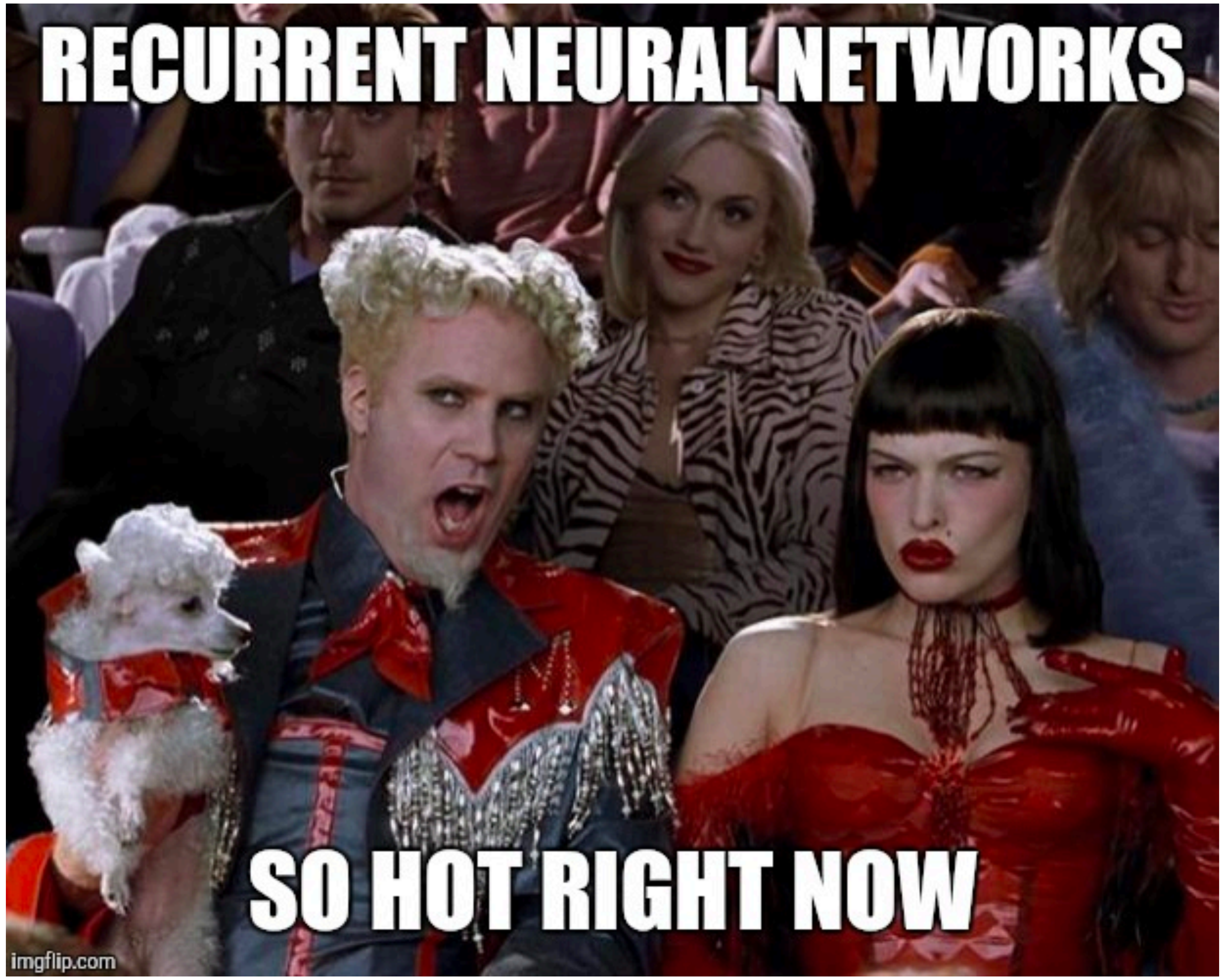
CS388: Natural Language Processing

Lecture 8: RNNs

Greg Durrett

The University of Texas at Austin



Credit: Chelsea Voss csvoss.com

Administrivia

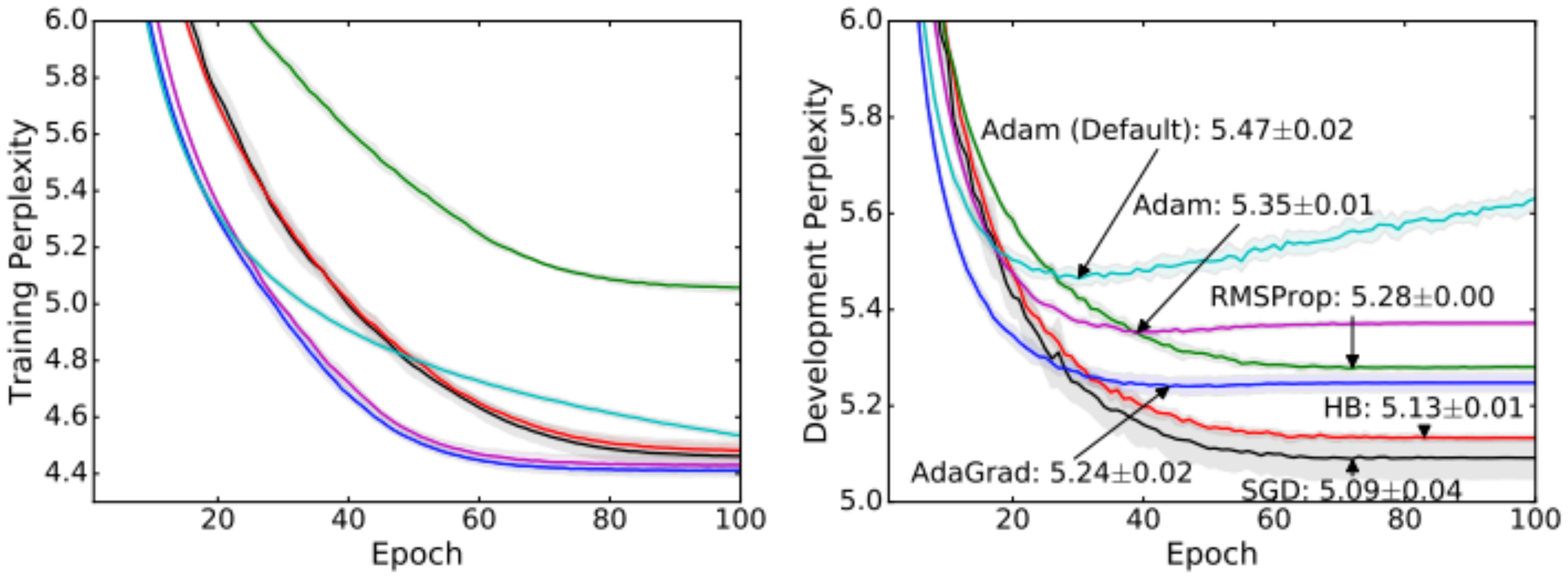
- Mini 1 results discussed at end of lecture
- Project 1 due **tonight**
- Mini 2 out Thursday

Recall: Training Tips

- Parameter initialization is critical to get good gradients, some useful heuristics (e.g., Glorot initializer)

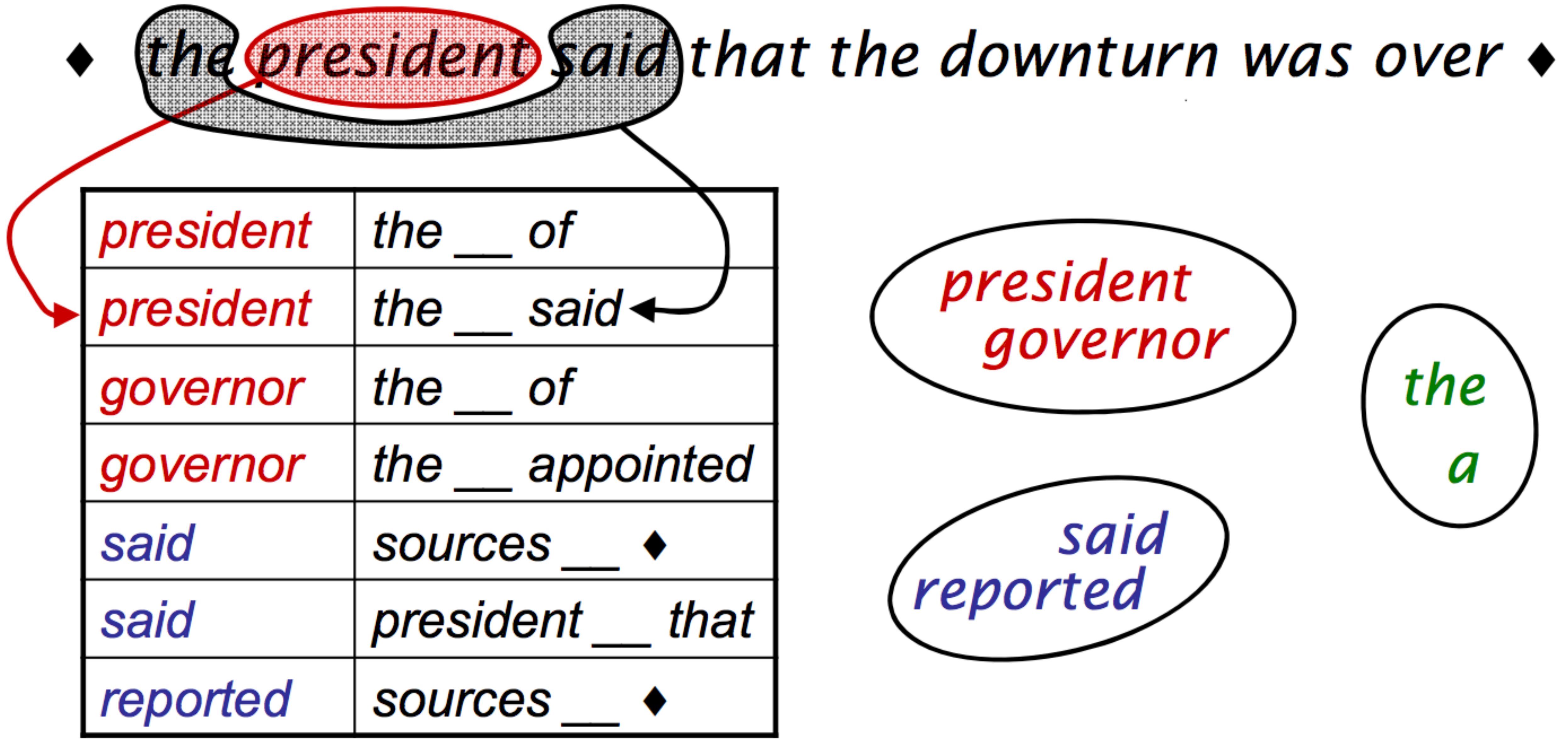
- Dropout is an effective regularizer, gradient clipping is useful

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

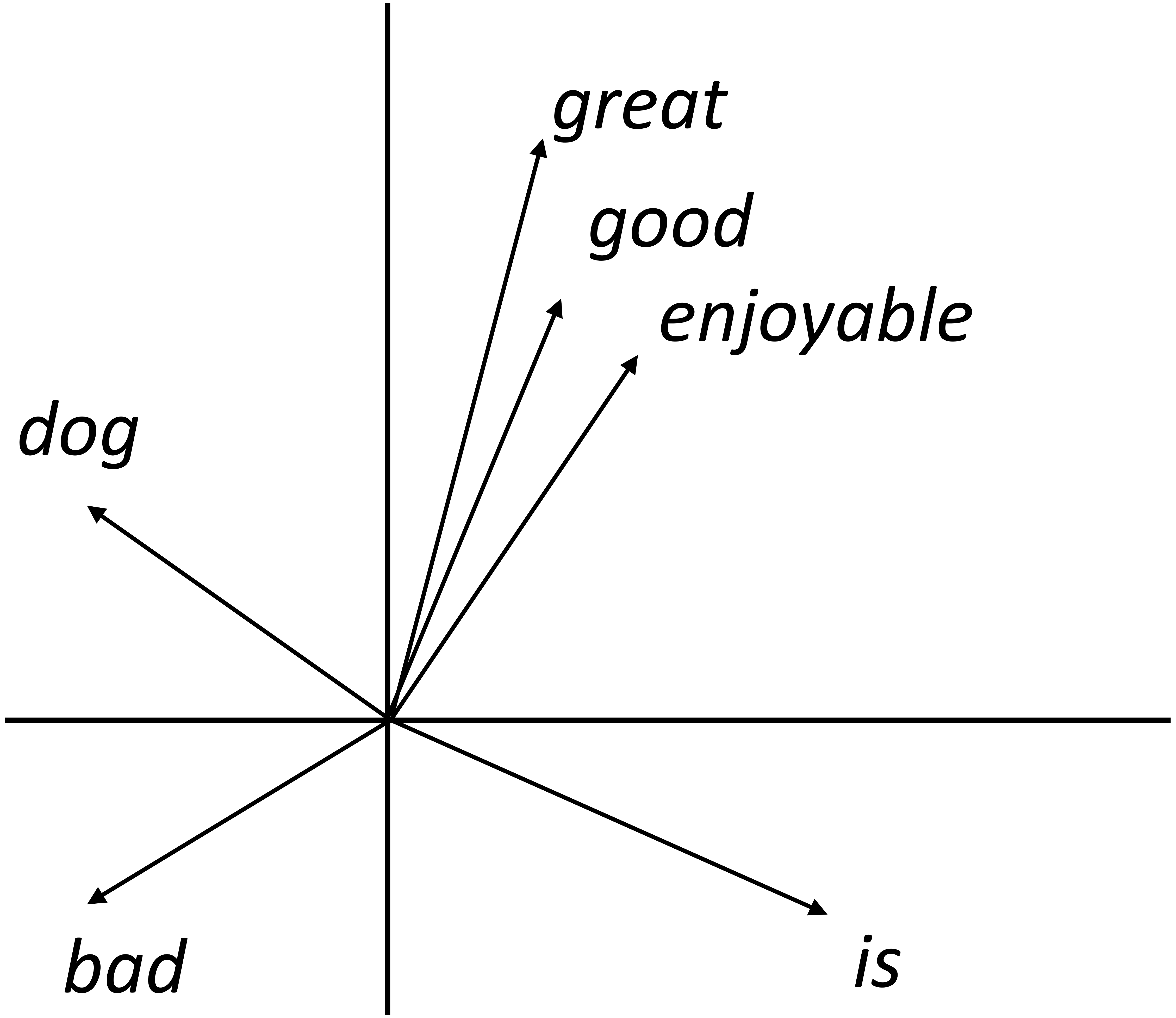


(e) Generative Parsing (Training Set)　(f) Generative Parsing (Development Set)

Recall: Word Vectors



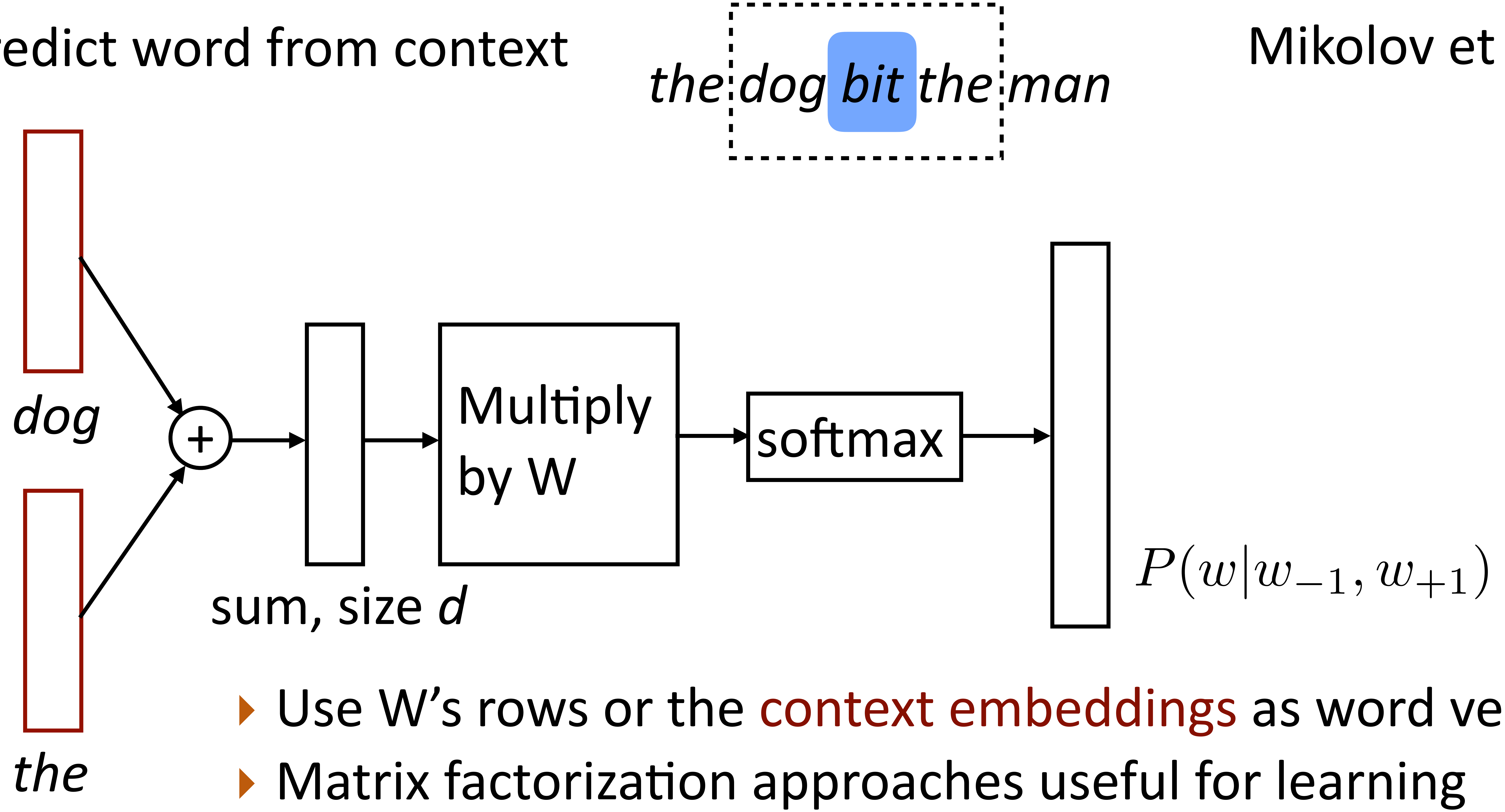
[Finch and Chater 92, Shuetze 93, many others]



Recall: Continuous Bag-of-Words

- Predict word from context

Mikolov et al. (2013)



- Use W's rows or the context embeddings as word vectors
- Matrix factorization approaches useful for learning vectors from really large data

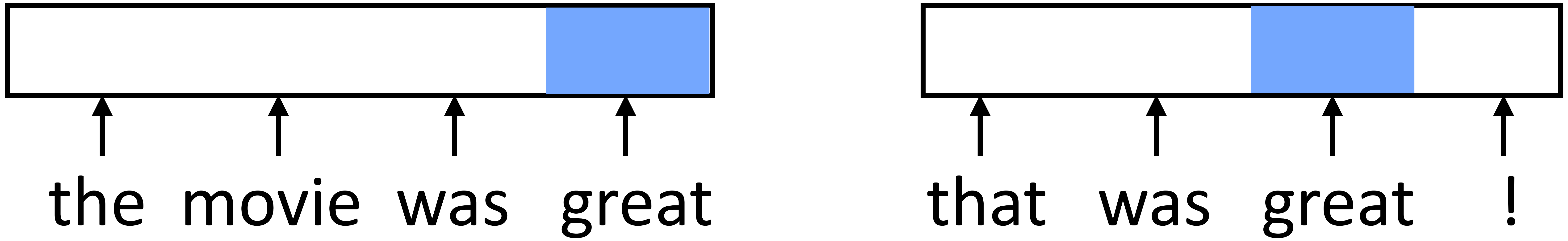
This Lecture

- 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



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

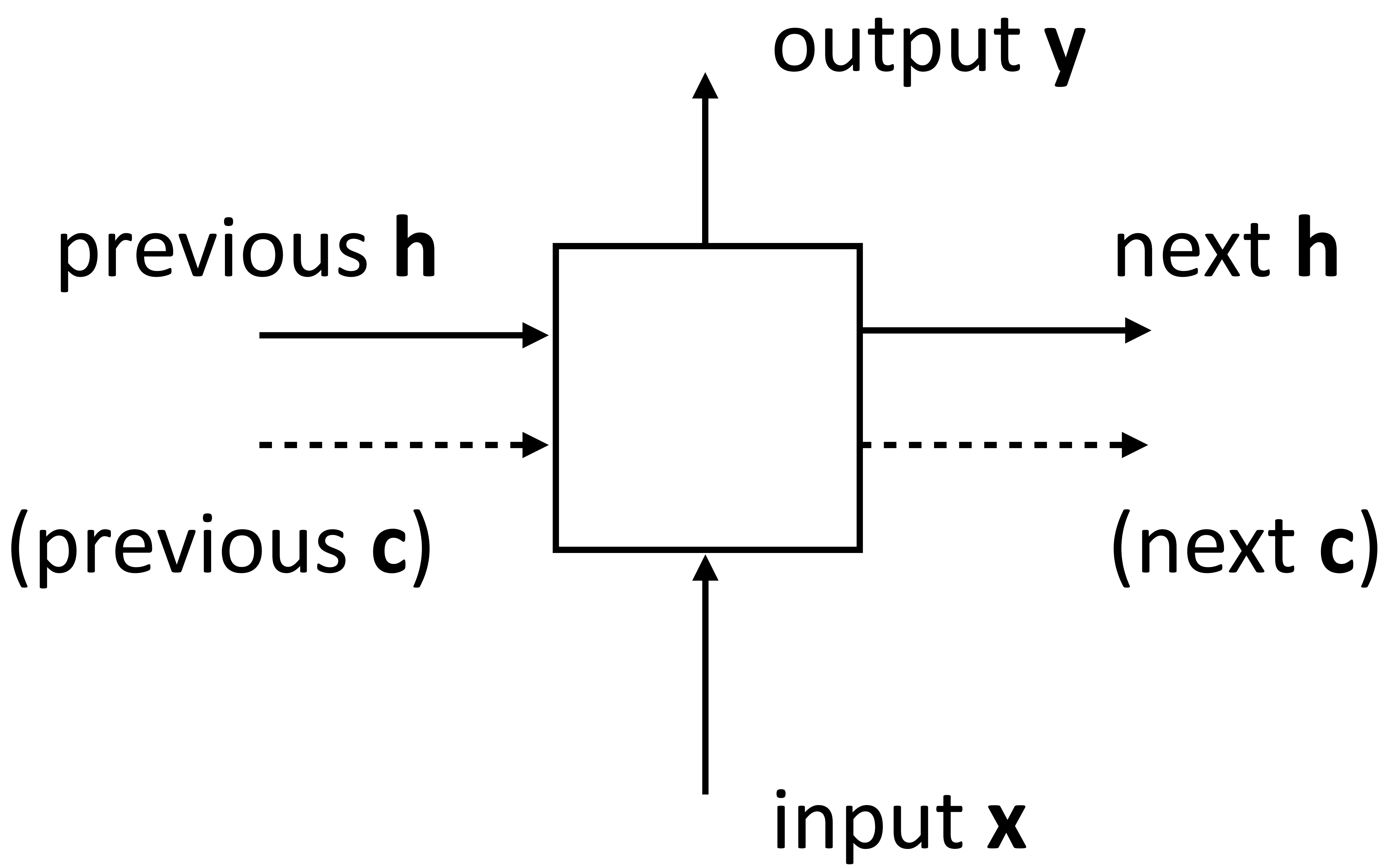
- Instead, we need to:

1) Process each word in a uniform way

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

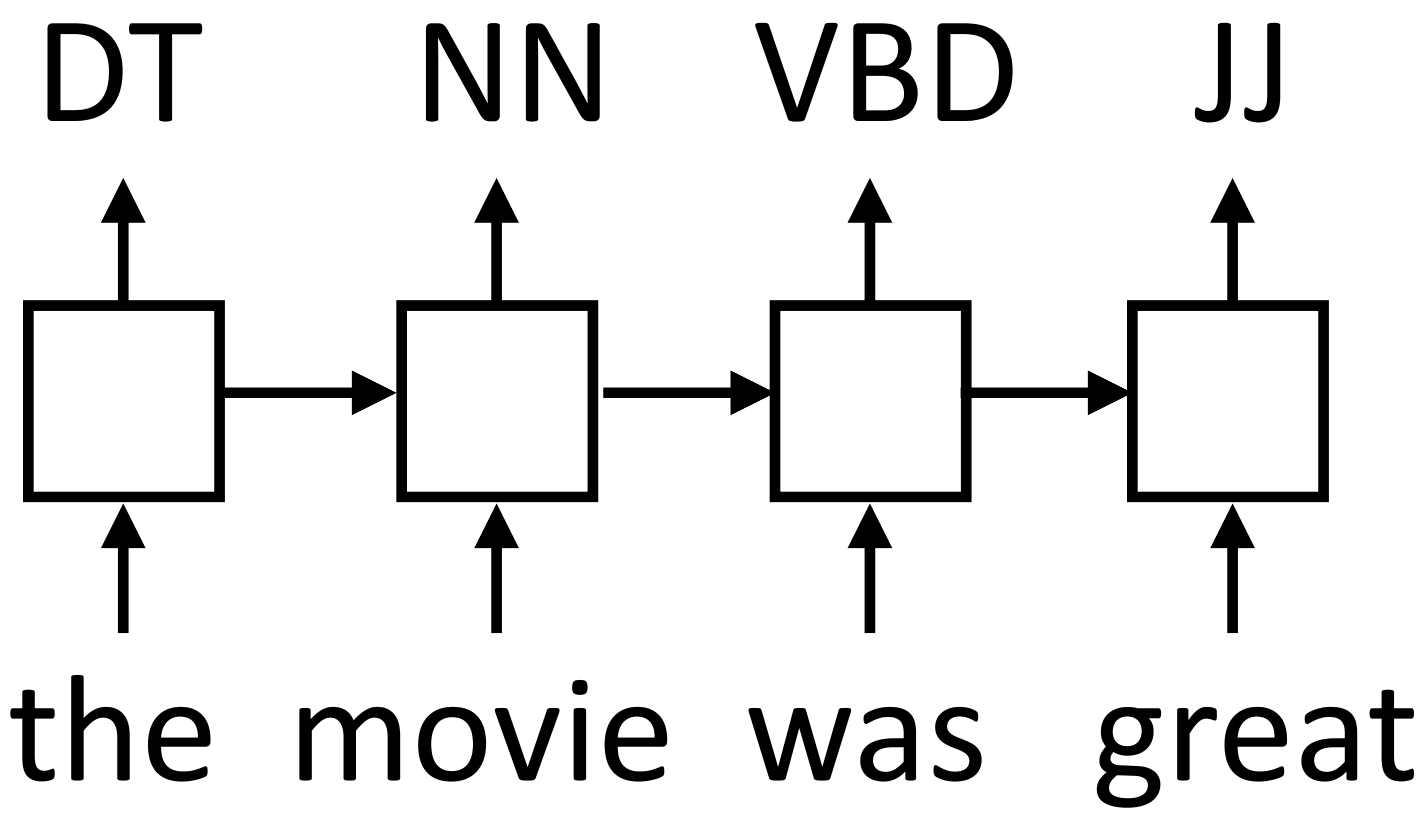
RNN Abstraction

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



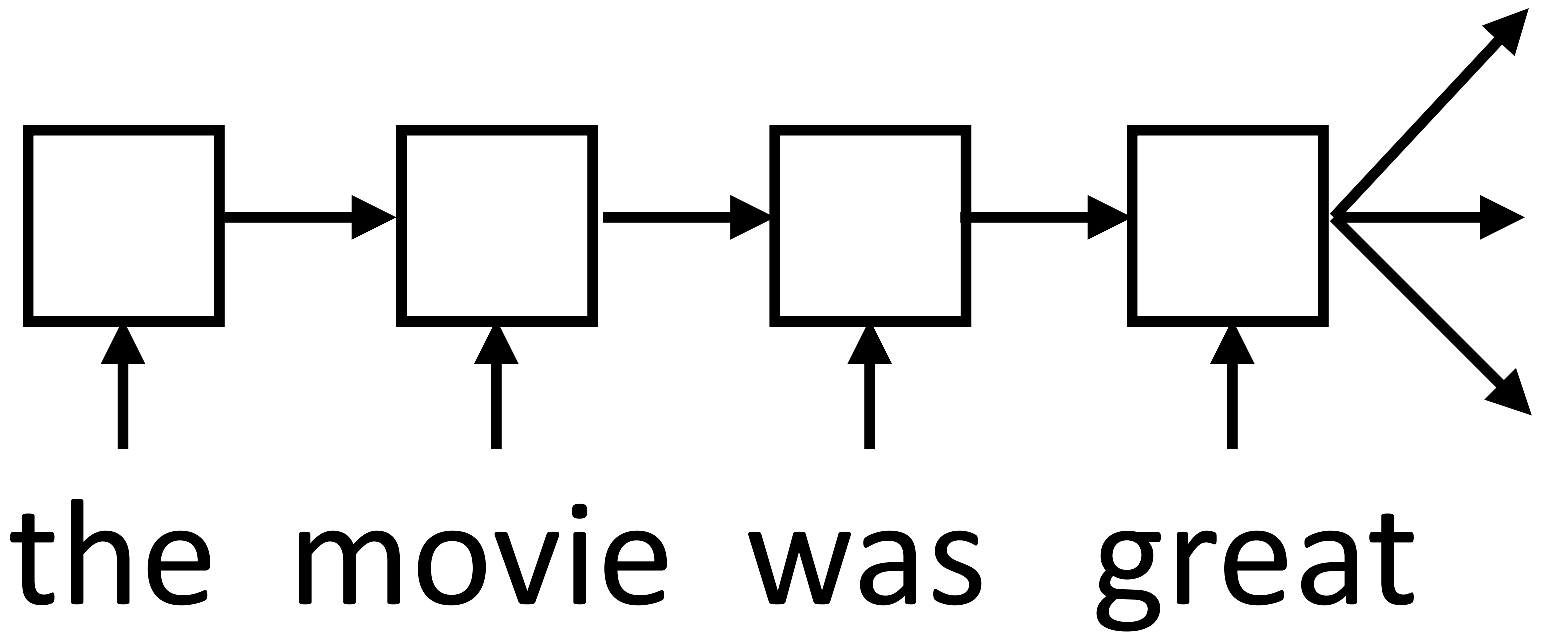
RNN Uses

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



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

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

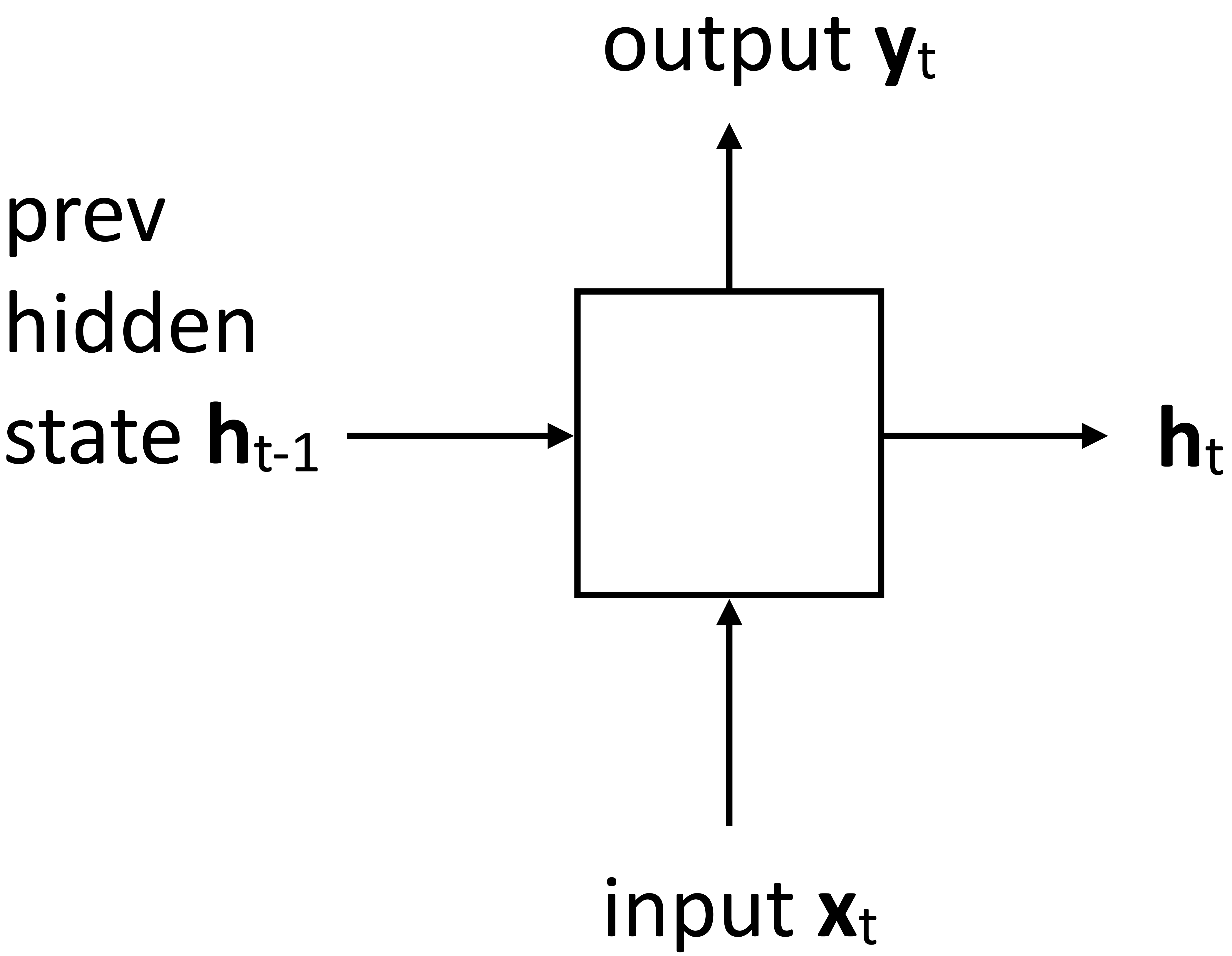


predict sentiment (matmul + softmax)

translate

paraphrase/compress

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

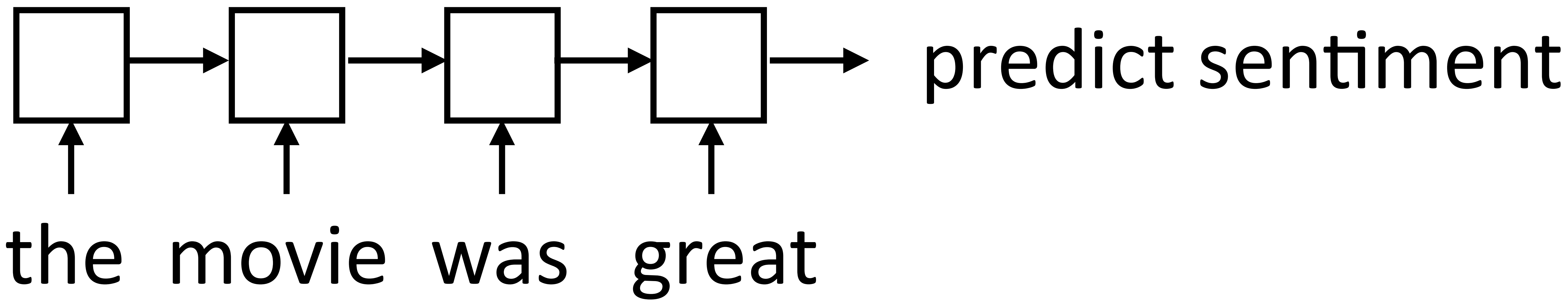
$$\mathbf{y}_t = \tanh(U\mathbf{h}_\mathbf{t} + \mathbf{b}_y)$$

- Computes output from hidden state

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

Elman (1990)

Training Elman Networks



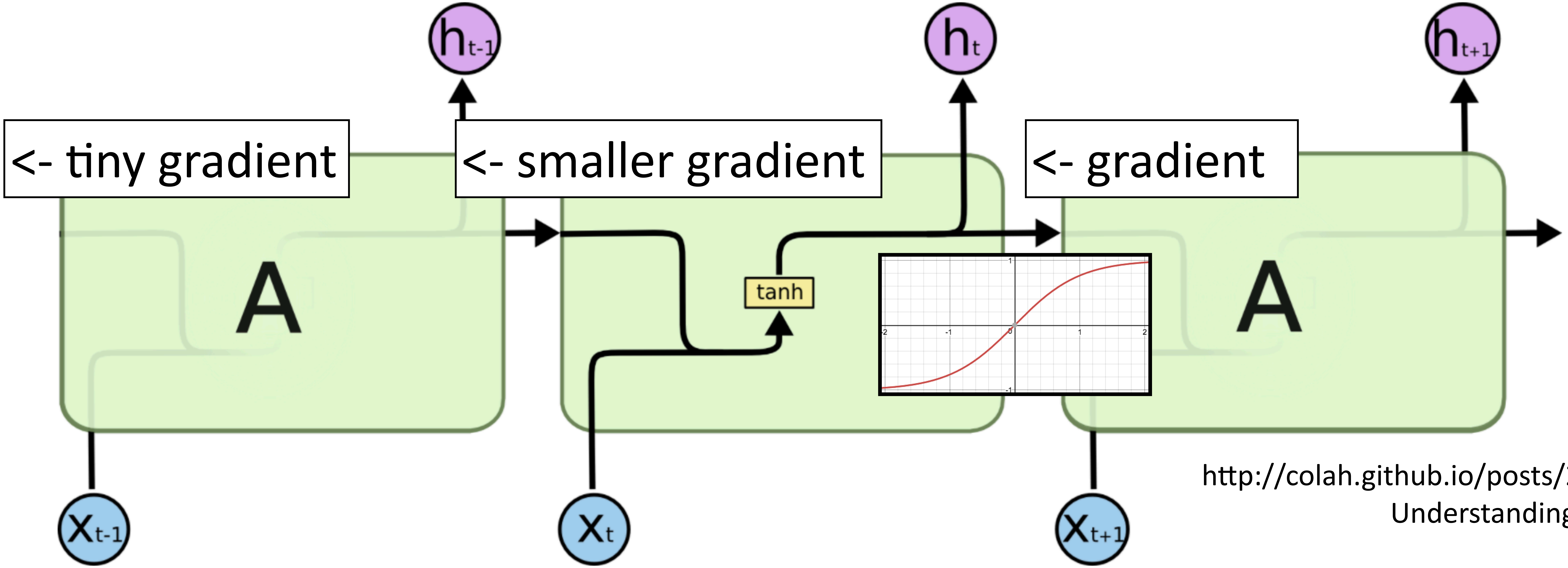
- "Backpropagation through time": build the network as one big computation graph, some parameters are shared

- 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*

Vanishing Gradient



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

- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0
- Repeated multiplication by V causes problems $\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$

LSTMs/GRUs

Gated Connections

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

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

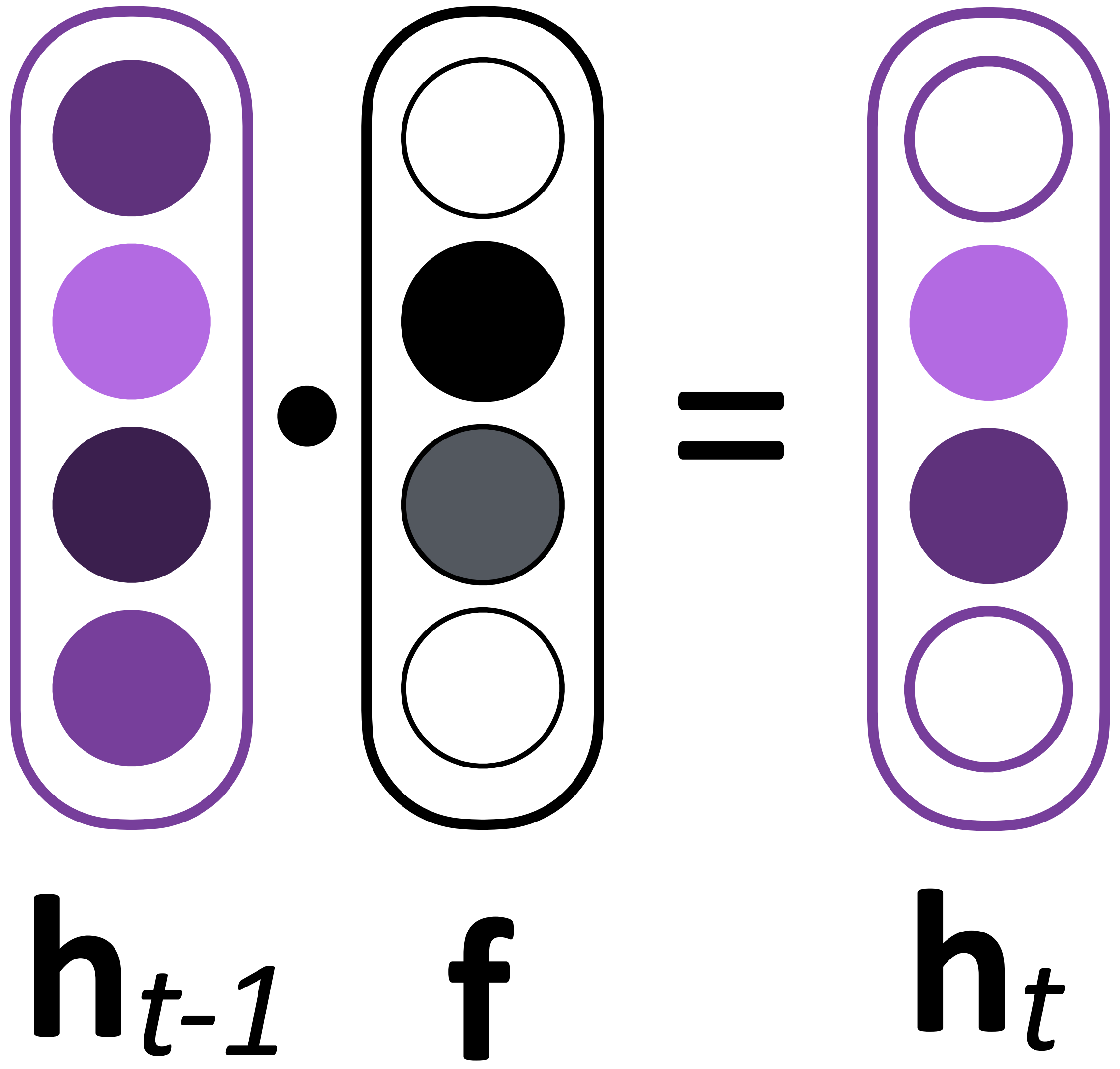
gated

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

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! More stable without matrix multiply (*V*) as well

LSTMs

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

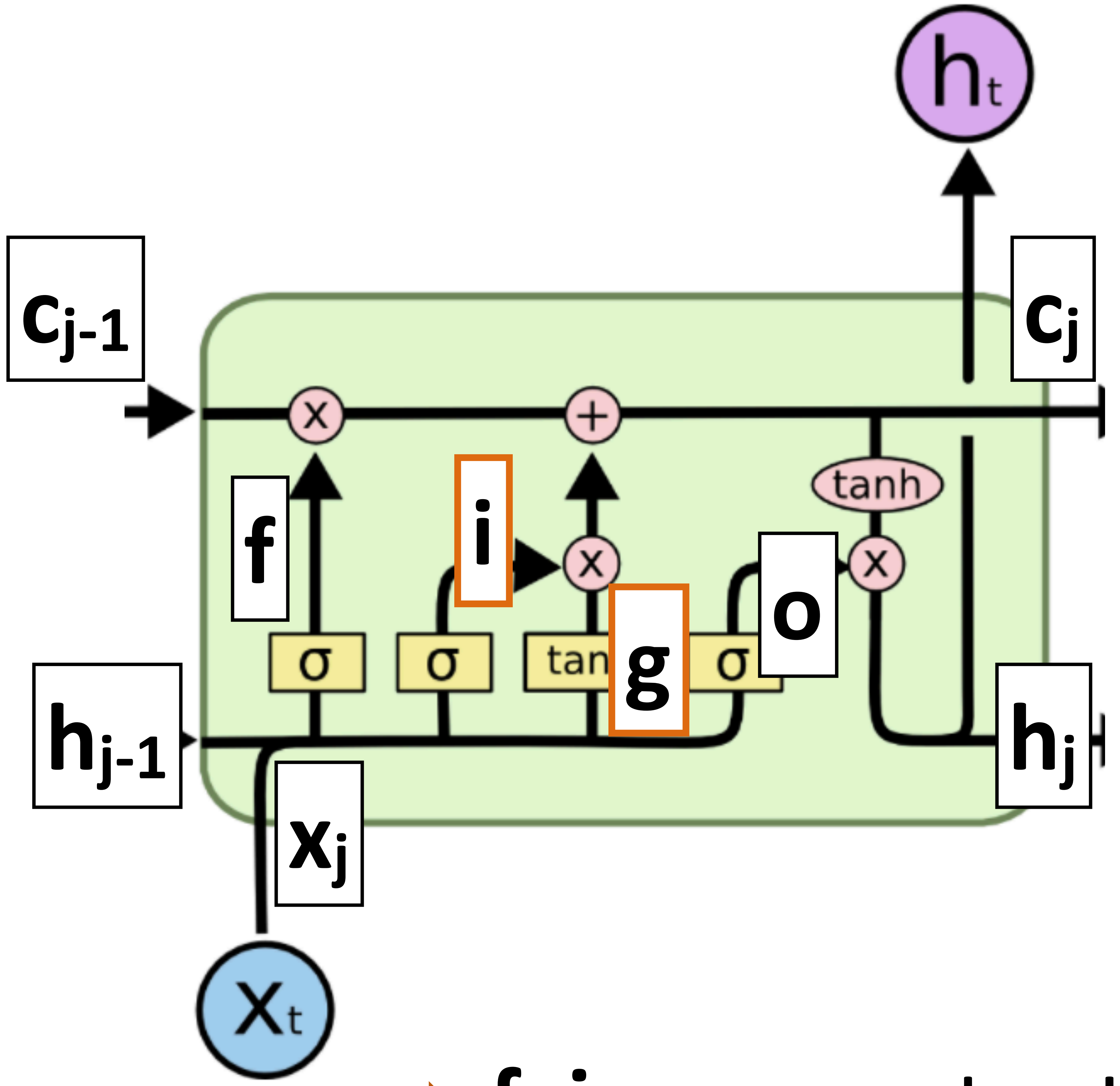
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- Vector-valued forget gate **f** 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



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

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

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

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

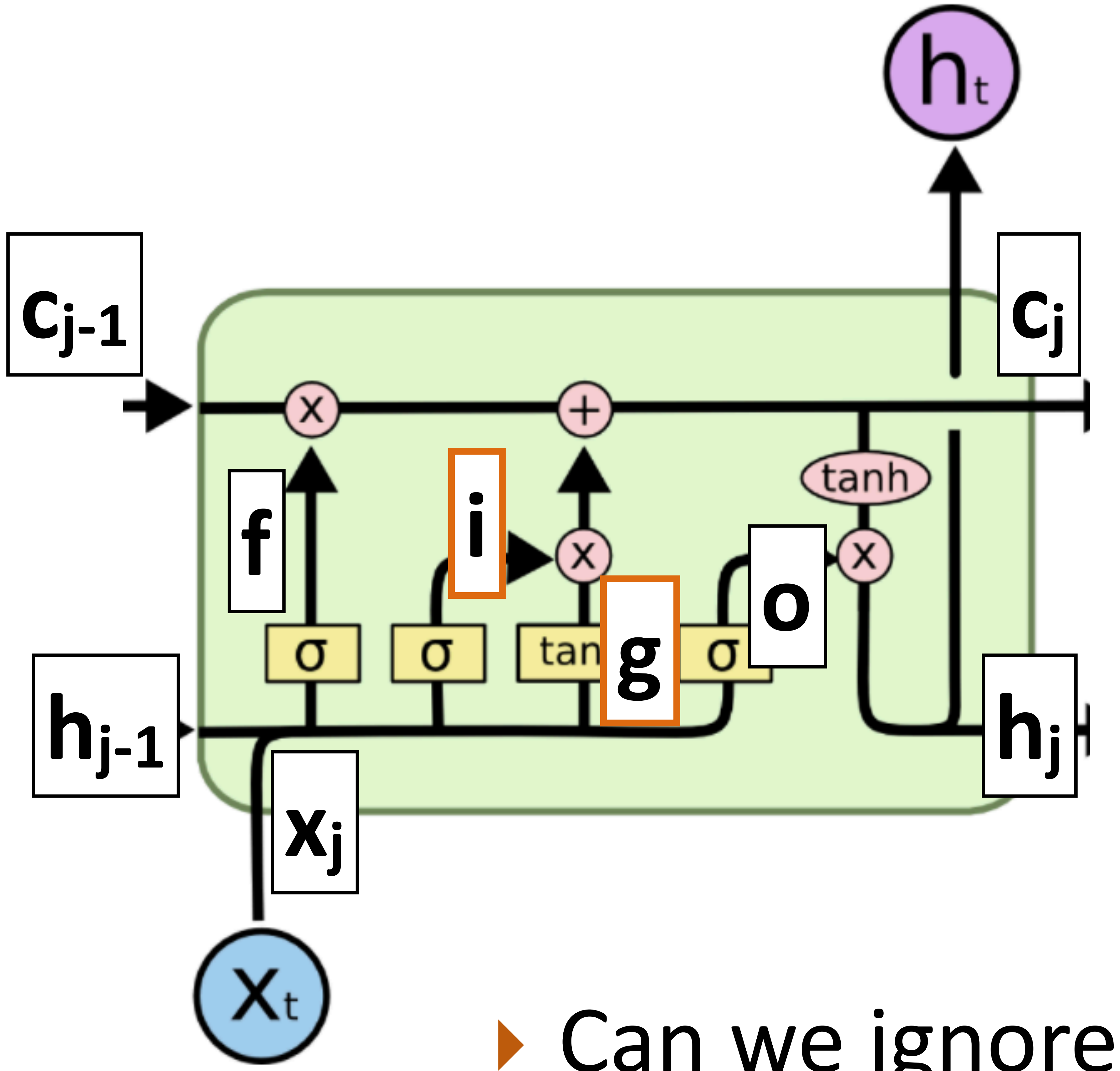
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

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

Goldberg lecture notes

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

LSTMs



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

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

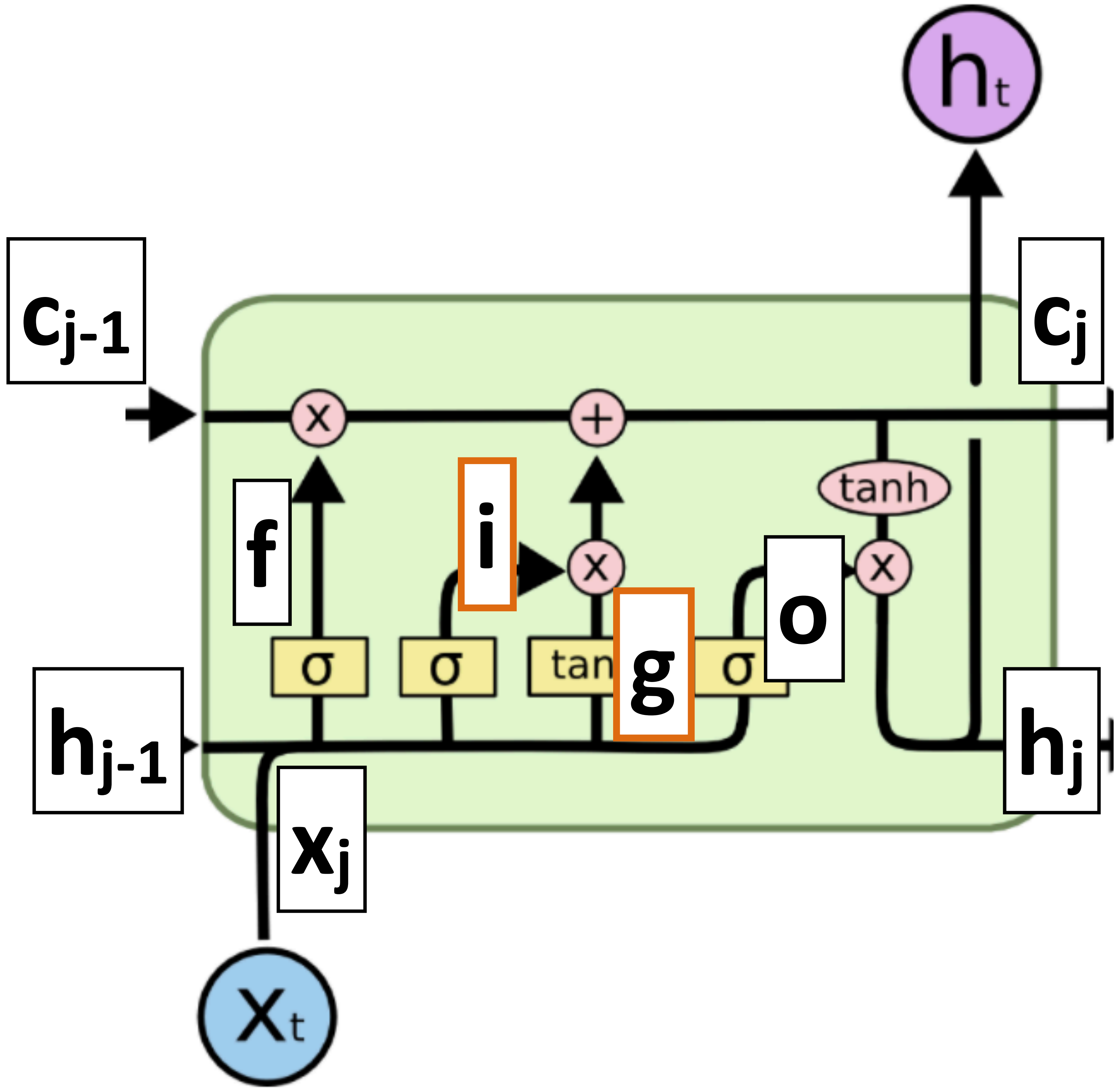
$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

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

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

- 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

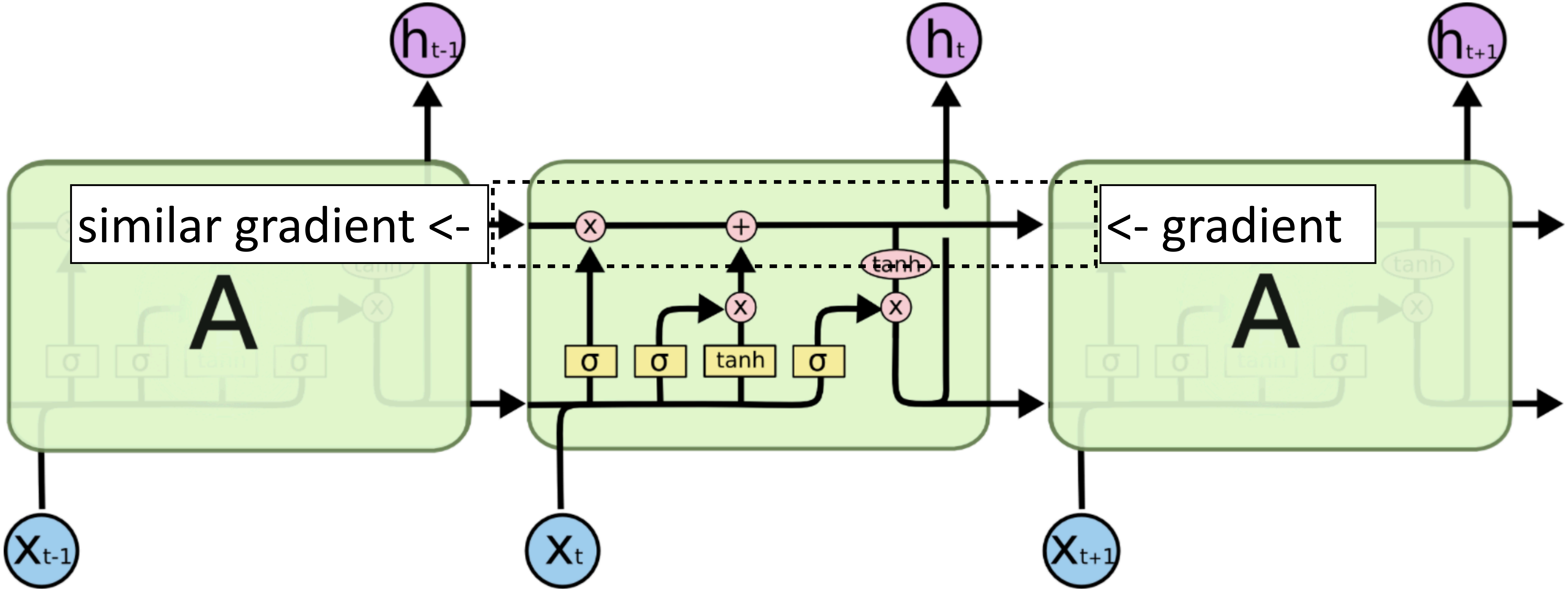


- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Goldberg lecture notes

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

LSTMs

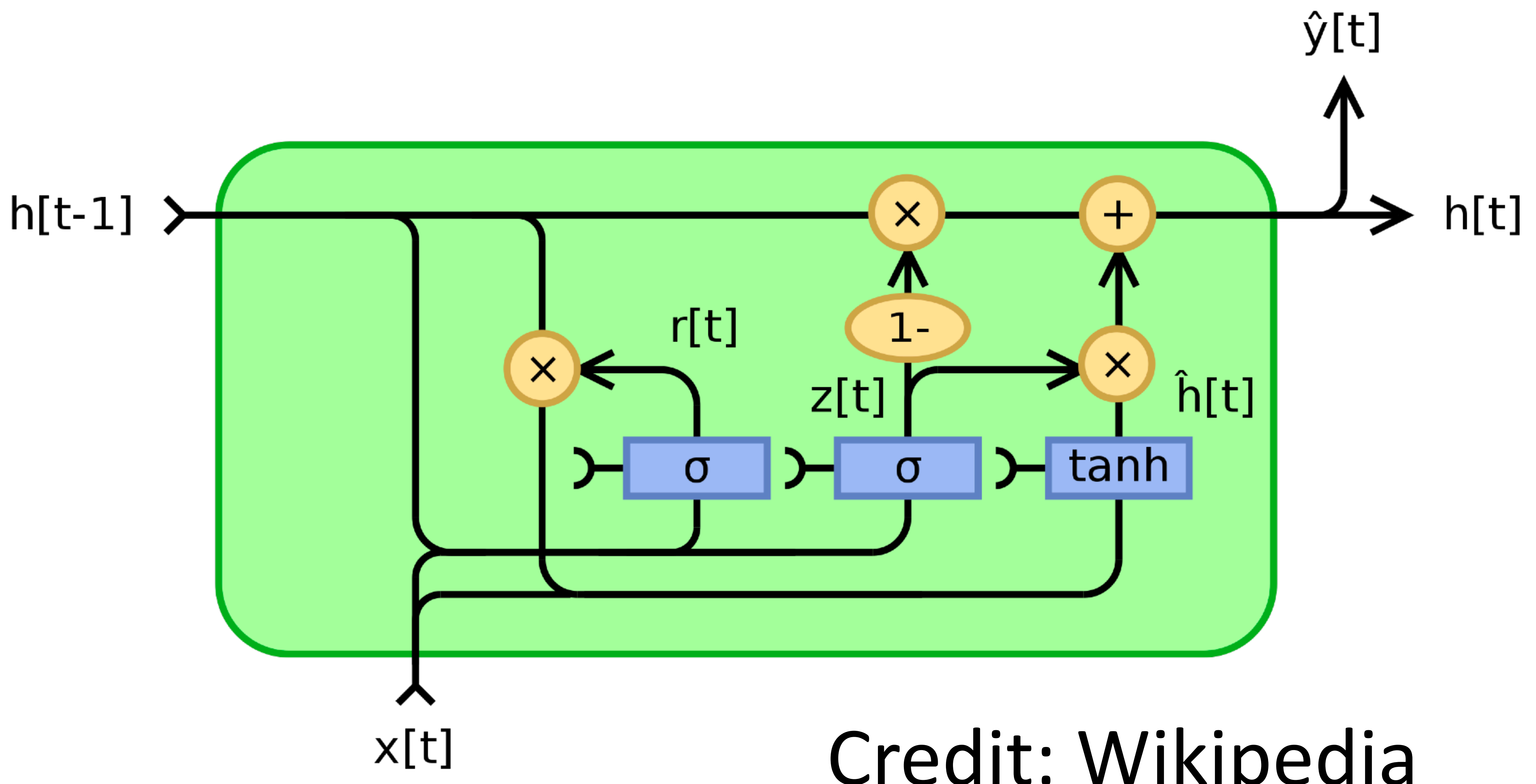


- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

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

GRUs

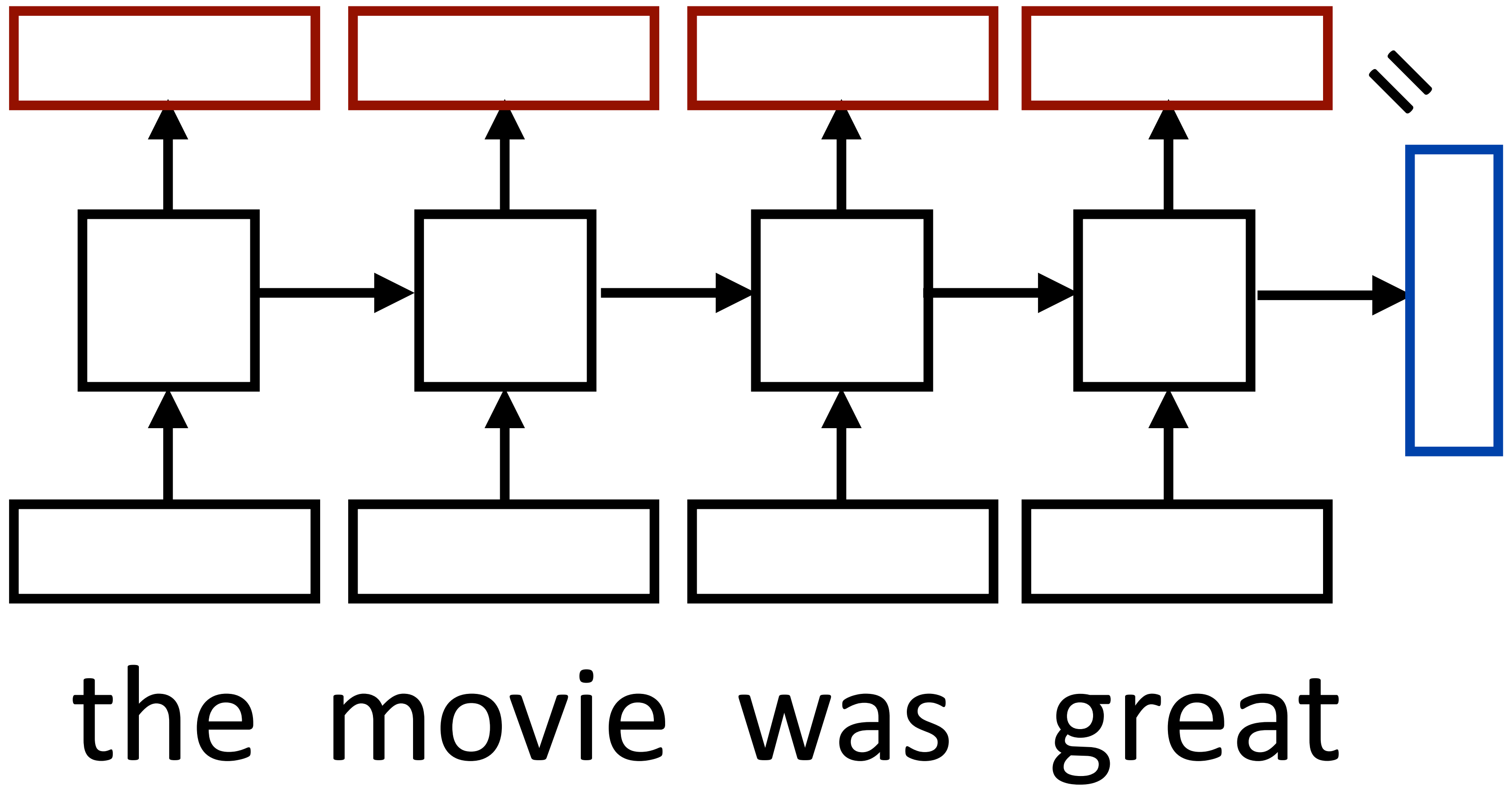
- **z** is update, **r** is reset
- The single hidden state and simpler update gate gives simpler mixing semantics than in LSTMs
- Faster to train and sometimes works better than LSTMs, often a tossup



Credit: Wikipedia

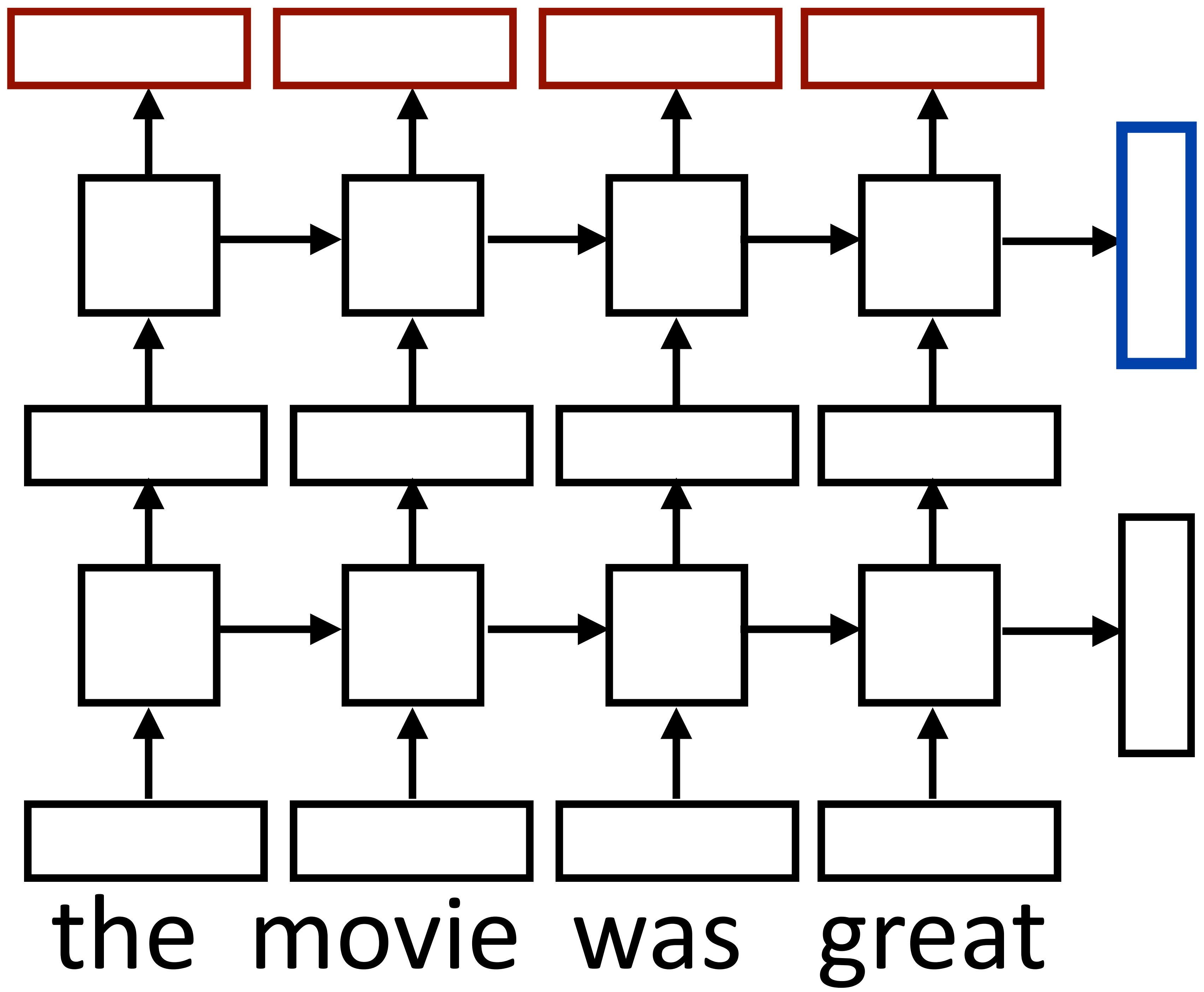
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

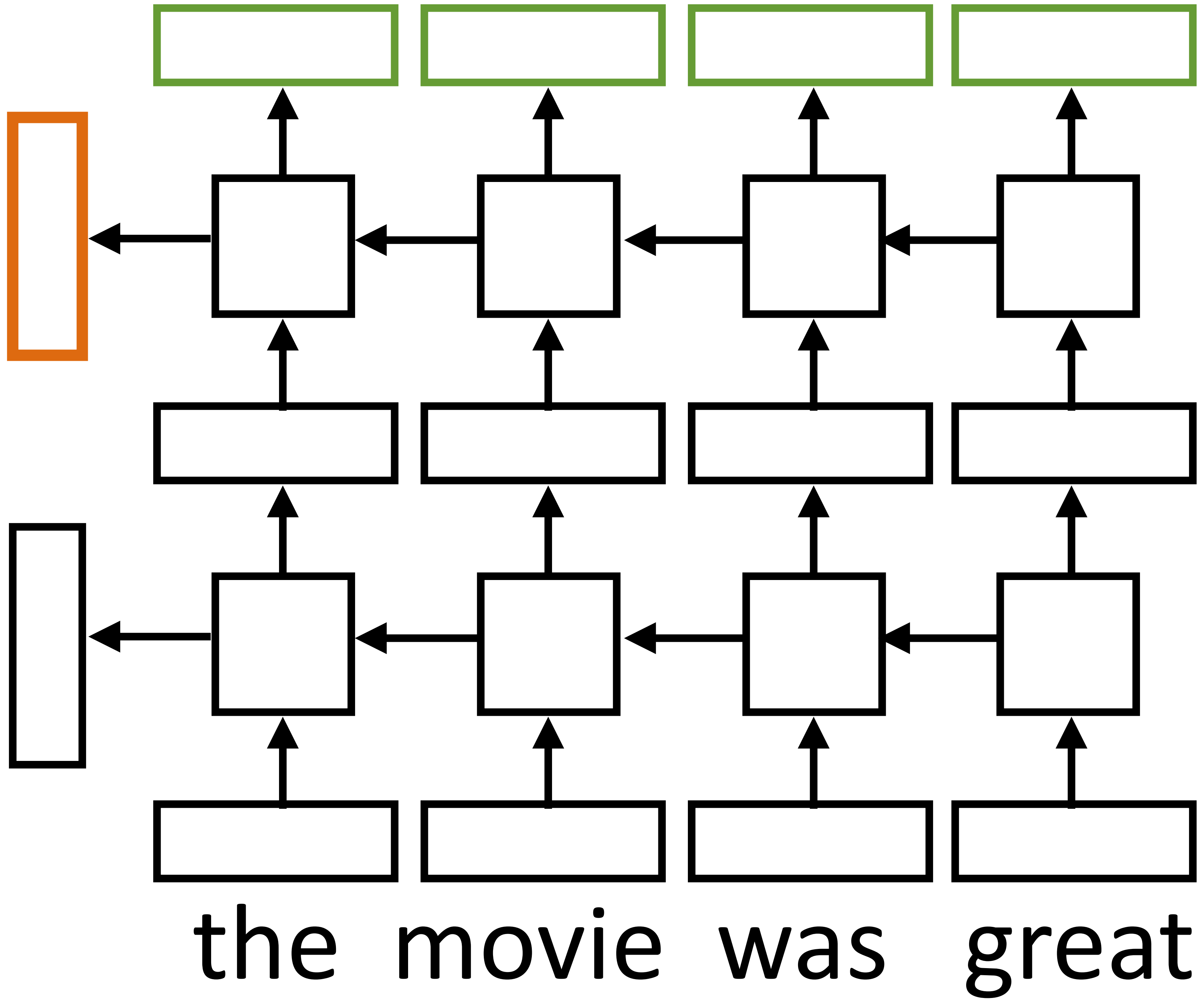
What do RNNs produce?



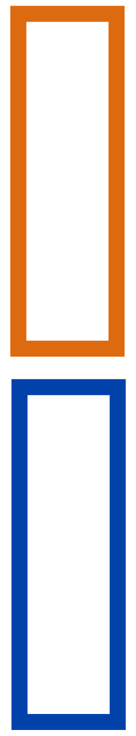
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors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

Multilayer Bidirectional RNN

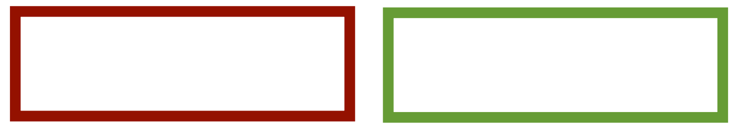




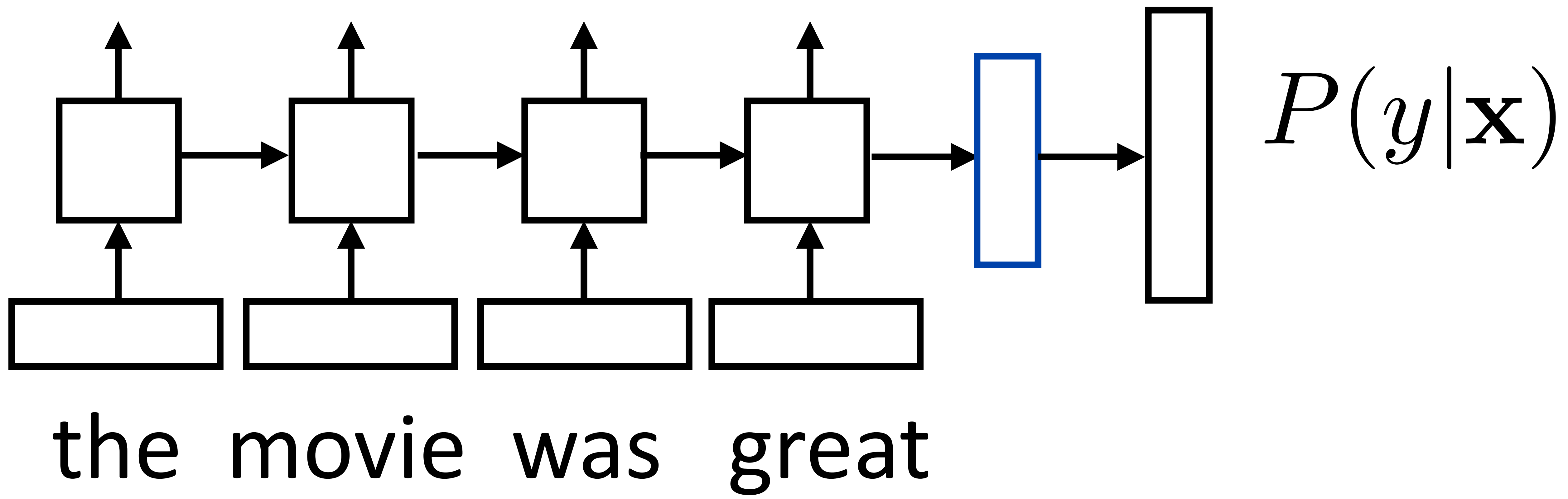
- Sentence classification based on concatenation of both final outputs



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

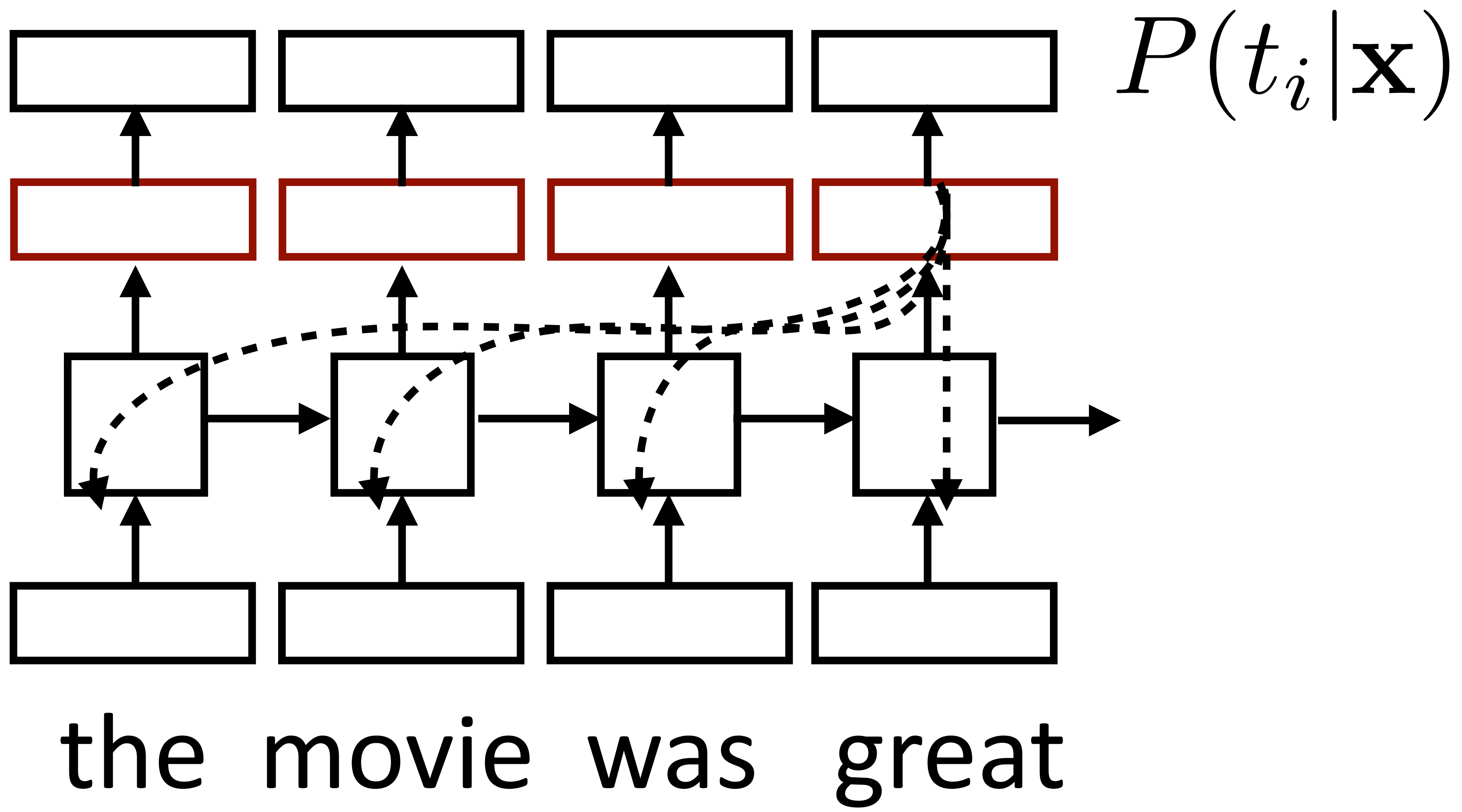


Training RNNs



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

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)

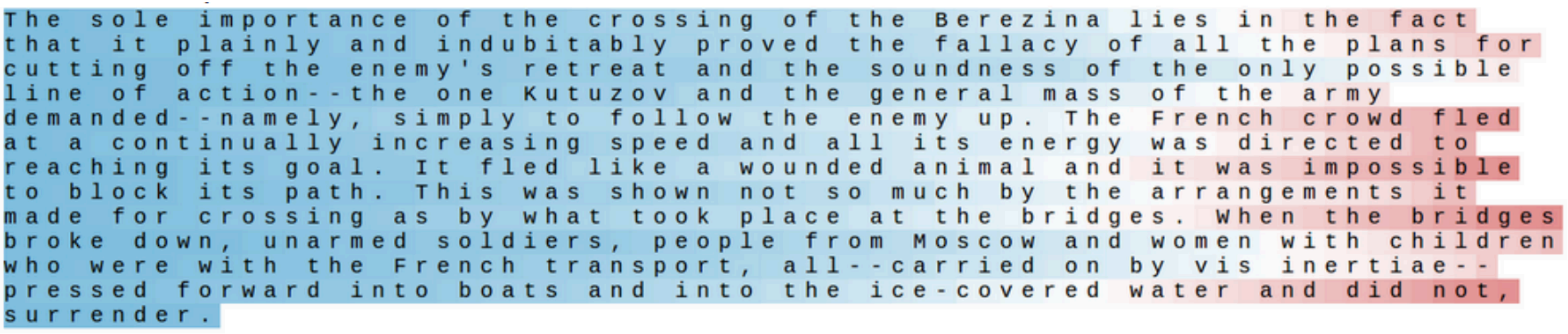
Applications

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

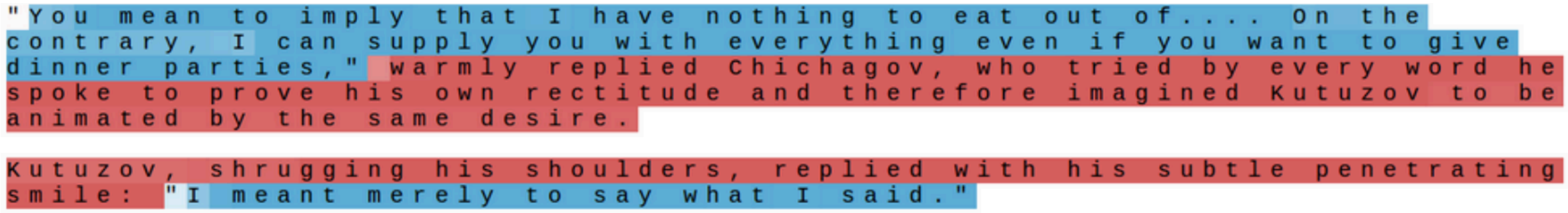
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n



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: tells us if we're in a quote or not



Karpathy et al. (2015)

Visualizing LSTMs

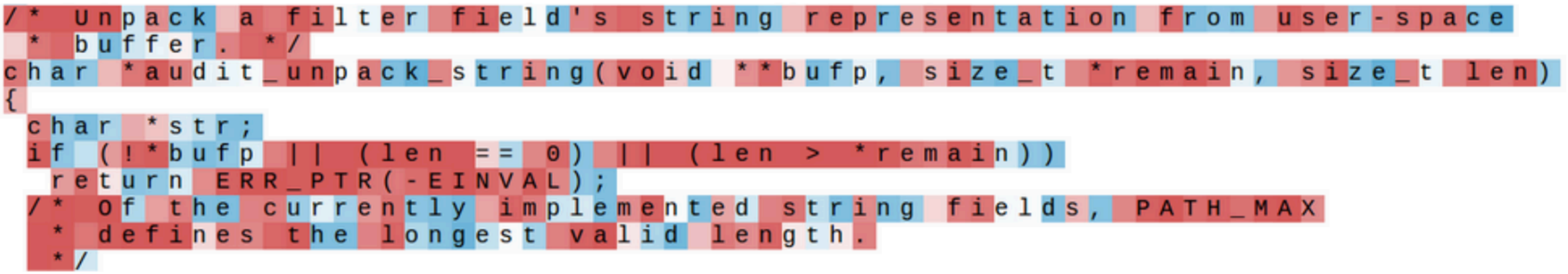
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#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;
}
```

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

Sentiment Analysis

- Semi-supervised method: initialize the language model by training to reproduce the document in a seq2seq fashion (discussed in a few lectures), called a sequential autoencoder

Model	Test error rate
LSTM with tuning and dropout	13.50%
LSTM initialized with word2vec embeddings	10.00%
LM-LSTM (see Section 2)	7.64%
SA-LSTM (see Figure 1)	7.24%
Full+Unlabeled+BoW [21]	11.11%
WRRBM + BoW (bnc) [21]	10.77%
NBSVM-bi (Naïve Bayes SVM with bigrams) [35]	8.78%
seq2-bown-CNN (ConvNet with dynamic pooling) [11]	7.67%
Paragraph Vectors [18]	7.42%

better than tuned Naive Bayes when using the SA trick

Dai and Le (2015)

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

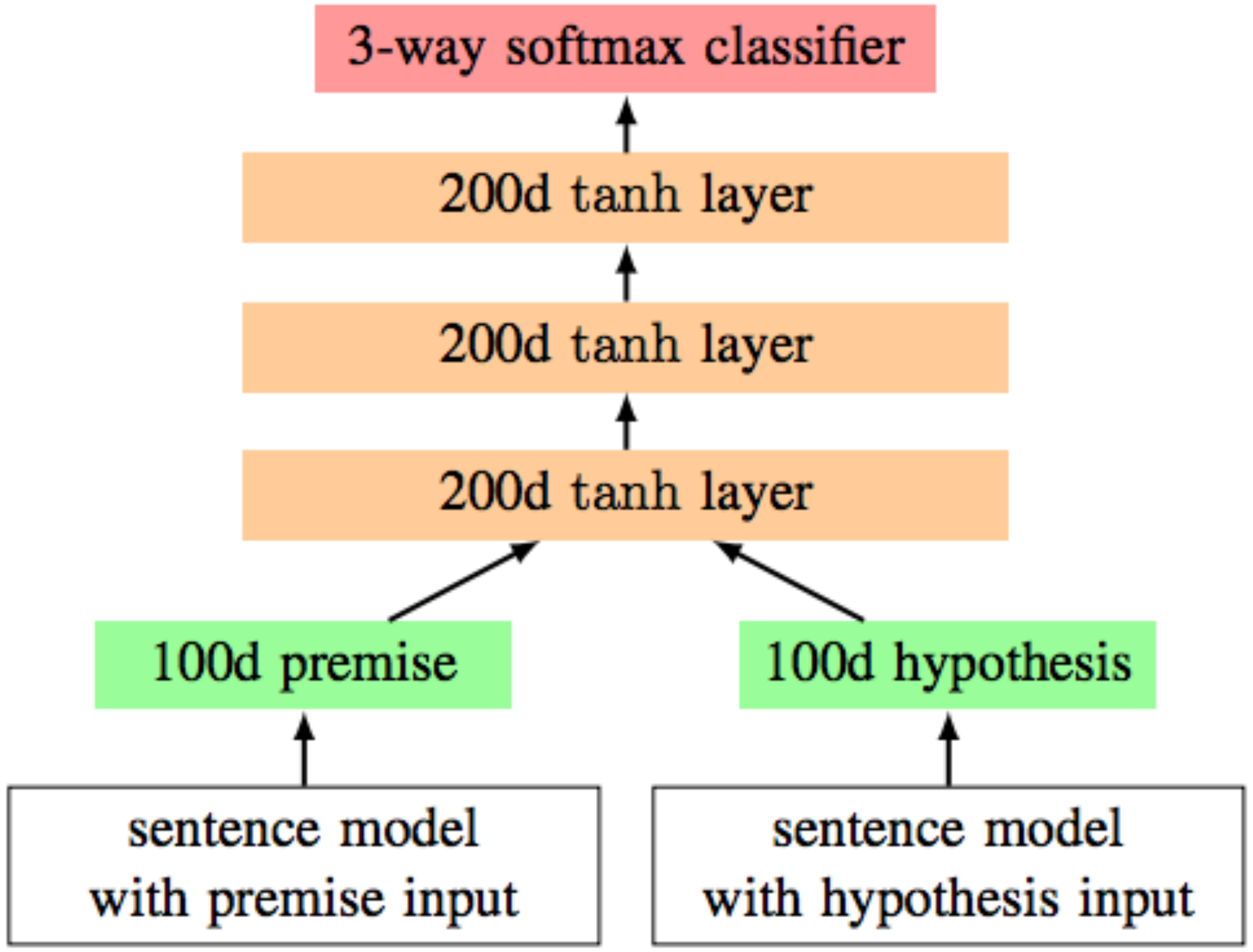
300D LSTM: 80% accuracy

(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

Mini 1 Results

- Mini 1 test F1 results:

Xiaoyang Shen 87.60

Rajat Jain 87.59

Kaj Bostrom 87.32

Yejin Cho 87.24

- L2 regularization, shuffling across epochs, class weighting from sk-learn, +/-2 words and prefixes+suffixes
- Adding indicator of whether it was PERSON (gazetteer) in train hurt performance
- POS=NNP feature

> 87: Anubrata Das, Rudrajit Das, Fengyu Deng, Chinmoy Samant, Ting-Yu Yen