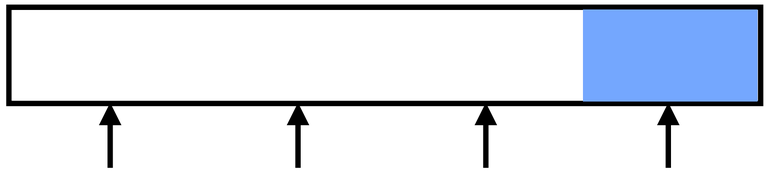
- What if we want embedding representations for whole sentences?
- Is there a way we can compose vectors to make sentence representations? Summing? Concatenating? RNNs?

RNN Basics

RNN Motivation

- Feedforward NNs isn't the best to handle sentences with variable length input, words with multiple senses, or take word order and context (e.g. "not good") into consideration





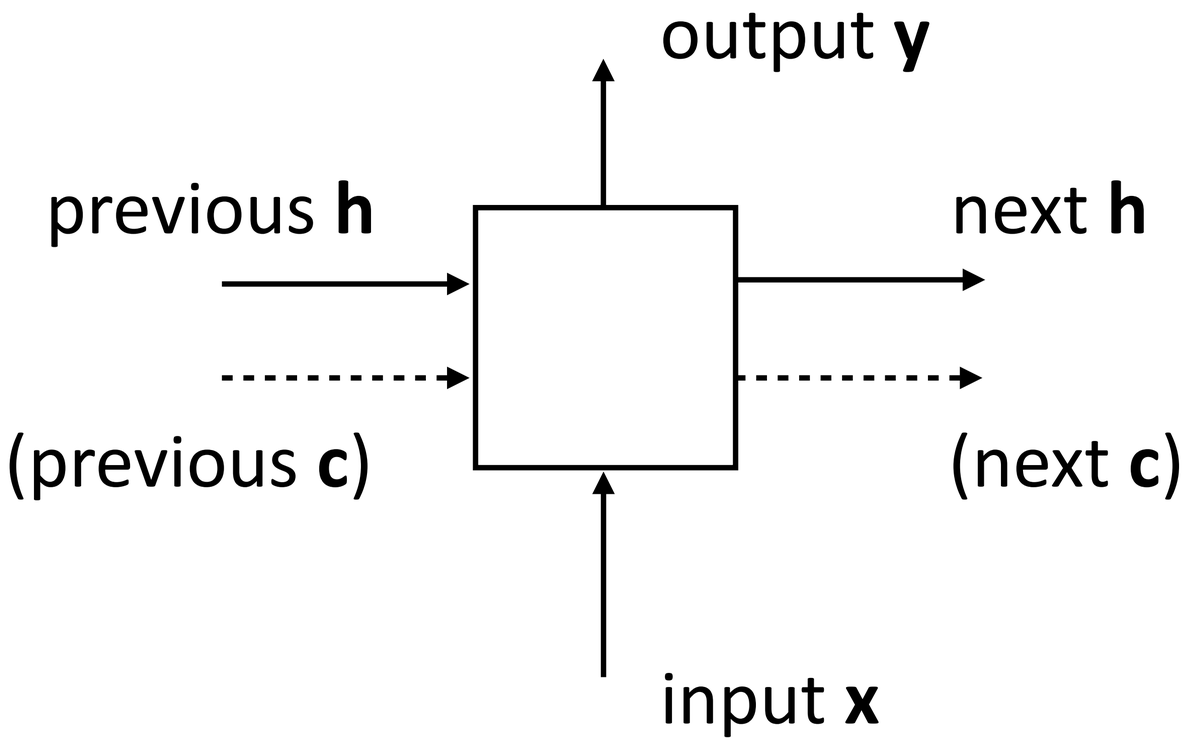
- Instead, we need to:

1) Process each word in a uniform way

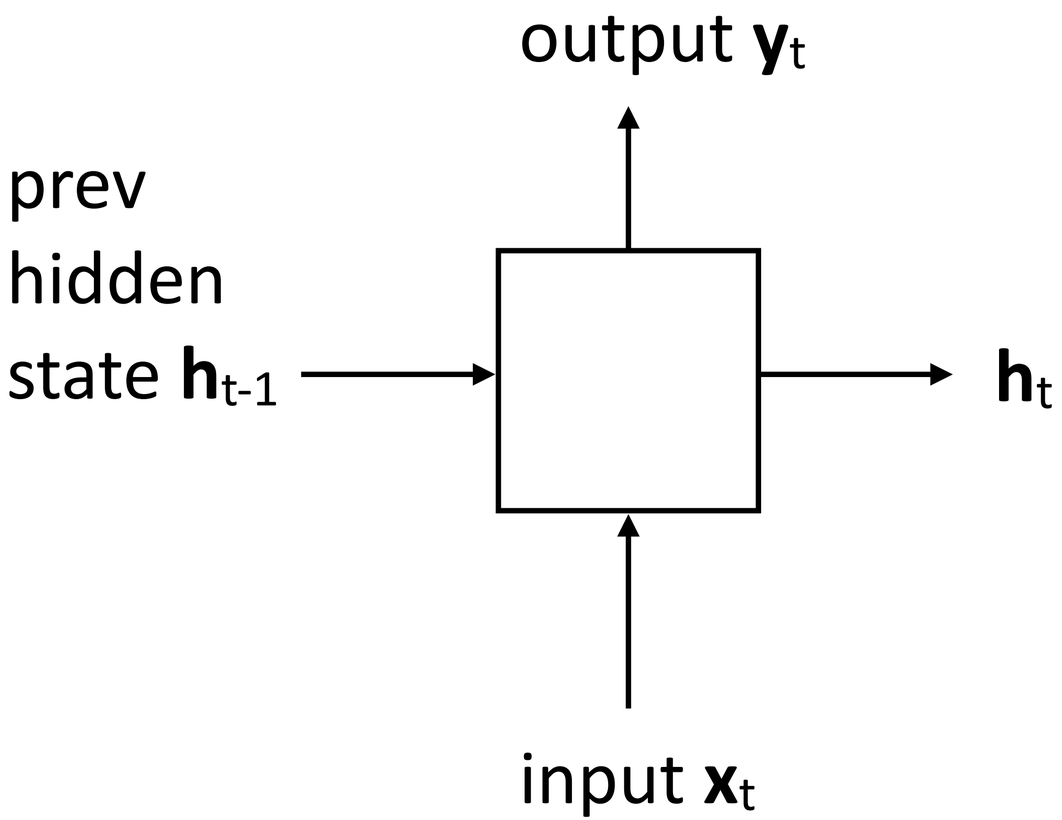
2) …while still exploiting the (structure of) context that that token occurs in

RNN Abstraction

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



Elman Networks



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

- Updates hidden state based on input and current hidden state

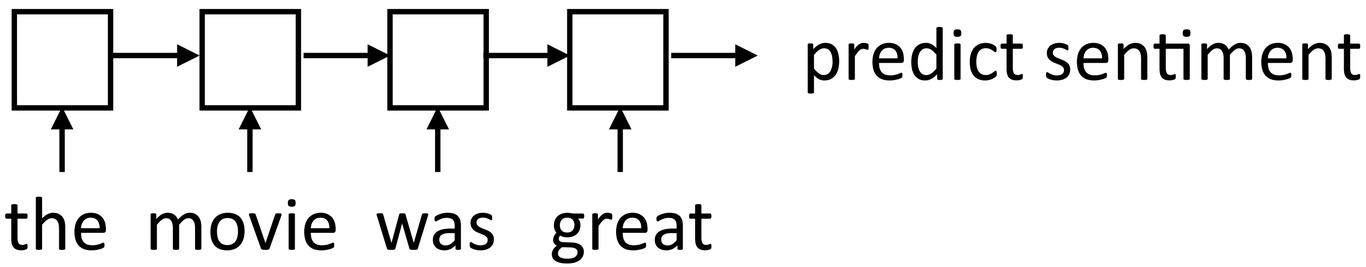
$$\mathbf{y}_t = \tanh(U\mathbf{h_t} + \mathbf{b}_y)$$

- Computes output from hidden state

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

Elman (1990)

Training Elman Networks



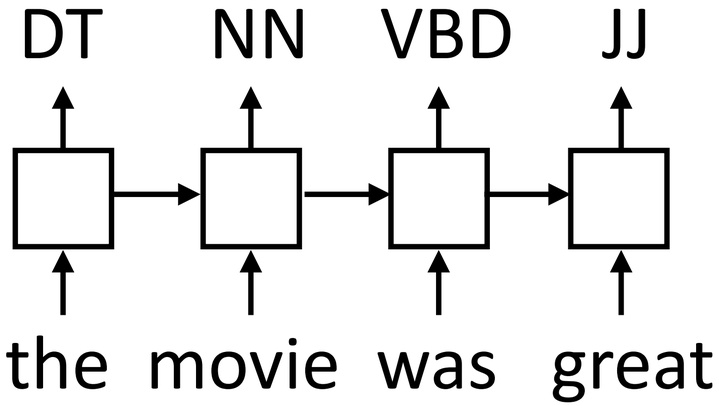
- "Backpropagation through time": build the network as one big computation graph, some parameters are shared
- RNN potentially needs to learn how to "remember" information for a long time!

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*

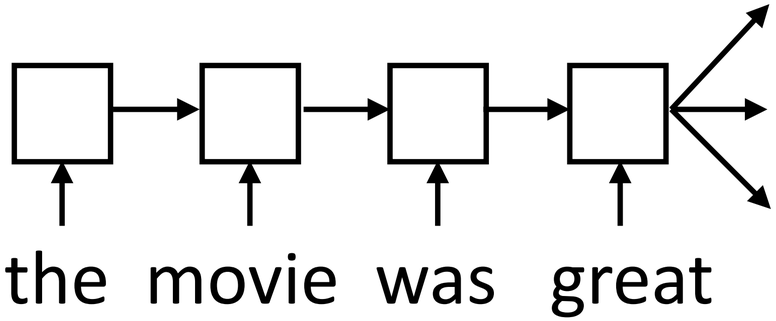
RNN Uses

- 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

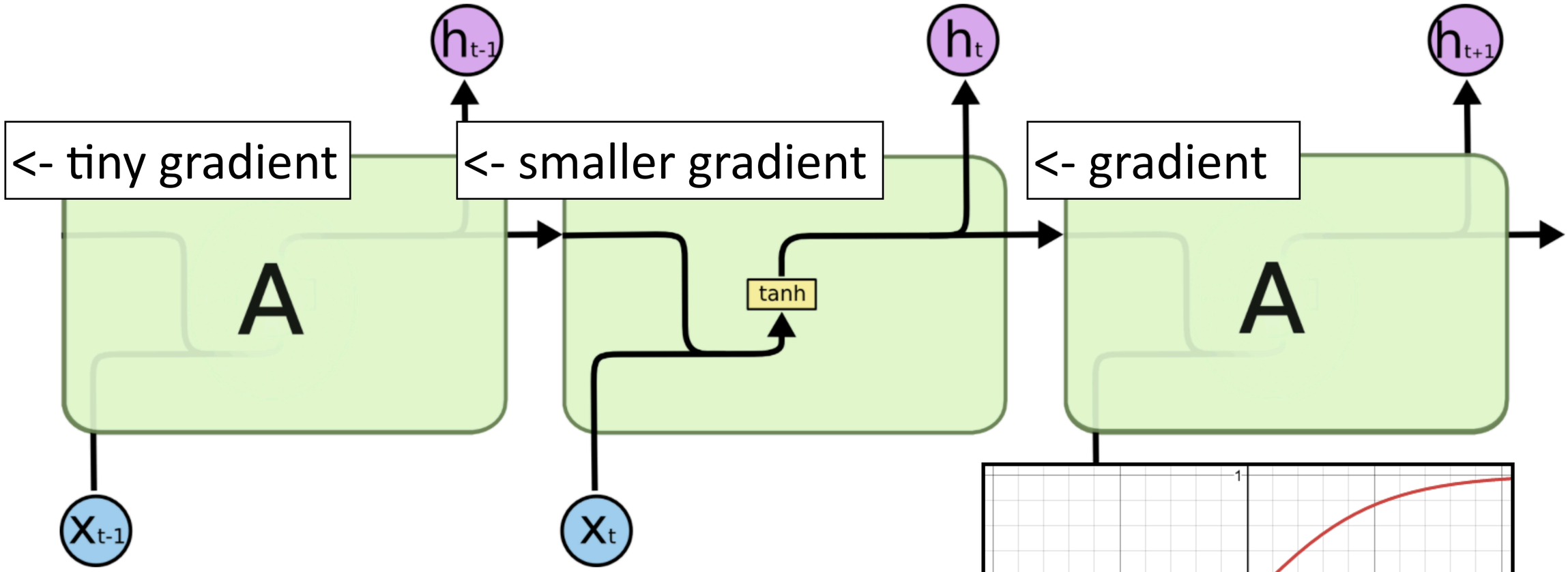


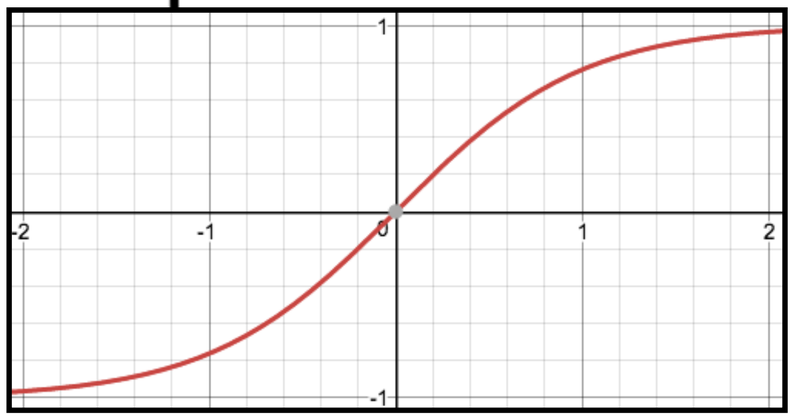
predict sentiment (matmul + softmax)

translate

paraphrase/compress

Vanishing Gradient





- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0

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

LSTMs/GRUs

Gated Connections

- Designed to fix "vanishing gradient" problem using *gates*

$$\mathbf{h}_t = \mathbf{h}_{t-1} \odot \mathbf{f} + \text{func}(\mathbf{x}_t)$$

gated

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

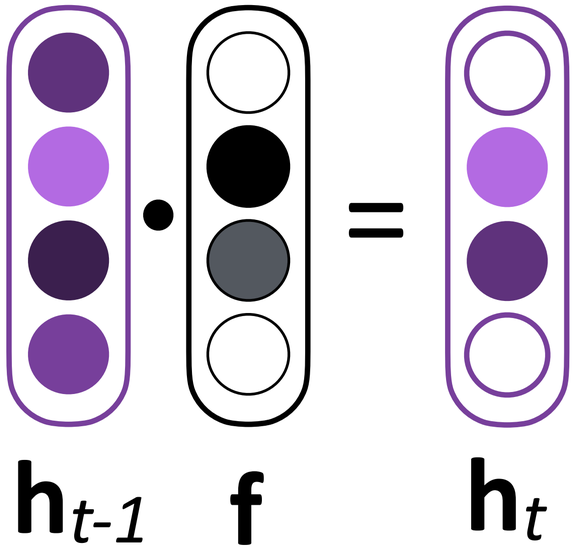
Elman

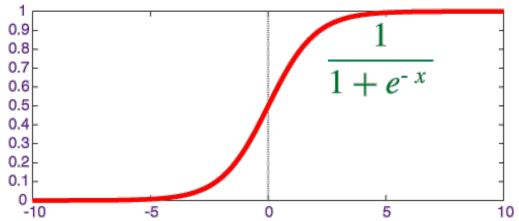
- Vector-valued "forget gate" **f** computed based on input and previous hidden state

$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$

- Sigmoid: elements of **f** are in (0, 1)

- If **f** ≈ **1**, we simply sum up a function of all inputs — gradient doesn't vanish!





LSTMs

- "Cell" **c** in addition to hidden state **h**

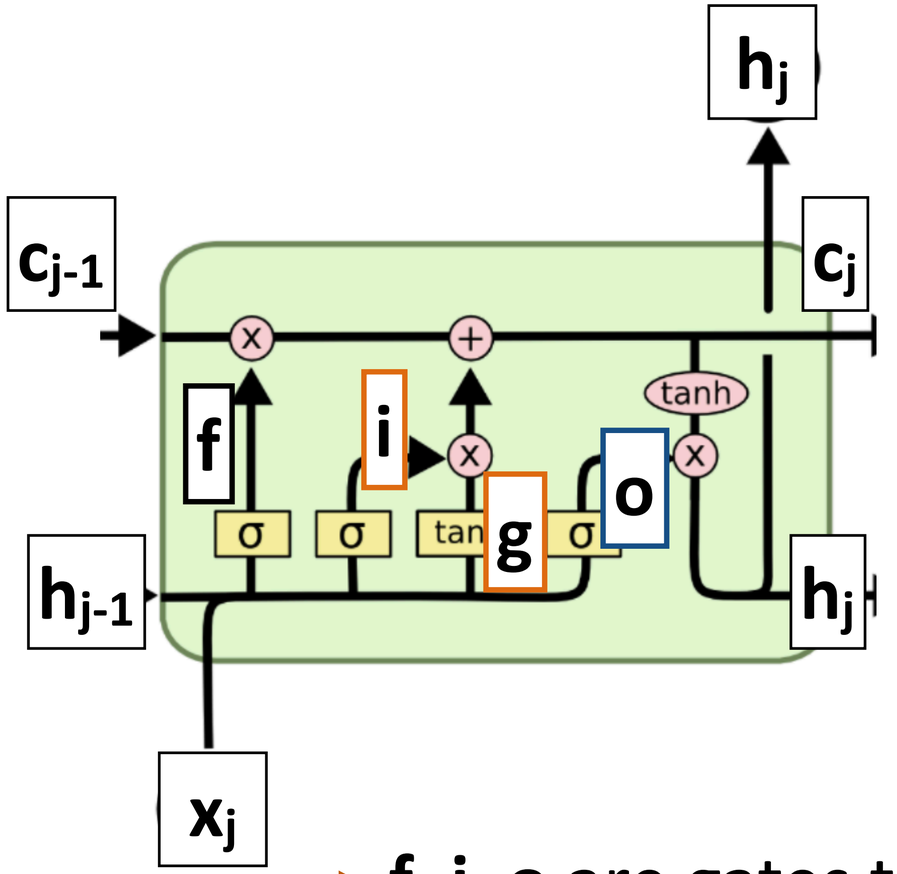
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\text{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- Vector-valued forget gate **f** depends on the **h** hidden state

$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$

- Basic communication flow: **x** -> **c** -> **h ->** output**,** each step of this process is gated in addition to gates from previous timesteps

LSTMs



$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\mathbf{g} = \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}})$$

$$\mathbf{i} = \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}})$$

$$\mathbf{h_j} = \tanh(\mathbf{c_j}) \odot \mathbf{o}$$

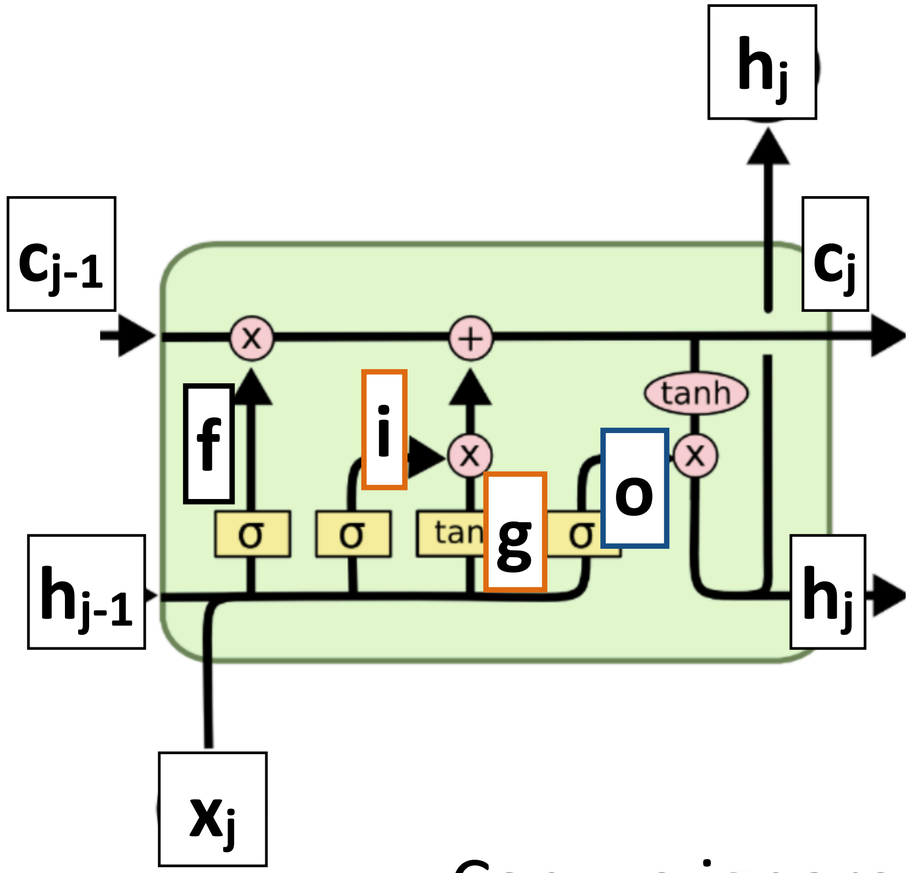
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **f**, **i**, **o** are gates that control information flow
- **g** reflects the main computation of the cell

Hochreiter & Schmidhuber (1997)

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

LSTMs



$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

$$\mathbf{g} = \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}})$$

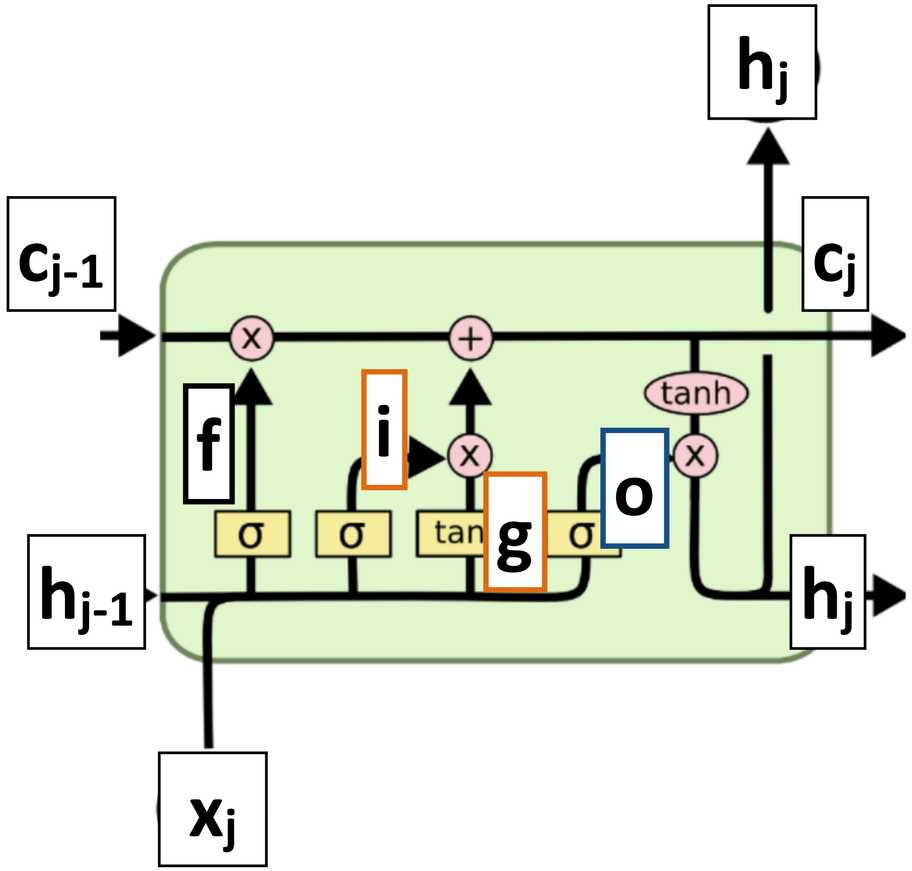
$$\mathbf{i} = \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}})$$

$$\mathbf{h_j} = \tanh(\mathbf{c_j}) \odot \mathbf{o}$$

$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- Can we ignore the old value of **c** for this timestep?
- Can an LSTM sum up its inputs **x**?
- Can we ignore a particular input **x**?
- Can we output something without changing **c**?

LSTMs

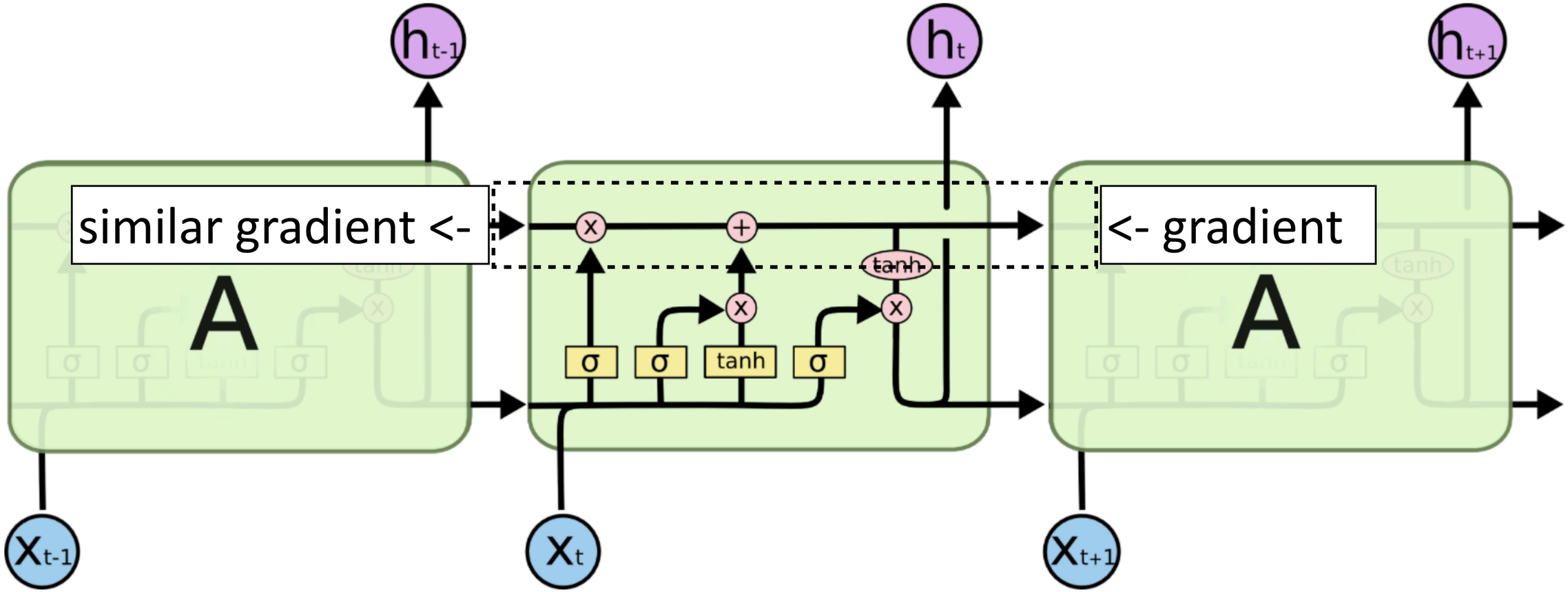


- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Goldberg lecture notes

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

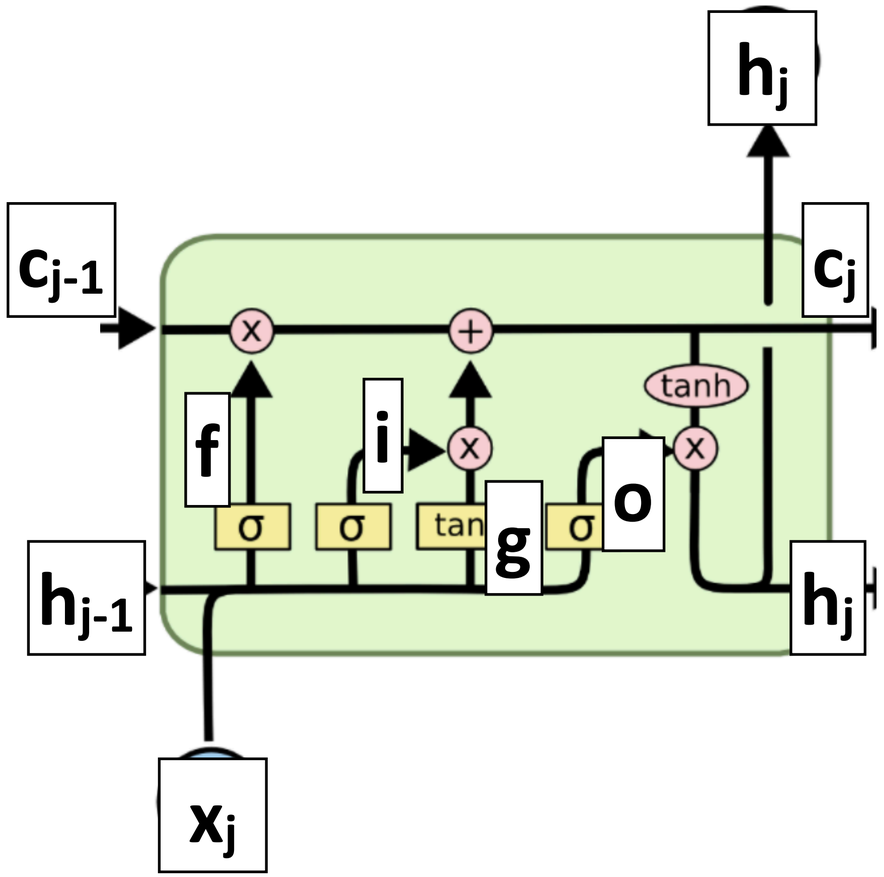
LSTMs



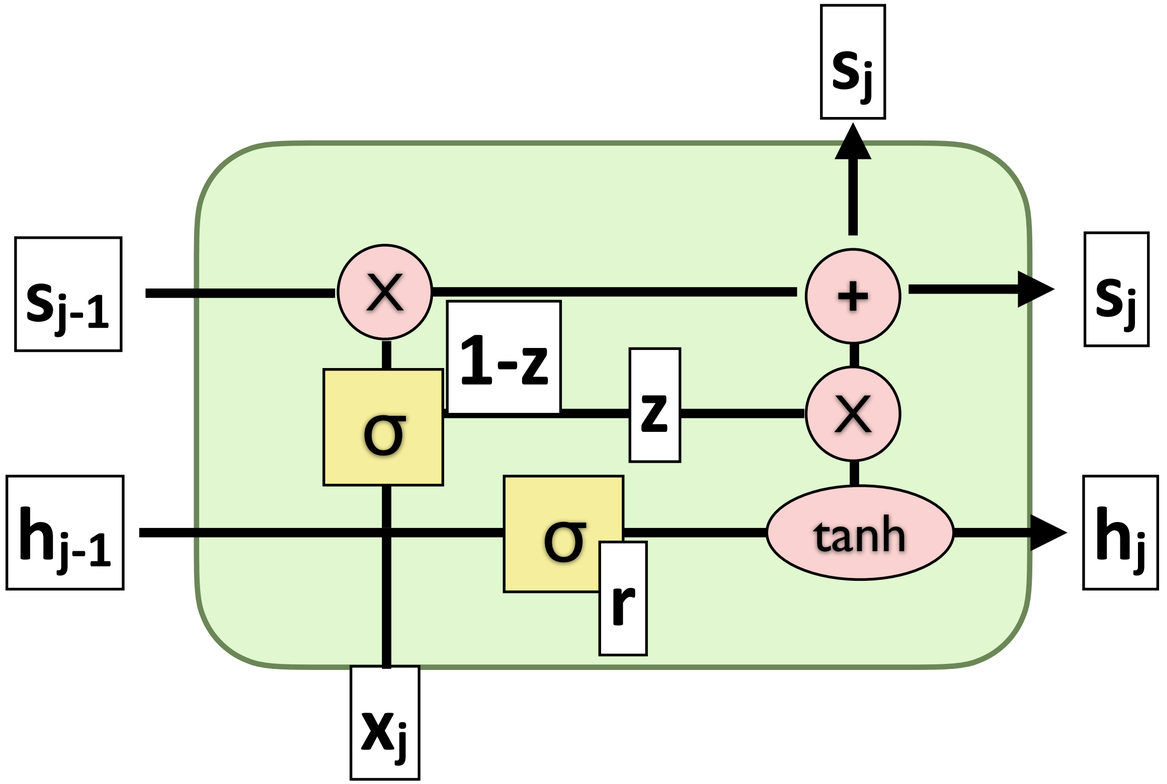
- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

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

GRUs



- LSTM: more complex and slower, may work a bit better



- GRU: faster, a bit simpler
- Two gates: **z** (forget, mixes **s** and **h**) and **r** (mixes **h** and **x**)

GRUs

- Also solves the vanishing gradient problem, simpler than LSTM

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \text{func}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

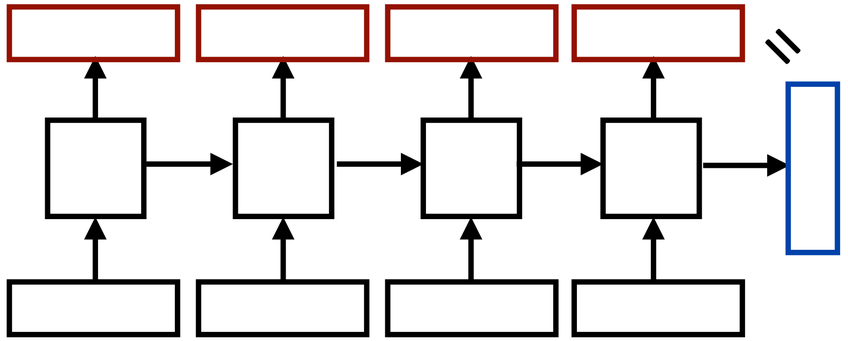
$$\mathbf{z} = \sigma(W\mathbf{x}_t + U\mathbf{h}_{t-1})$$

- **z** controls mixing of hidden state **h** with new input **x**

- Faster to train and sometimes work better than LSTMs

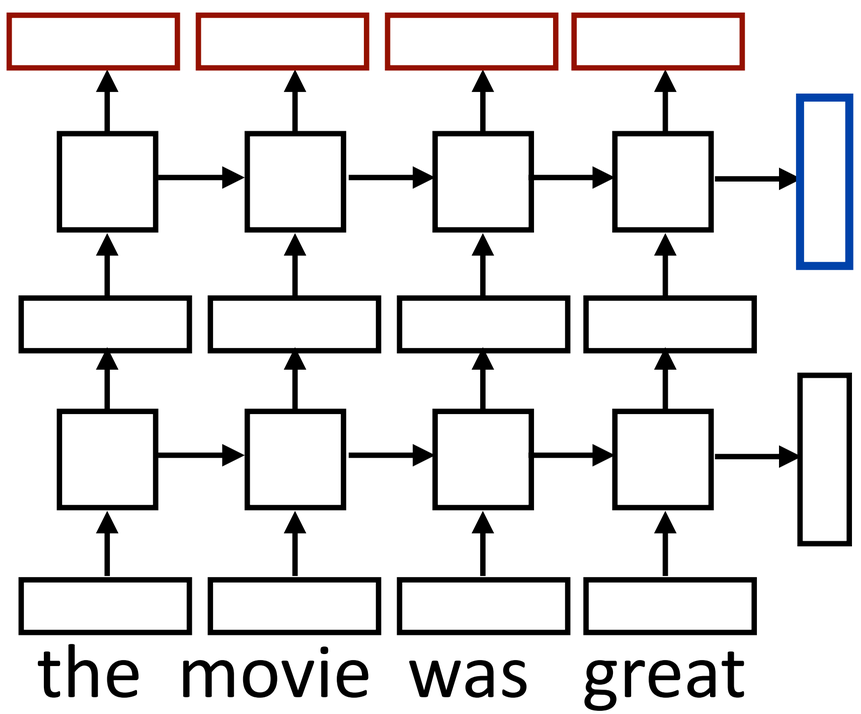
Cho et al. (2014)

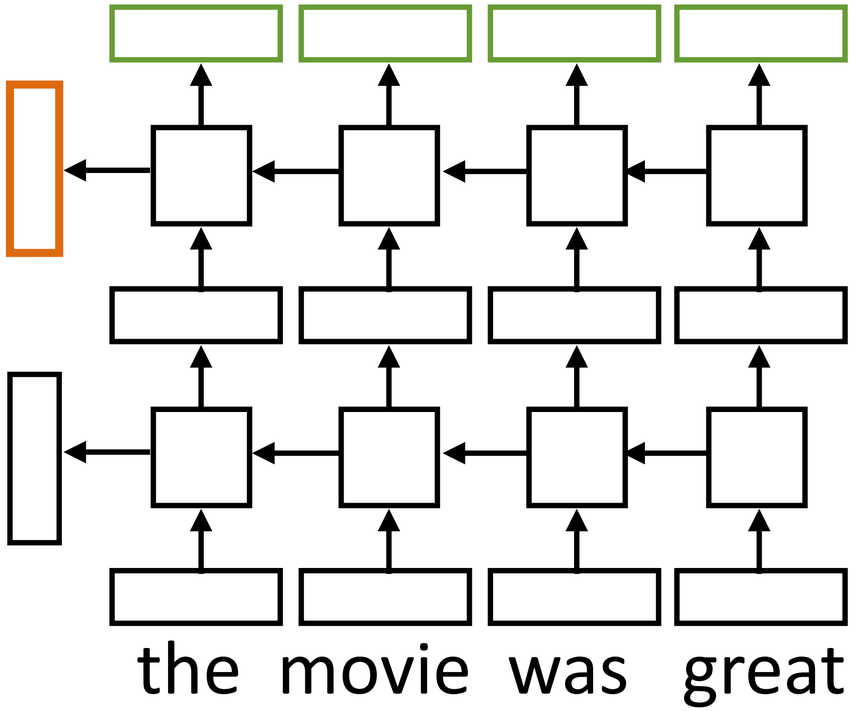
What do RNNs produce?



- 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 Bidirectional RNN

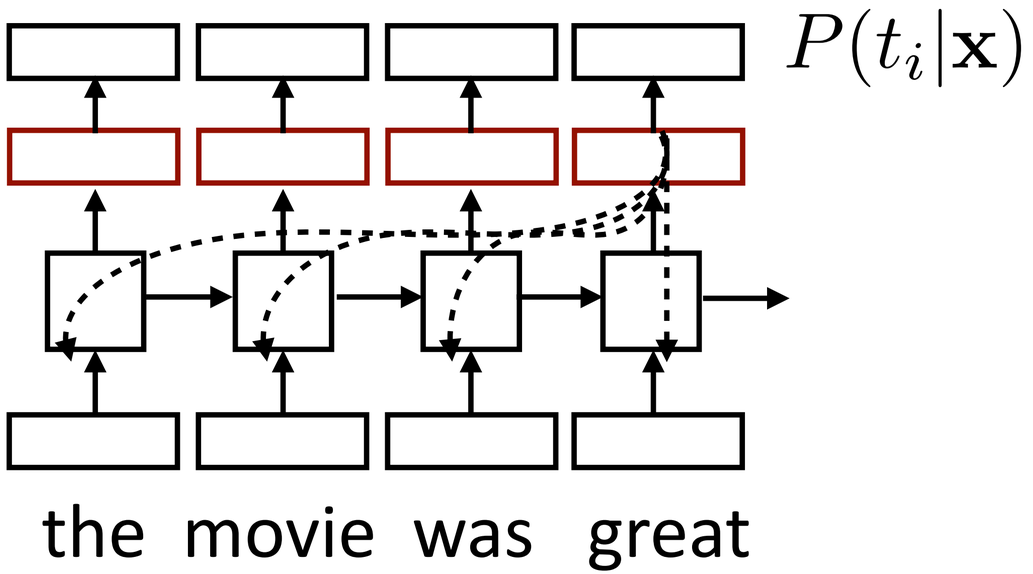




- Sentence classification based on concatenation of both final outputs

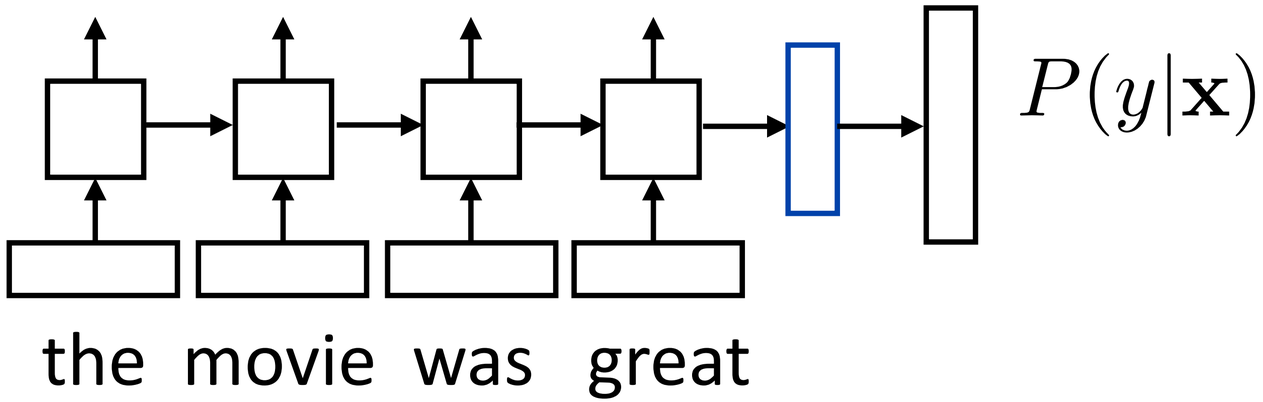
- Token classification based on concatenation of both directions' token representations

Training RNNs



- 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 RNNs



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

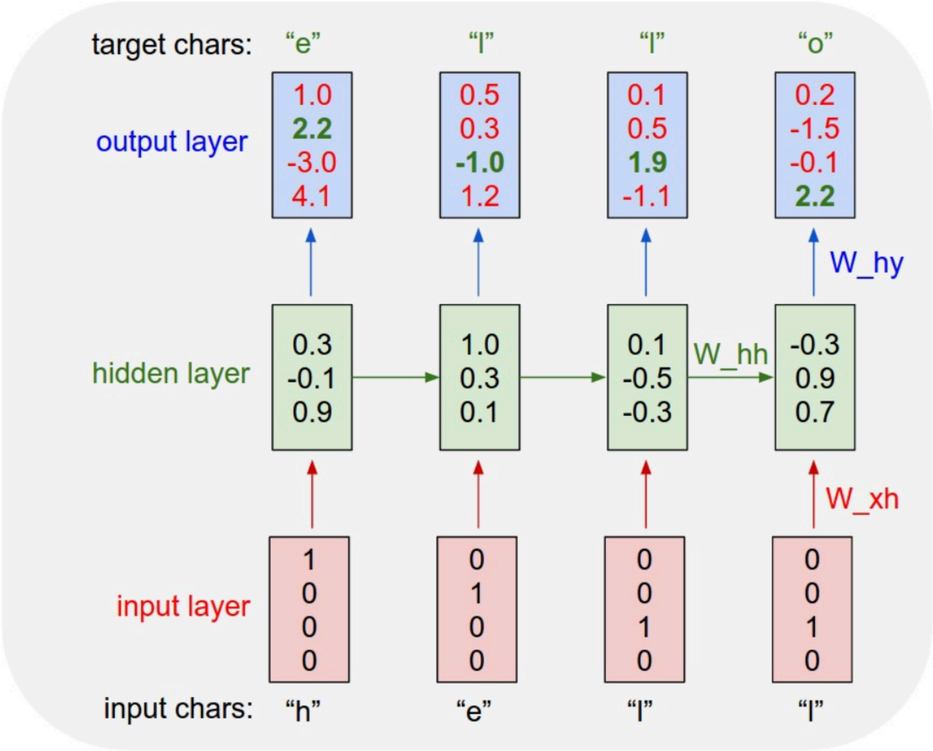
Applications

What can LSTMs model?

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

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

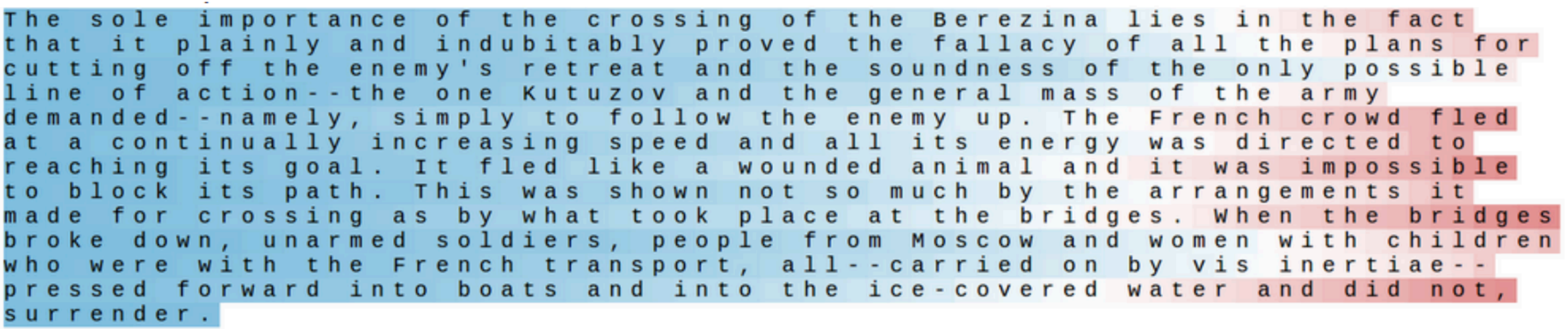


An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

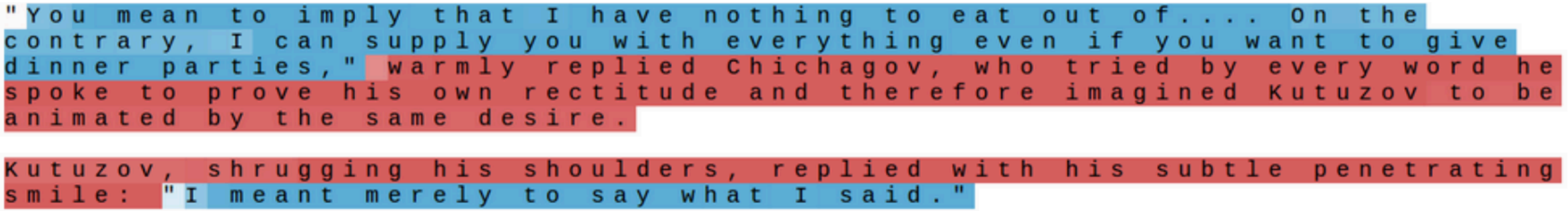
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

- 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



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

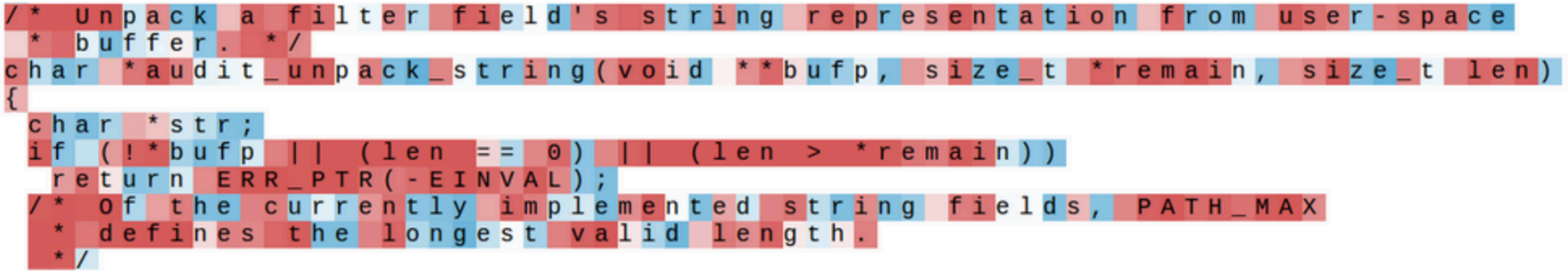
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Visualizing LSTMs

- 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



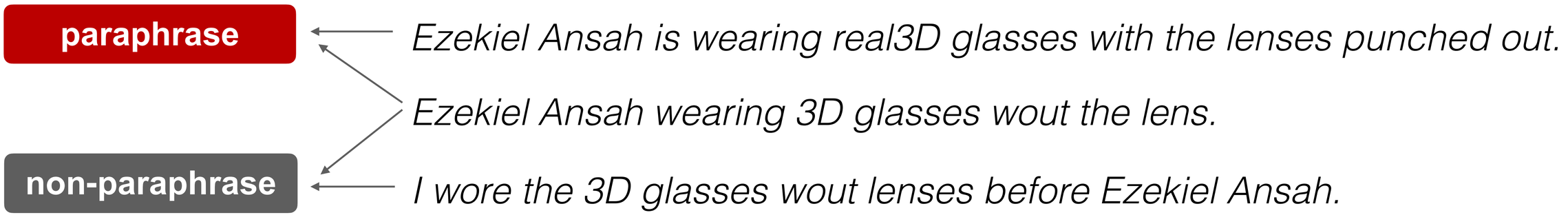
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

What can LSTMs model?

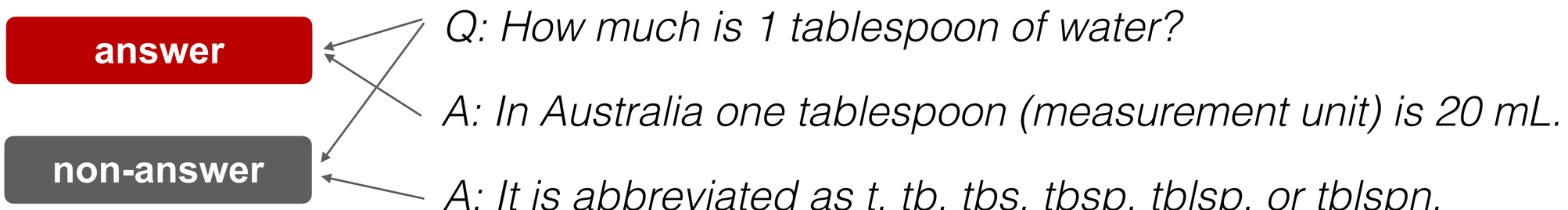
- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (next lecture)
- Textual entailment/similarity
  - Encode two sentences, predict

Semantic Similarity

Paraphrase Identification



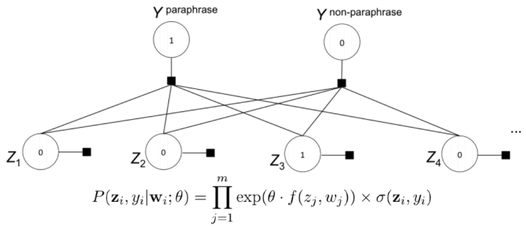
Question Answering



Wuwei Lan, Wei Xu. "Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering"
Best Paper Award at COLING (2018)

Semantic Similarity

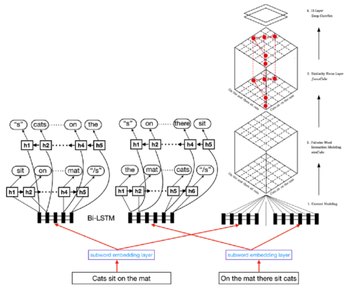
Multi-instance Learning
(TACL 2014)



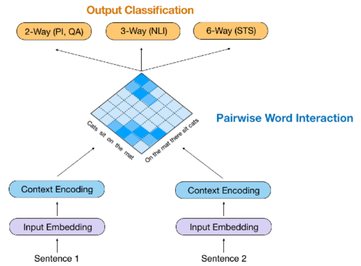
Twitter Paraphrase Corpus
(BUCC 2013; SemEval 2015;
EMNLP 2017; ongoing)

related to natural language generation

Multi-task Subword Model
(NAACL 2018)



Pairwise Interaction Models
(COLING 2018; ACL 2021)



Xu et al. (2013, 2014, 2015), Lan et al. (2017, 2018)

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

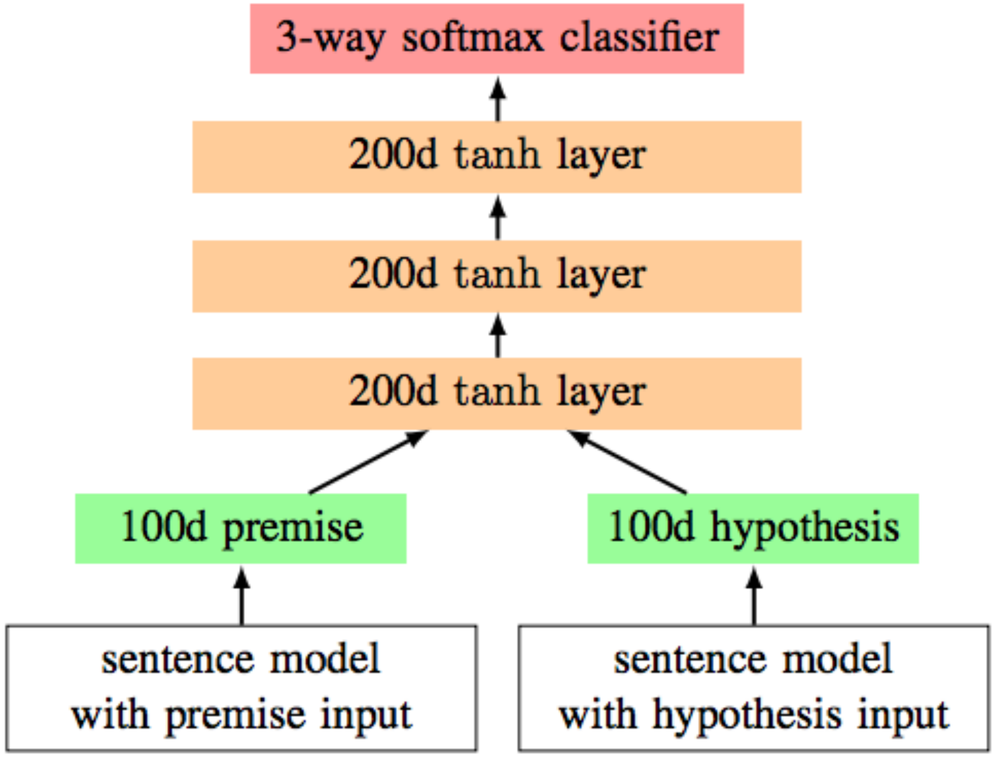
- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process

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
- Next time: CNNs and neural CRFs