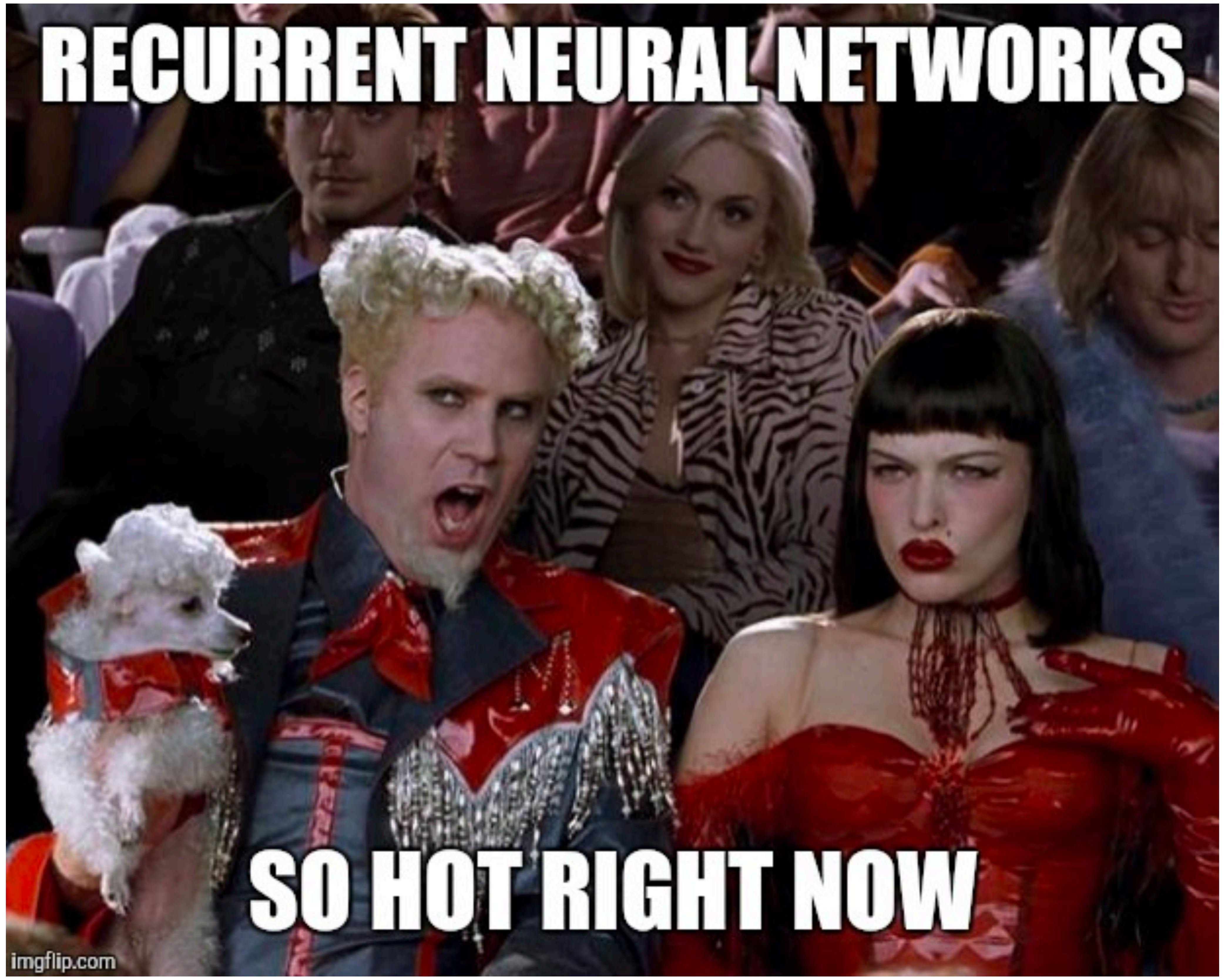
# CS388: Natural Language Processing

## Lecture 8: RNNs


Greg Durrett

TEXAS

The University of Texas at Austin

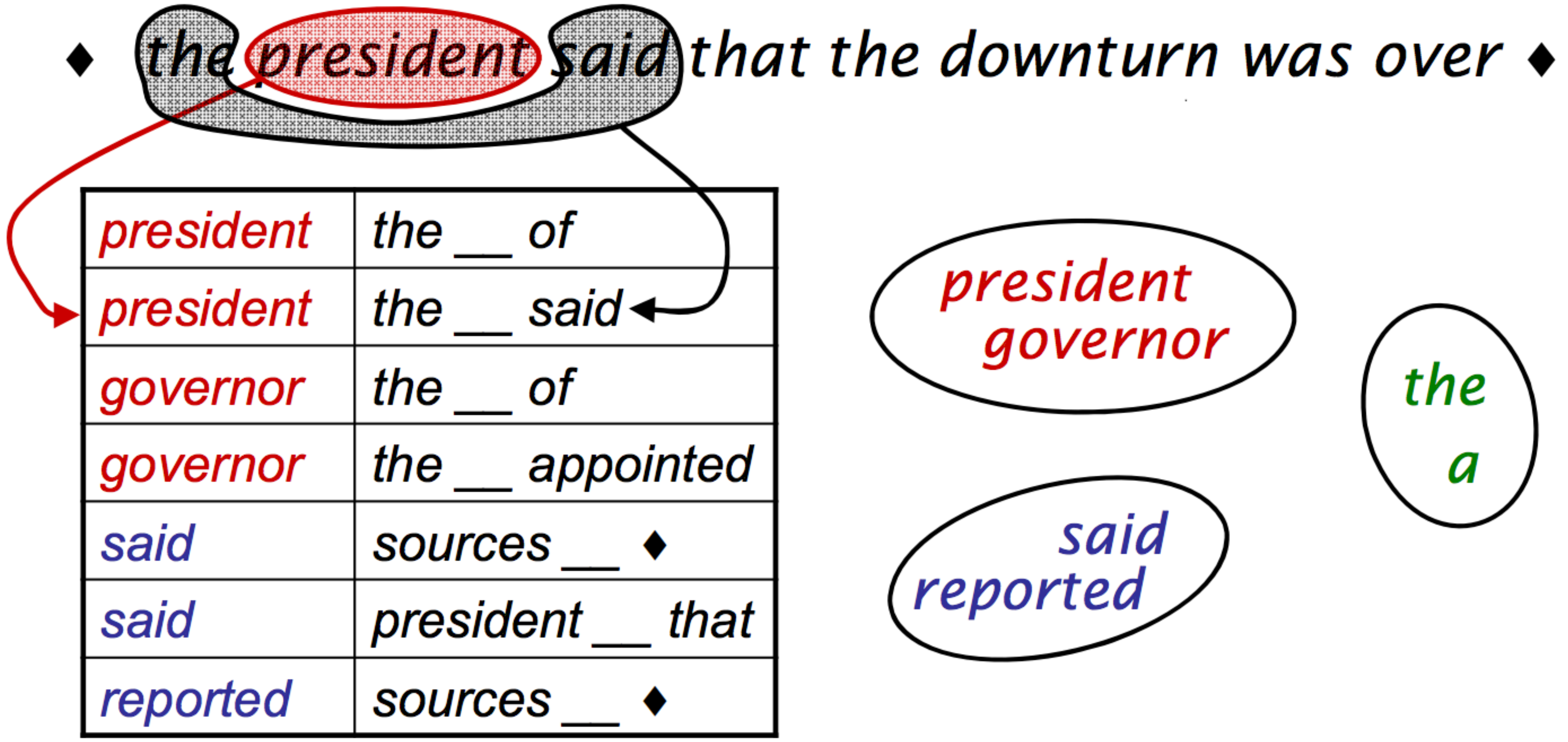




Credit: Chelsea Voss csvoss.com

# Administrivia

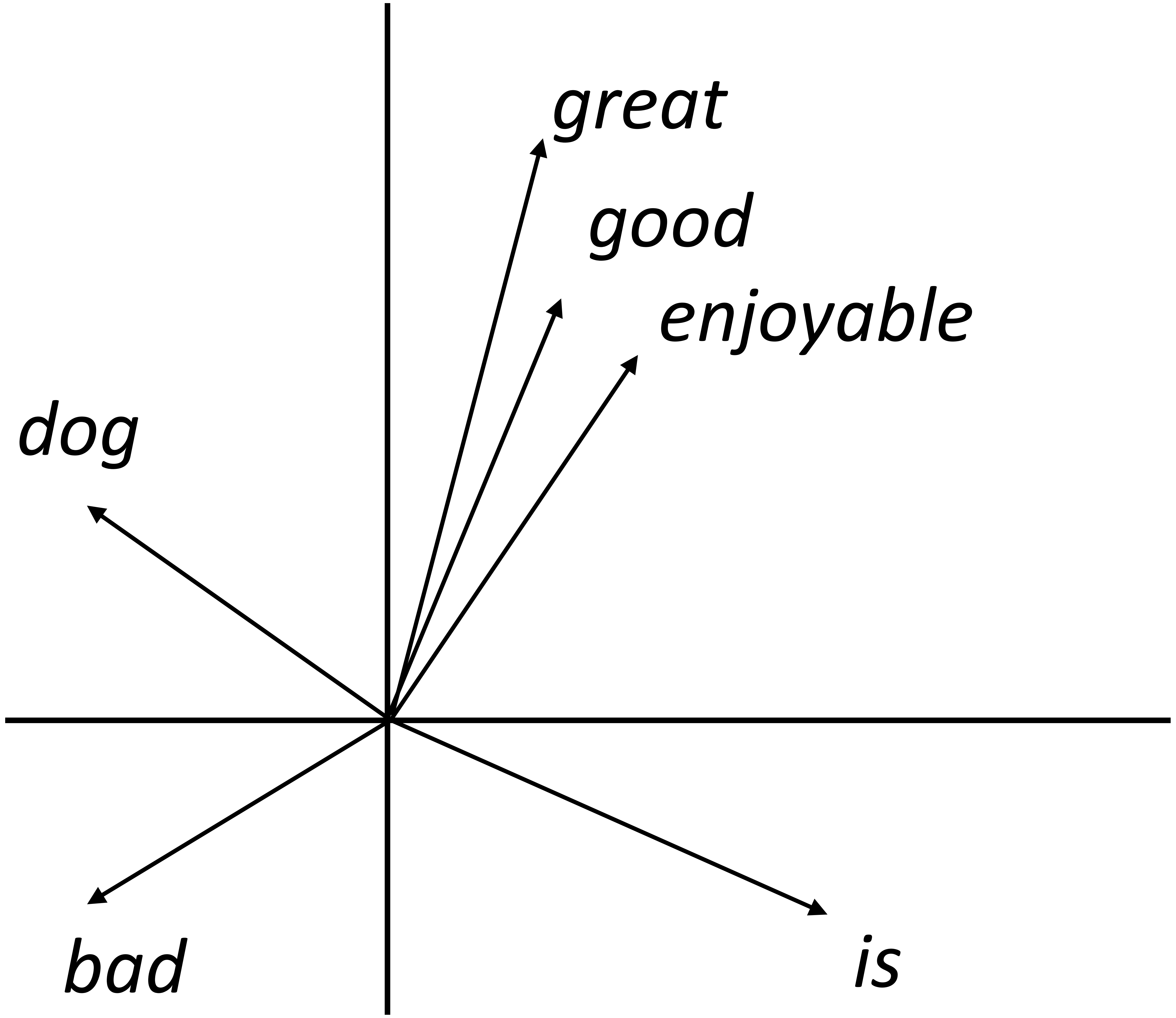
- Mini 1 back today
- Project 1 due tonight
- Mini 2 out tonight

# Recall: Word Vectors




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

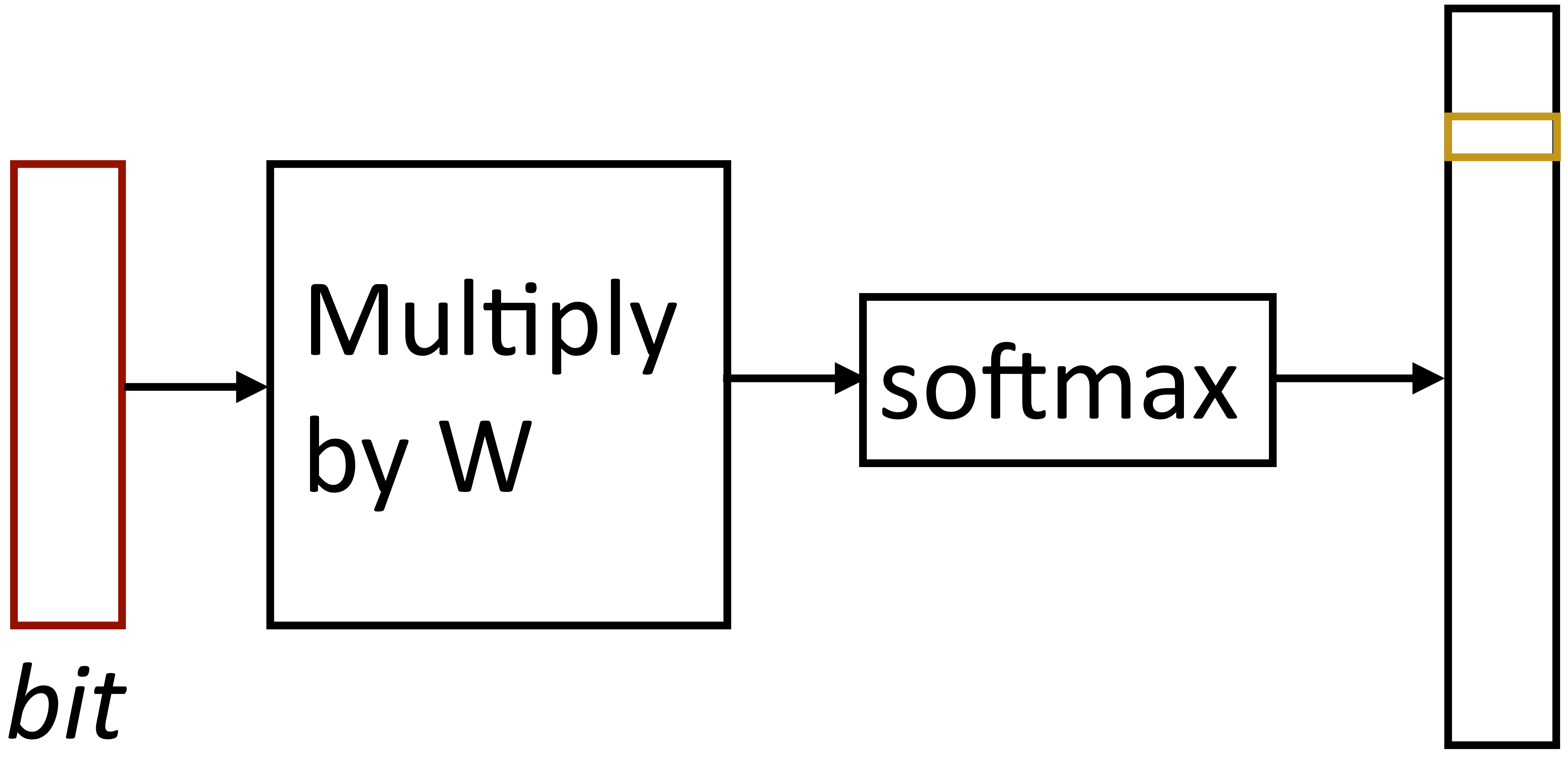
[Finch and Chater 92, Shuetze 93, many others]



# Recall: Skip-Gram

- Predict one word of context from word

*the dog bit the man*




gold = *dog*

$$P(w'|w) = \text{softmax}(We(w))$$

- Another training example: *bit -> the*
- Parameters: *d* x |V| vectors, |V| x *d* output parameters (W) (also usable as vectors!)

Mikolov et al. (2013)

# This Lecture

- Evaluating word embeddings

- Recurrent neural networks: basics, issues

- LSTMs / GRUs

- Applications / visualizations

# Evaluating Word Embeddings

# Evaluating Word Embeddings

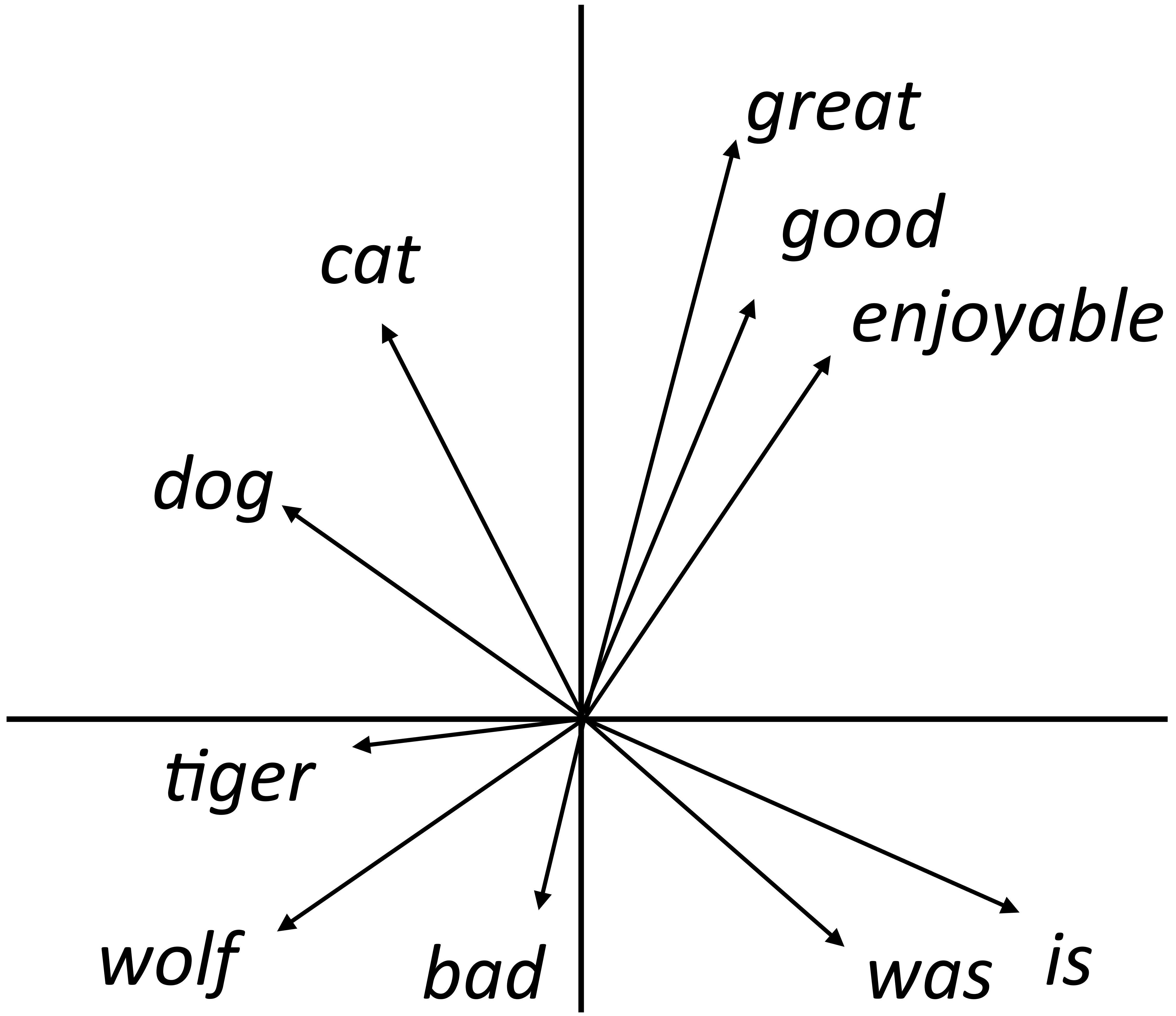
- What properties of language should word embeddings capture?

- Similarity: similar words are close to each other

- Analogy:

  good is to best as smart is to ???

  Paris is to France as Tokyo is to ???



# Similarity

| Method | WordSim Similarity | WordSim Relatedness | Bruni et al. MEN | Radinsky et al. M. Turk | Luong et al. Rare Words | Hill et al. SimLex |
|---|---|---|---|---|---|---|
| PPMI | .755 | **.697** | .745 | .686 | .462 | .393 |
| SVD | **.793** | .691 | **.778** | .666 | **.514** | .432 |
| SGNS | **.793** | .685 | .774 | **.693** | .470 | **.438** |
| GloVe | .725 | .604 | .729 | .632 | .403 | .398 |

- SVD = singular value decomposition on PMI matrix
- GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don't matter in practice

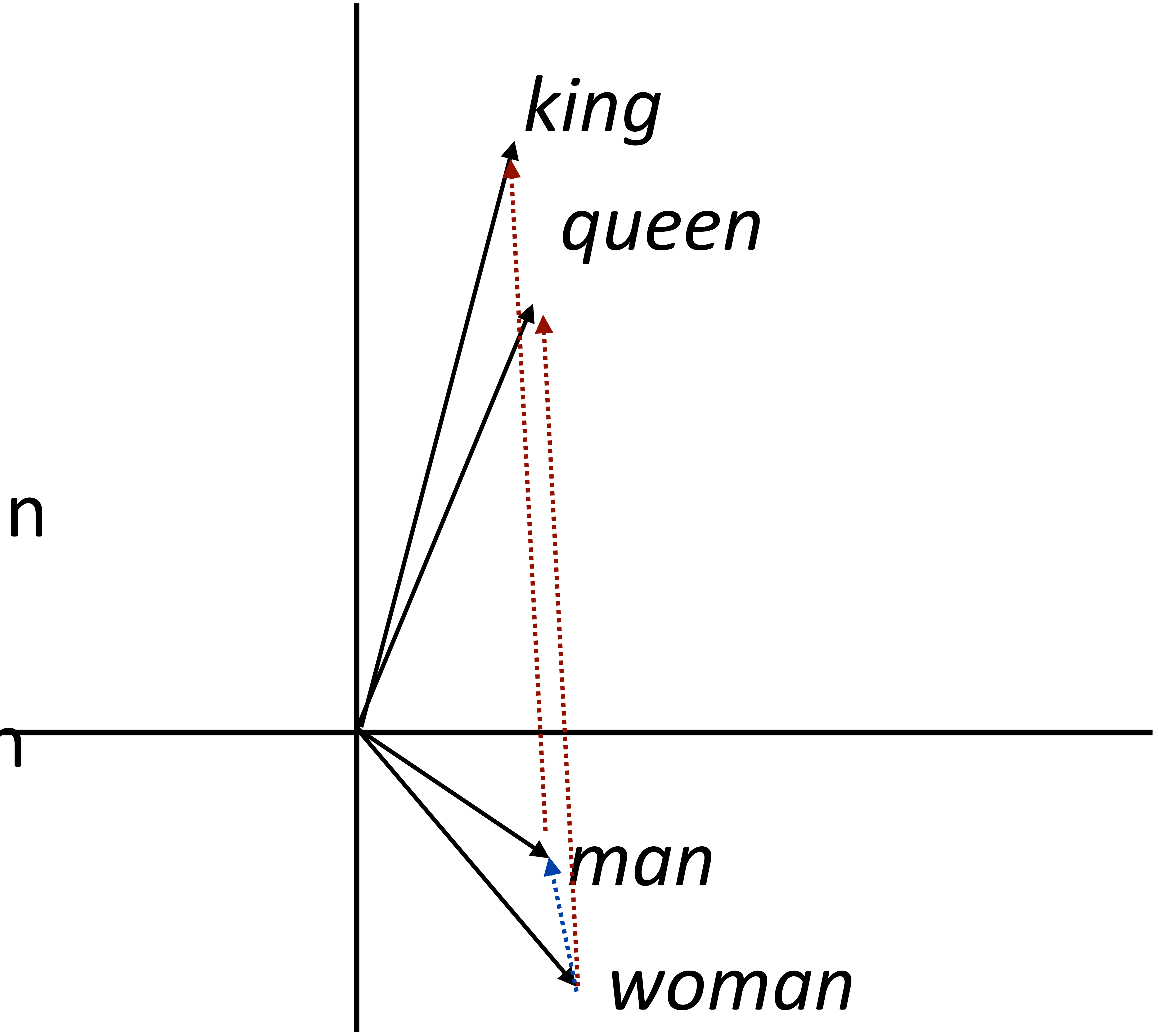
Levy et al. (2015)

# Analogies

*(king - man) + woman = queen*

*king + (woman - man) = queen*

- Why would this be?
- woman - man captures the difference in the contexts that these occur in
- Dominant change: more "he" with man and "she" with woman — similar to difference between king and queen
- Can evaluate on this as well



# What can go wrong with word embeddings?

- What’s wrong with learning a word’s “meaning” from its usage?
- What data are we learning from?
- What are we going to learn from this data?

# What do we mean by bias?

- Identify *she* - *he* axis in word vector space, project words onto this axis

**Extreme *she* occupations**

| | | |
|---|---|---|
| 1. homemaker | 2. nurse | 3. receptionist |
| 4. librarian | 5. socialite | 6. hairdresser |
| 7. nanny | 8. bookkeeper | 9. stylist |
| 10. housekeeper | 11. interior designer | 12. guidance counselor |

**Extreme *he* occupations**

| | | |
|---|---|---|
| 1. maestro | 2. skipper | 3. protege |
| 4. philosopher | 5. captain | 6. architect |
| 7. financier | 8. warrior | 9. broadcaster |
| 10. magician | 11. figher pilot | 12. boss |

Bolukbasi et al. (2016)

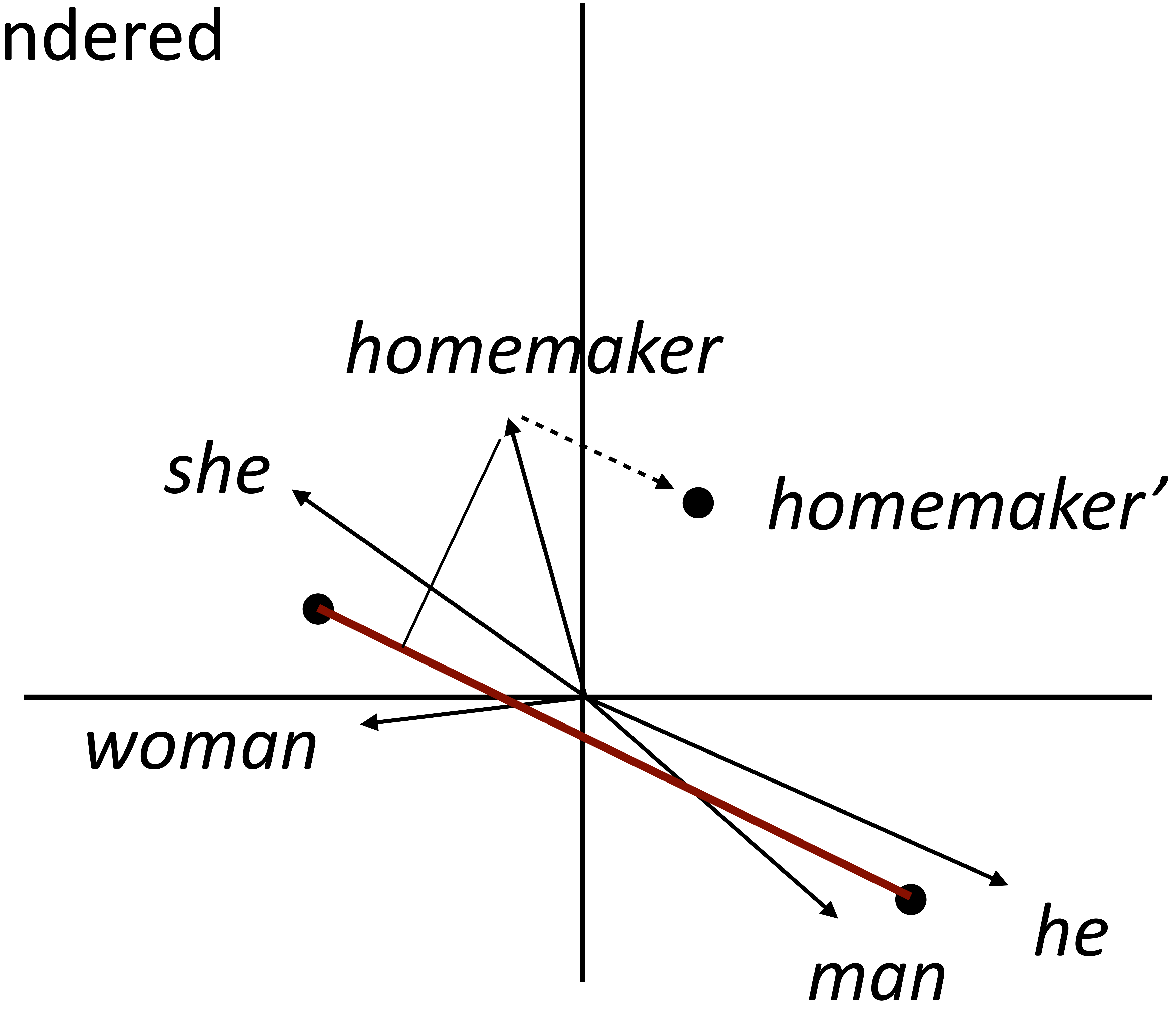
- Nearest neighbor of (b - a + c)

| **Racial Analogies** | |
|---|---|
| black → homeless | caucasian → servicemen |
| caucasian → hillbilly | asian → suburban |
| asian → laborer | black → landowner |
| **Religious Analogies** | |
| jew → greedy | muslim → powerless |
| christian → familial | muslim → warzone |
| muslim → uneducated | christian → intellectually |

Manzini et al. (2019)

# Debiasing

- Identify gender subspace with gendered words
- Project words onto this subspace
- Subtract those projections from the original word

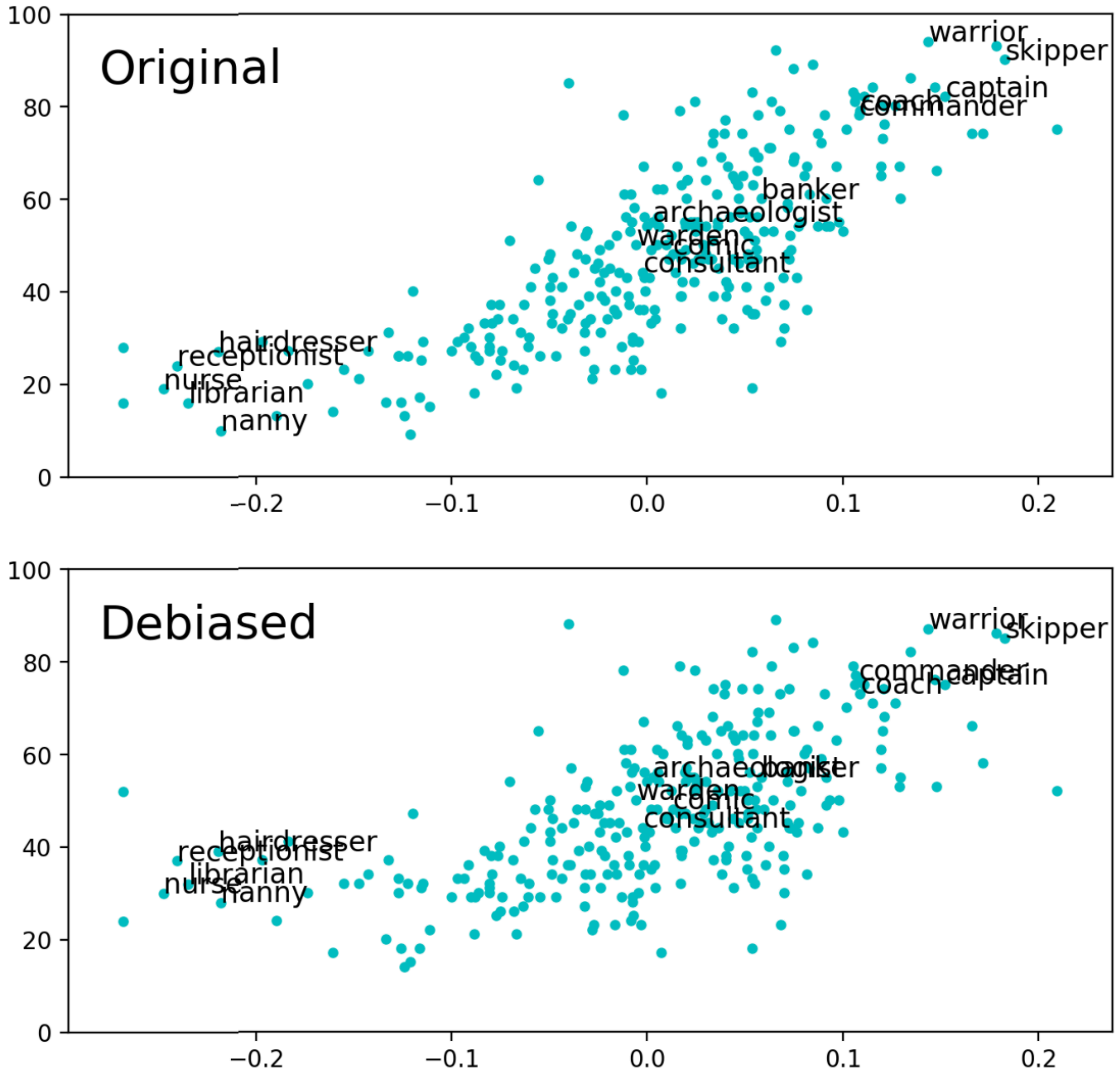



Bolukbasi et al. (2016)

# Hardness of Debiasing

- Not that effective…and the male and female words are still clustered together

- Bias pervades the word embedding space and isn't just a local property of a few words



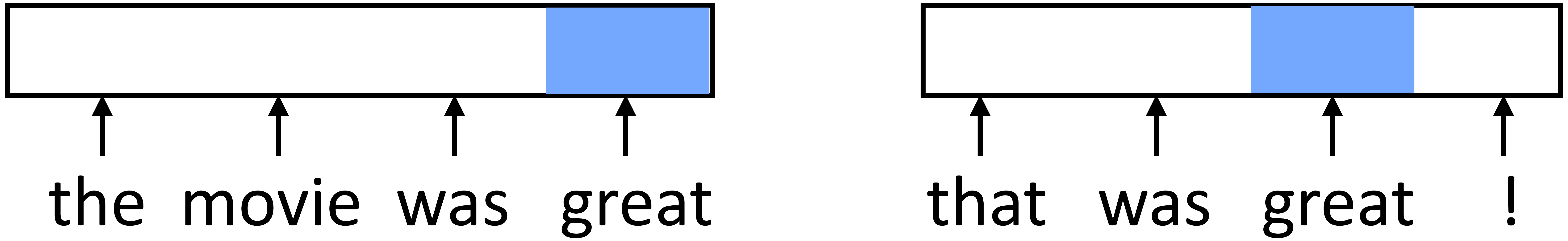

(a) The plots for HARD-DEBIASED embedding, before (top) and after (bottom) debiasing.

Gonen and Goldberg (2019)

# RNN Basics

# RNN Motivation

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



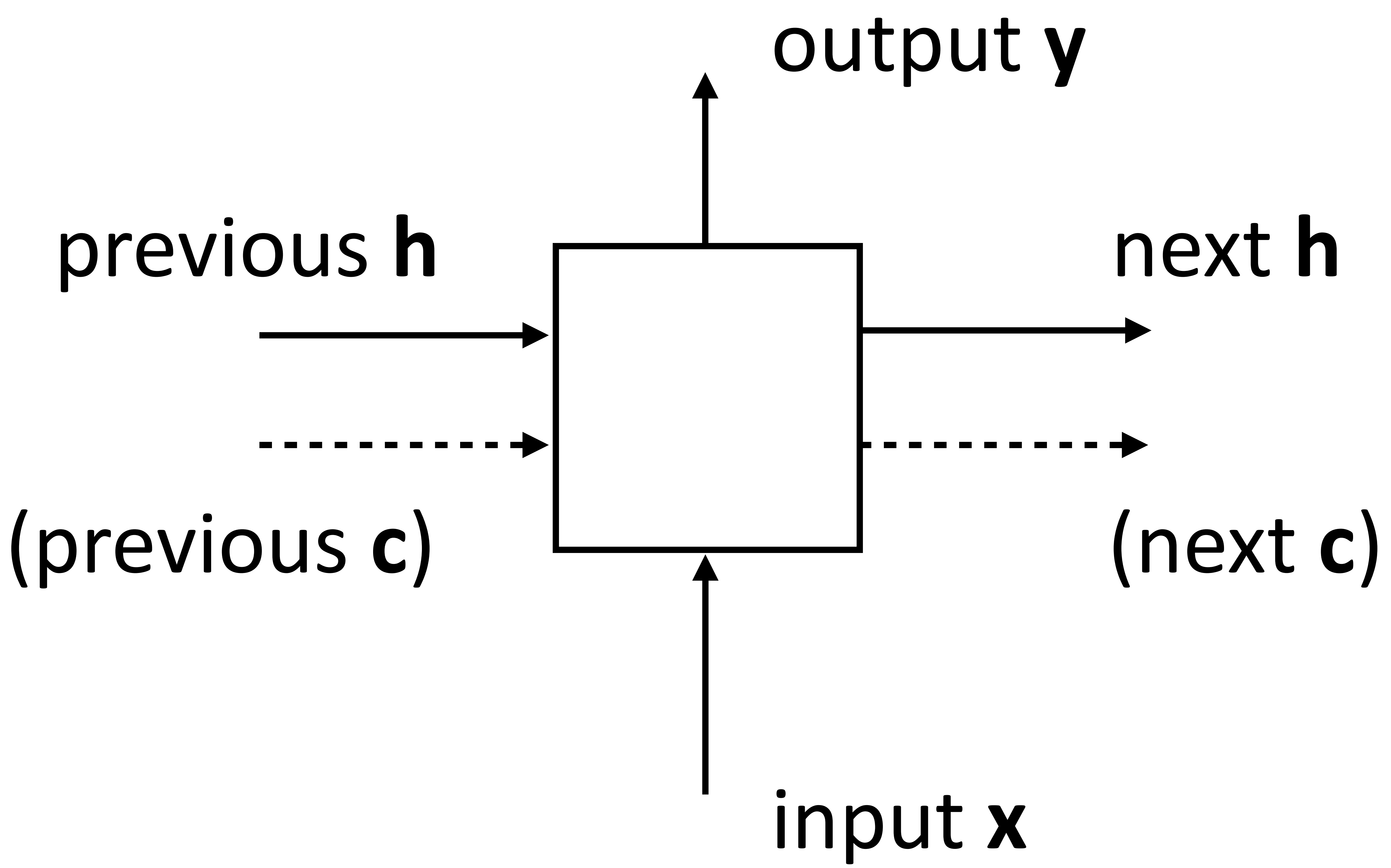

- These don’t look related (*great* is in two different orthogonal subspaces)
- Instead, we need to:

1) Process each word in a uniform way

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

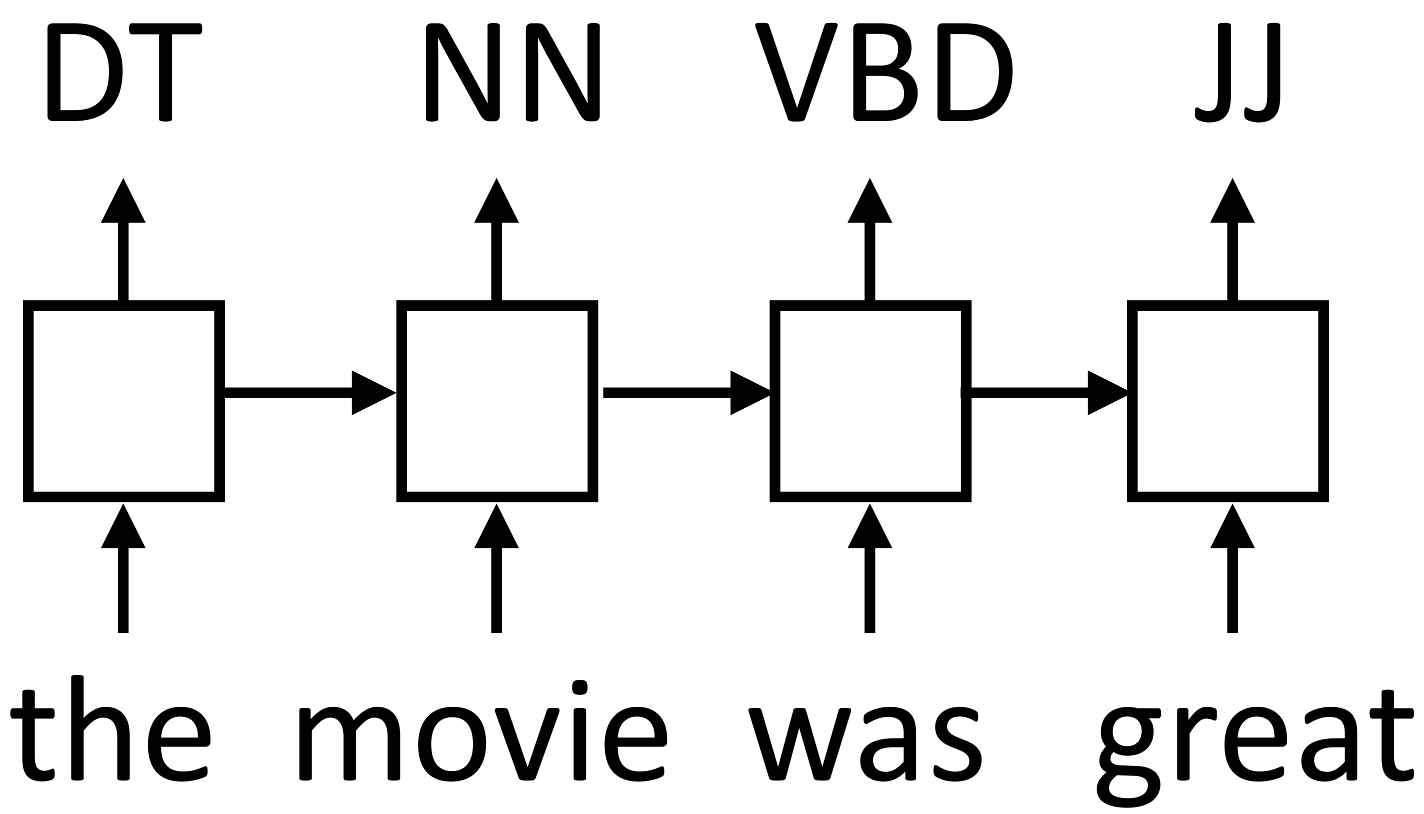
# RNN Abstraction

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



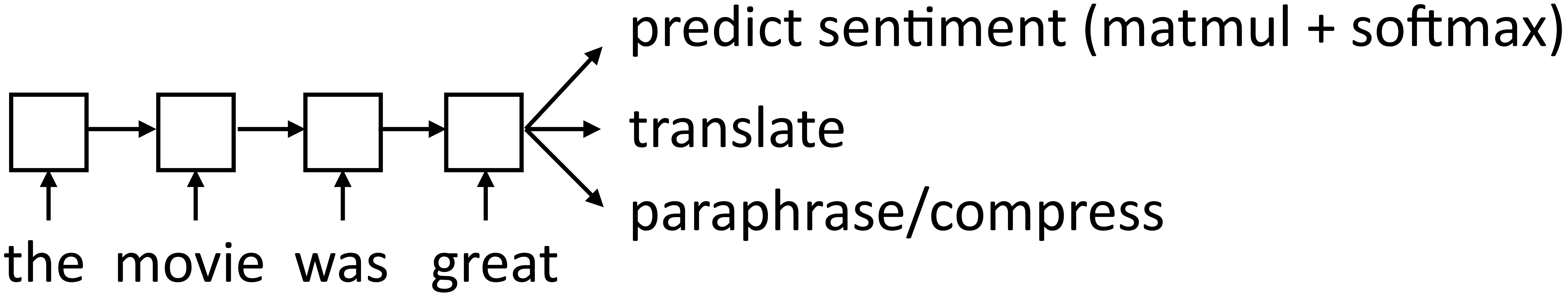
# RNN Uses

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

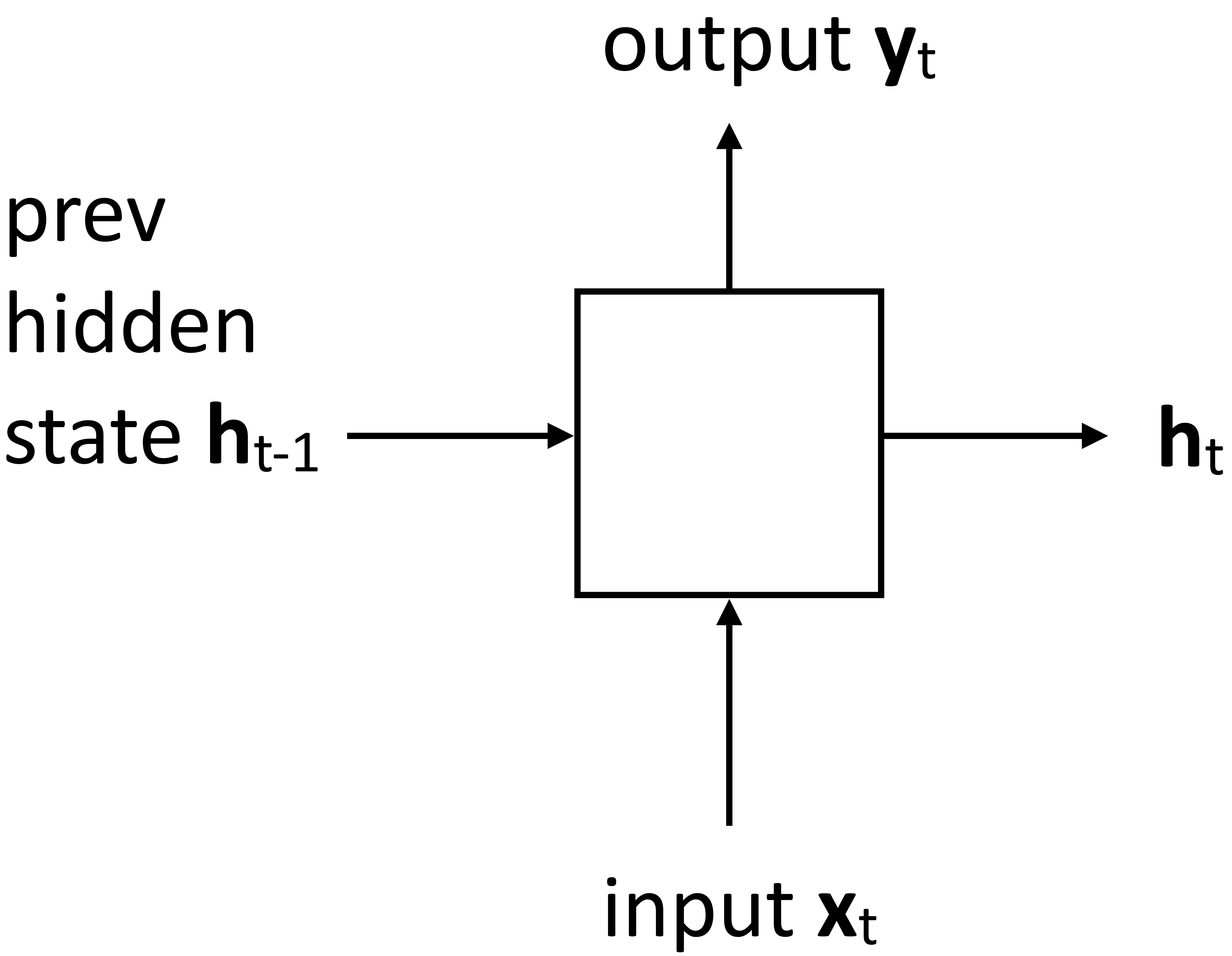



output **y** = score for each tag, then softmax

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



# Elman Networks




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

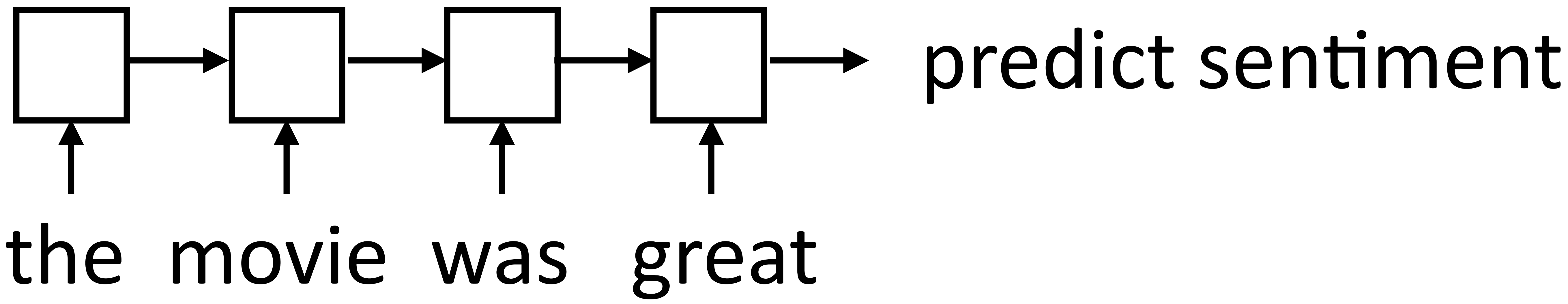
- Updates hidden state based on input and current hidden state

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

- Computes output from hidden state

Elman (1990)

# Training Elman Networks



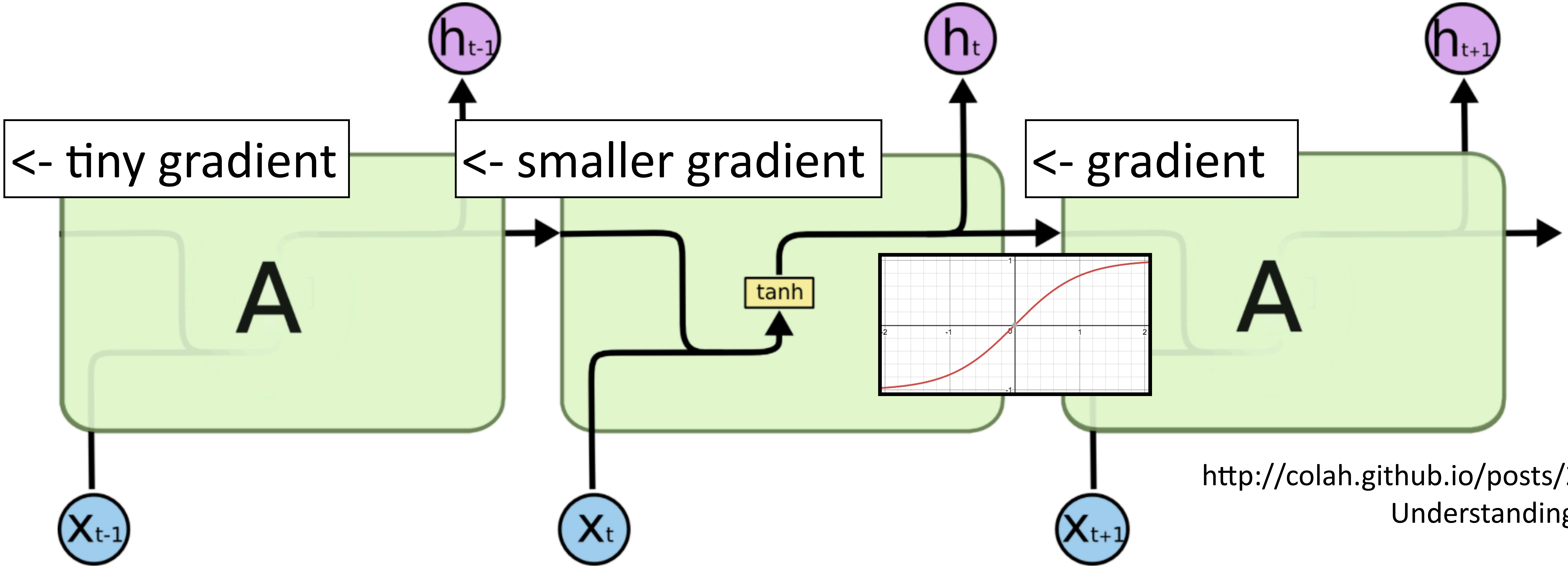

- "Backpropagation through time": build the network as one big computation graph, some parameters are shared

- 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*

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

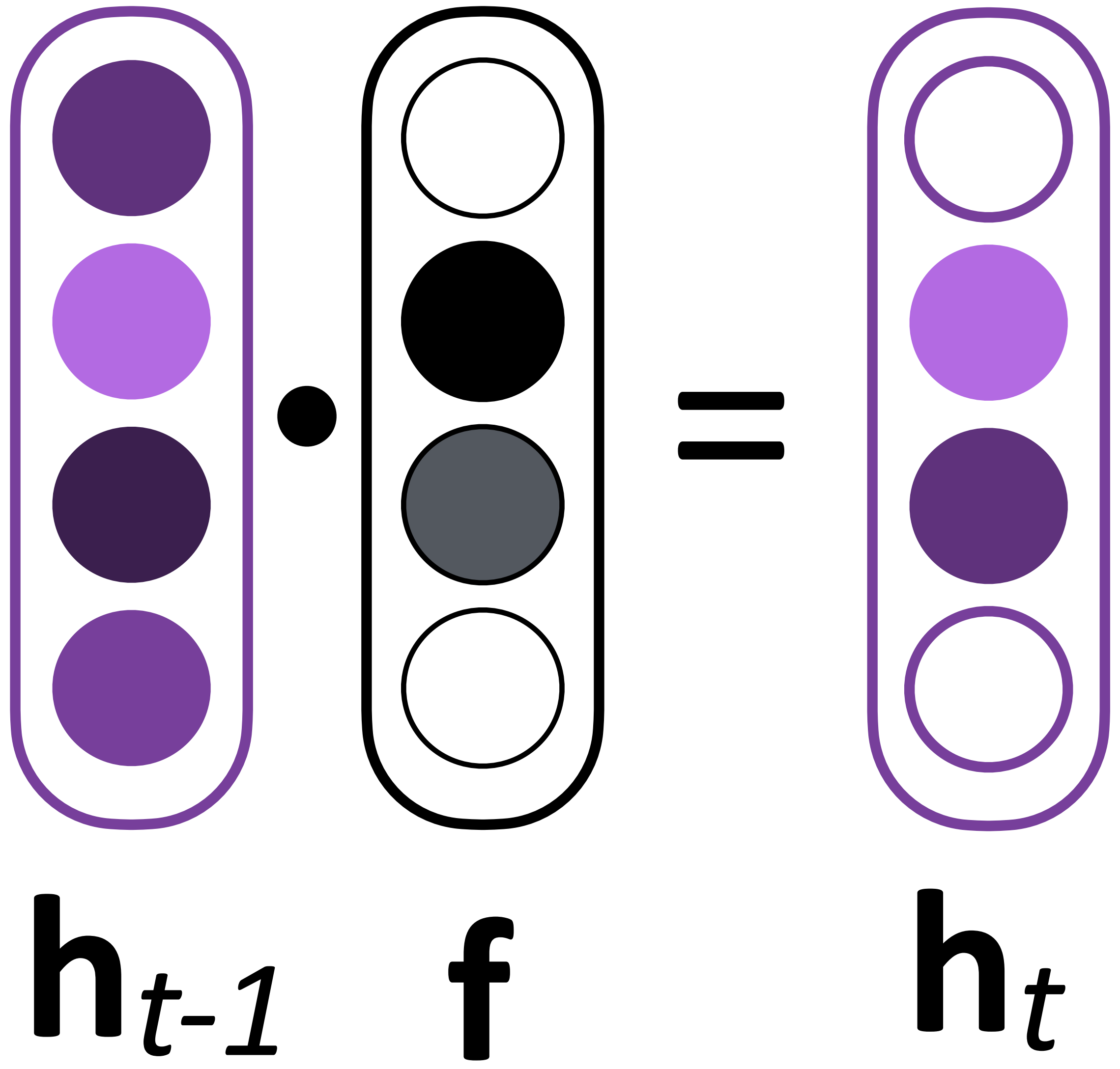
gated

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

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

- "Long short-term memory" network: hidden state is a "short-term" memory

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

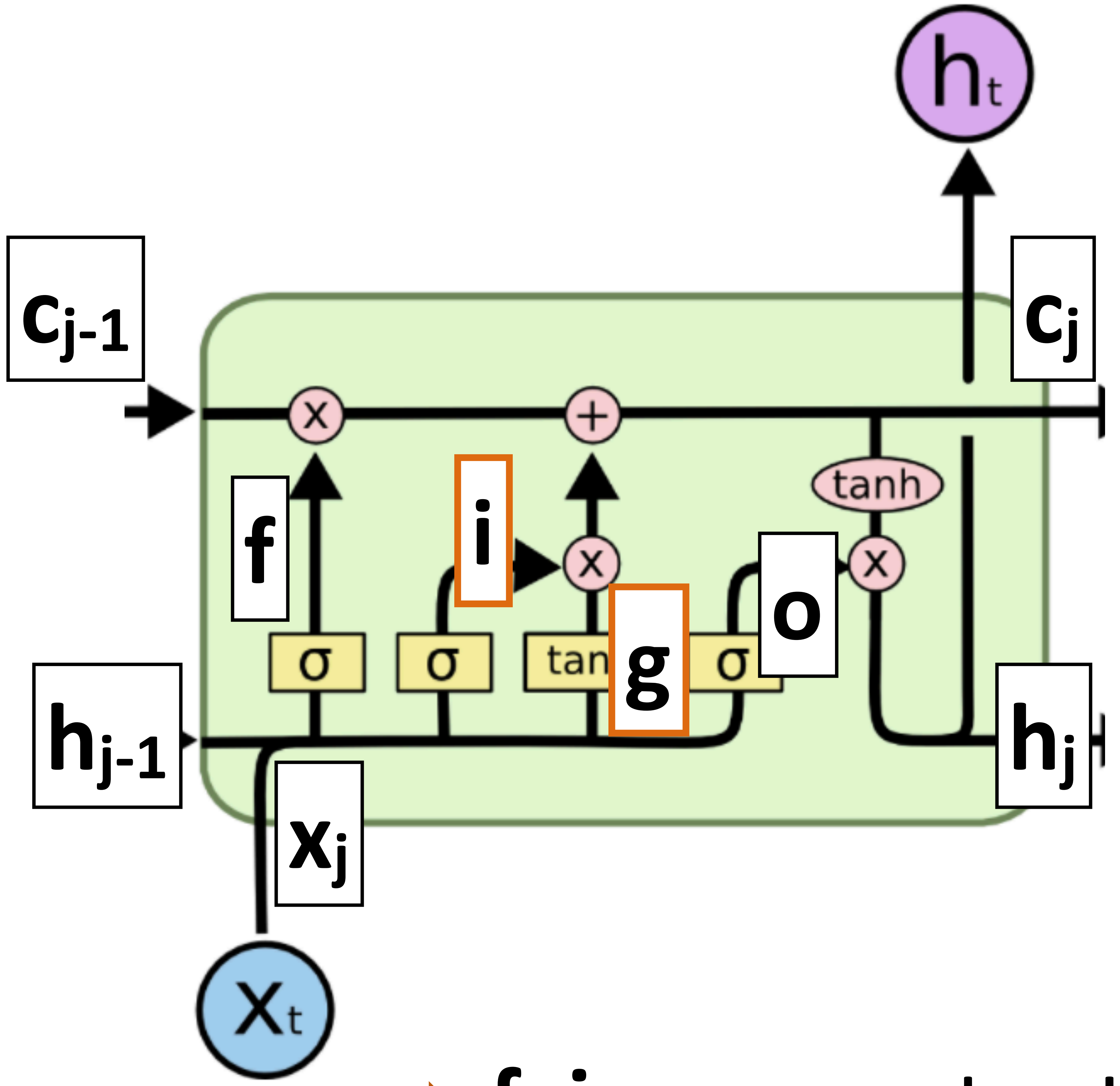
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- 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




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

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

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

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

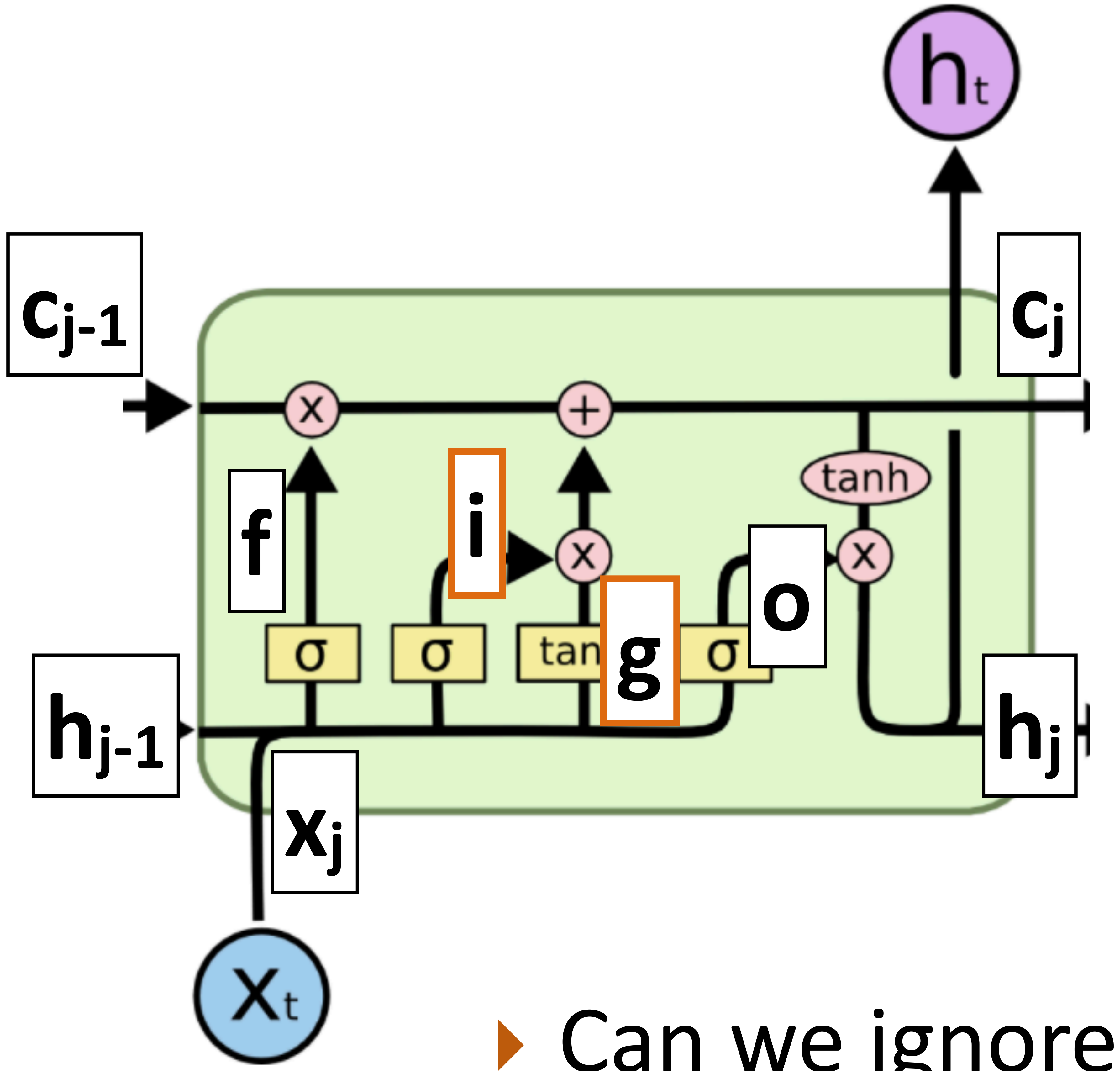
$$\mathbf{o} = \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})$$

- **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




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

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

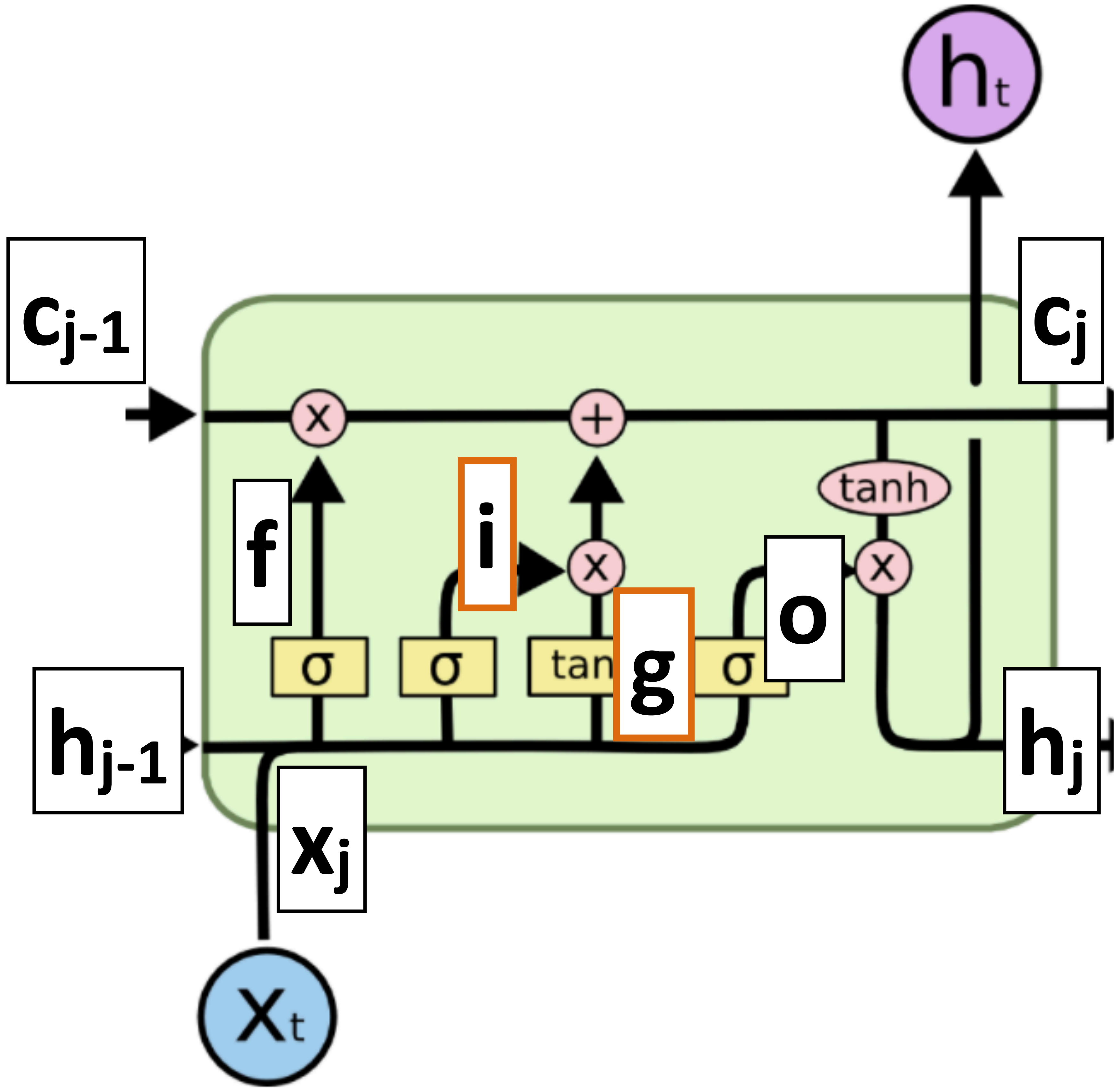
$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j} \mathbf{W^{xg}} + \mathbf{h_{j-1}} \mathbf{W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j} \mathbf{W^{xi}} + \mathbf{h_{j-1}} \mathbf{W^{hi}}) \end{aligned}}$$

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

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

- 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

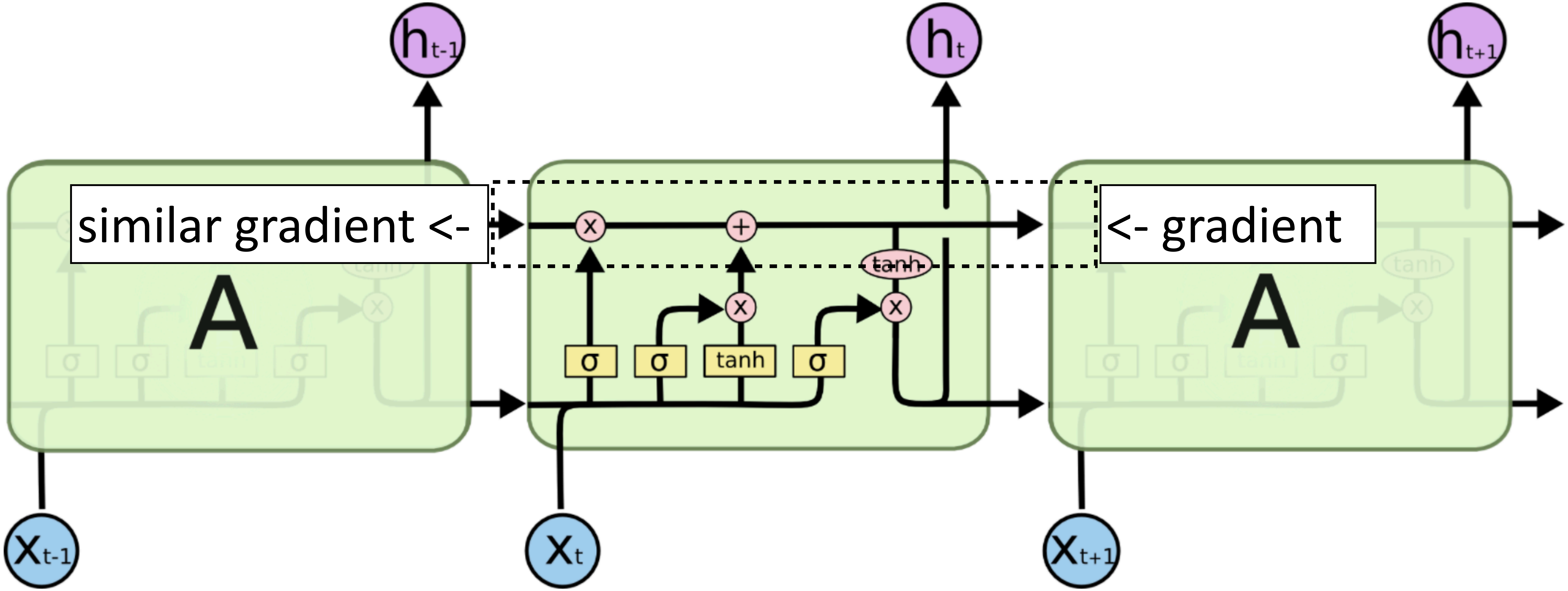



- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Goldberg lecture notes

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

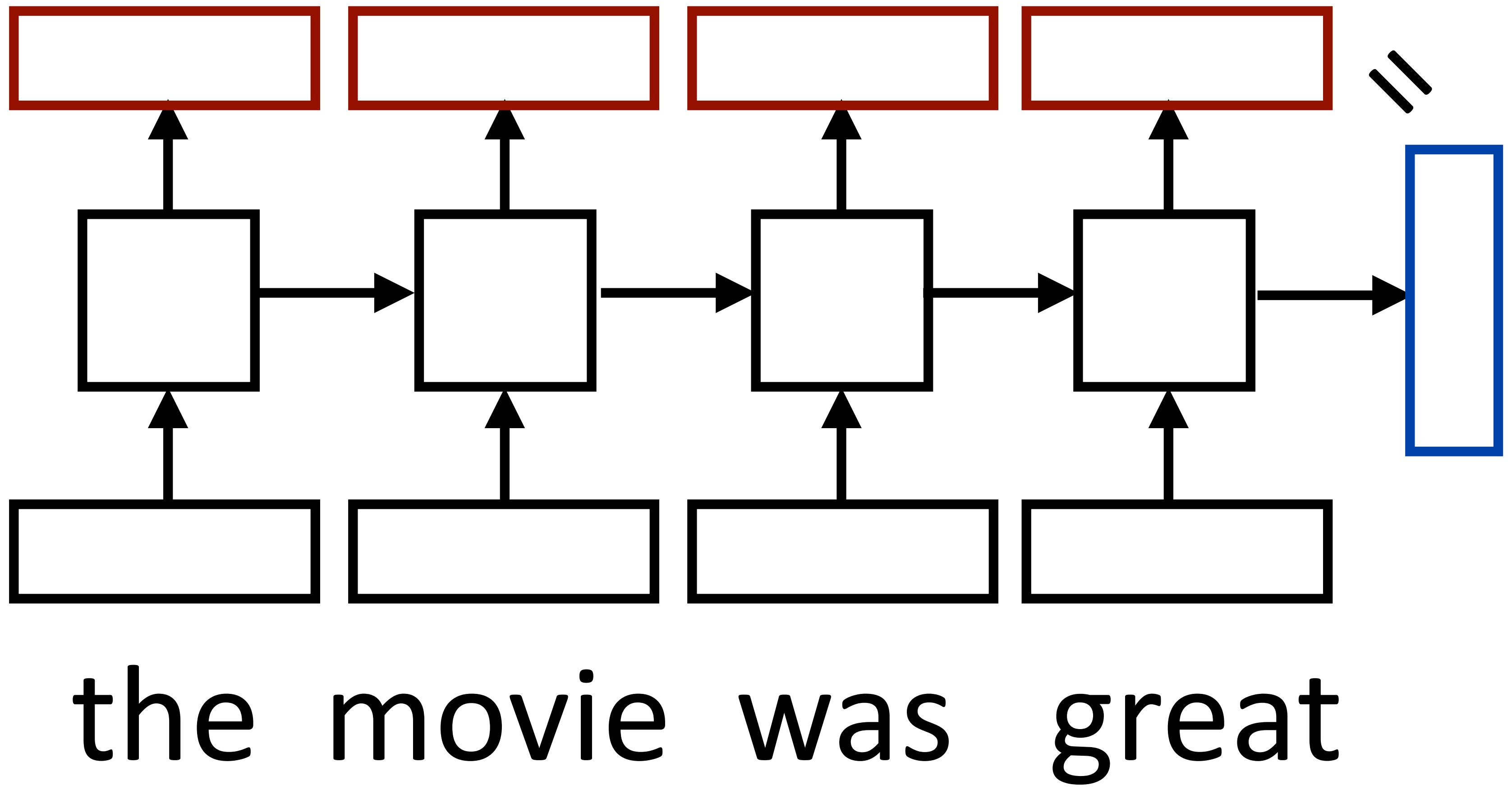
# LSTMs




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

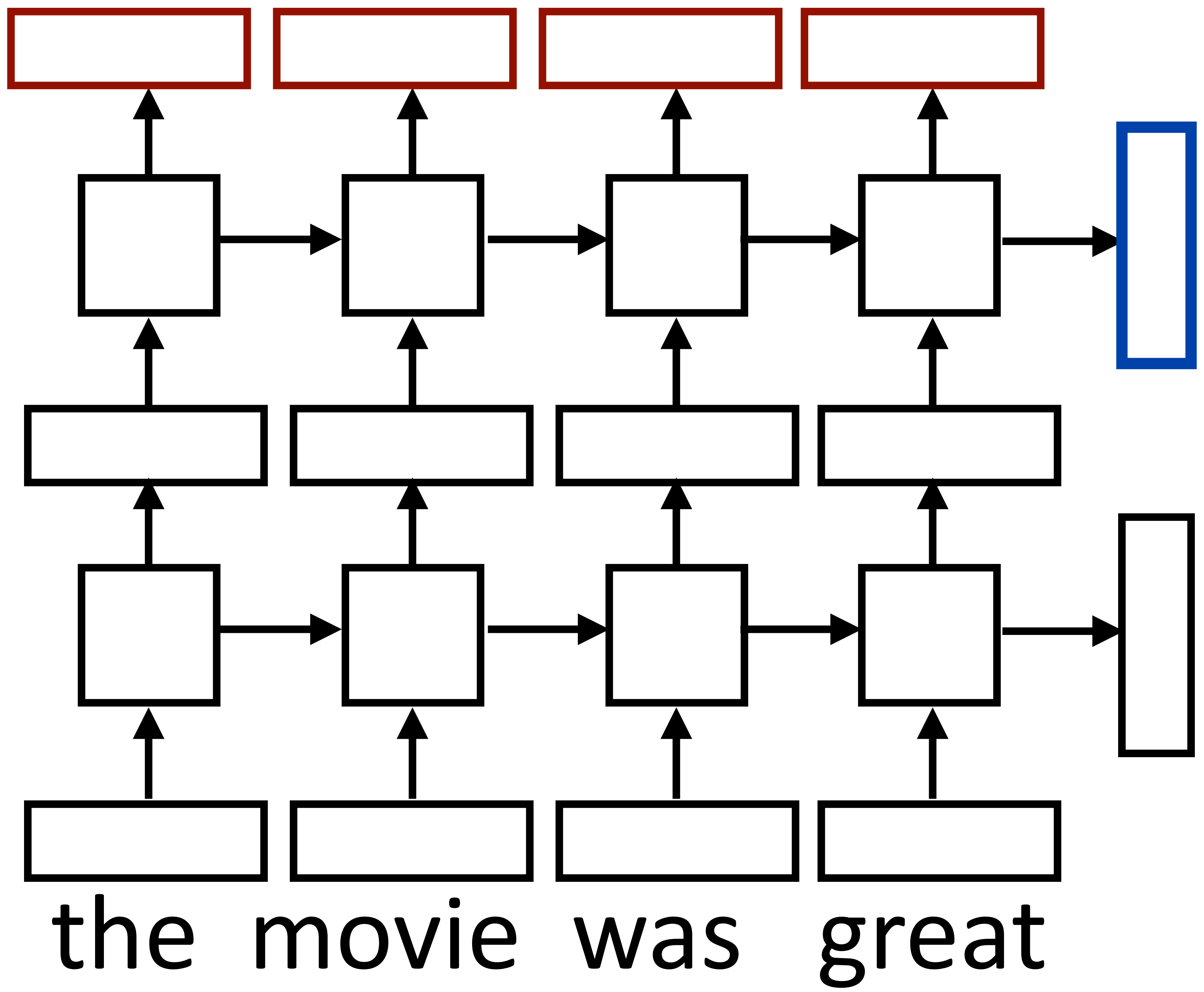
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

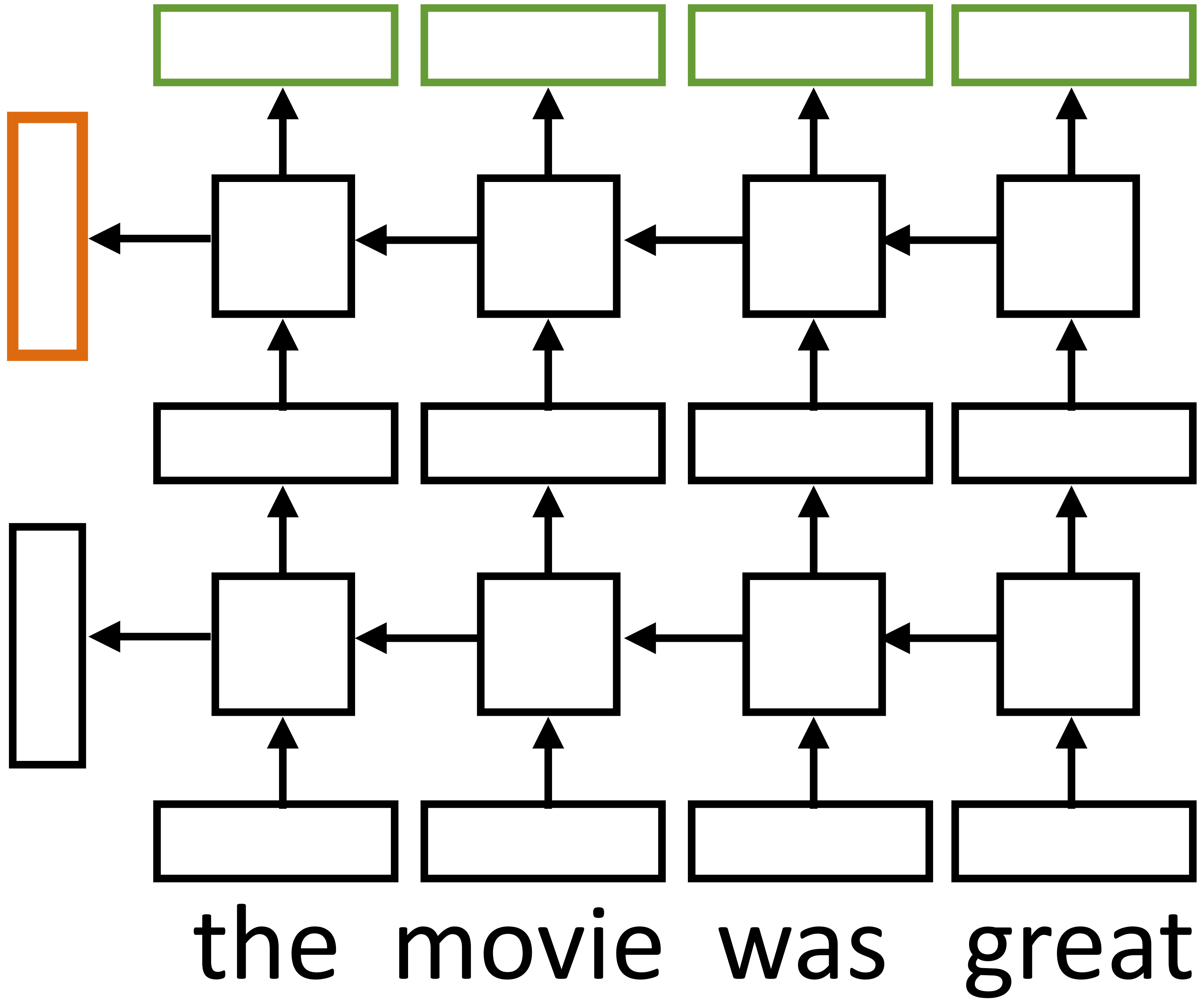
# What do RNNs produce?



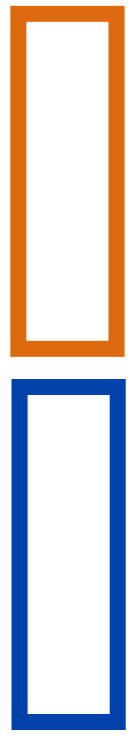

- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors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

# Multilayer Bidirectional RNN

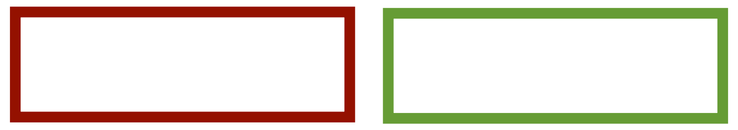


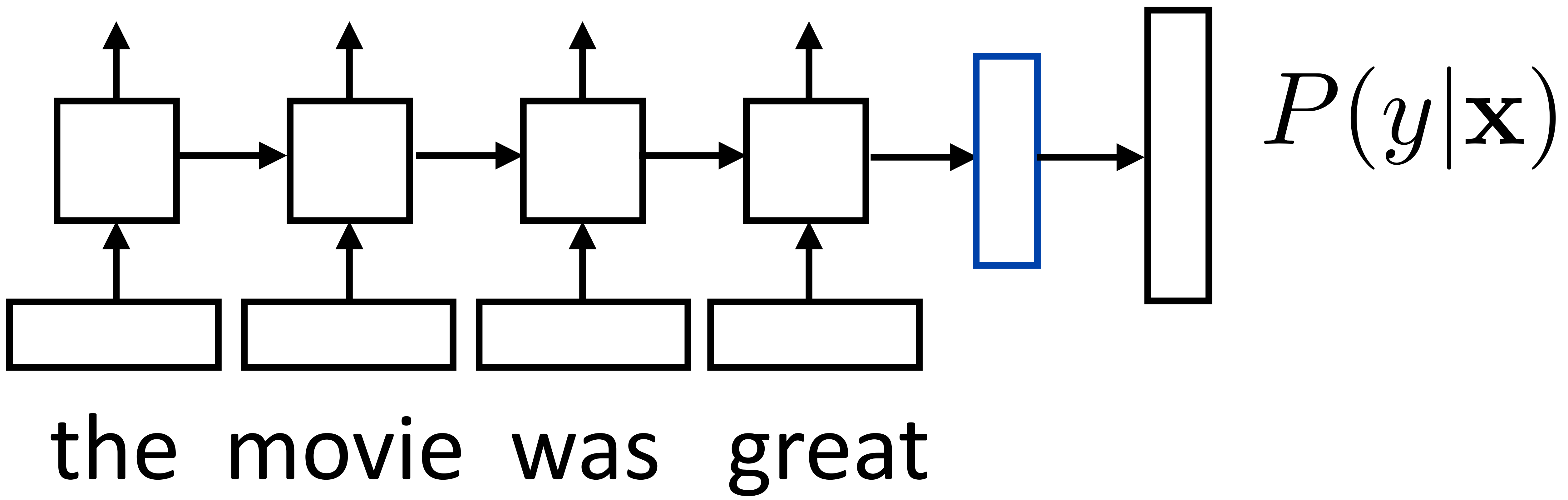

- Sentence classification based on concatenation of both final outputs



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

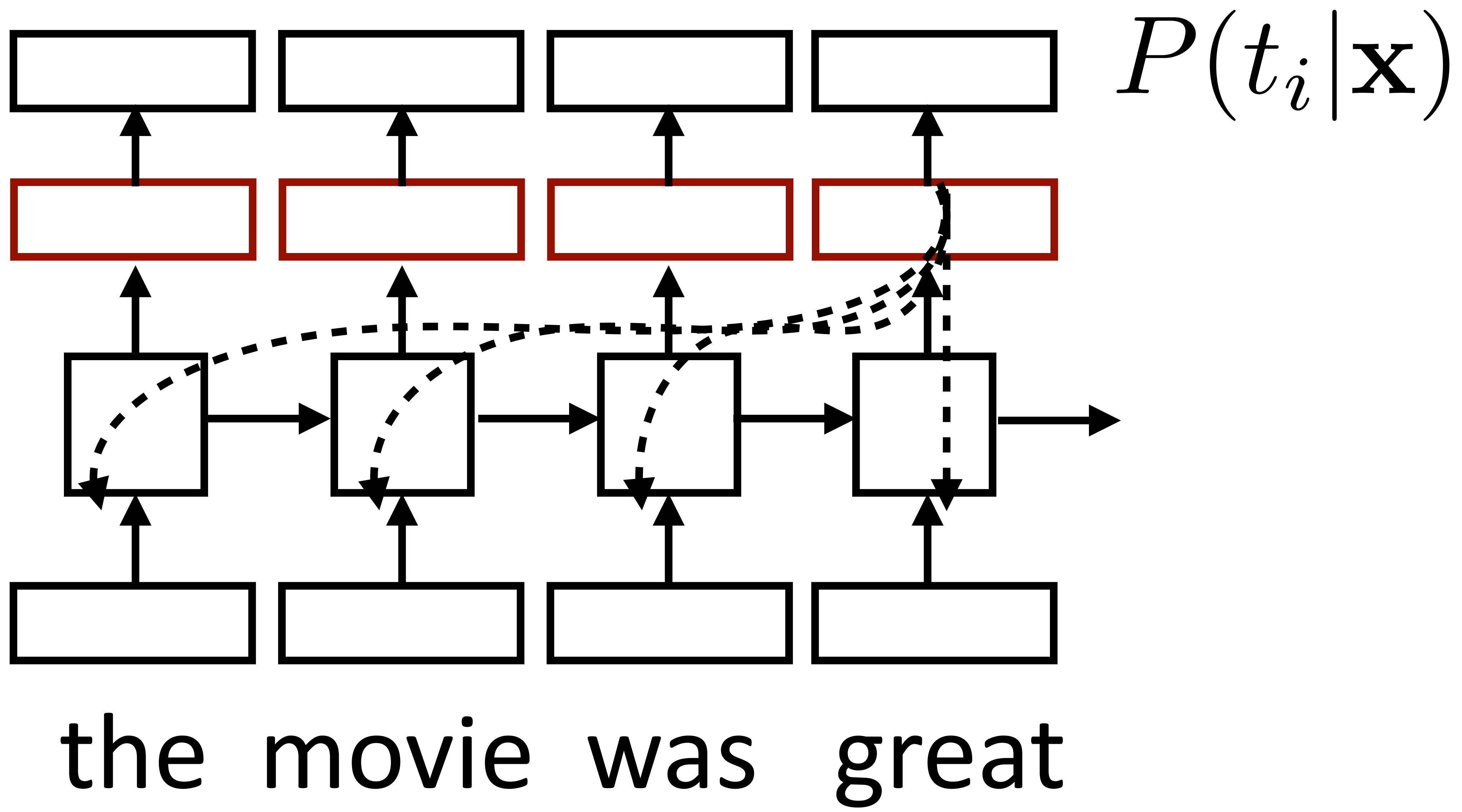


# Training RNNs




- 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) or POS tagging

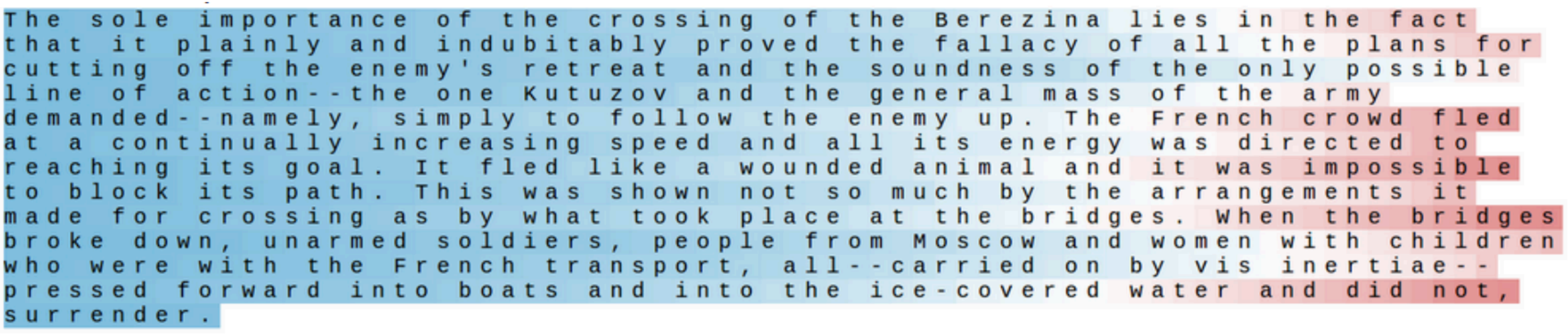
# Applications

# 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

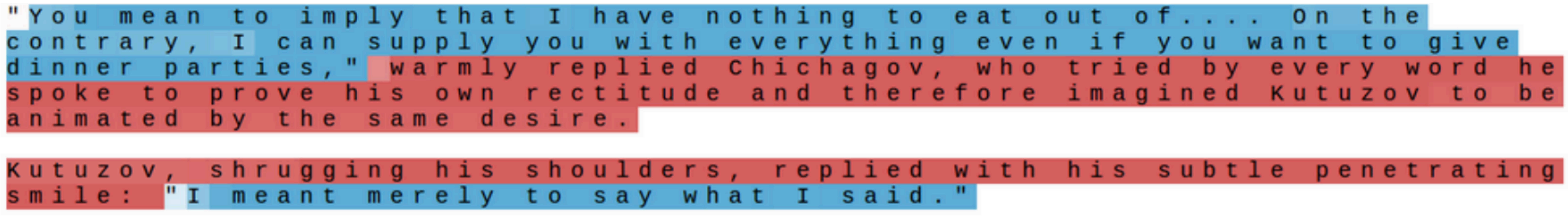
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n




Karpathy et al. (2015)

# Visualizing LSTMs

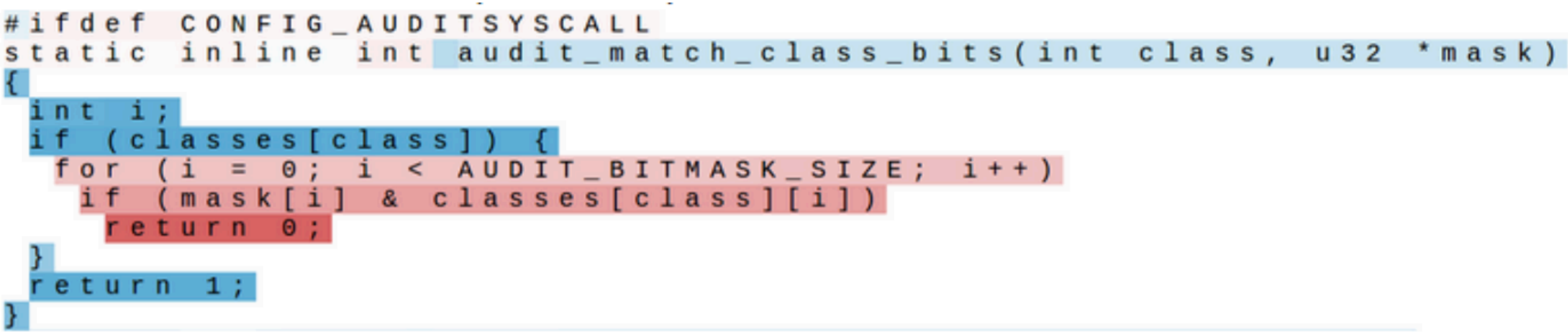
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: tells us if we're in a quote or not




Karpathy et al. (2015)

# Visualizing LSTMs

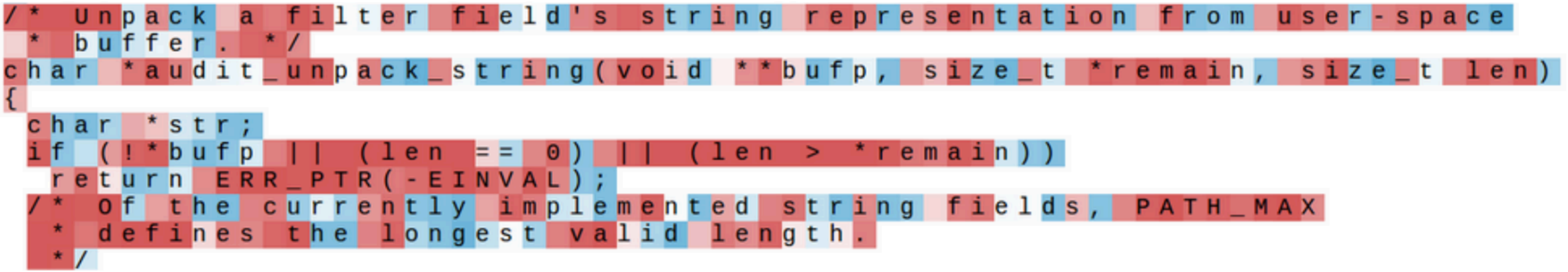
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation




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 (later in the course)
- 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 (a type of pre-training 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% |

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

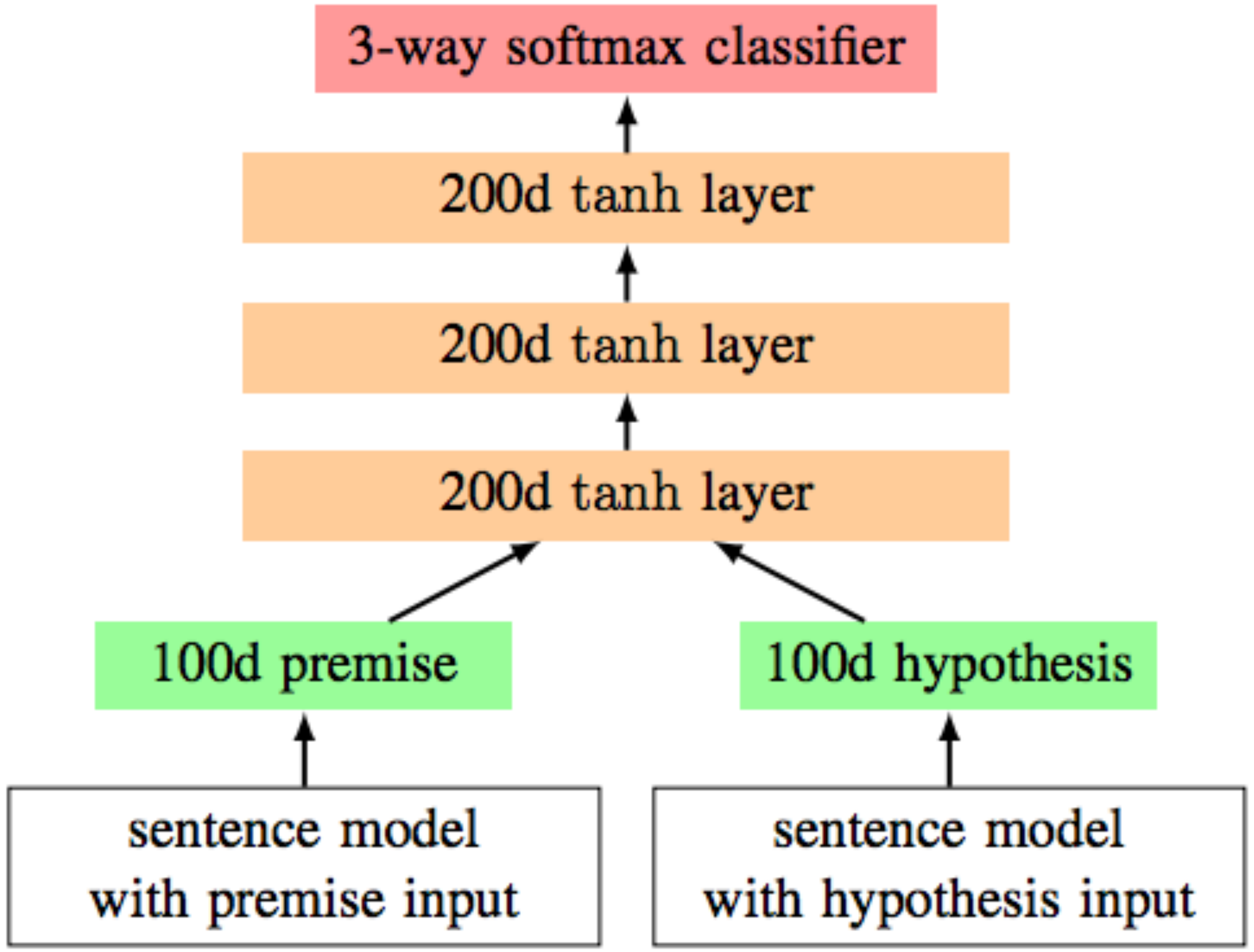
300D LSTM: 80% accuracy

(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