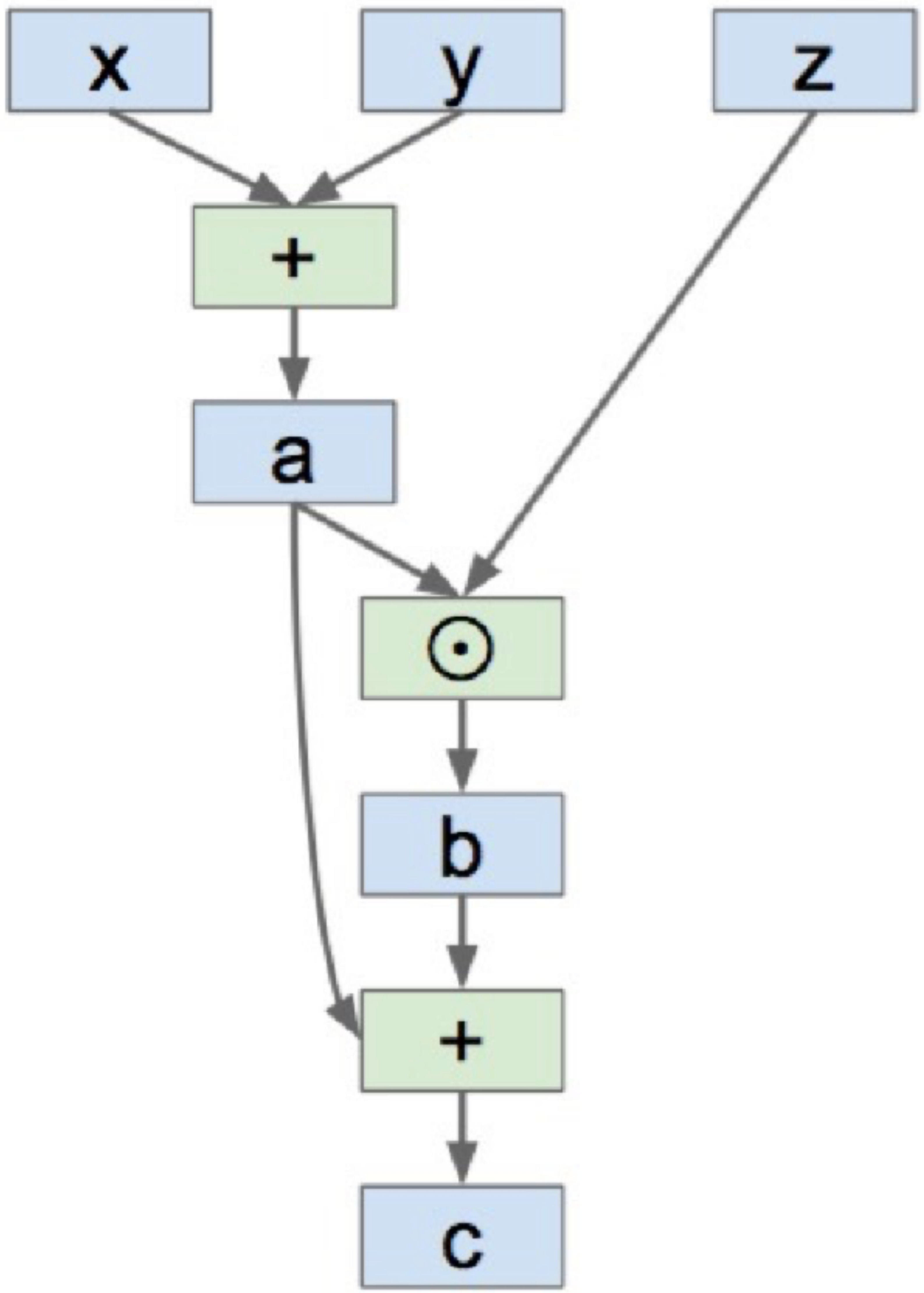
CS395T: Structured Models for NLP
Lecture 15: RNNs I

Greg Durrett

Recall: Computation Graphs



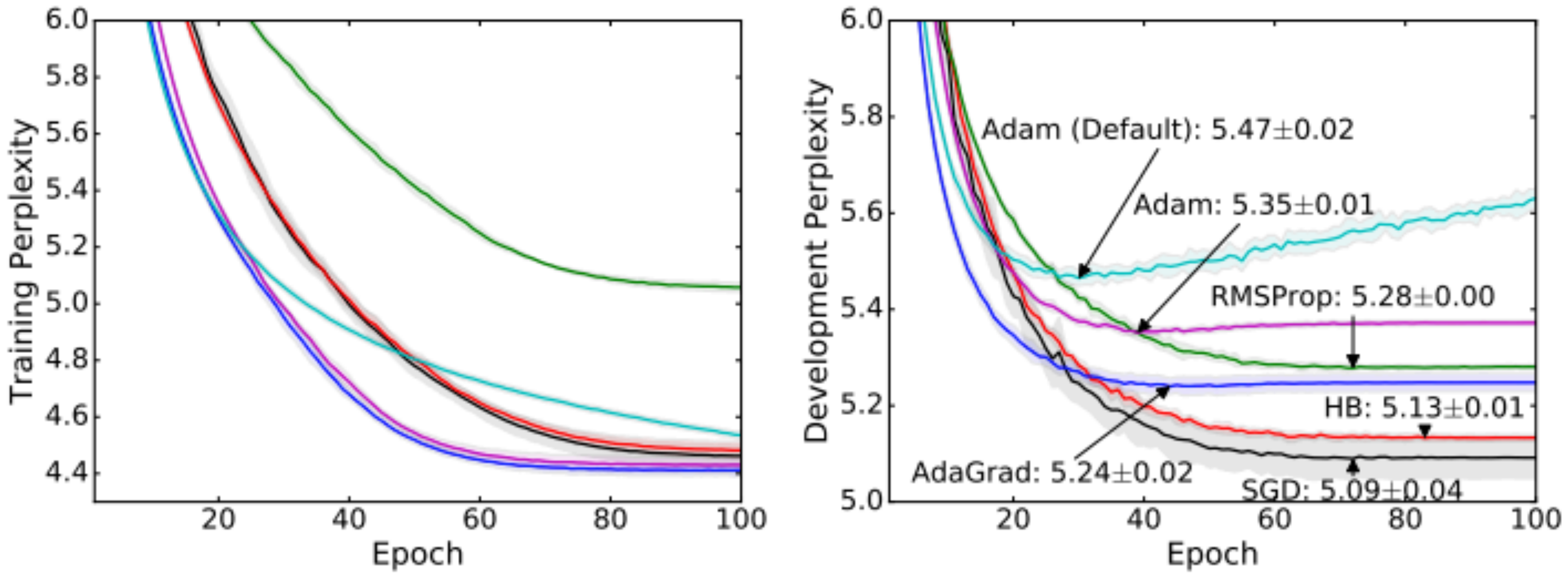

```
x = tf.placeholder("x")
y = tf.placeholder("y")
z = tf.placeholder("z")
a = tf.add(x, y)
b = tf.multiply(a, z)
c = tf.add(b, a)

with tf.Session() as sess:
    output = sess.run([a,c],
                      dict_of_input_values)
```

Recall: Training Tips

- Glorot initializer: $U\left[-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, +\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}\right]$

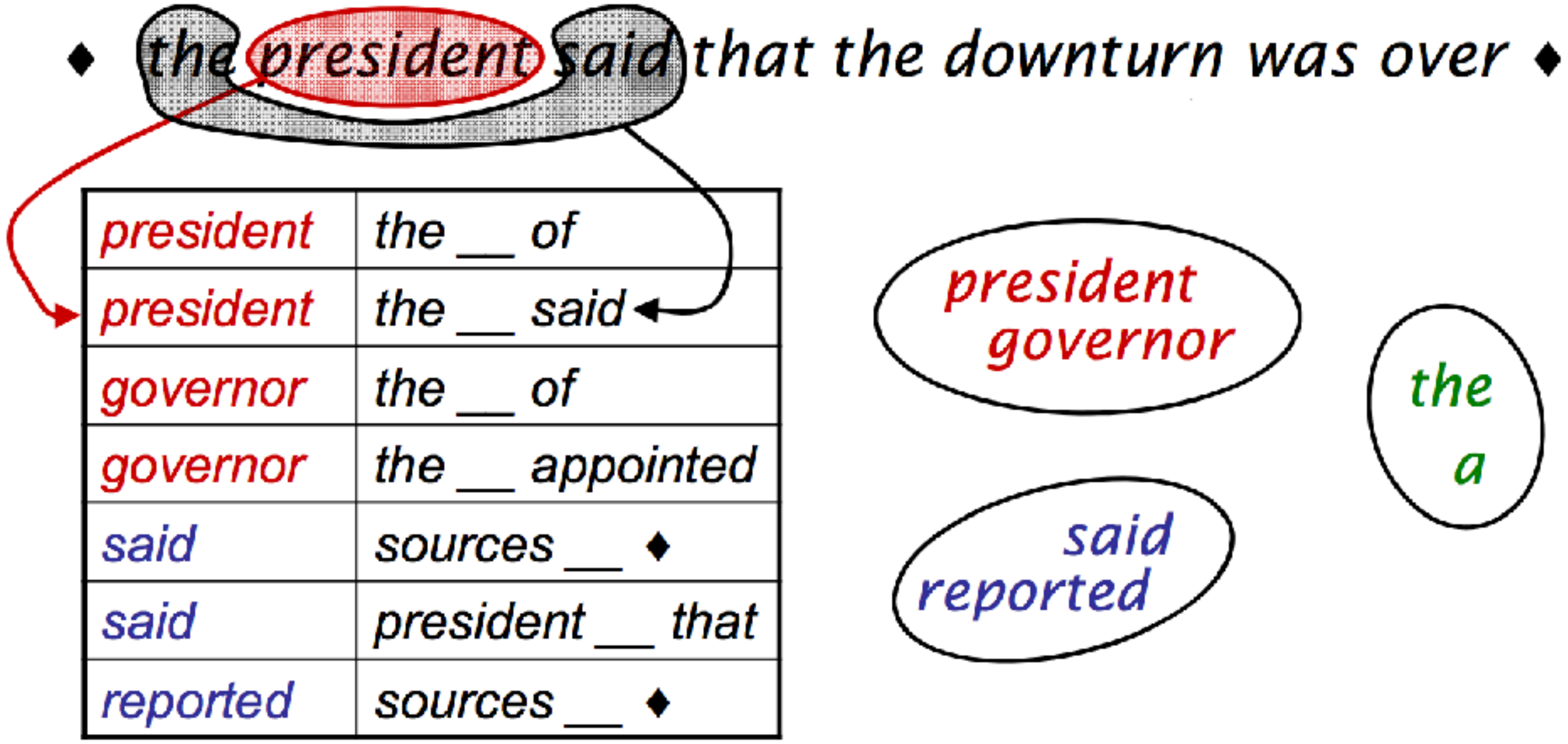
- Use dropout for regularization

- Think about your optimizer: Adam or tuned SGD work well



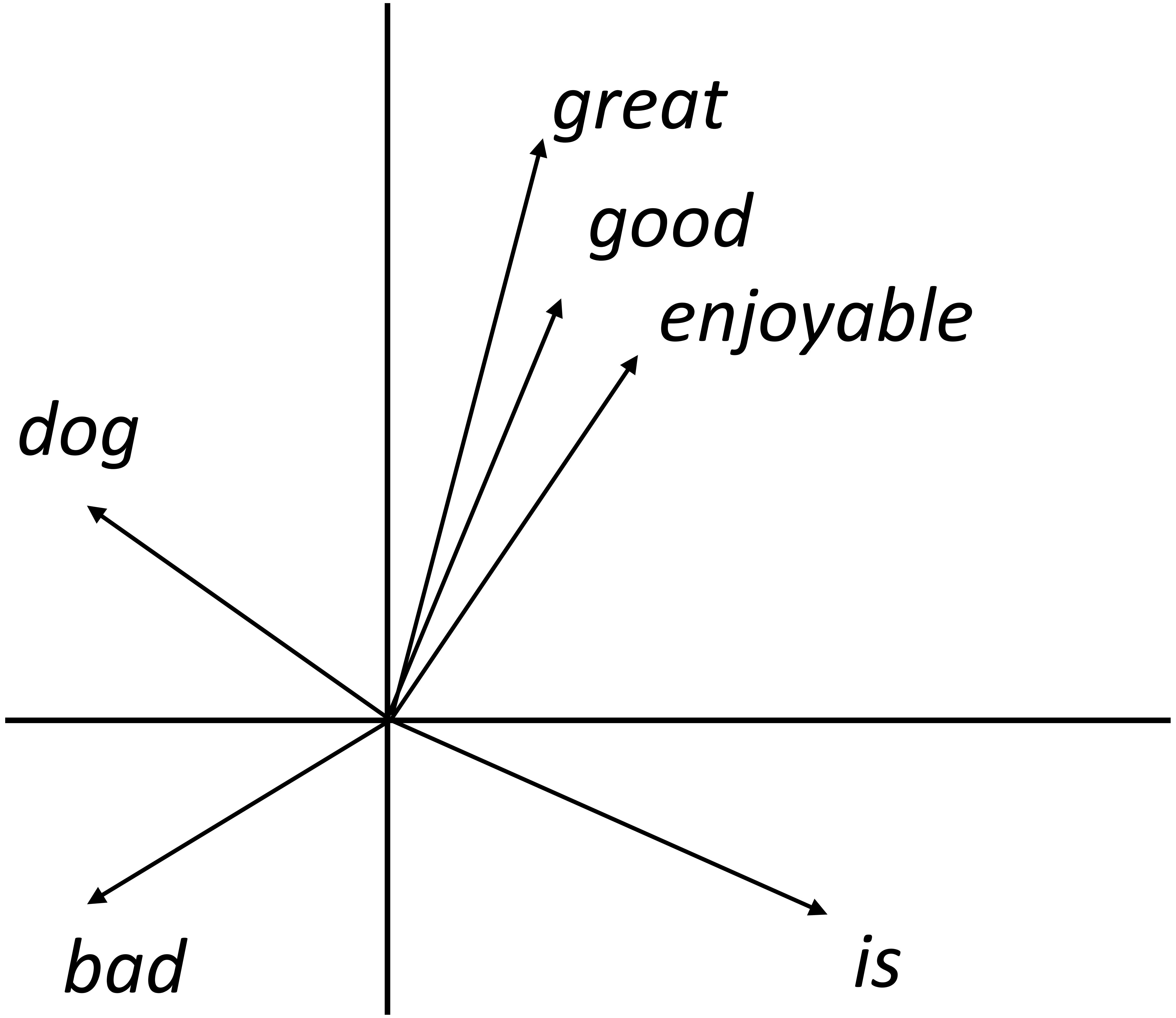
(e) Generative Parsing (Training Set) (f) Generative Parsing (Development Set)

Recall: Word Vectors



president	the __ of
president	the __ said
governor	the __ of
governor	the __ appointed
said	sources __ ♦
said	president __ that
reported	sources __ ♦

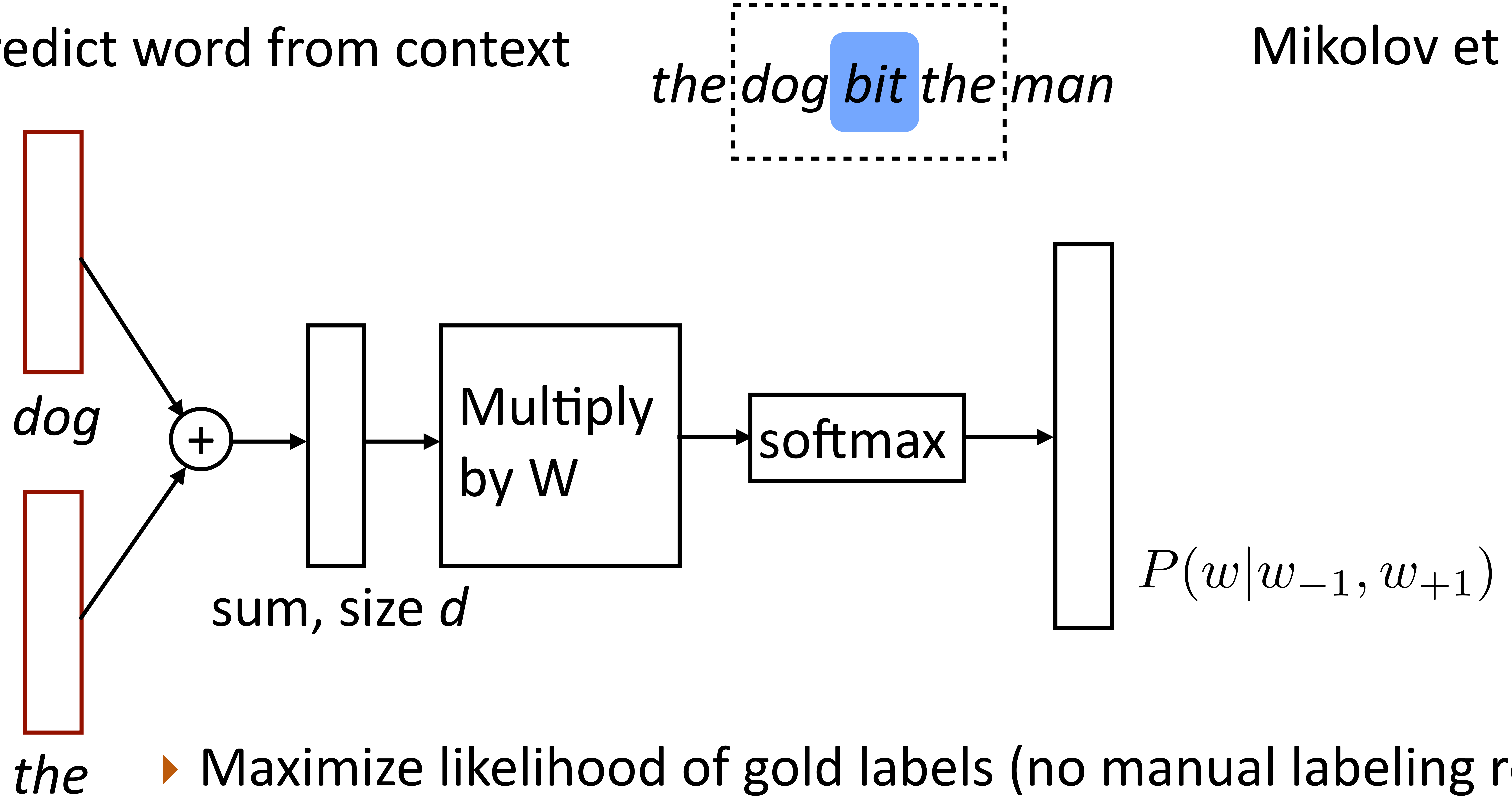
[Finch and Chater 92, Shuetze 93, many others]



Recall: Word Vectors

- Predict word from context

Mikolov et al. (2013)



- Maximize likelihood of gold labels (no manual labeling required!)

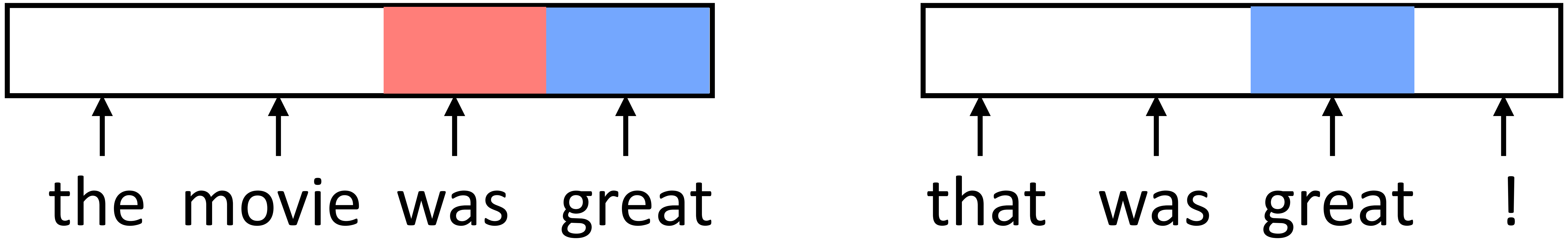
This Lecture

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

RNN Basics

RNN Motivation

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



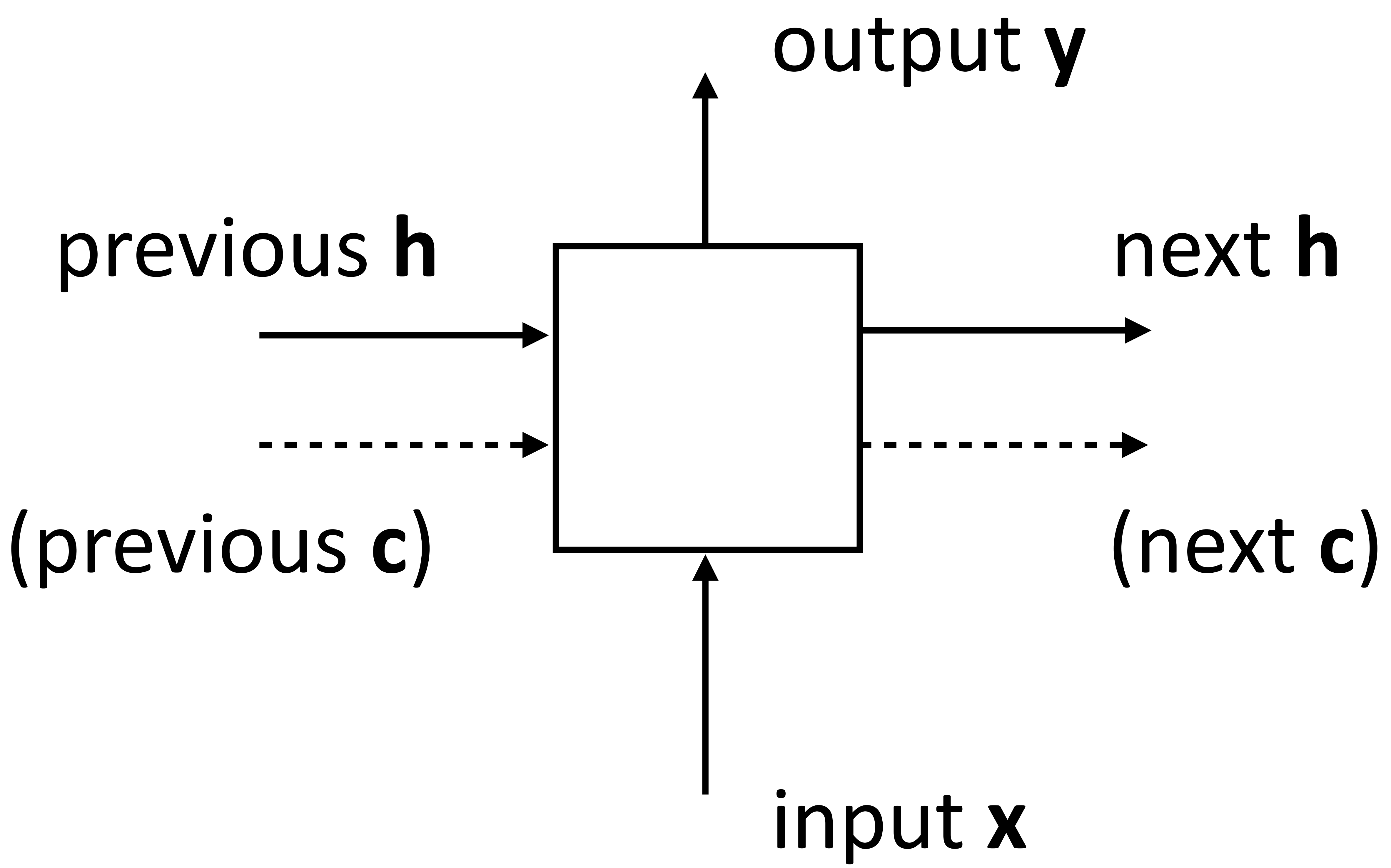
- Instead, we need to:

1) Process each element in a uniform way

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

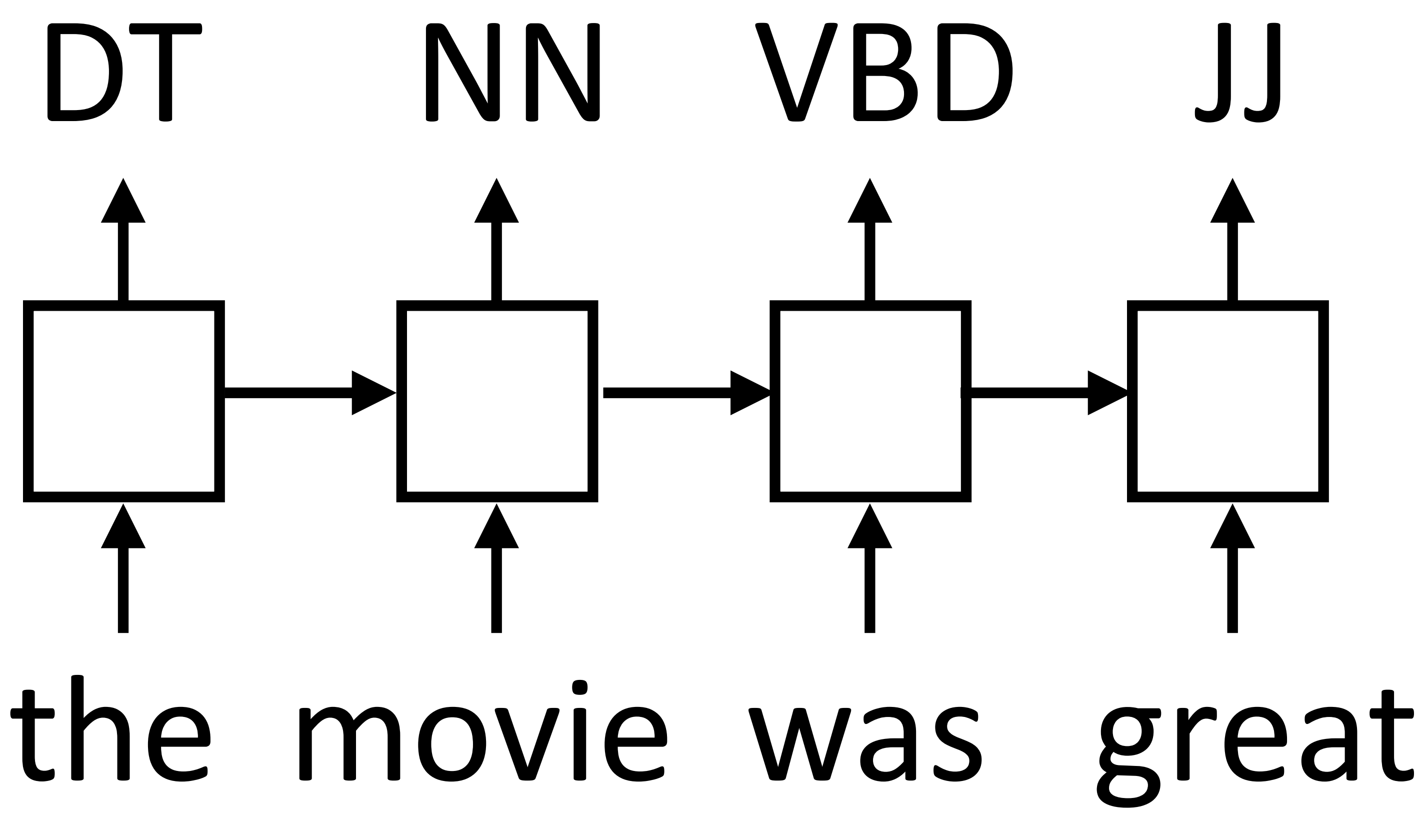
RNN Abstraction

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

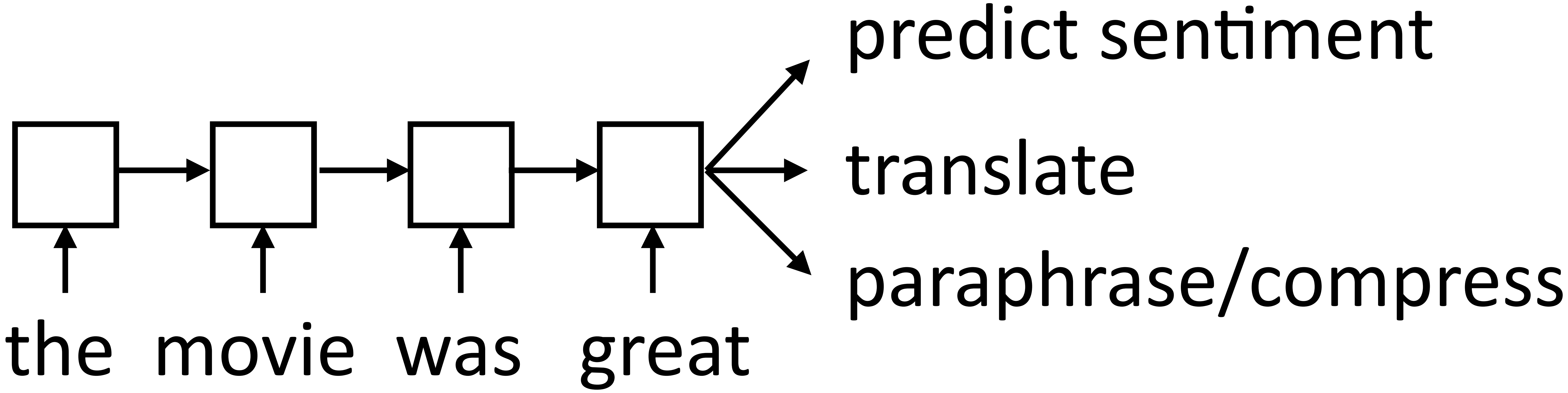


RNN Uses

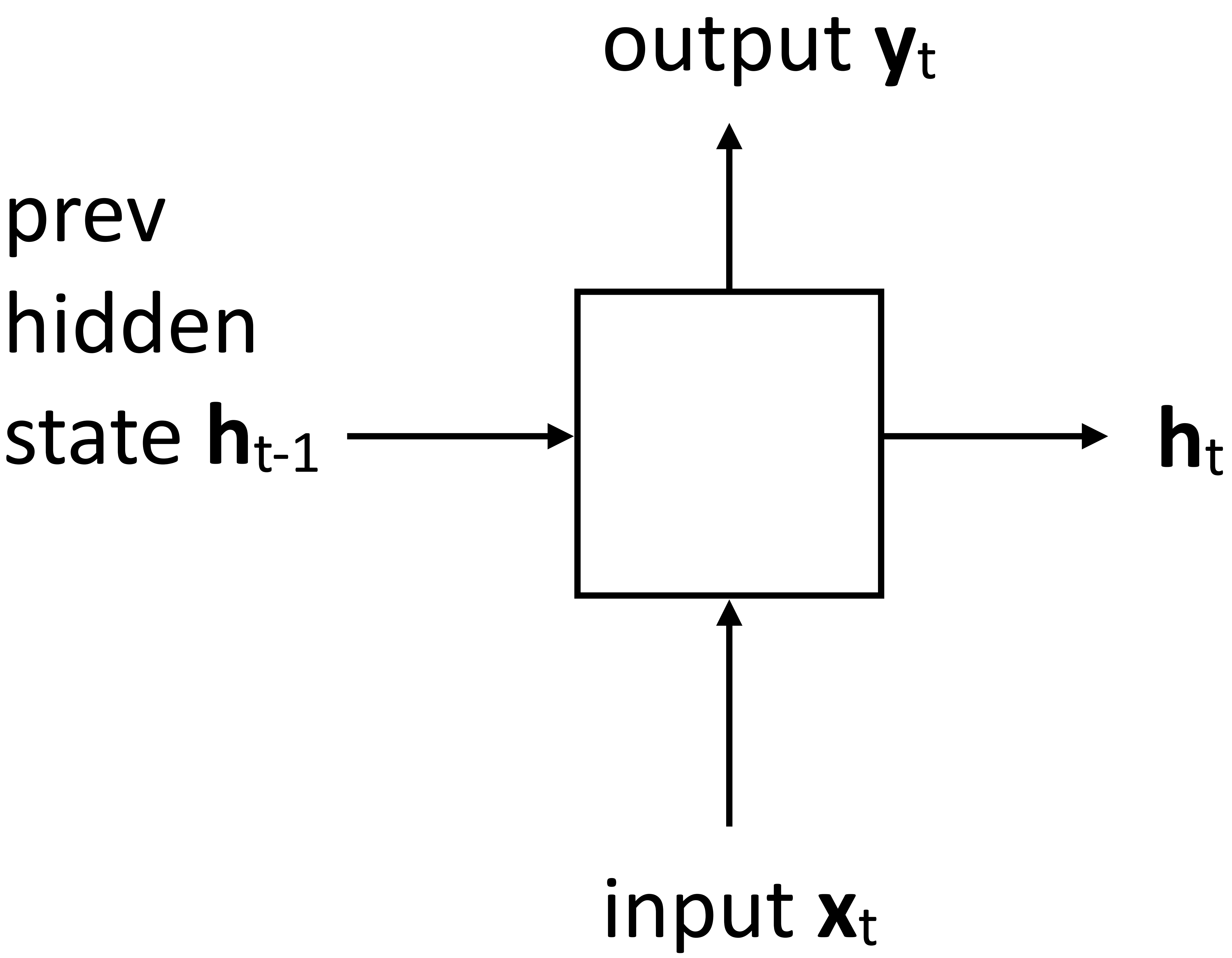
- Transducer: make some prediction for each element in a sequence



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



Elman Networks



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

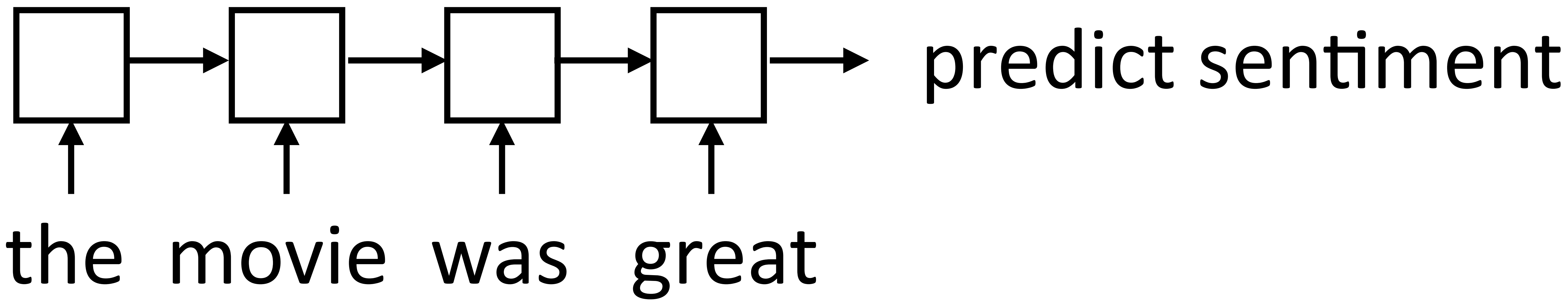
- Updates hidden state based on input and current hidden state

$$\mathbf{y}_t = \tanh(U\mathbf{h}_{\mathbf{t}} + \mathbf{b}_y)$$

- Computes output from hidden state

- Long history! (invented in the 1980s)

Training Elman Networks

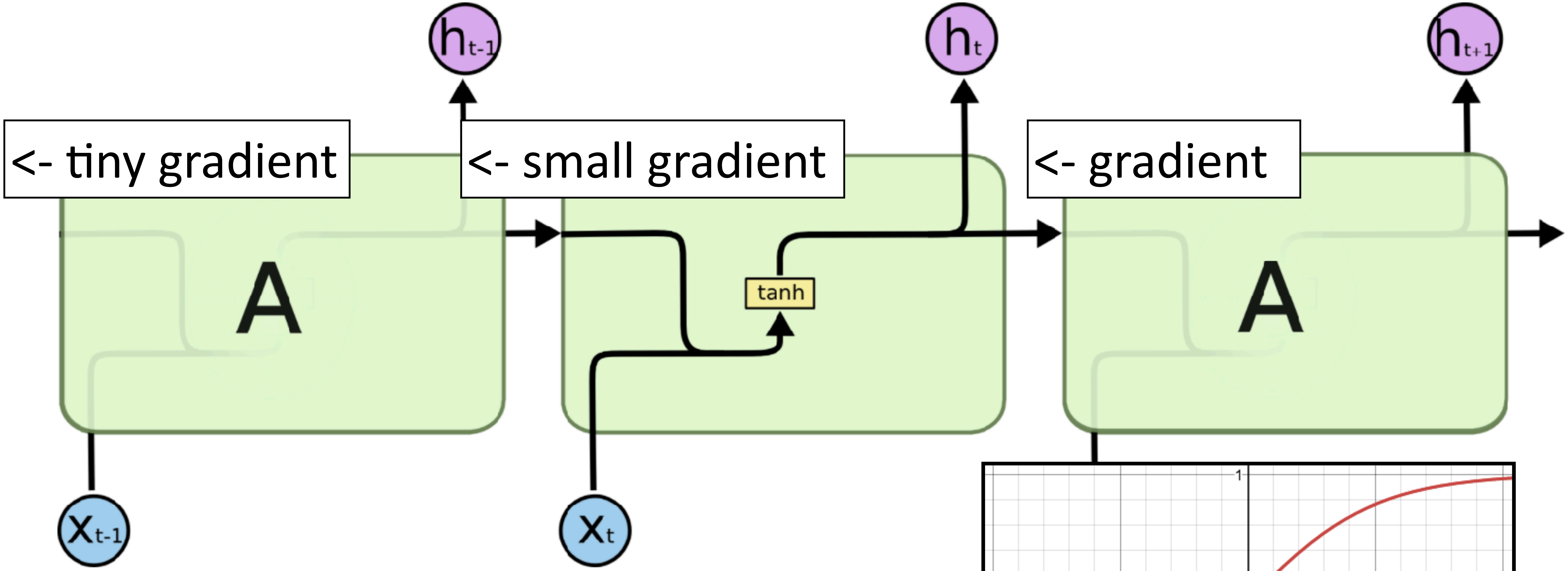


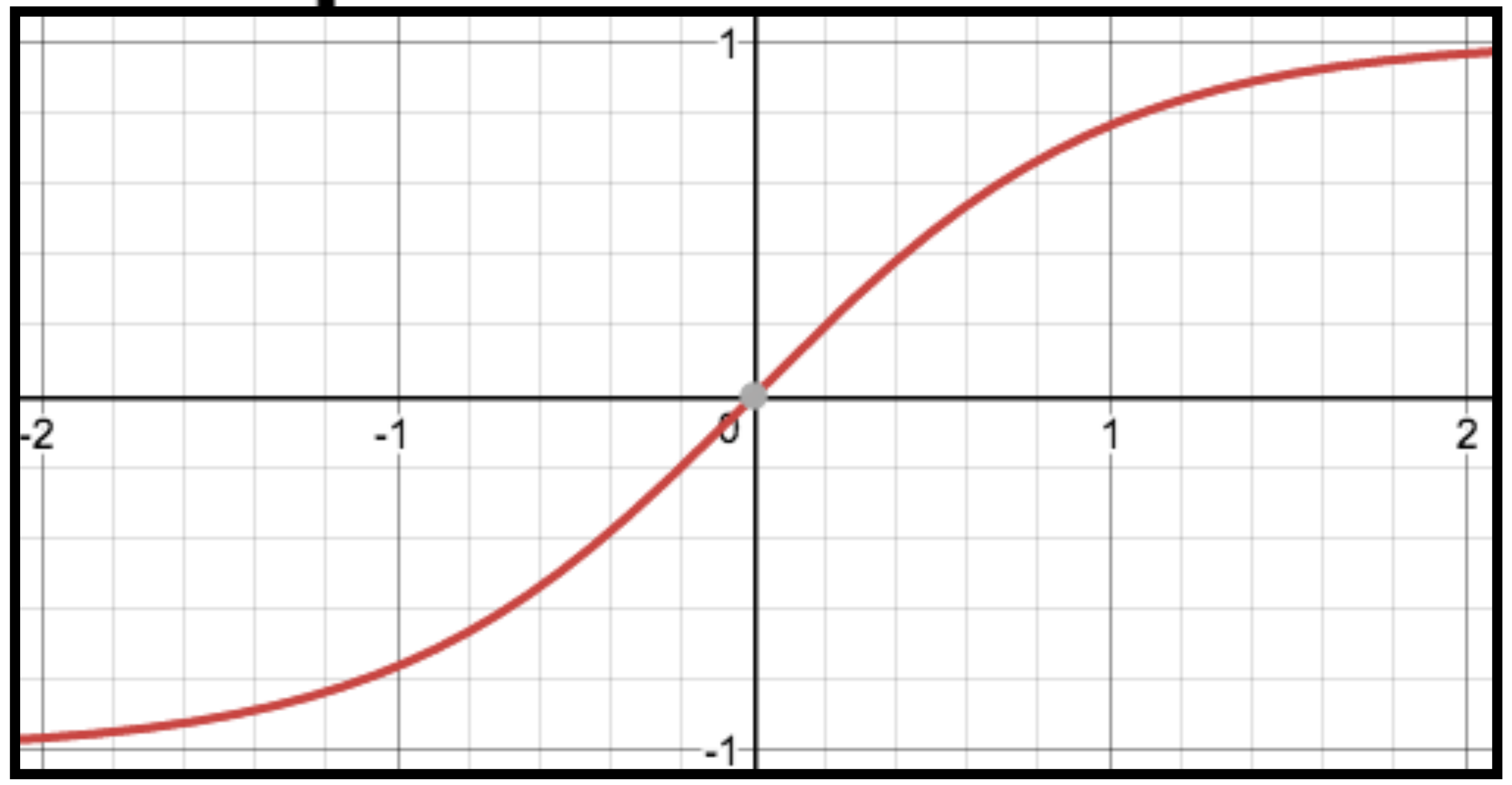
- Need to backpropagate through the whole network from the end
- RNN potentially needs to learn how to “remember” information for a long time!

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*

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

- Designed to fix "vanishing gradient" problem

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

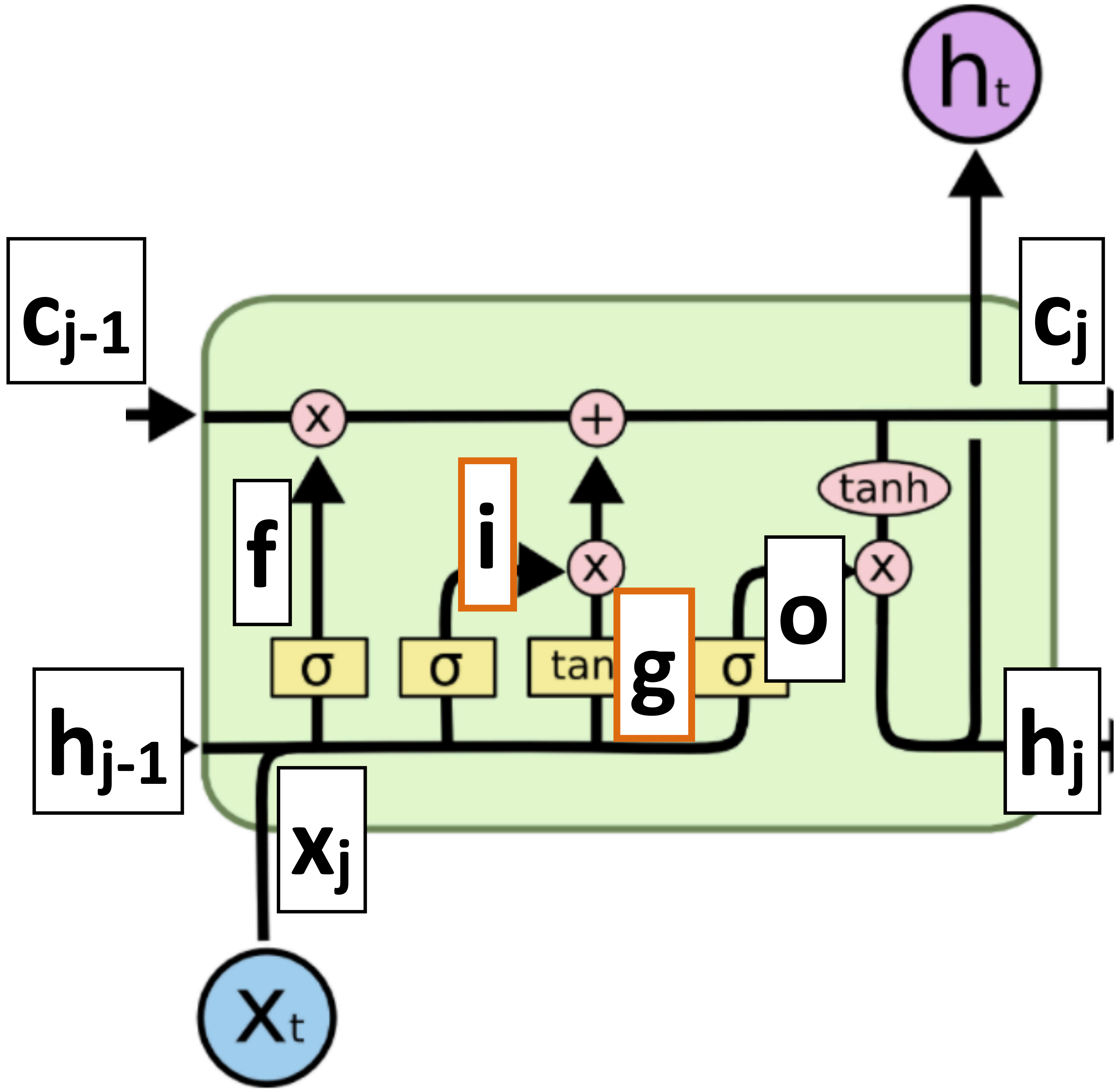
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- Vector-valued forget gate **f** computed based on input and 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



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

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

$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

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

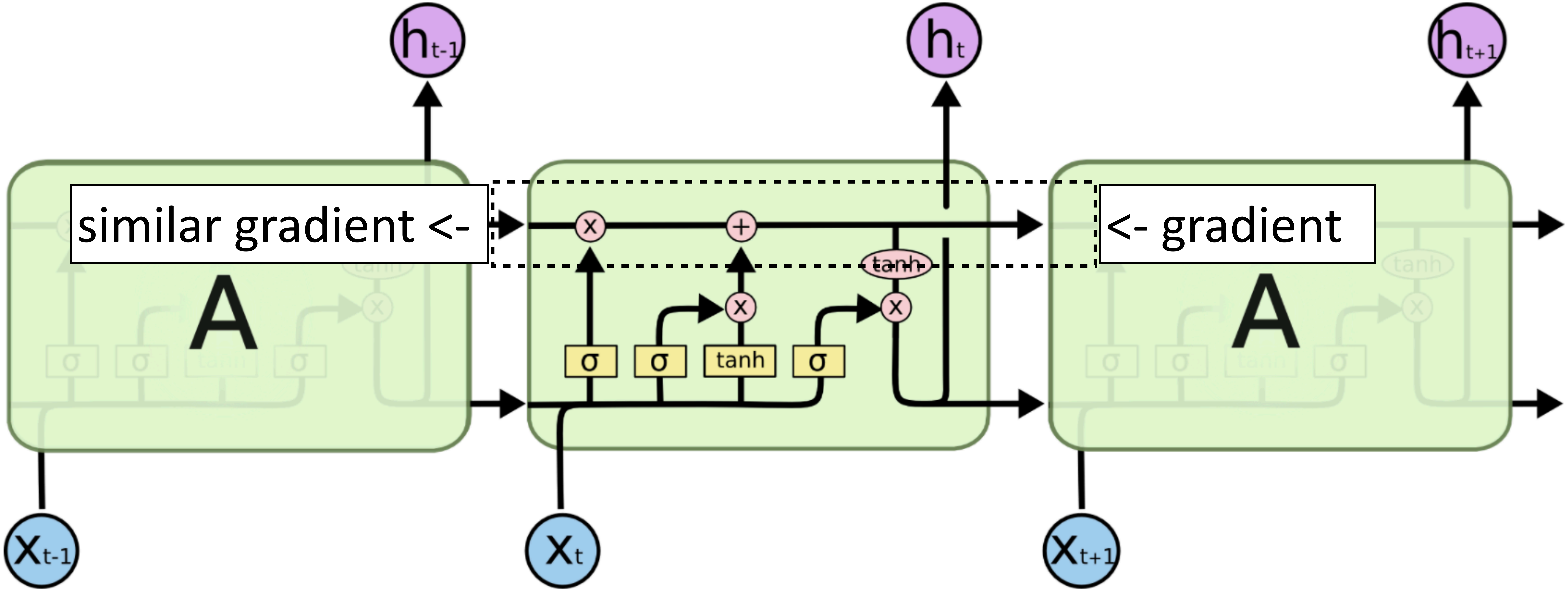
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

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

Goldberg lecture notes

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

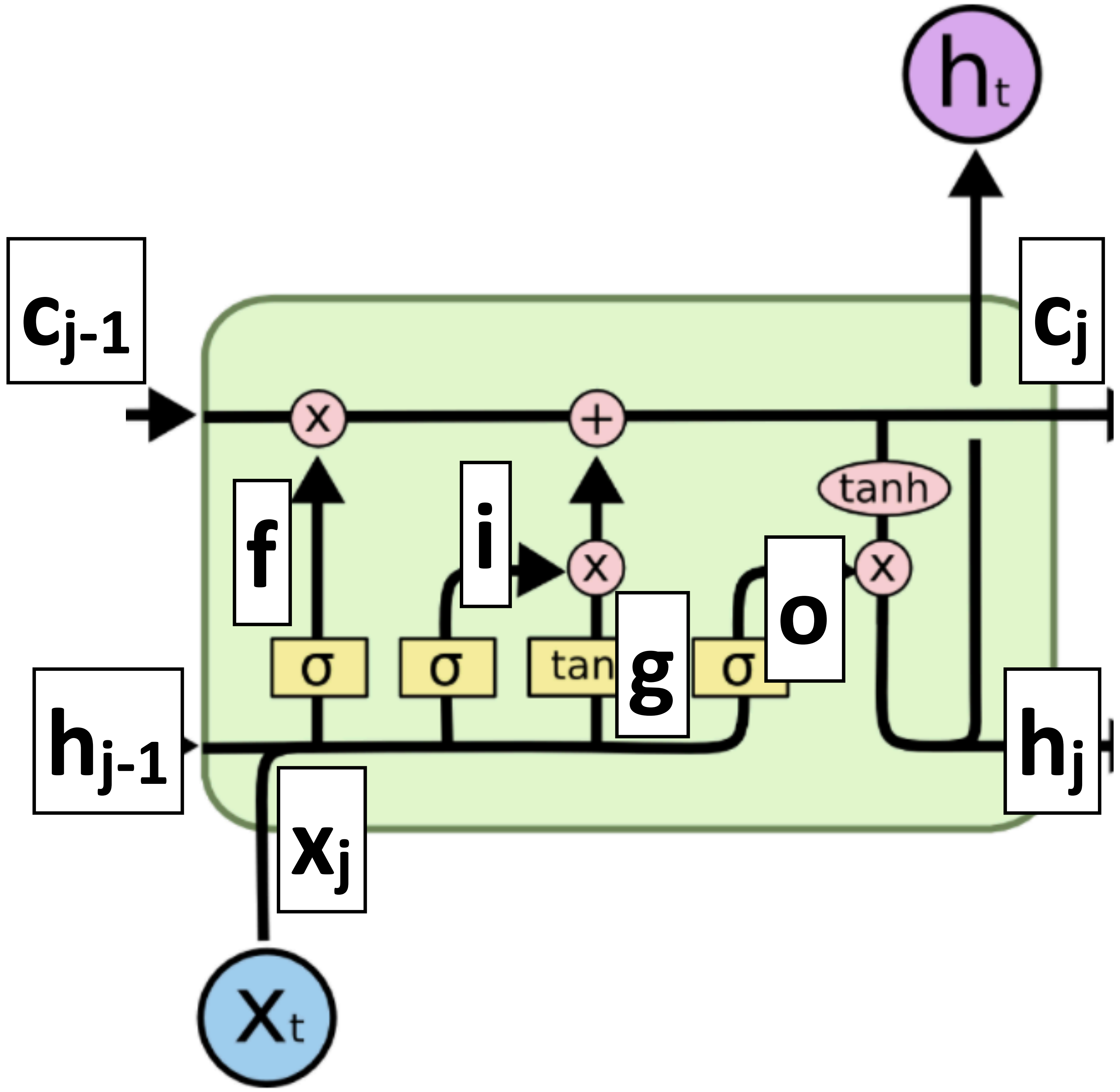
LSTMs



- 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/

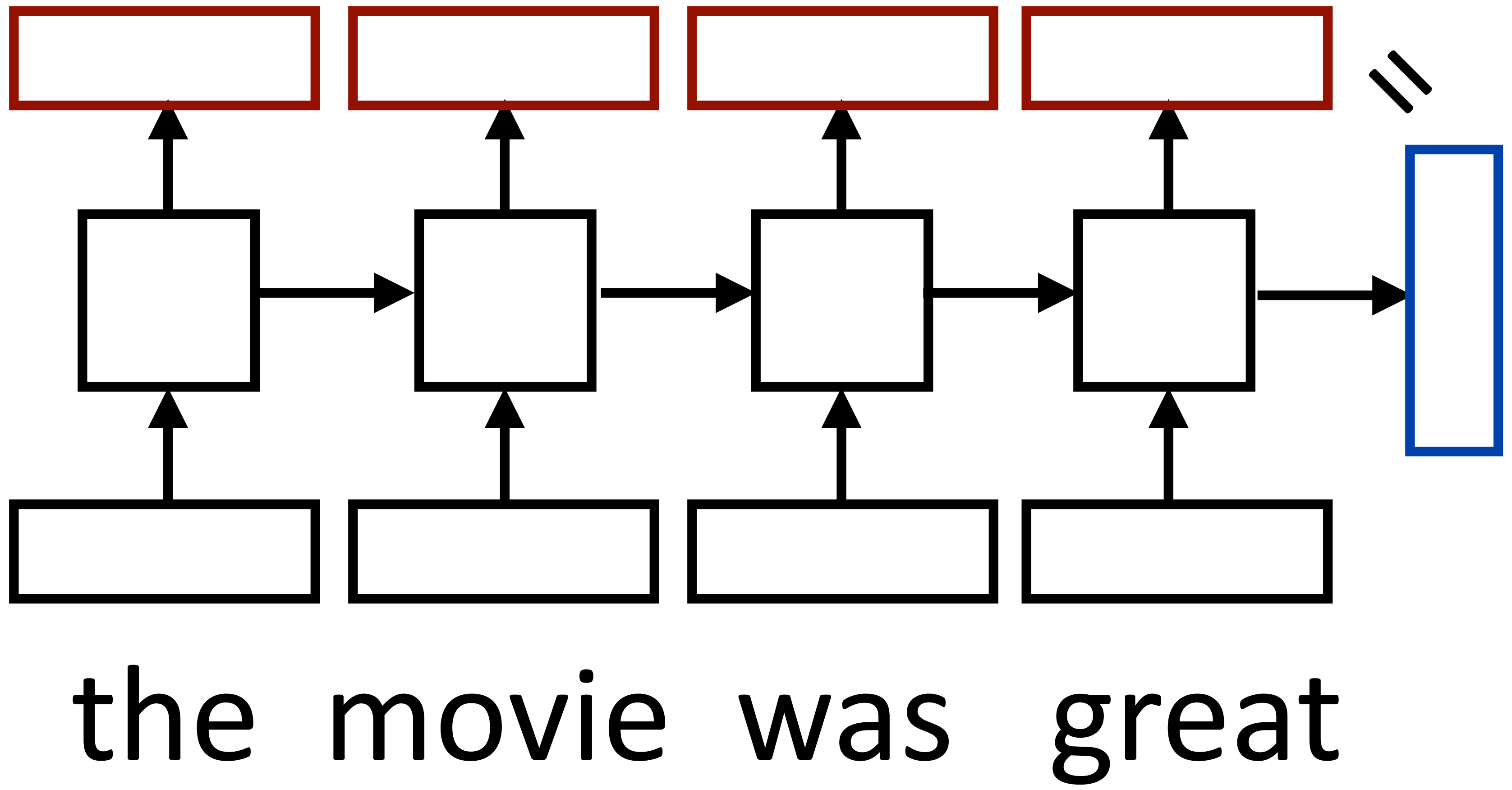
Understanding LSTM Parameters



- Initialize hidden layer randomly
- Need to learn how the gates work: what do we forget/remember?
- **h** and **x** affect **i** and **f**: based on state and input, do we remember the current state or incorporate new input?
- **g** uses an arbitrary nonlinearity, this is the "layer" of the cell

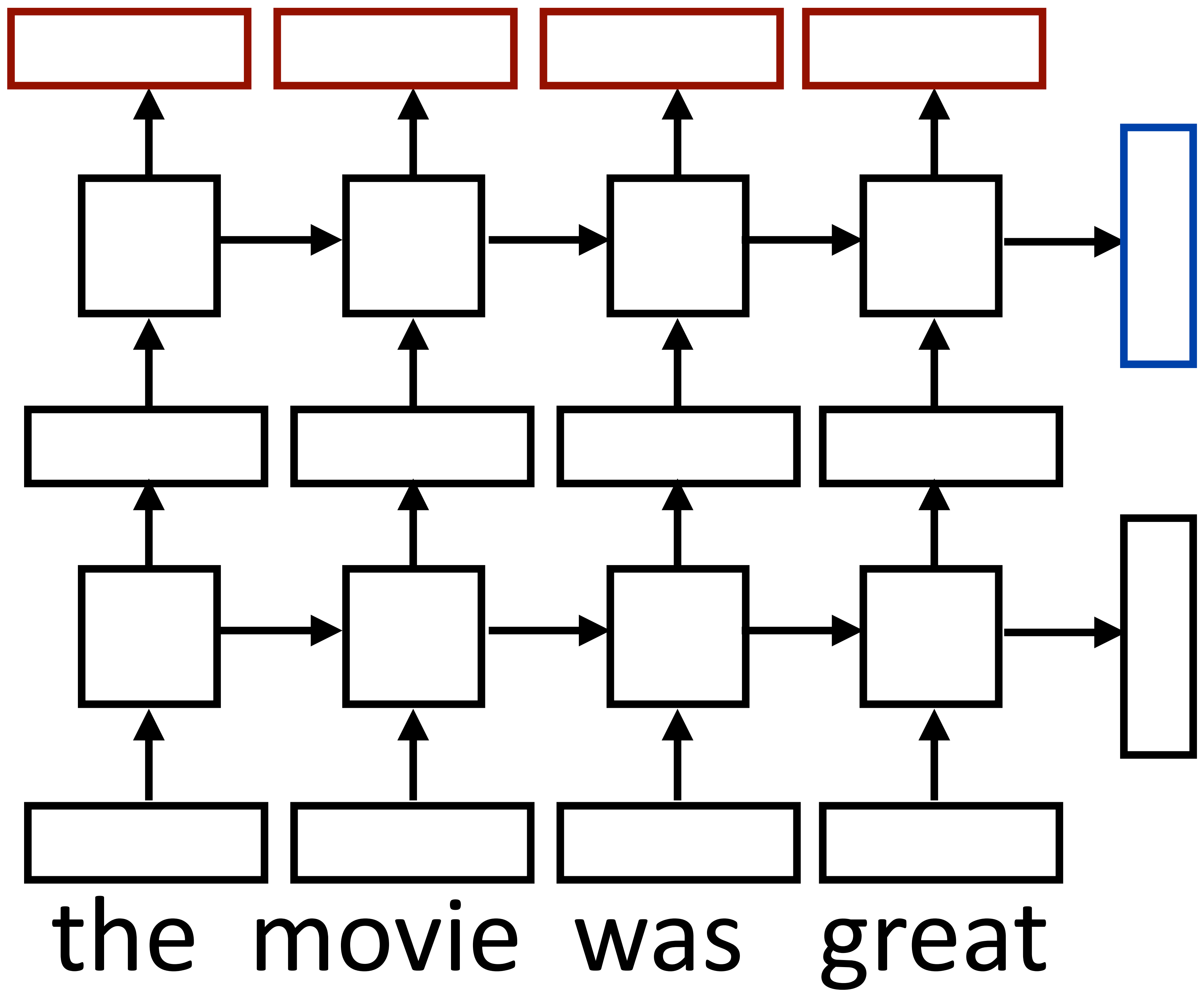
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

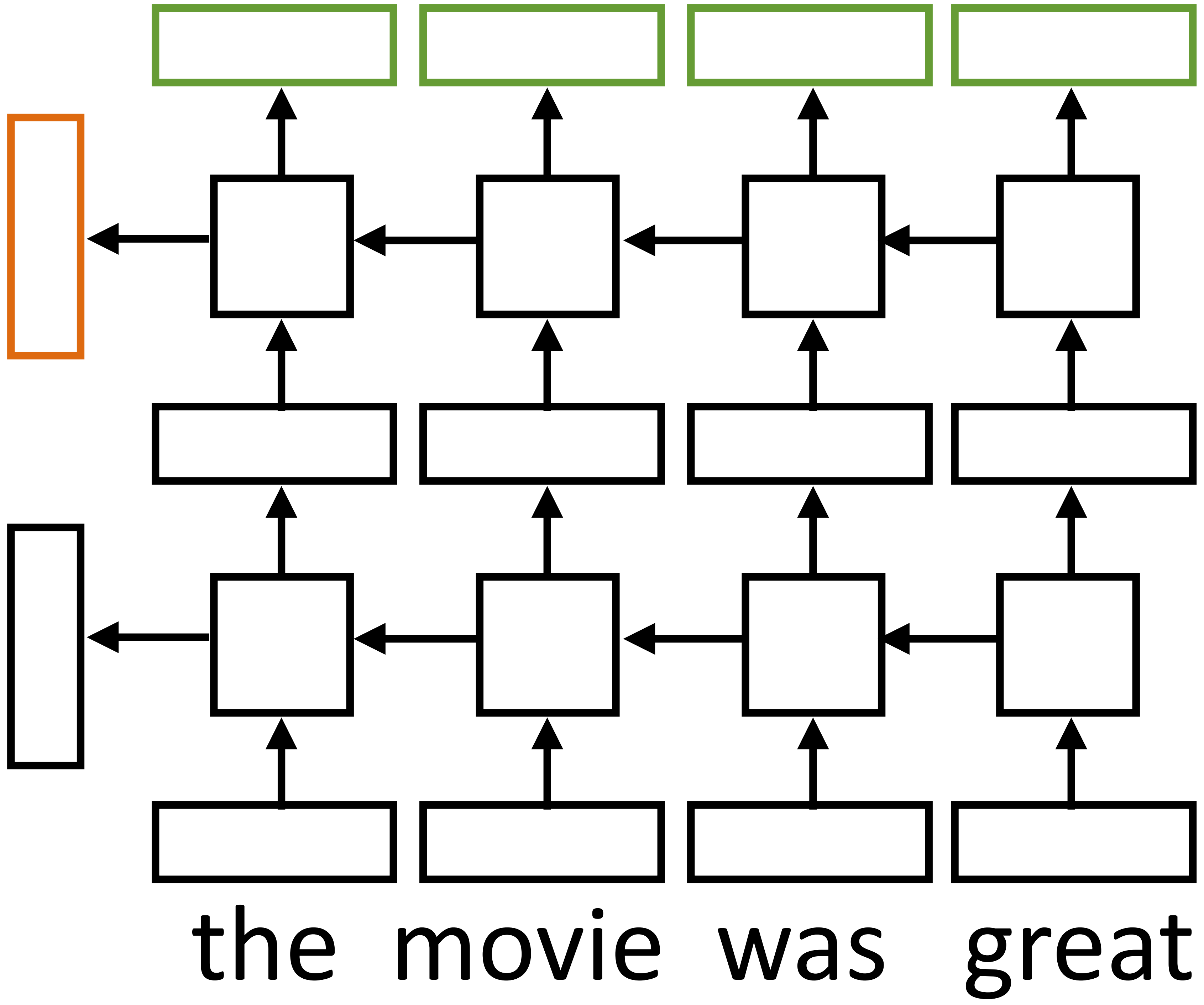
What do LSTMs produce?



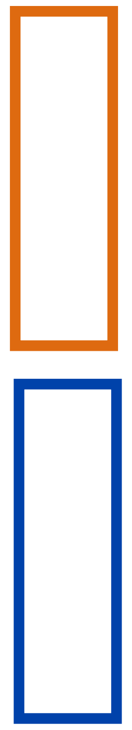
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- LSTM can be viewed as a transformation of a sequence of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

Multilayer Bidirectional LSTM

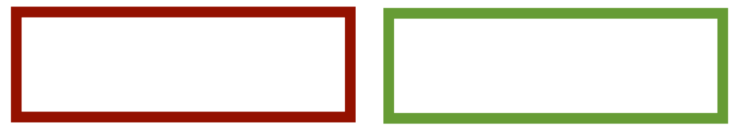




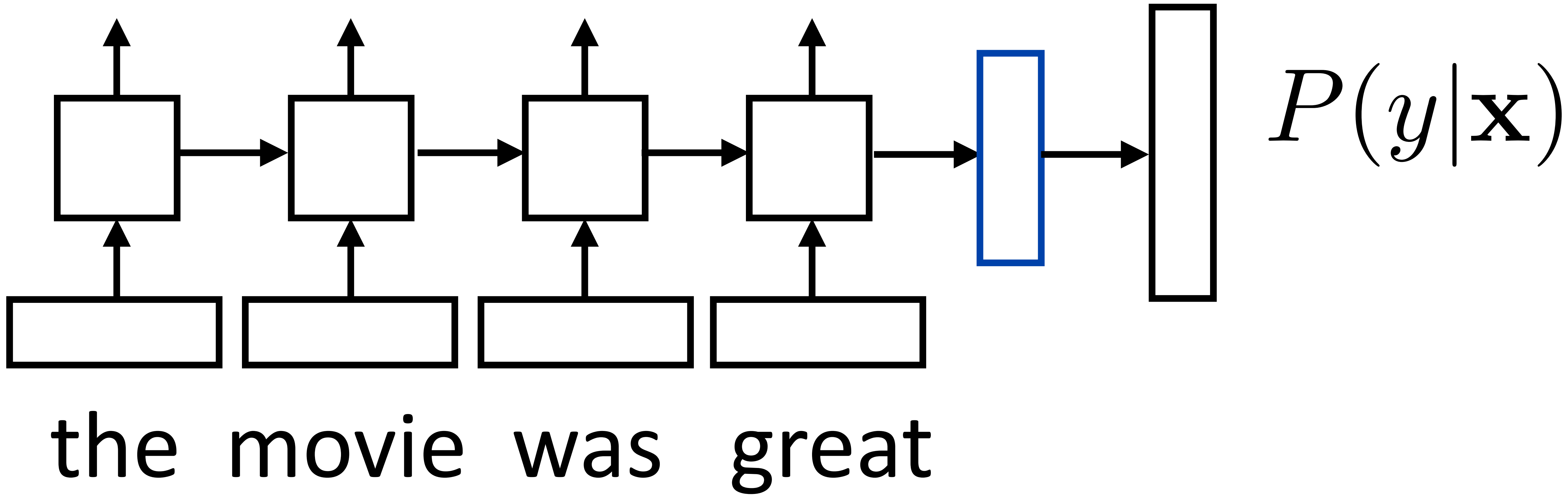
- Sentence classification based on concatenation of both final outputs



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

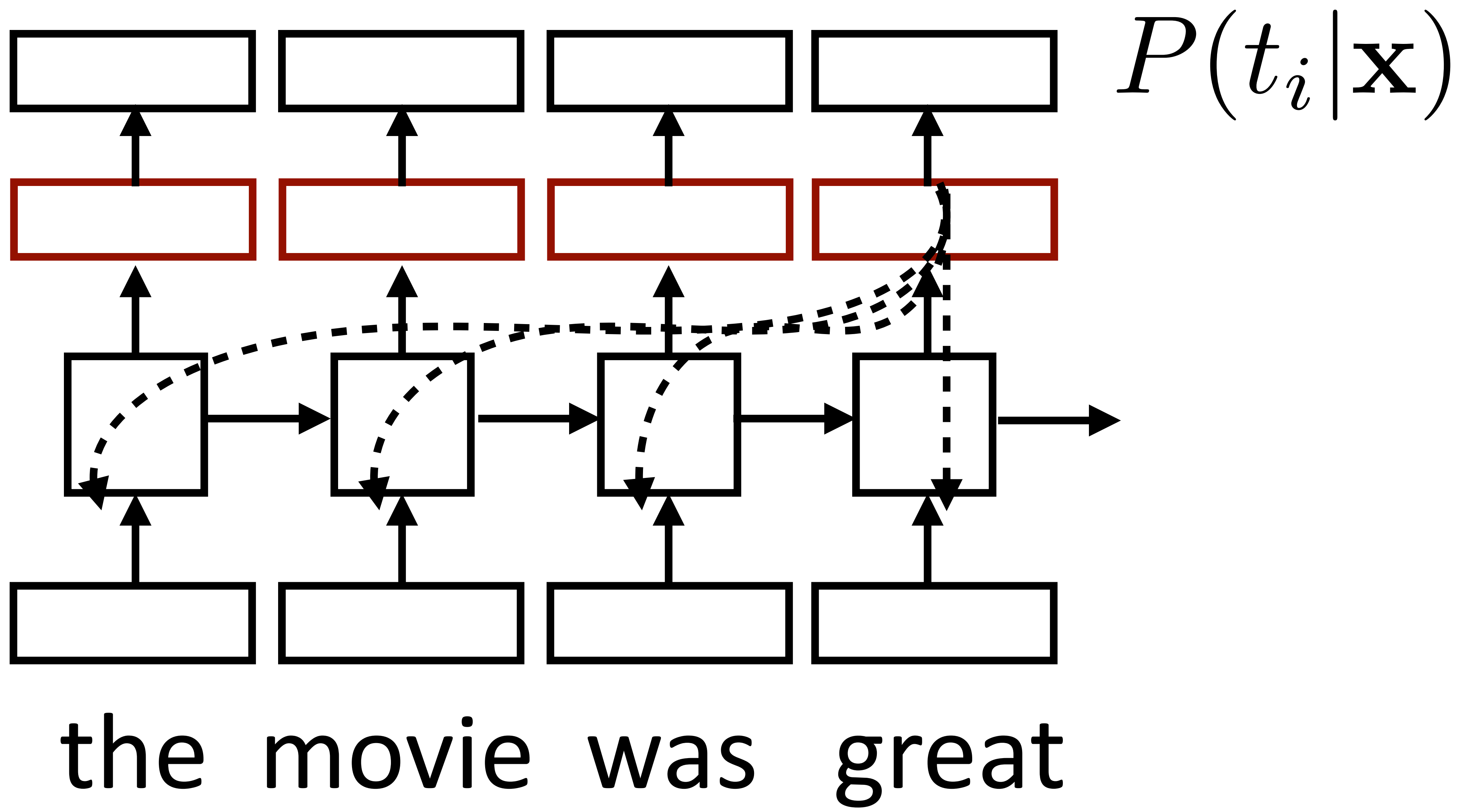


Training LSTMs



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

Training LSTMs



- 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)

GRUs

- Also solves the vanishing gradient problem, simpler than LSTM

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

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

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

- Faster to train and often works better — consider using these for the project!

Cho et al. (2014)

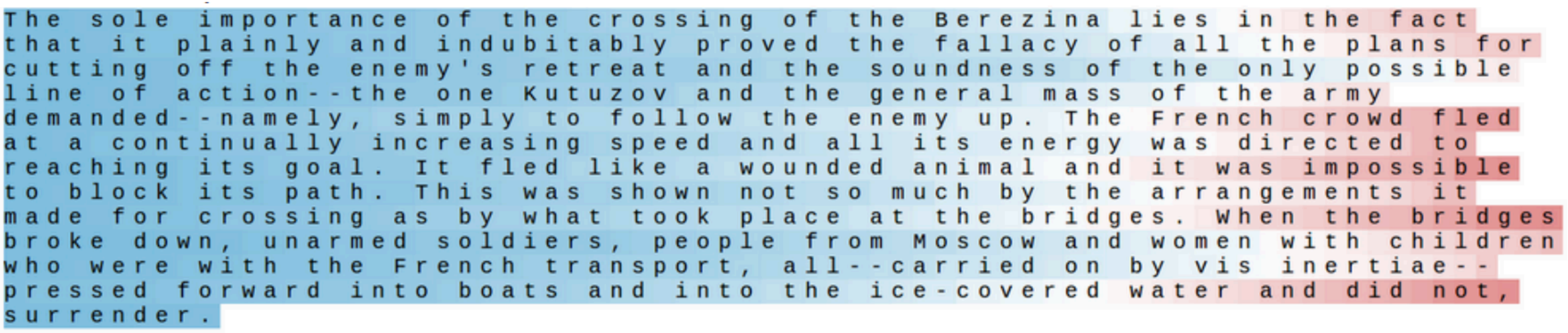
Applications

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 (next lecture)

Visualizing LSTMs

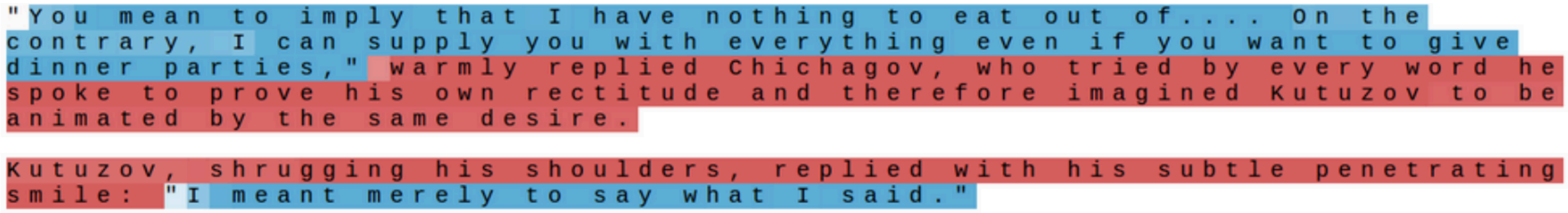
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Counter: know when to generate \n



Karpathy et al. (2015)

Visualizing LSTMs

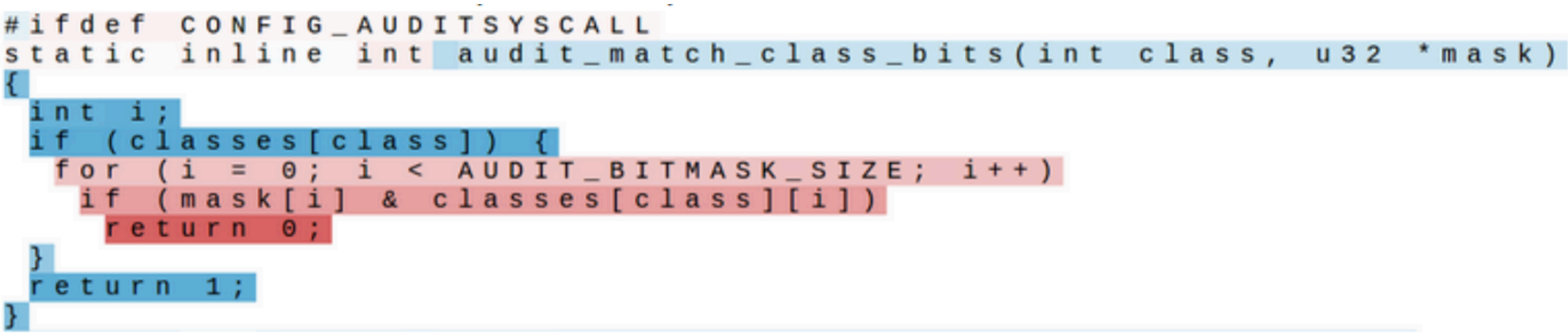
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: know when to generate ”



Karpathy et al. (2015)

Visualizing LSTMs

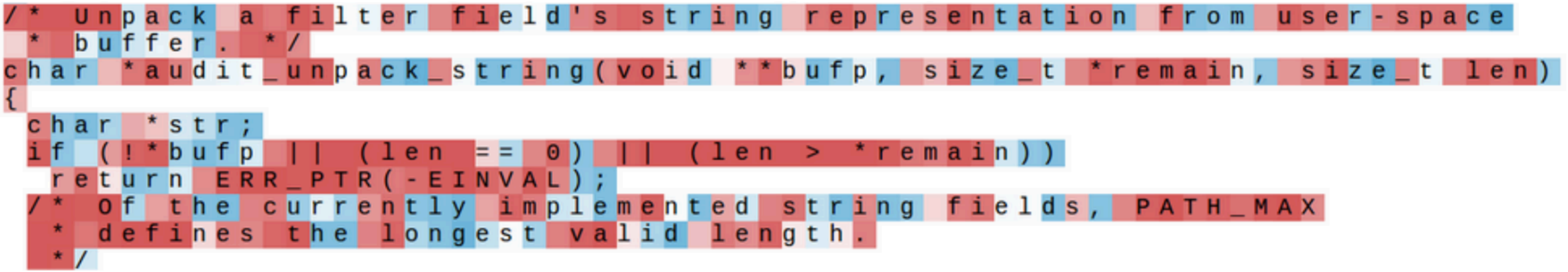
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation



Karpathy et al. (2015)

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



Karpathy et al. (2015)

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
  - 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	*neutral*	The man is sleeping
An older and younger man smiling	*contradicts*	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.)

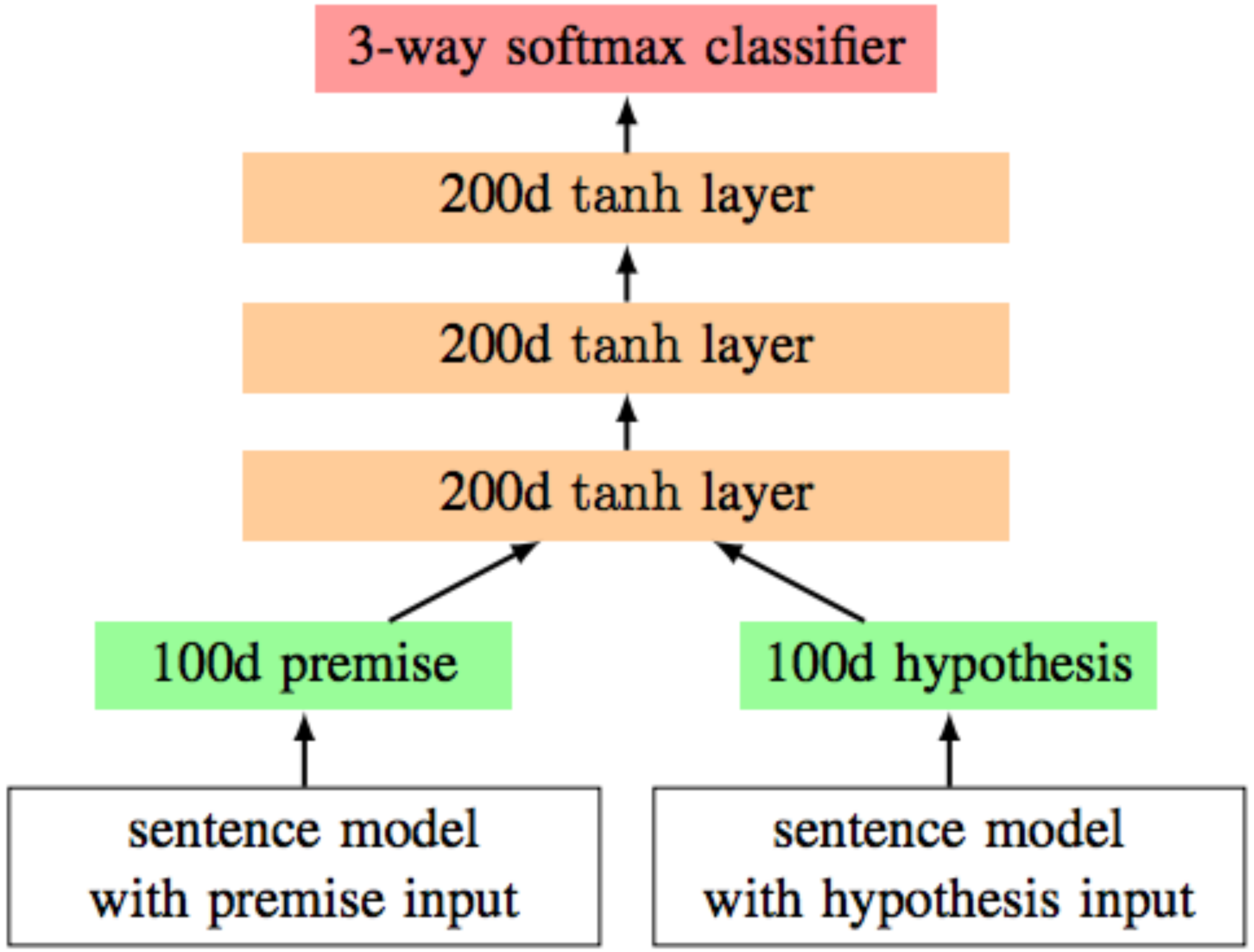
SNLI Dataset

- 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)

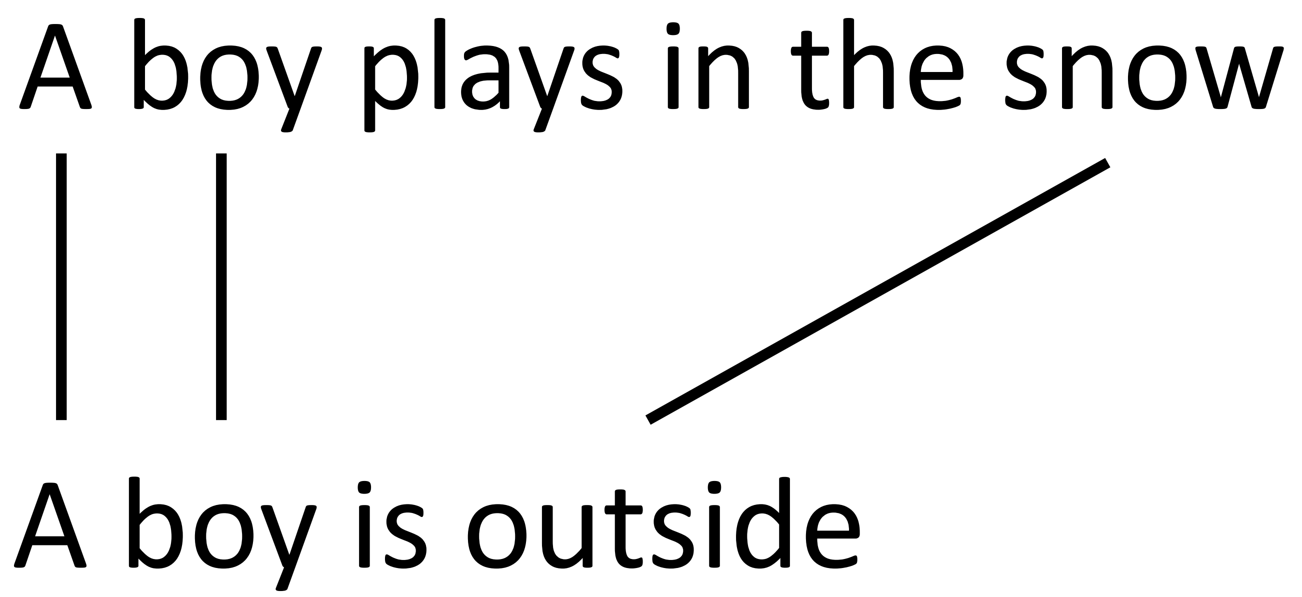
300D BiLSTM: 83% accuracy
(Liu et al., 2016)



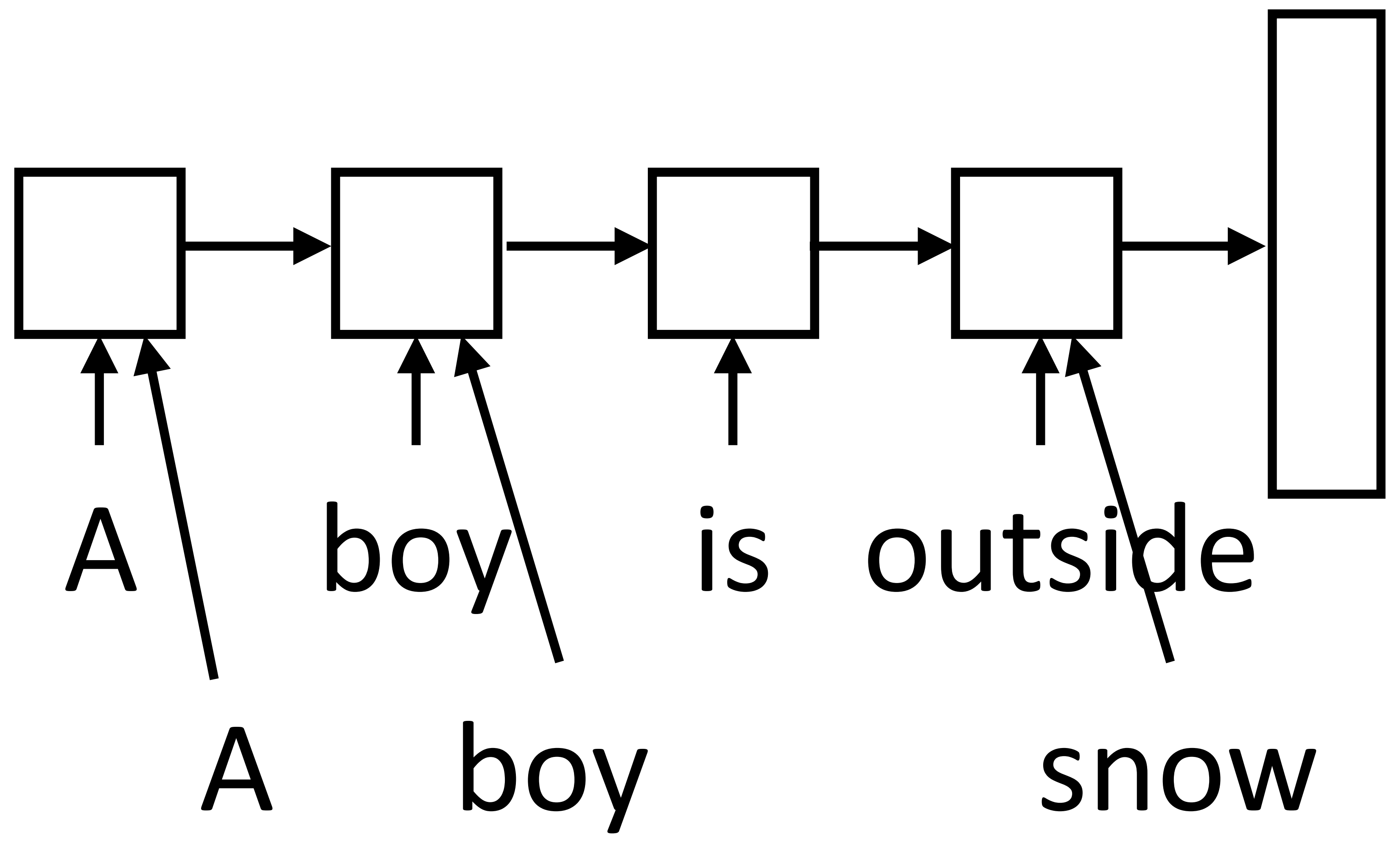
Bowman et al. (2015)

Aligned Inputs

- Two statements often have a natural alignment between them



- Process the hypothesis with knowledge of the premise

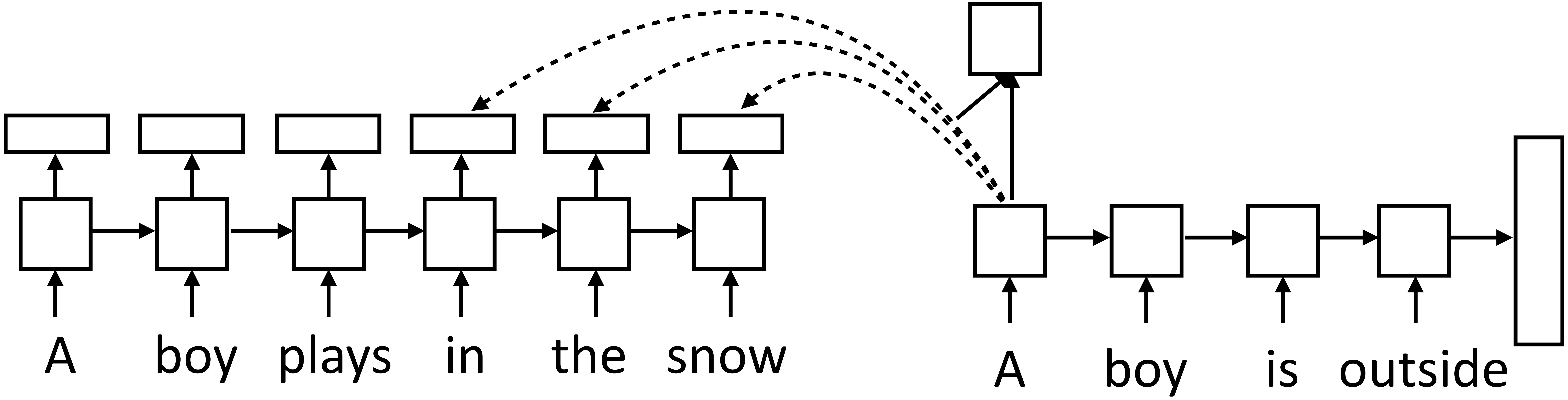


- Seeing the alignment lets you make entailment judgments as you're reading the sentence

Bowman et al. (2015)

Attention Mechanism

- *Learned* notion of alignment to some input



- Compare hidden state to encoded input vectors to compute alignment, use that to compute an input to further processing
- Attention models: 85-86% on SNLI, SOTA = 88%

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: encoder-decoder (seq2seq) models, machine translation
- Attention: critical idea that really makes it work!