Lecture 8: RNNs

Alan Ritter

(many slides from Greg Durrett)

Recall: Training Tips

Recall: Training Tips

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

Recall: Training Tips

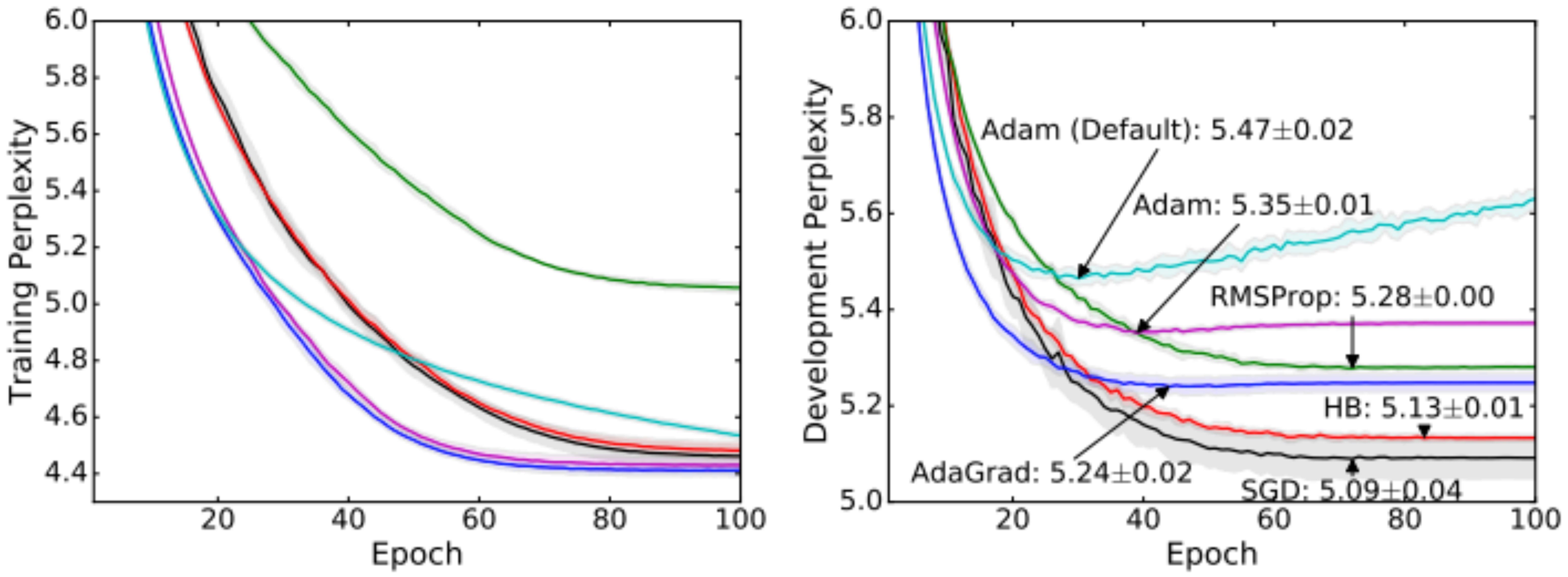
- Parameter initialization is critical to get good gradients, some useful heuristics (e.g., Xavier initializer)
- Dropout is an effective regularizer

Recall: Training Tips

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

- Dropout is an effective regularizer

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

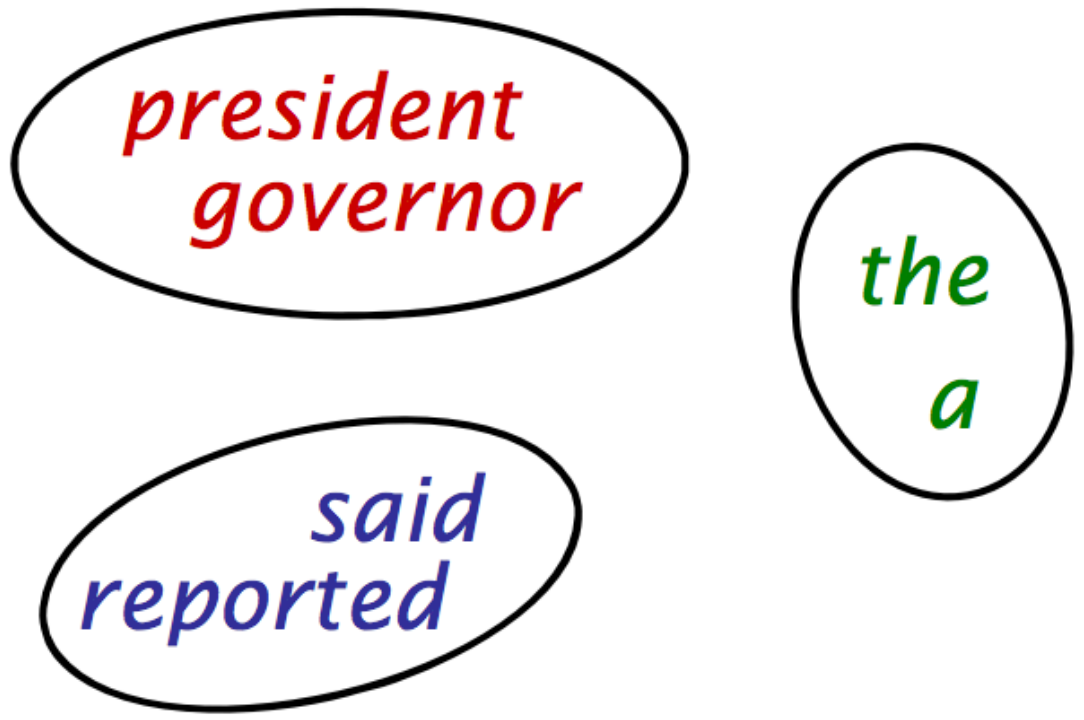


(e) Generative Parsing (Training Set) (f) Generative Parsing (Development Set)

Recall: Word Vectors

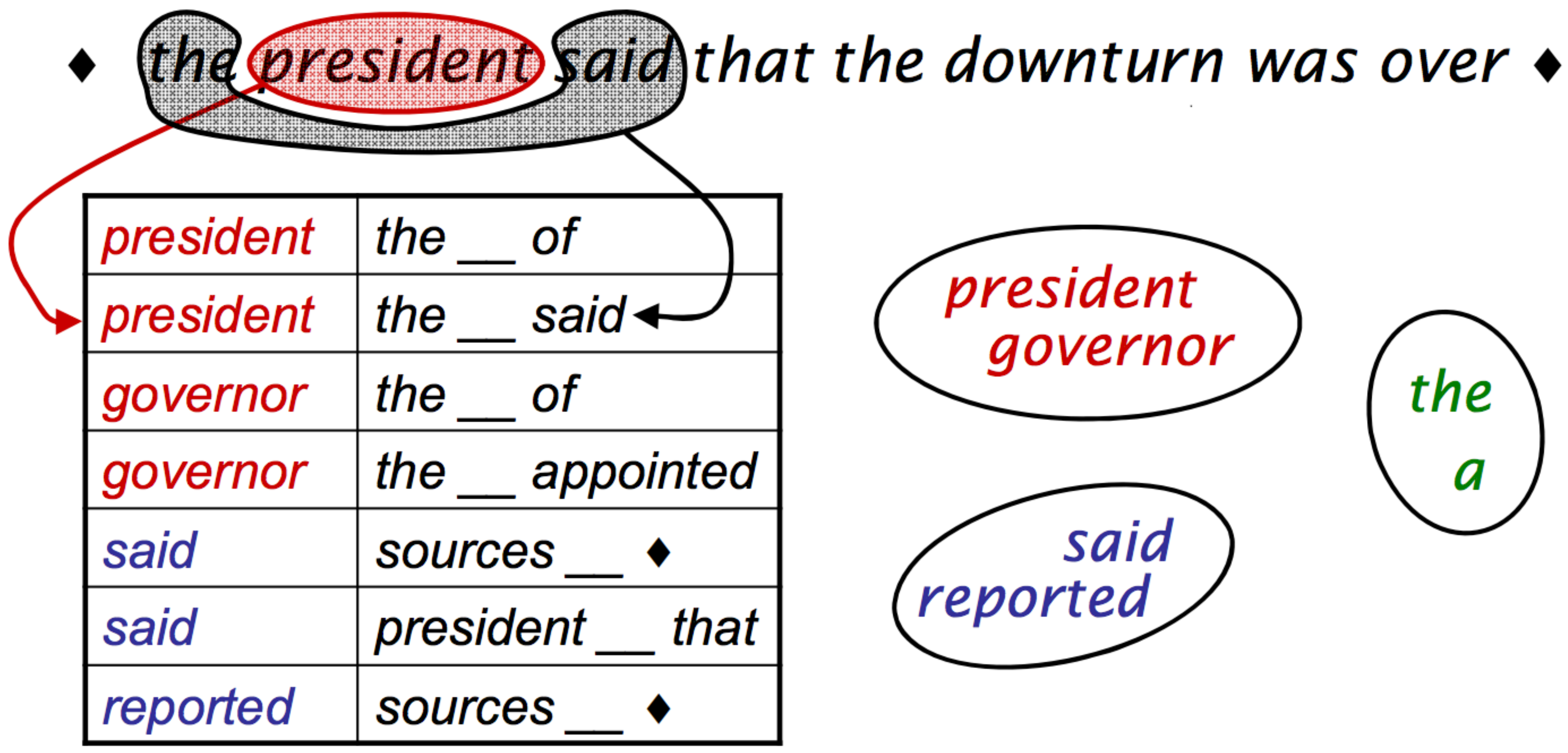
♦ *the president said that the downturn was over* ♦

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



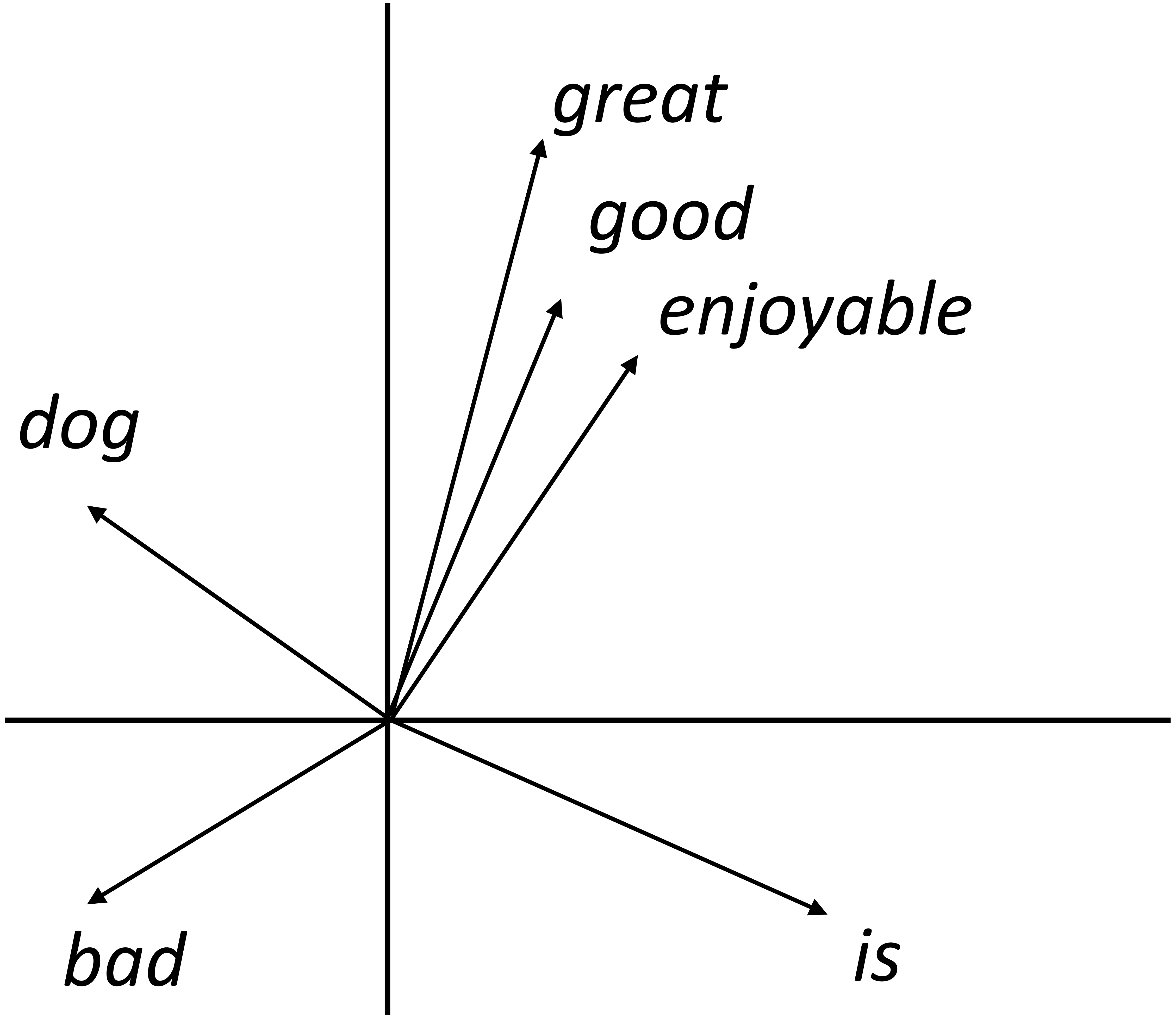
[Finch and Chater 92, Shuetze 93, many others]

Recall: Word Vectors



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: Continuous Bag-of-Words

- Predict word from context

the $\boxed{\textit{dog}\ \textit{bit}\ \textit{the}}$ *man*

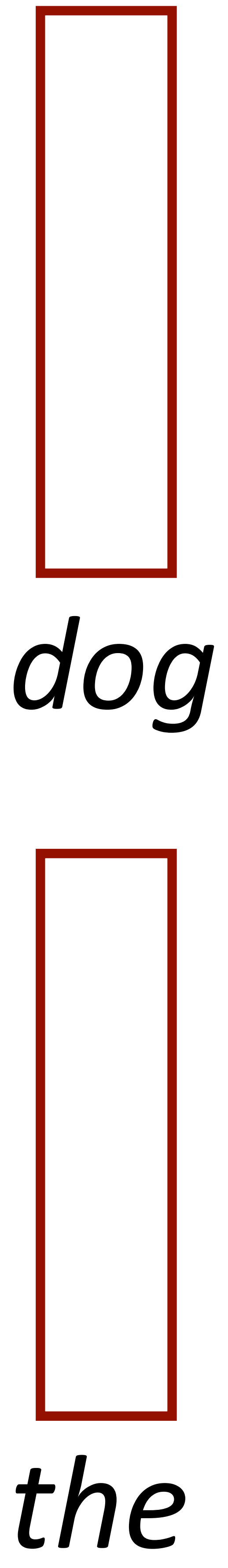
Mikolov et al. (2013)

Recall: Continuous Bag-of-Words

- Predict word from context

the dog bit the man

Mikolov et al. (2013)

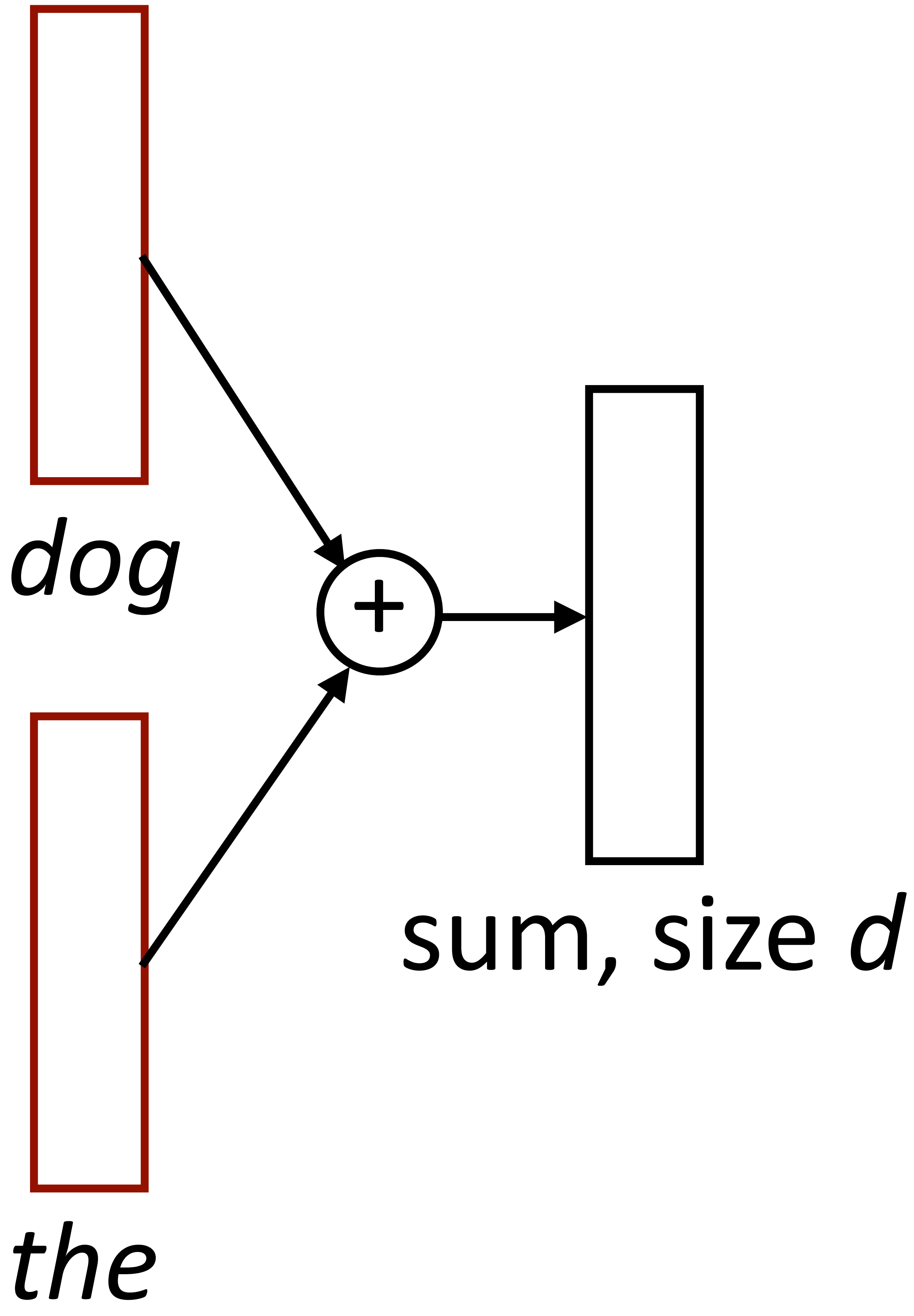


Recall: Continuous Bag-of-Words

- Predict word from context

Mikolov et al. (2013)

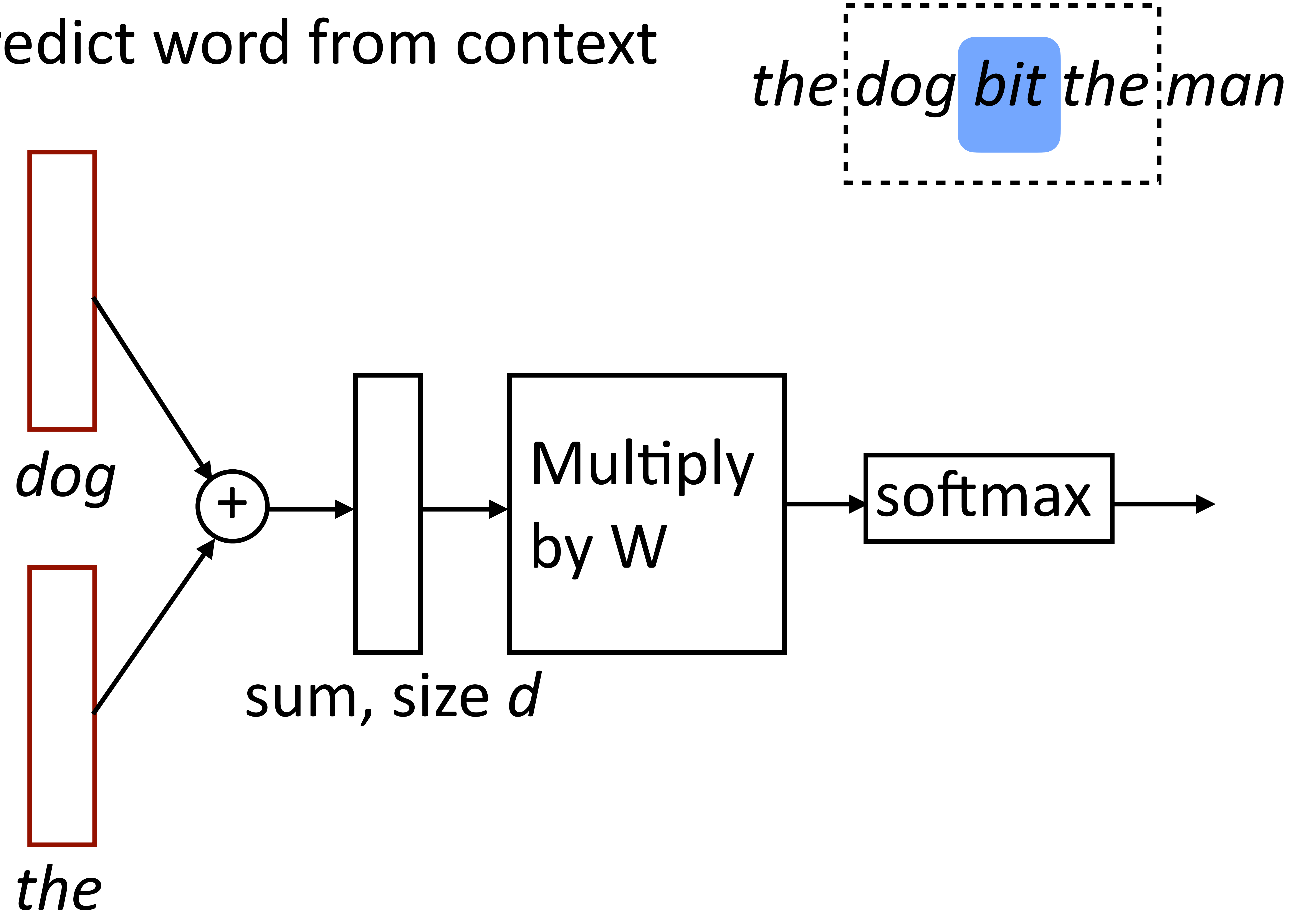
the dog bit the man



Recall: Continuous Bag-of-Words

- Predict word from context

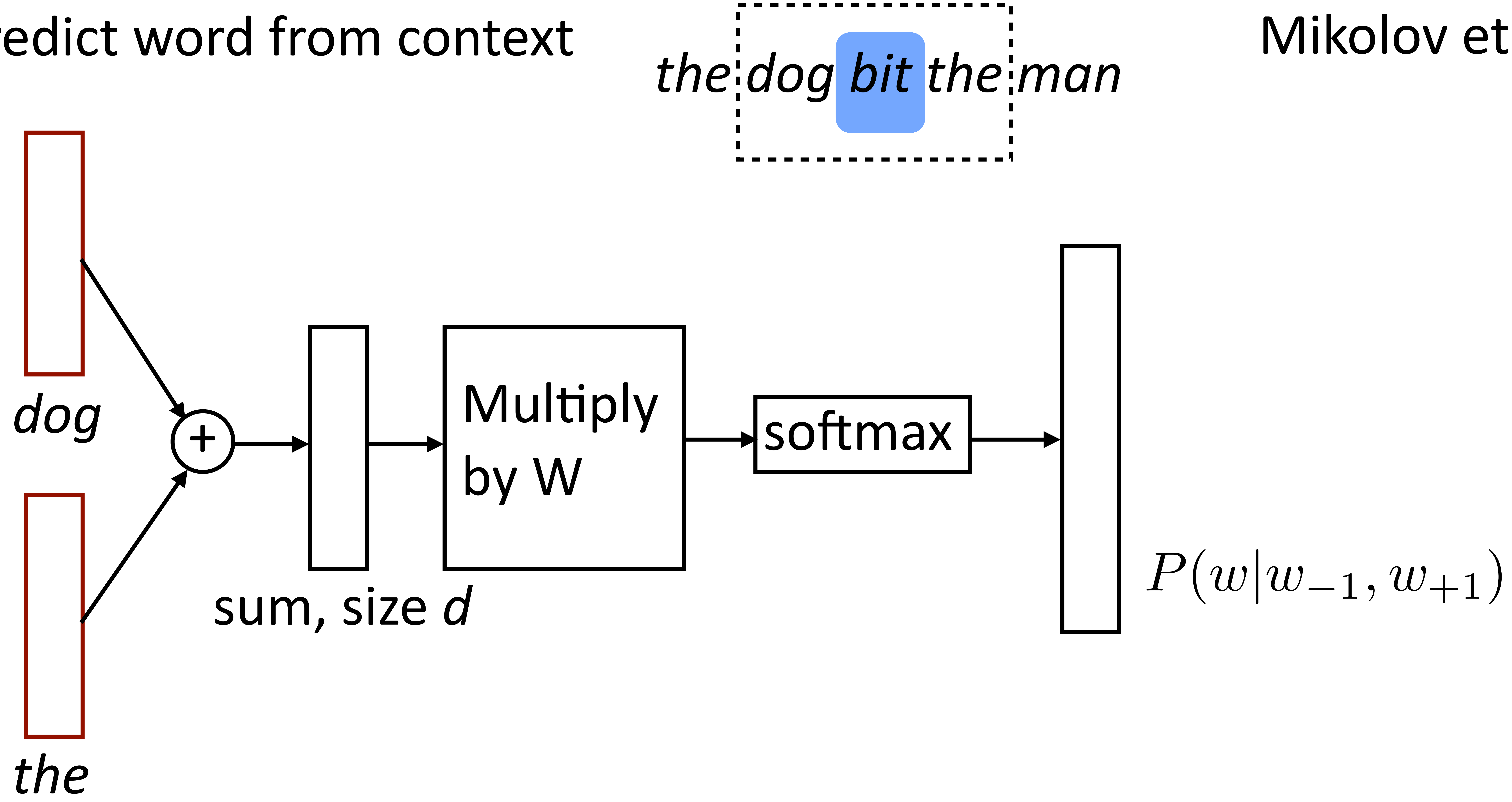
Mikolov et al. (2013)



Recall: Continuous Bag-of-Words

- Predict word from context

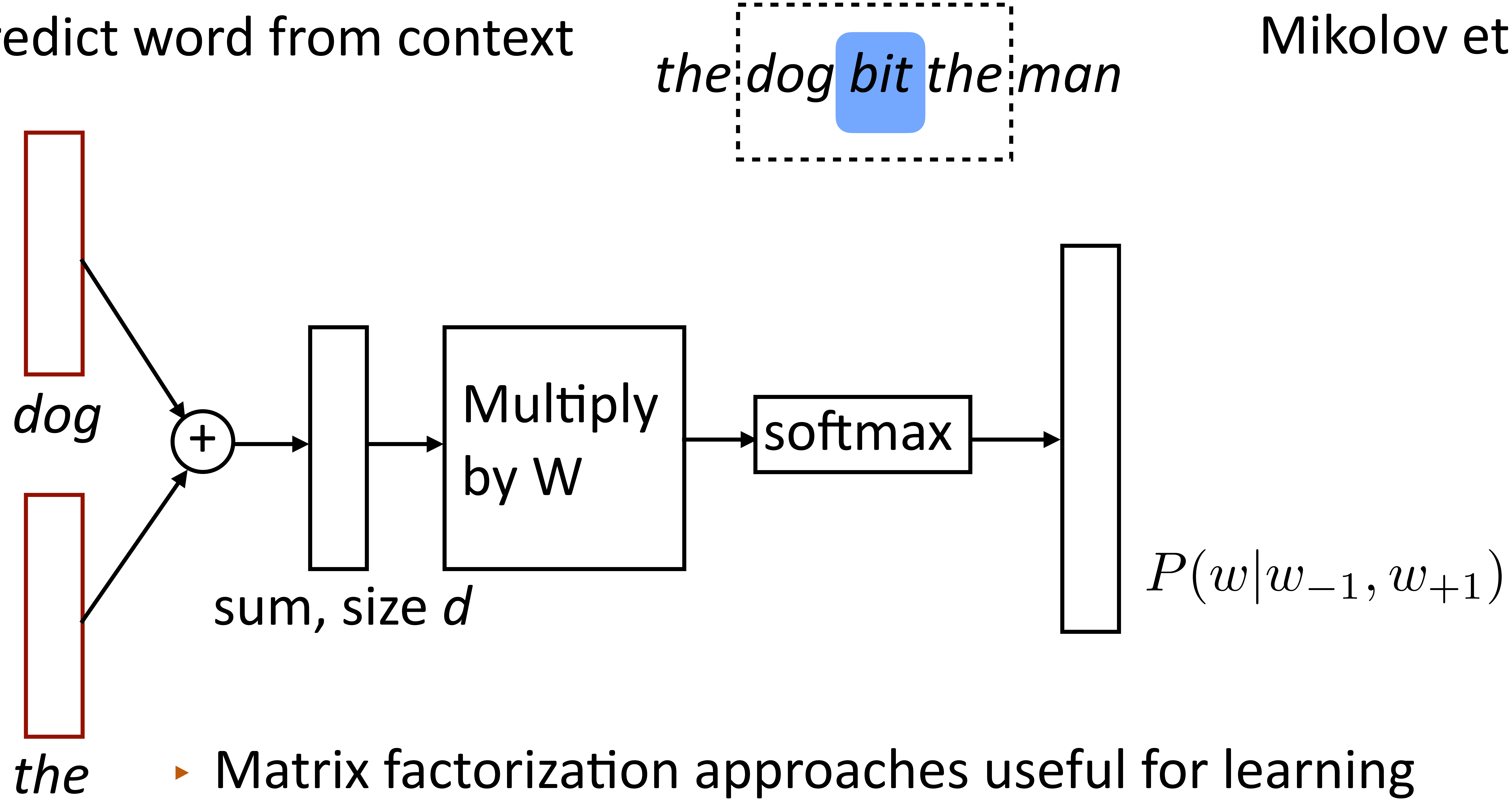
Mikolov et al. (2013)



Recall: Continuous Bag-of-Words

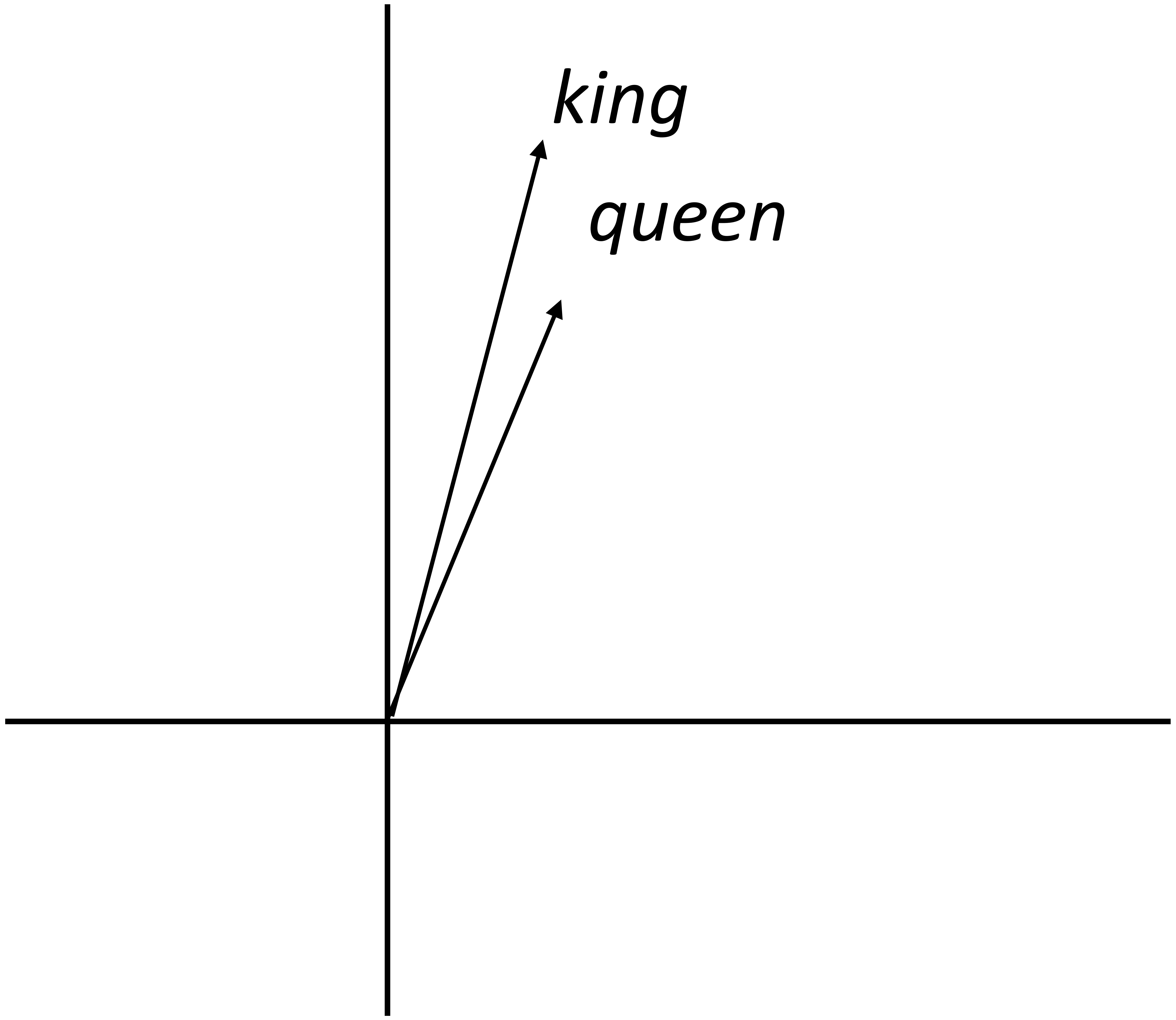
- Predict word from context

Mikolov et al. (2013)

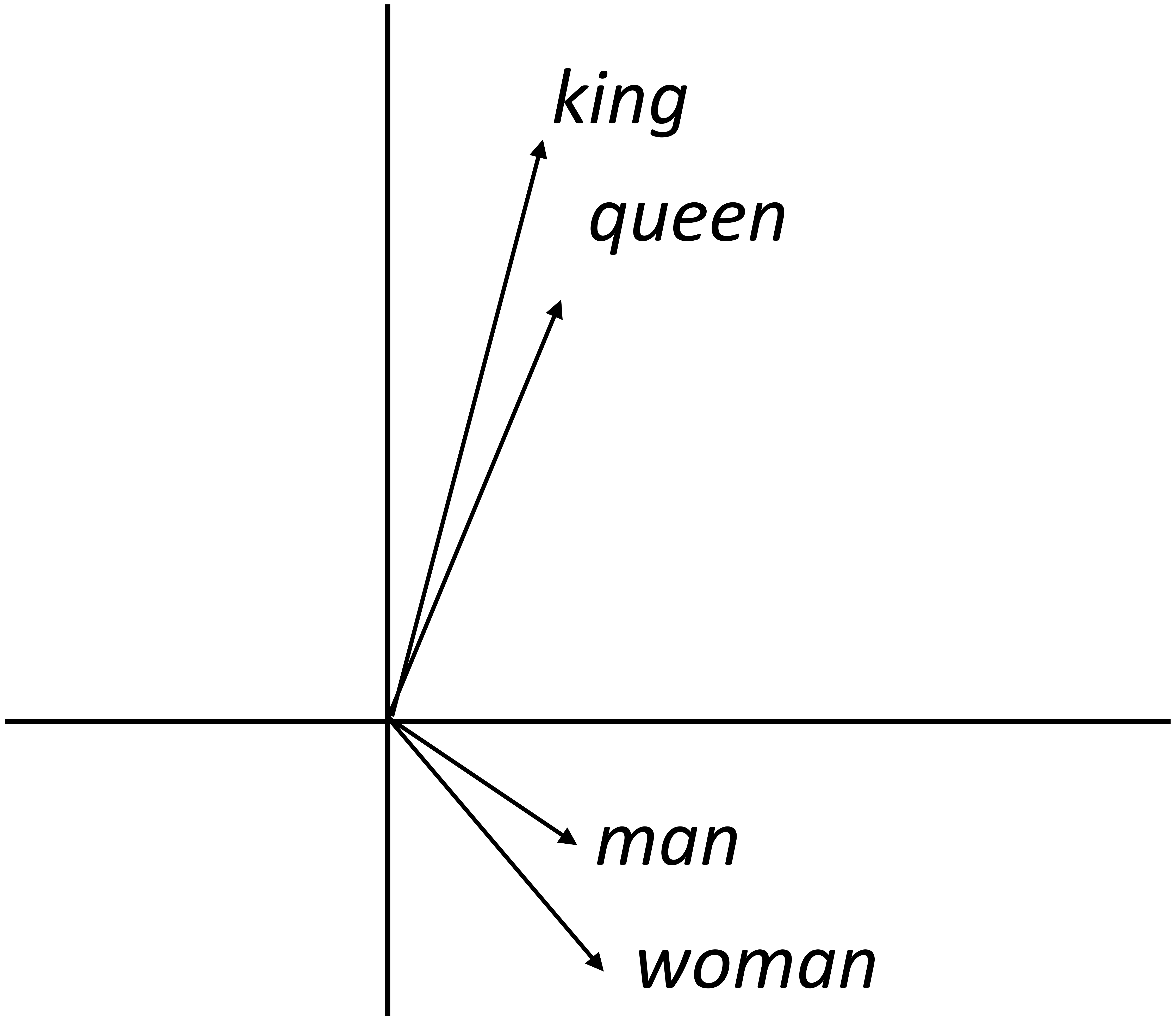


- Matrix factorization approaches useful for learning vectors from really large data

Analogies

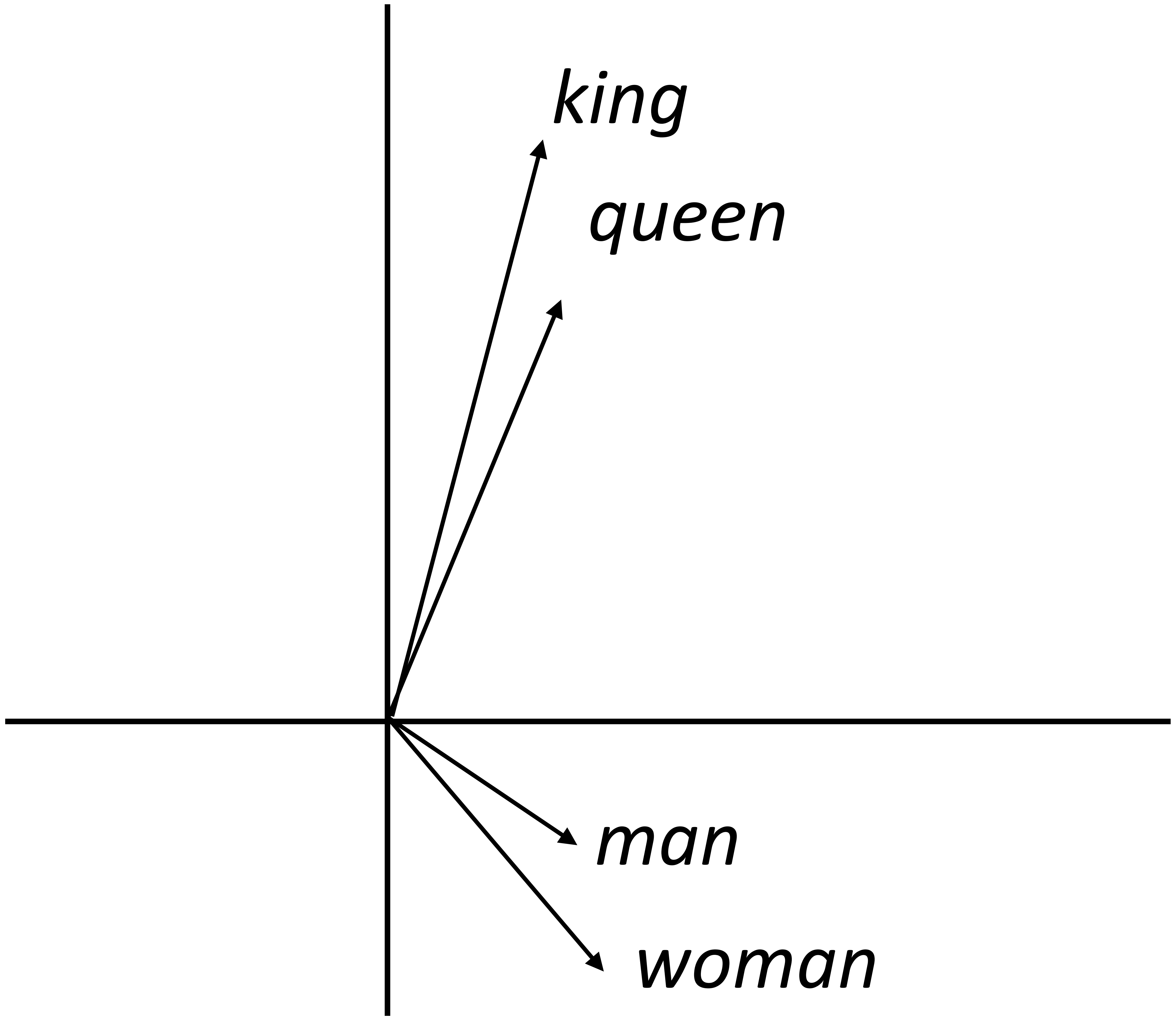


Analogies



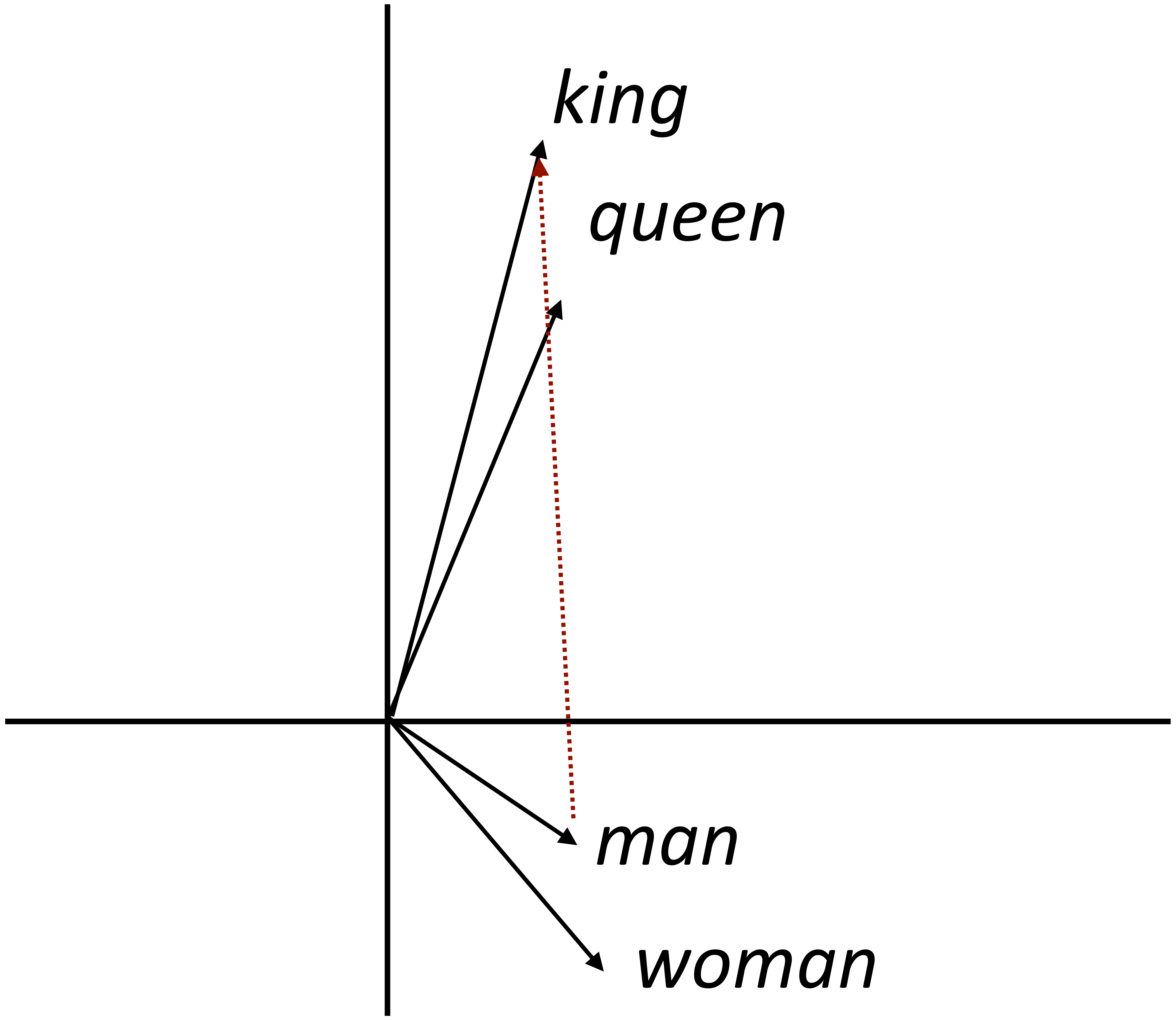
Analogies

(king - man) + woman = queen



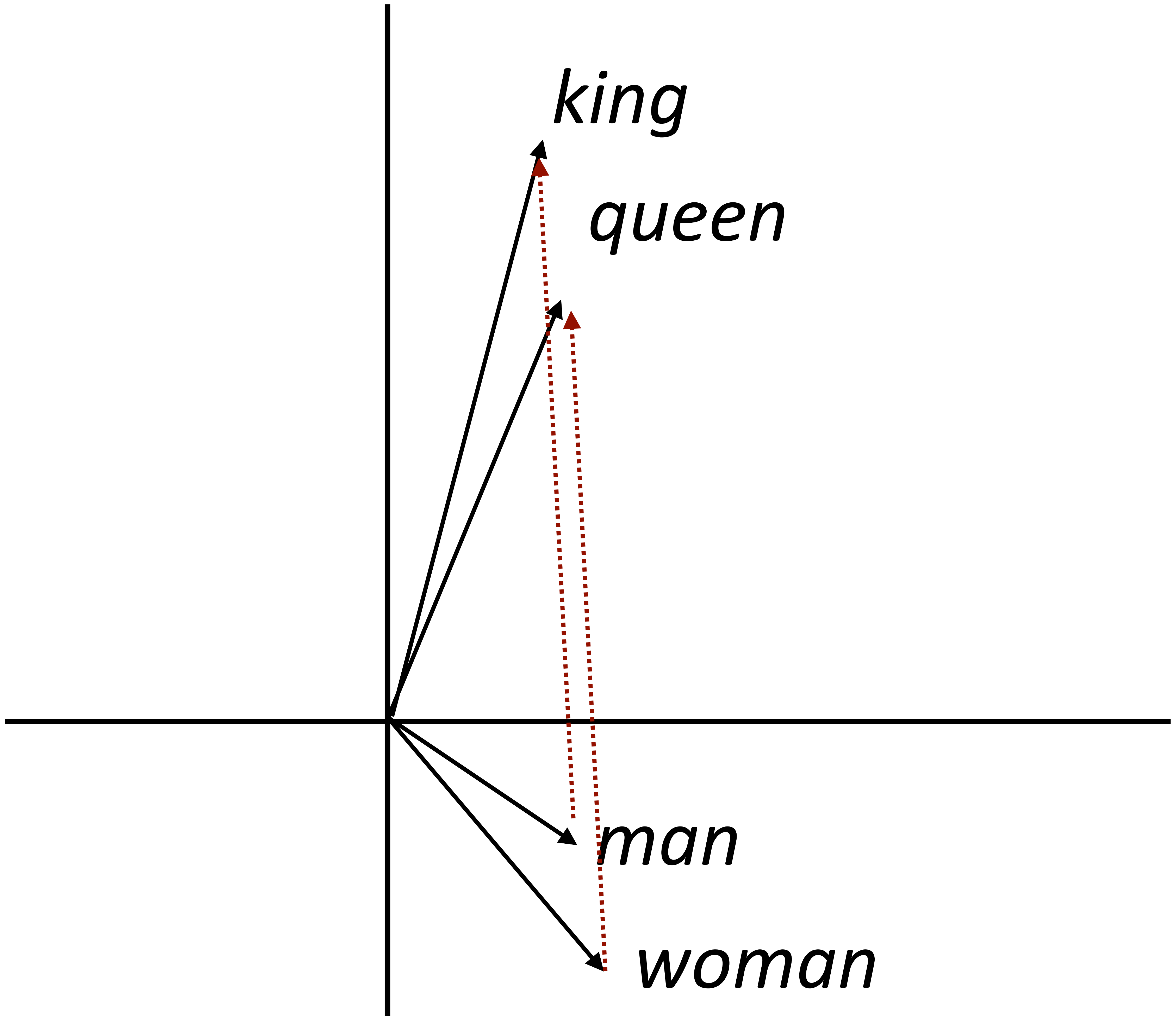
Analogies

(king - man) + woman = queen



Analogies

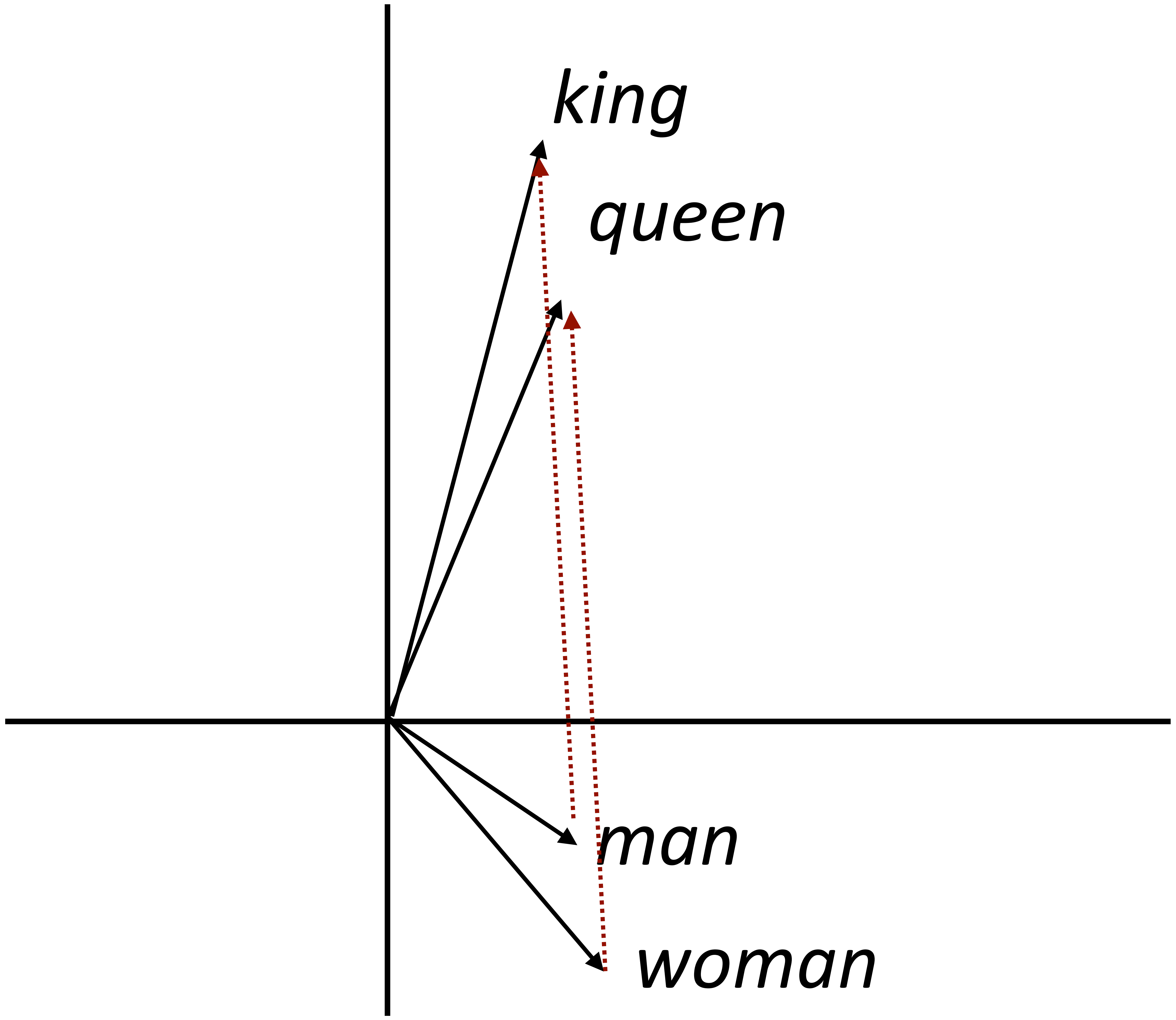
(king - man) + woman = queen



Analogies

(king - man) + woman = queen

king + (woman - man) = queen

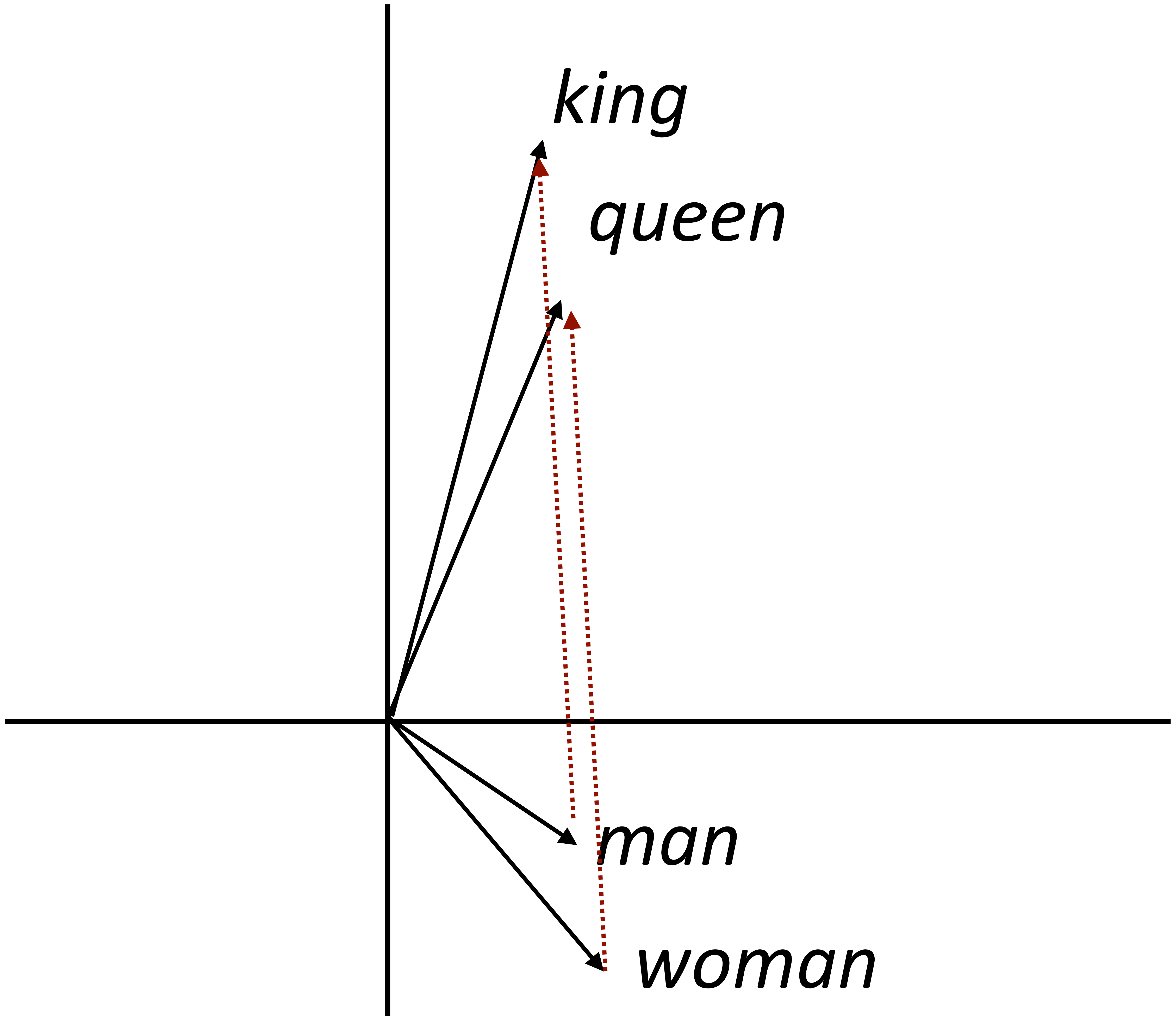


Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?

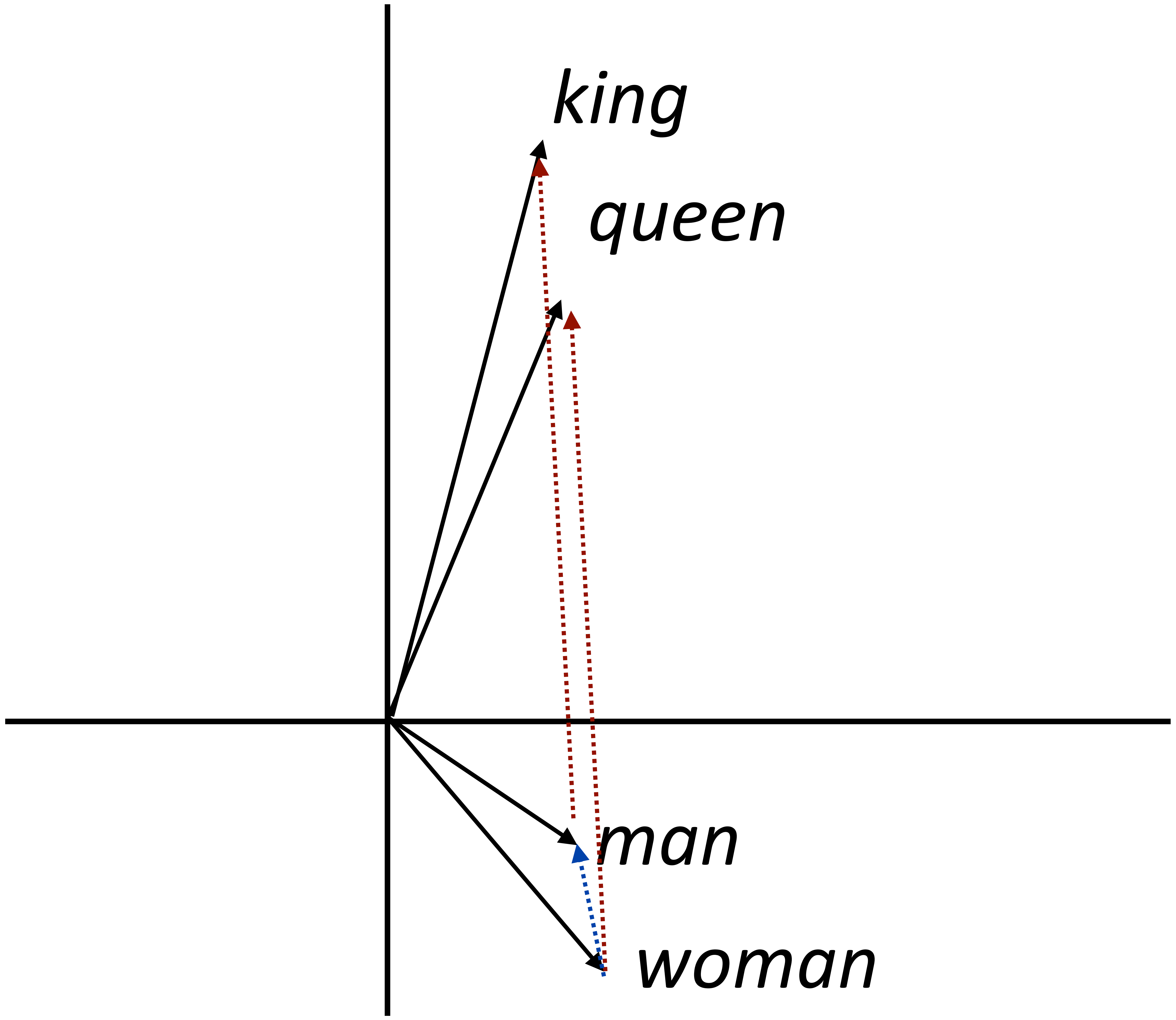


Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?

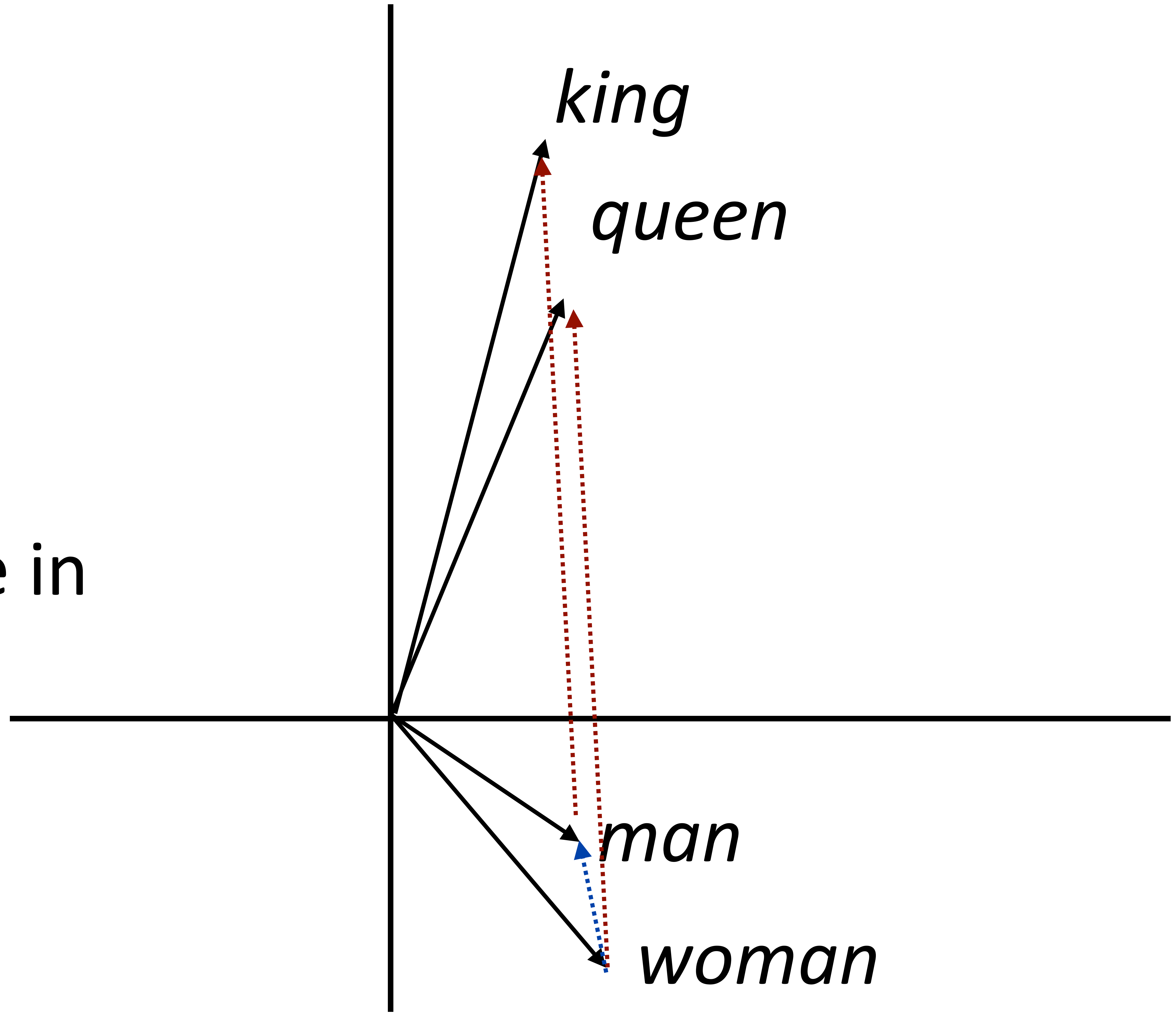


Analogies

(king - man) + woman = queen

king + (woman - man) = queen

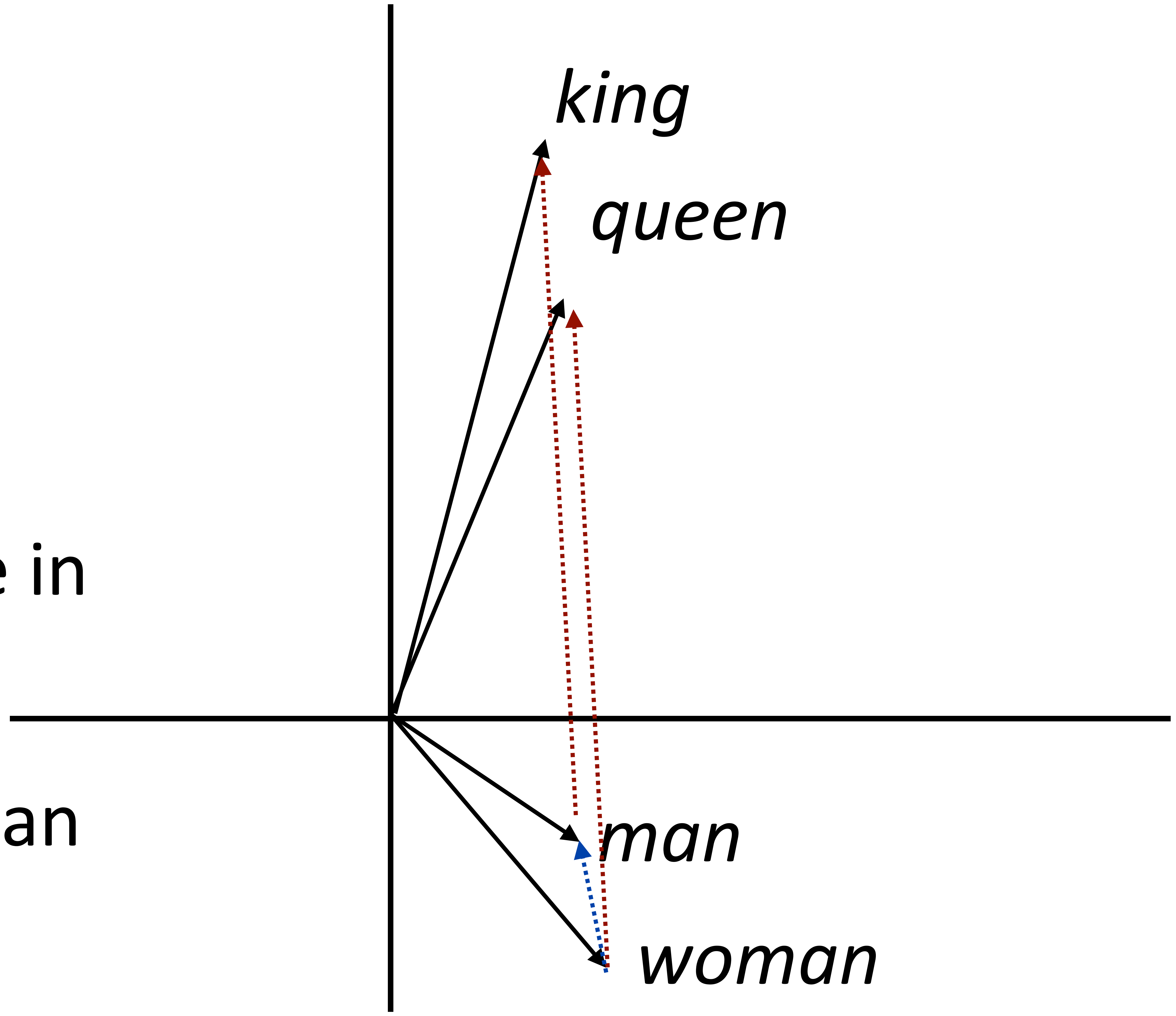
- Why would this be?
- woman - man captures the difference in the contexts that these occur in



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

Analogies

Method	Google Add / Mul	MSR Add / Mul
PPMI	.553 / .679	.306 / .535
SVD	.554 / .591	.408 / .468
SGNS	.676 / **.688**	.618 / **.645**
GloVe	.569 / .596	.533 / .580

Levy et al. (2015)

Analogies

Method	Google Add / Mul	MSR Add / Mul
PPMI	.553 / .679	.306 / .535
SVD	.554 / .591	.408 / .468
SGNS	.676 / **.688**	.618 / **.645**
GloVe	.569 / .596	.533 / .580

- These methods can perform well on analogies on two different datasets using two different methods

Levy et al. (2015)

Analogies

Method	Google Add / Mul	MSR Add / Mul
PPMI	.553 / .679	.306 / .535
SVD	.554 / .591	.408 / .468
SGNS	.676 / **.688**	.618 / **.645**
GloVe	.569 / .596	.533 / .580

- These methods can perform well on analogies on two different datasets using two different methods

Maximizing for *b*: Add = $\cos(b, \underbrace{a_2}_{\text{king}} - \underbrace{a_1}_{\text{man}} + \underbrace{b_1}_{\text{woman}})$ Mul = $\dfrac{\cos(b_2, a_2)\cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}$

Levy et al. (2015)

Using Word Embeddings

Using Word Embeddings

- Approach 1: learn embeddings directly from data in your neural model, no pretraining
  - Often works pretty well

Using Word Embeddings

- Approach 1: learn embeddings directly from data in your neural model, no pretraining
  - Often works pretty well
- Approach 2: pretrain using GloVe, keep fixed
  - Faster because no need to update these parameters
  - Need to make sure GloVe vocabulary contains all the words you need

Using Word Embeddings

- Approach 1: learn embeddings directly from data in your neural model, no pretraining
  - Often works pretty well
- Approach 2: pretrain using GloVe, keep fixed
  - Faster because no need to update these parameters
  - Need to make sure GloVe vocabulary contains all the words you need
- Approach 3: initialize using GloVe, fine-tune
  - Not as commonly used anymore

This Lecture

- Recurrent neural networks

- Vanishing gradient problem

- LSTMs / GRUs

- Applications / visualizations

RNN Basics

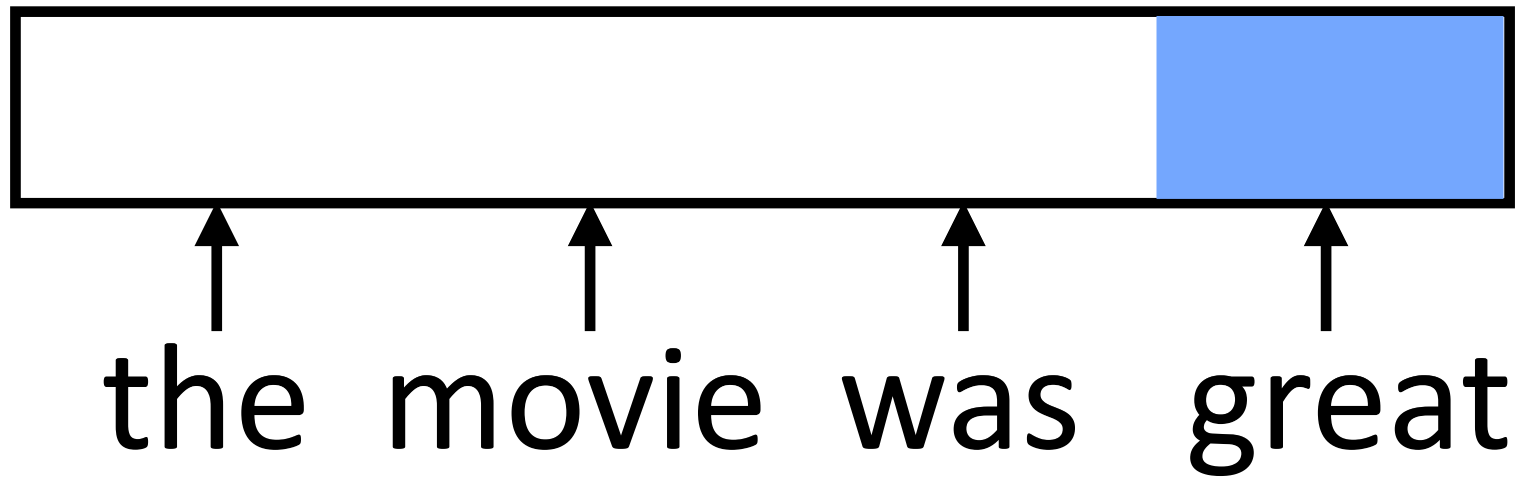
RNN Motivation

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

the movie was great

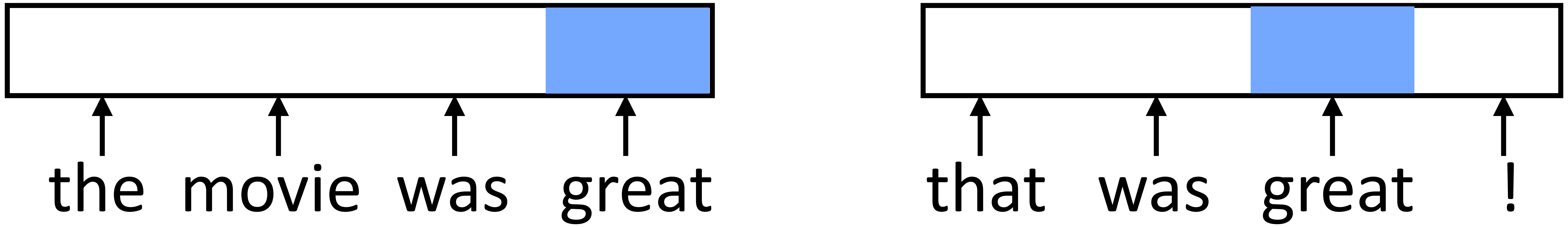
RNN Motivation

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



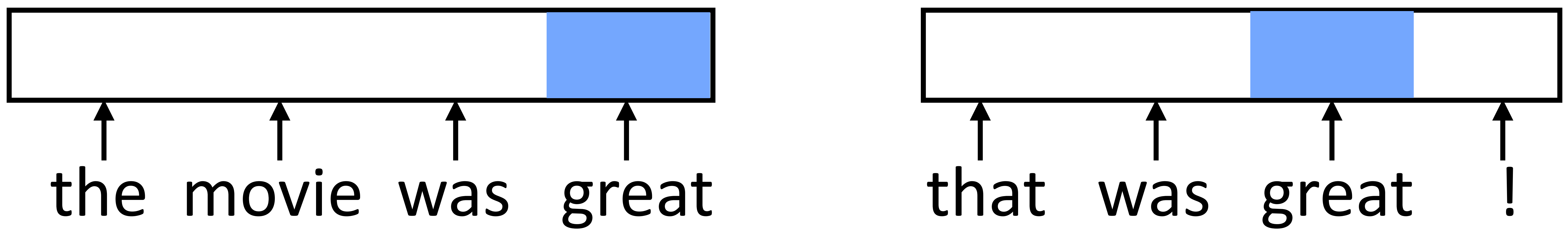
RNN Motivation

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



RNN Motivation

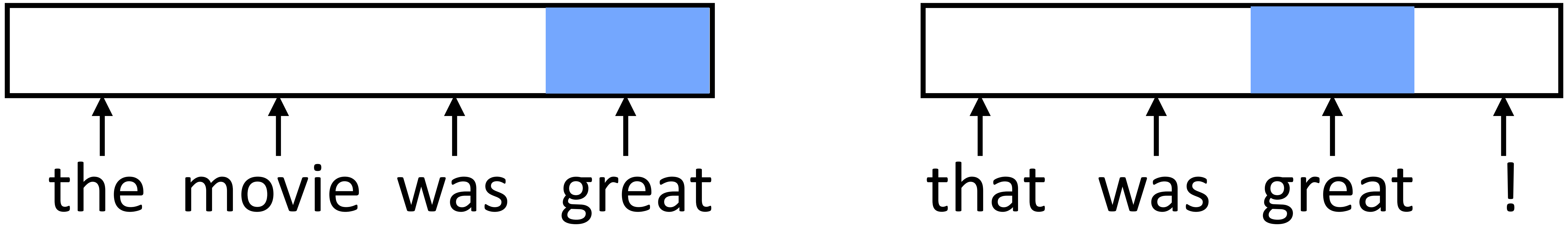
- Feedforward NNs can't handle variable length input: each position in the feature vector has fixed semantics



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

RNN Motivation

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

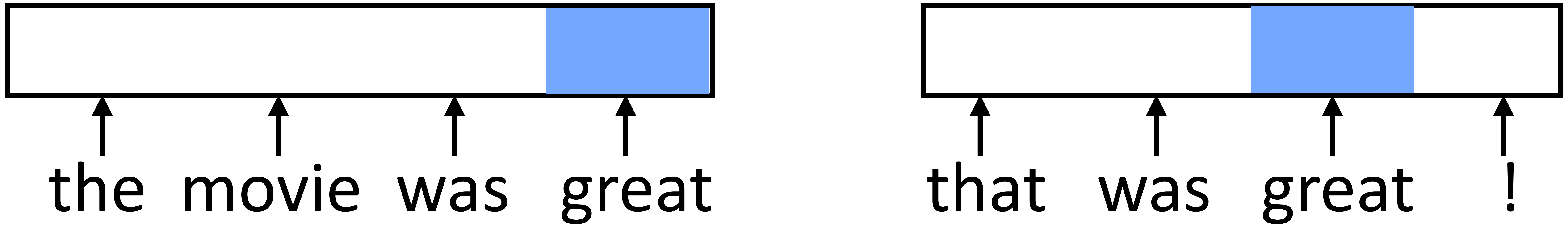


- These don't look related (*great* is in two different orthogonal subspaces)

- Instead, we need to:

RNN Motivation

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



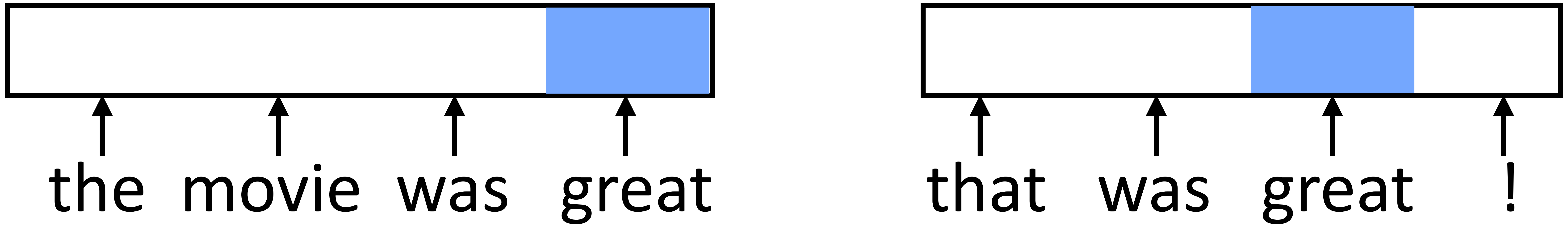
- These don't look related (*great* is in two different orthogonal subspaces)

- Instead, we need to:

1) Process each word in a uniform way

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)

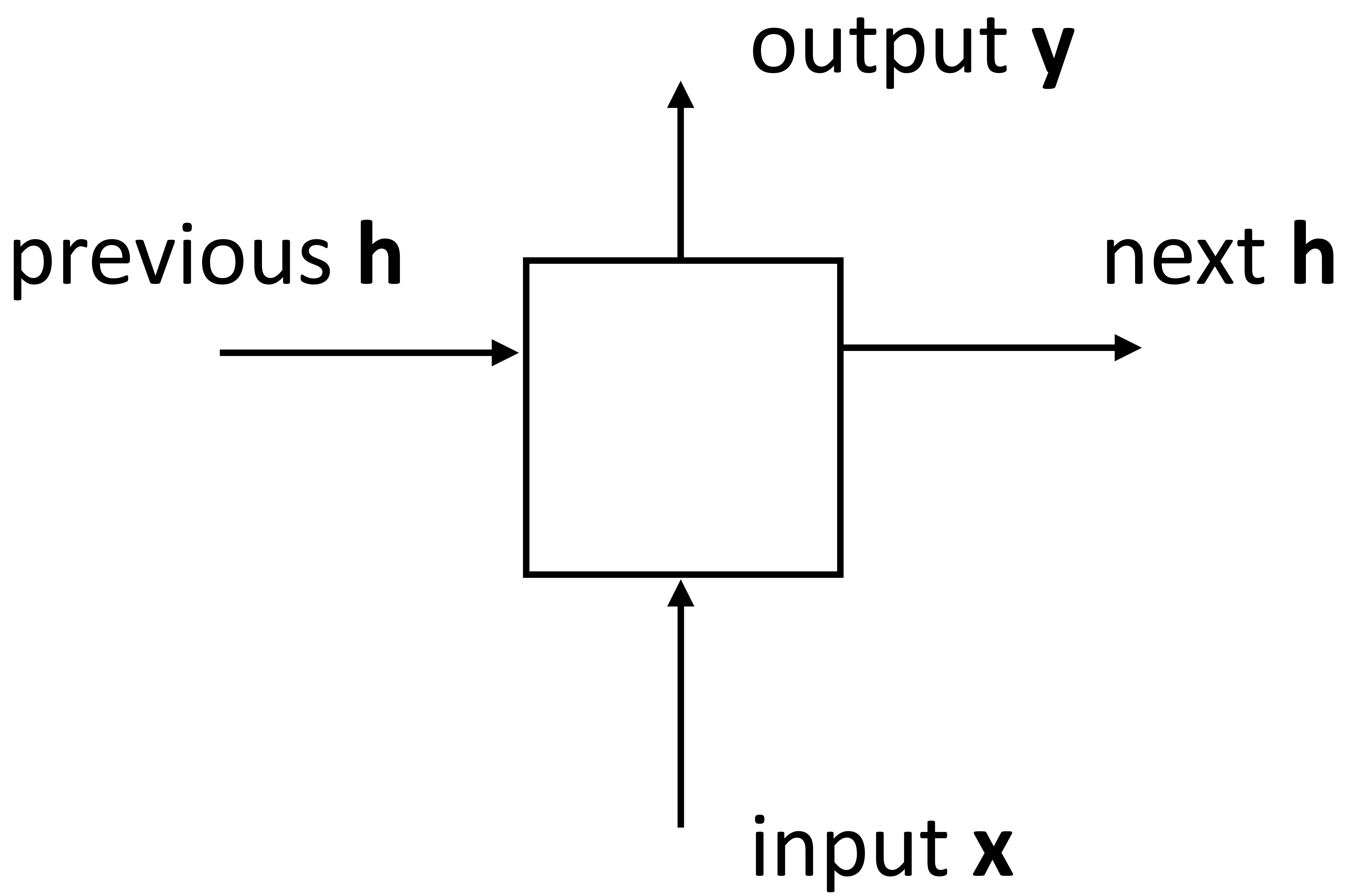
- Instead, we need to:

1) Process each word in a uniform way

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

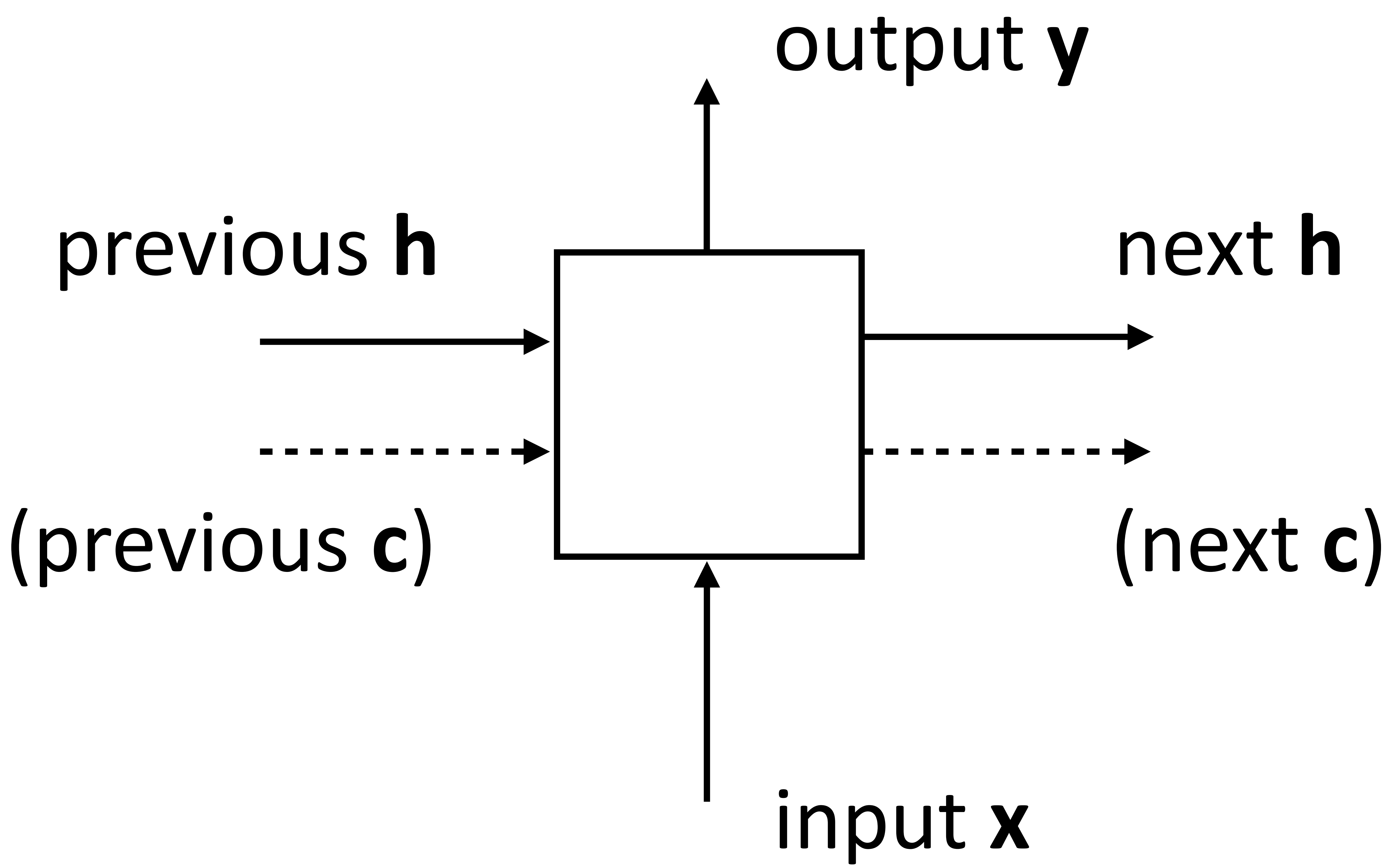
RNN Abstraction

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



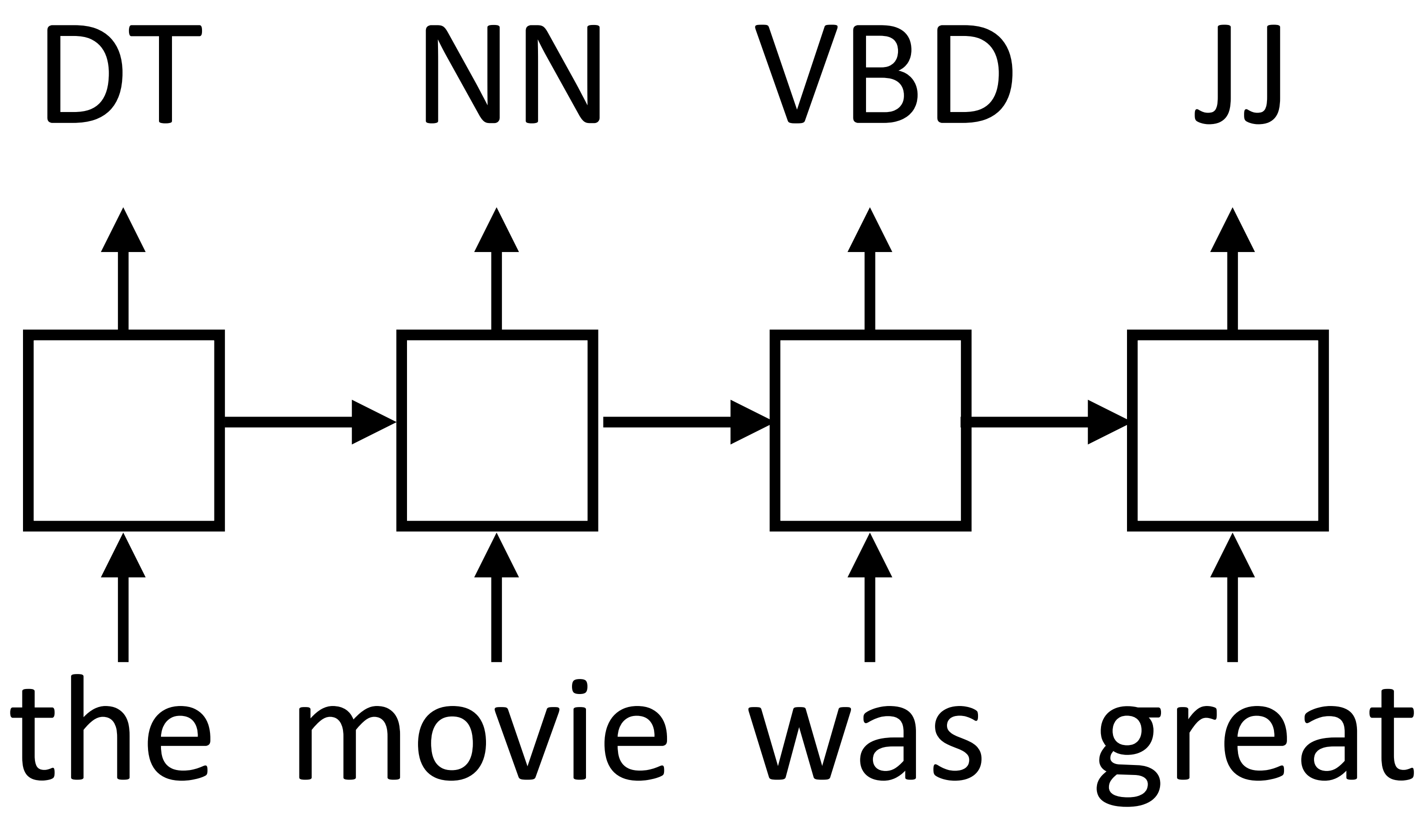
RNN Abstraction

- 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

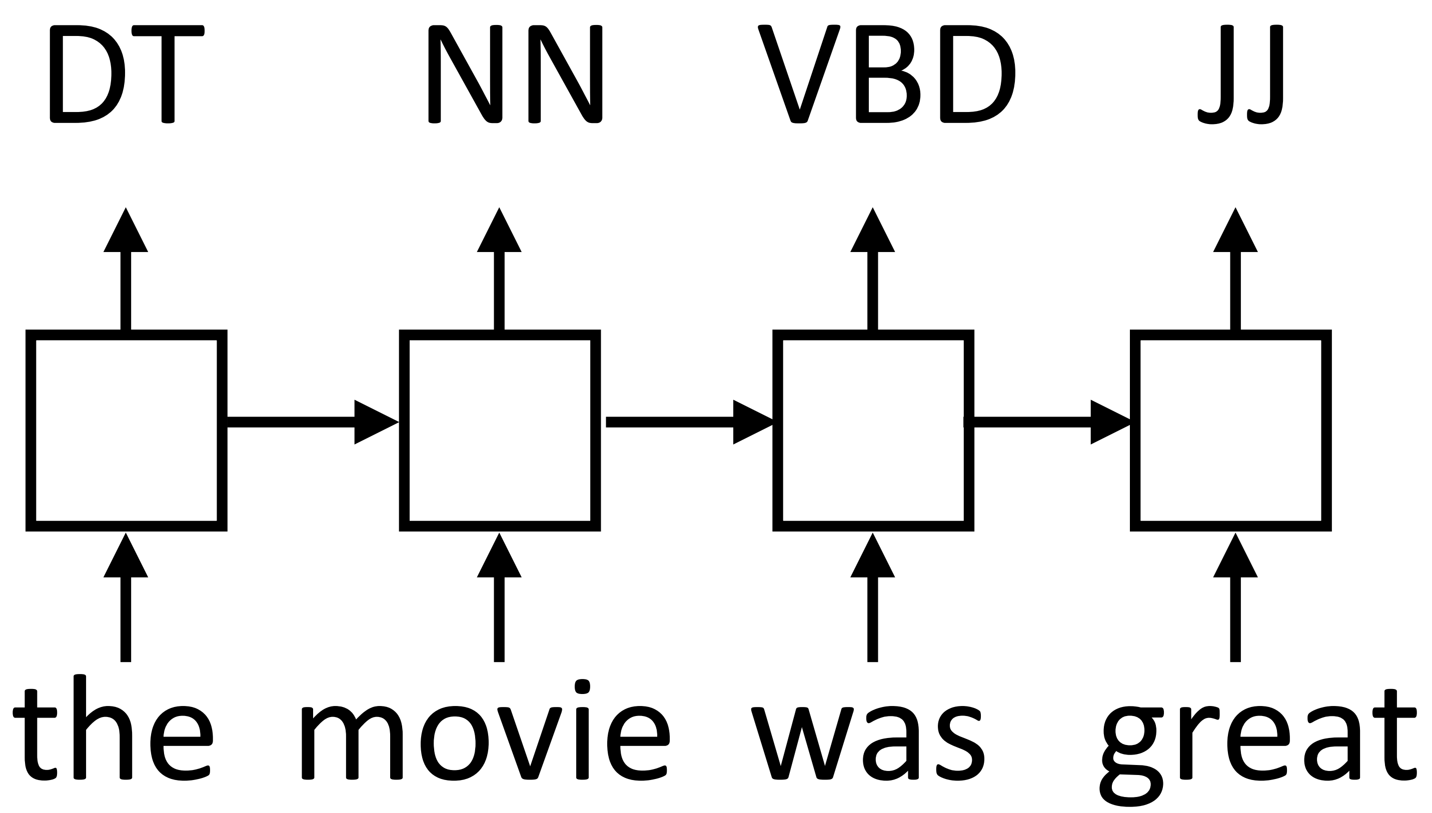
- Transducer: make some prediction for each element in a sequence



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

RNN Uses

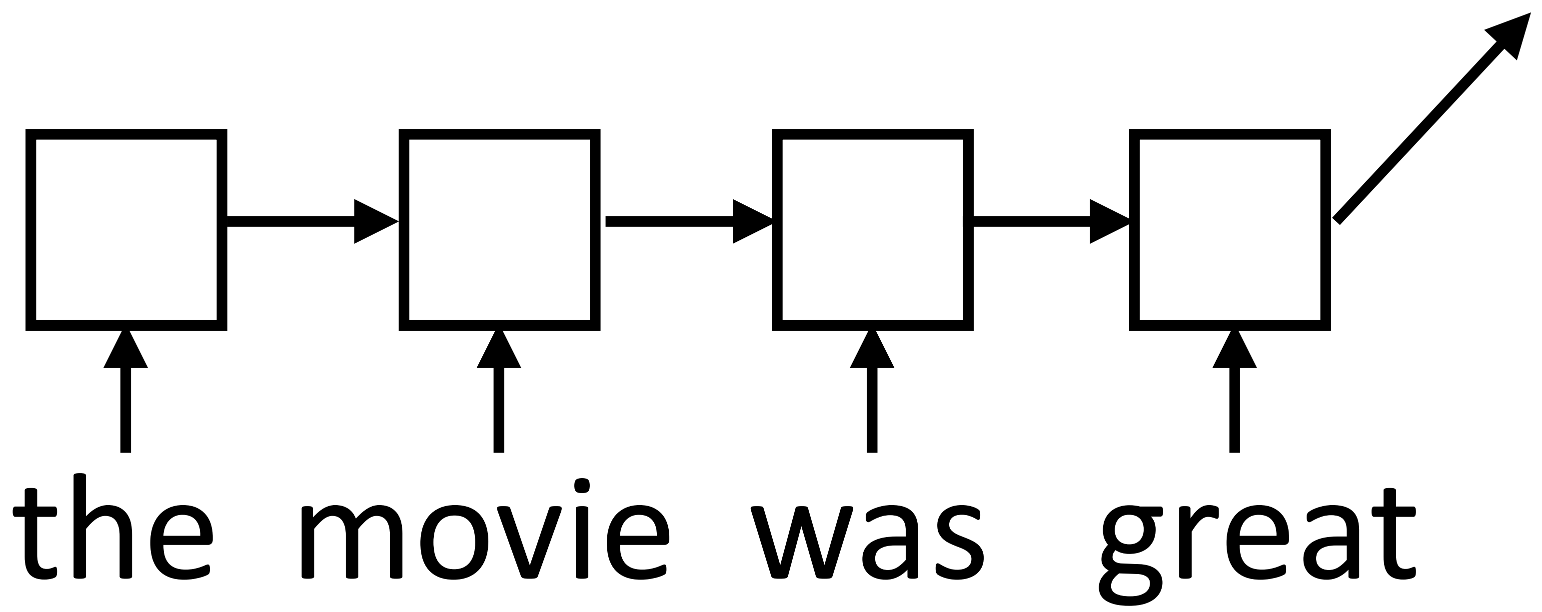
- Transducer: make some prediction for each element in a sequence



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

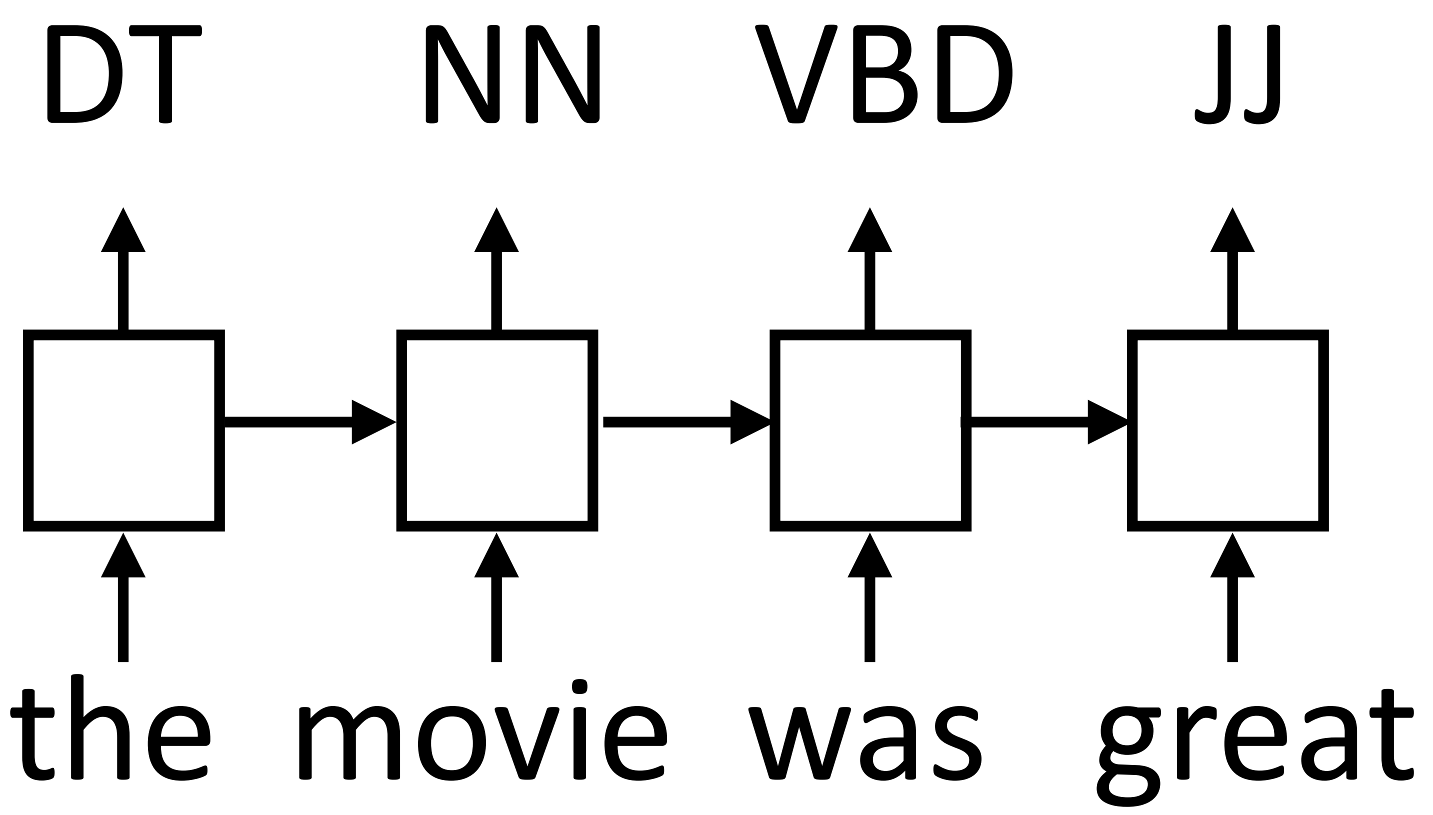
- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

predict sentiment (matmul + softmax)



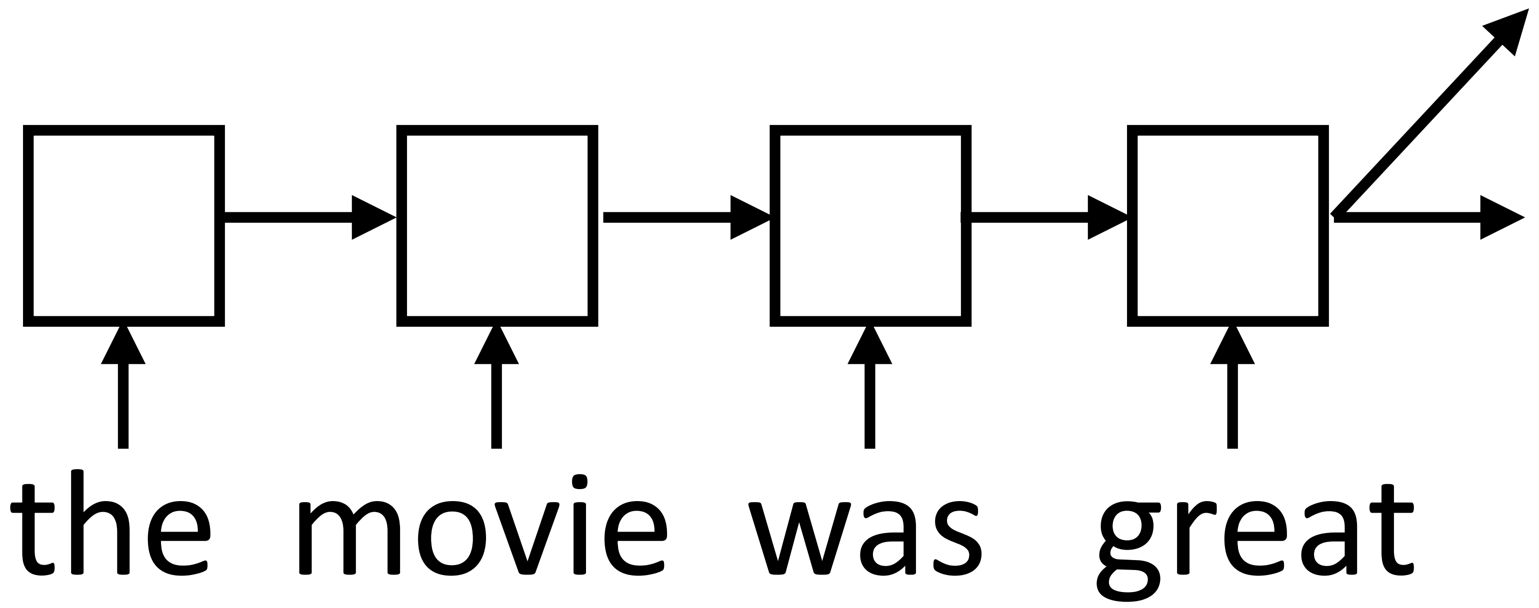
RNN Uses

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



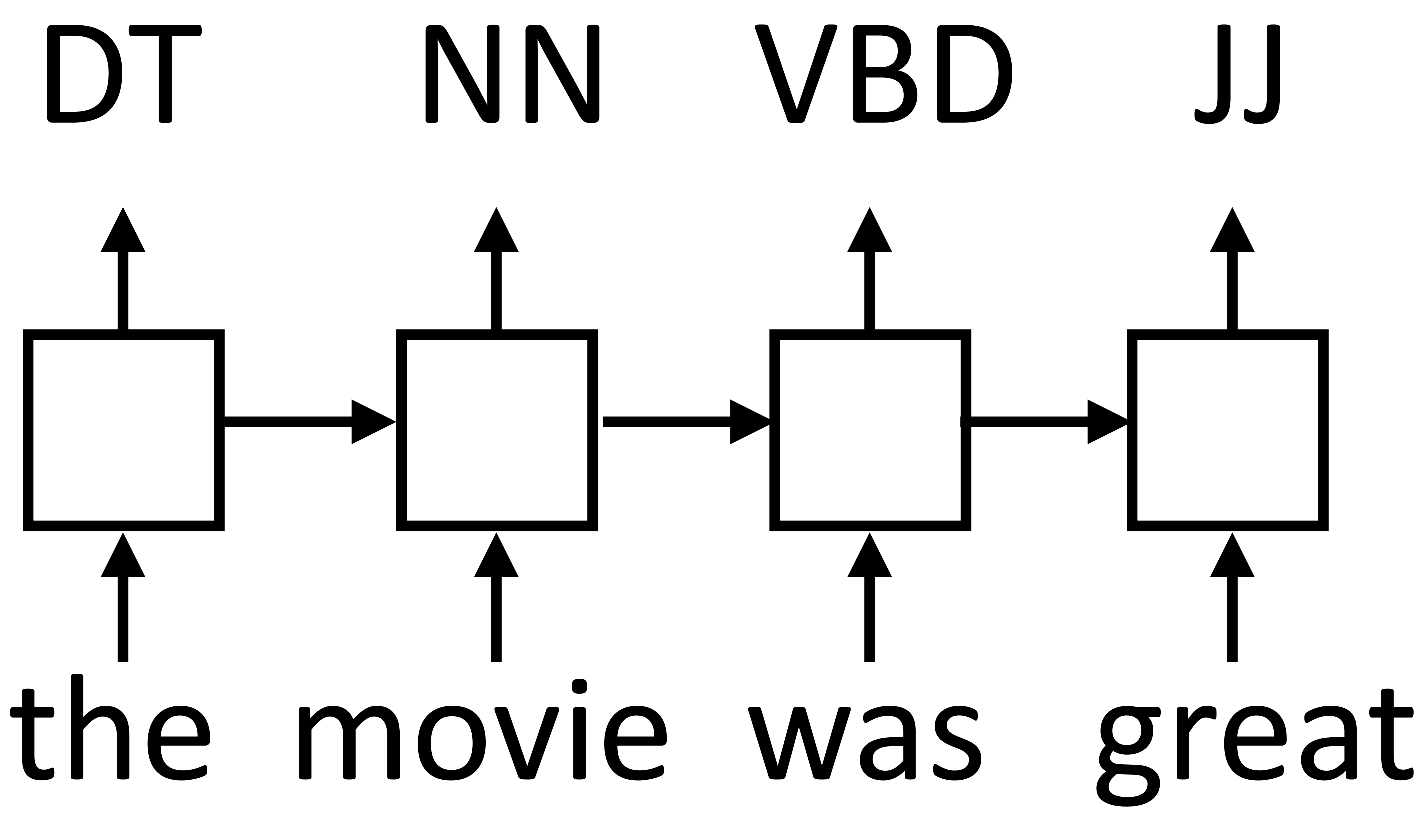
output **y** = score for each tag, then softmax

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



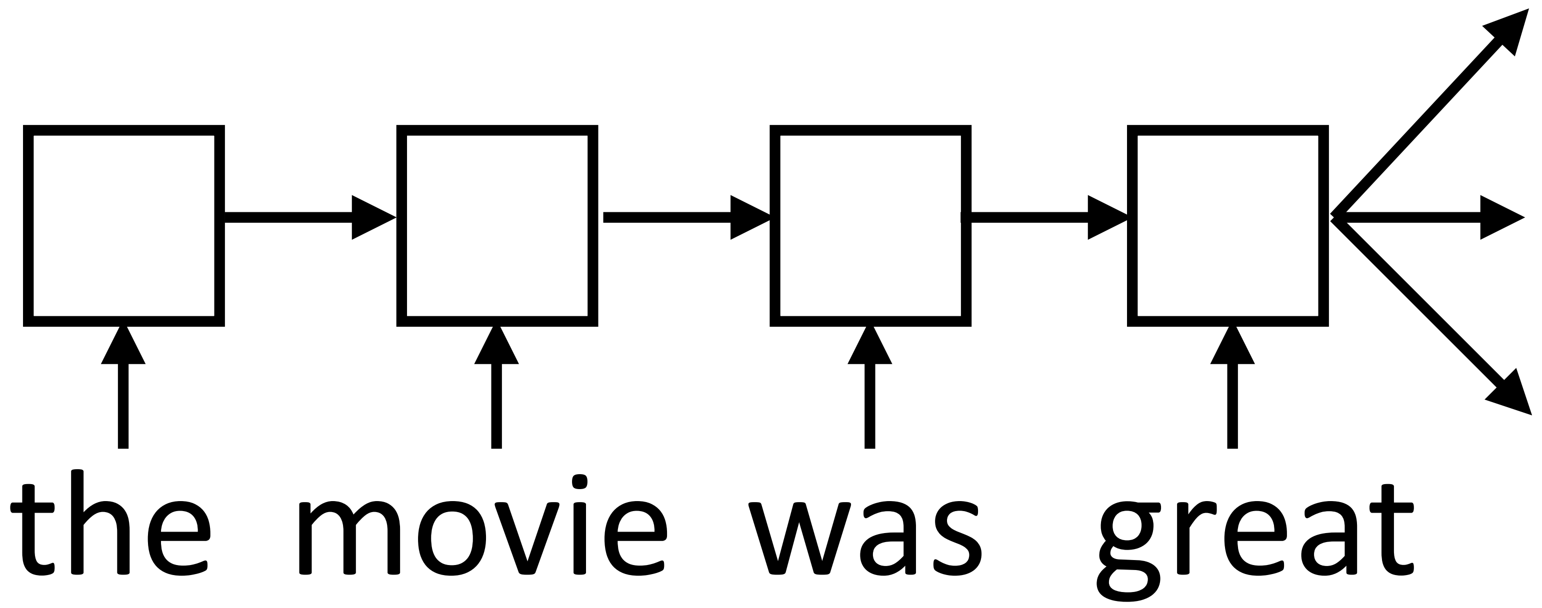
RNN Uses

- 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

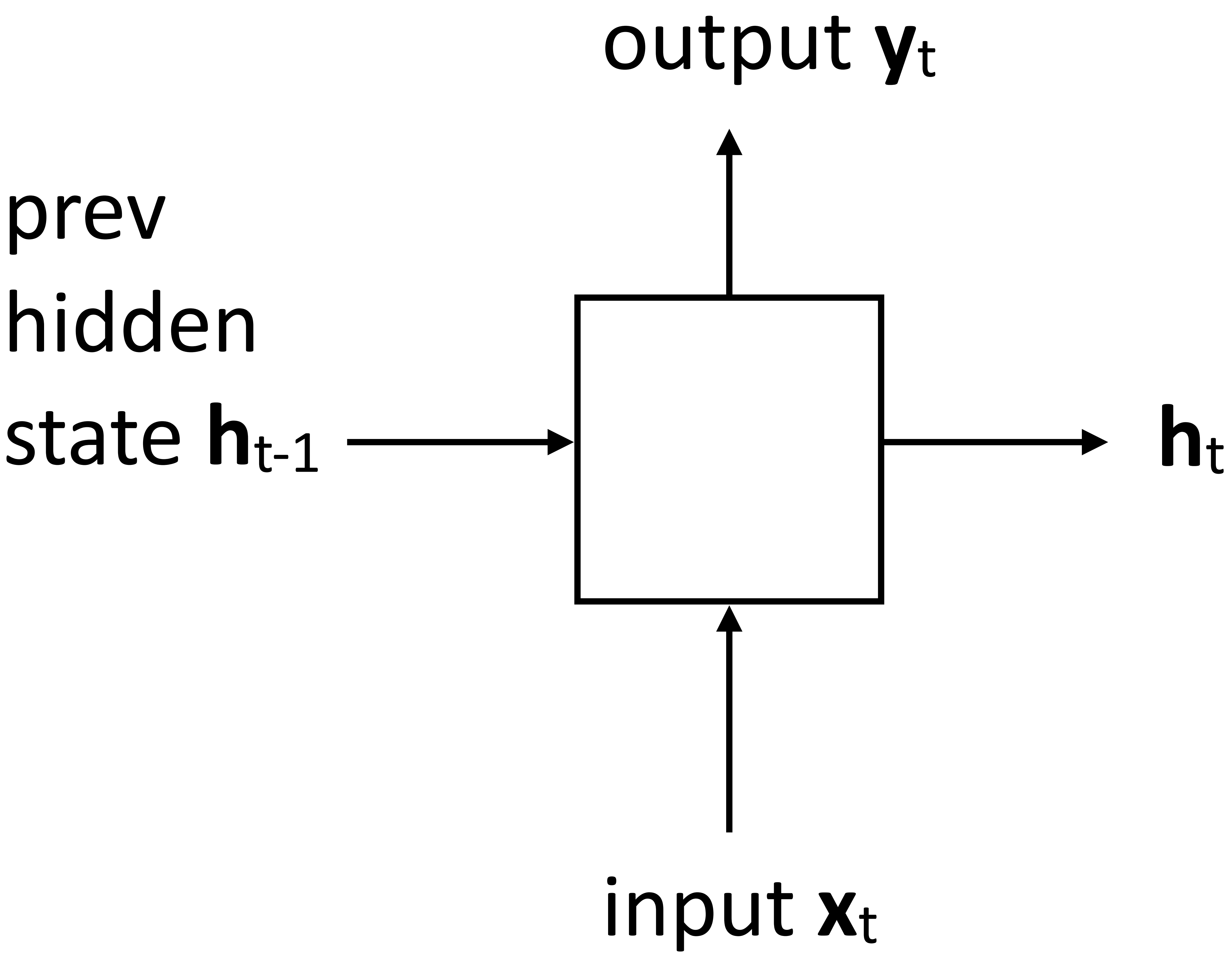


predict sentiment (matmul + softmax)

translate

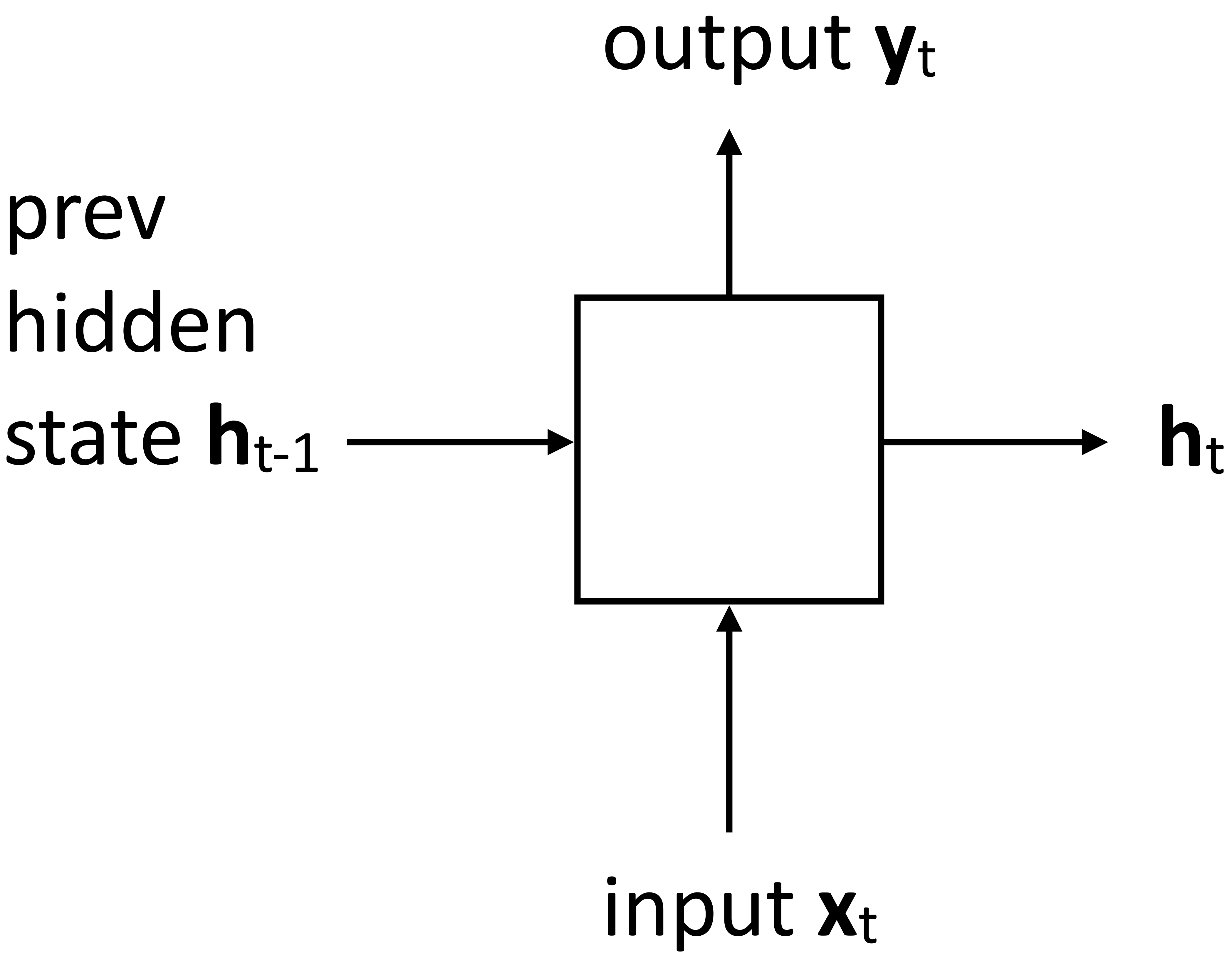
paraphrase/compress

Elman Networks



Elman (1990)

Elman Networks

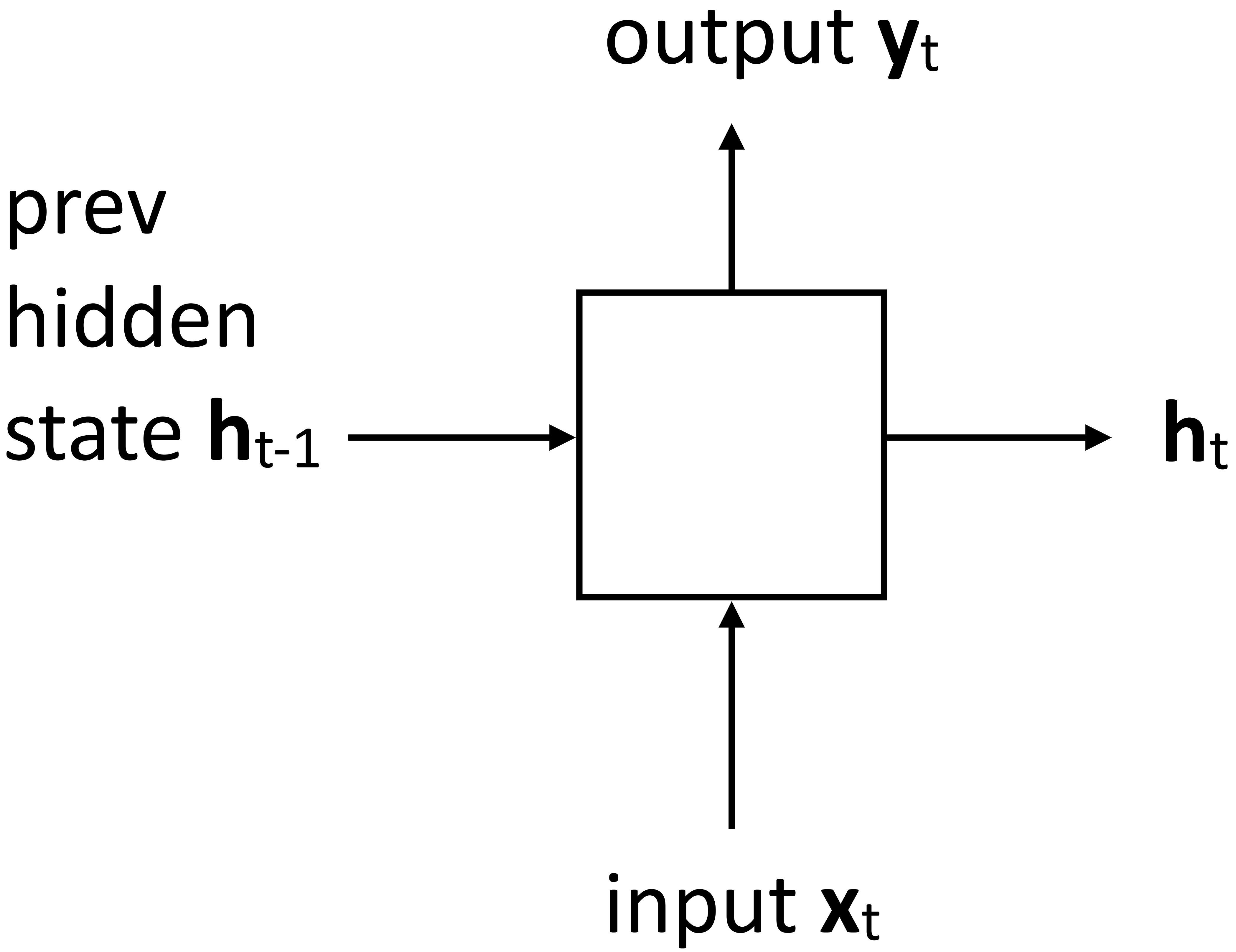


$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

- Updates hidden state based on input and current hidden state

Elman (1990)

Elman Networks



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

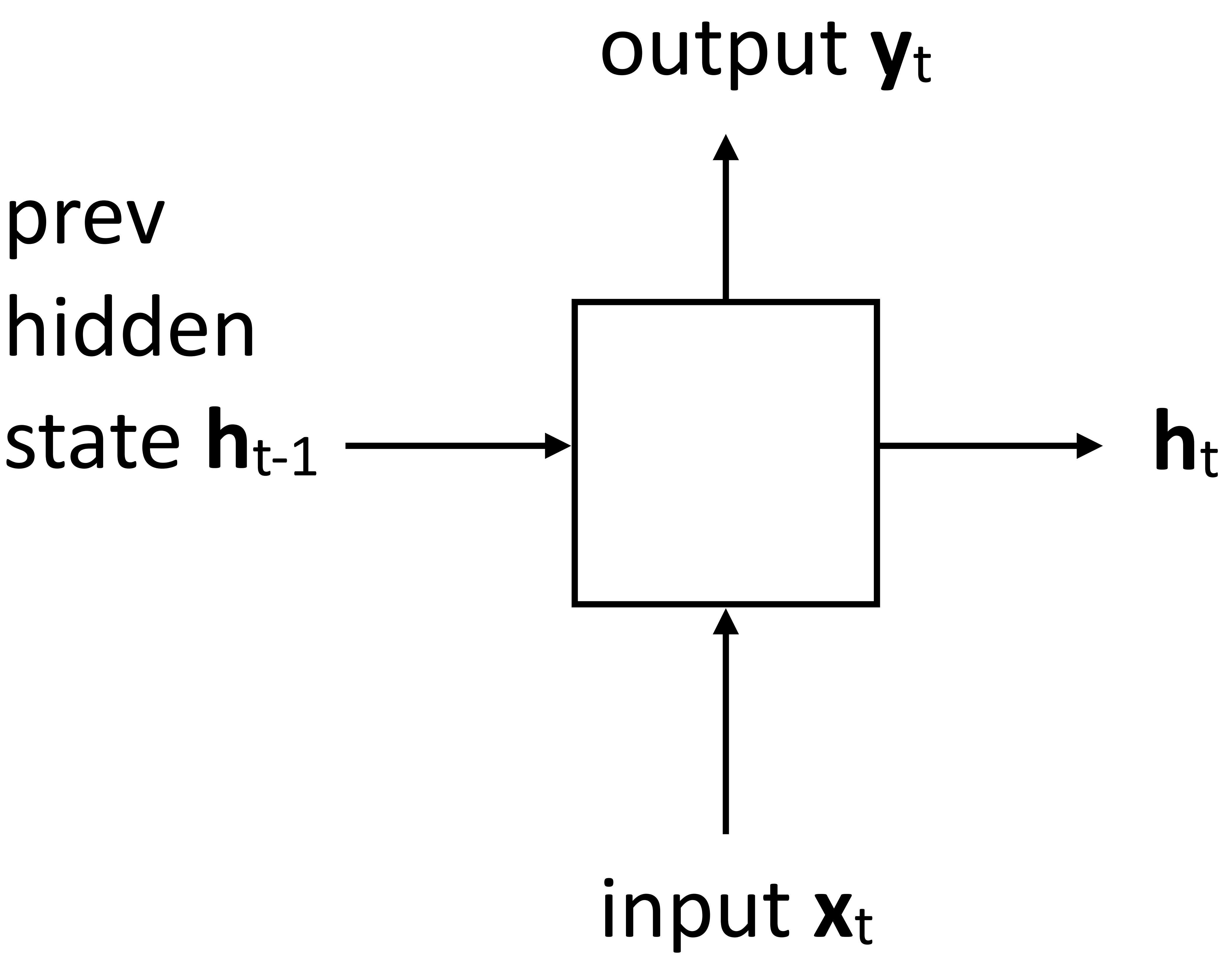
- Updates hidden state based on input and current hidden state

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

- Computes output from hidden state

Elman (1990)

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

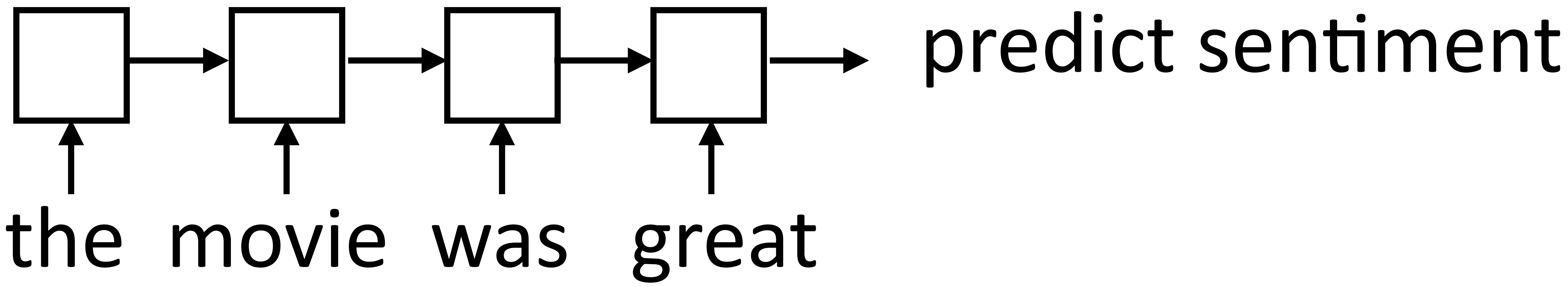
$$\mathbf{y}_t = \tanh(U\mathbf{h_t} + \mathbf{b}_y)$$

- Computes output from hidden state

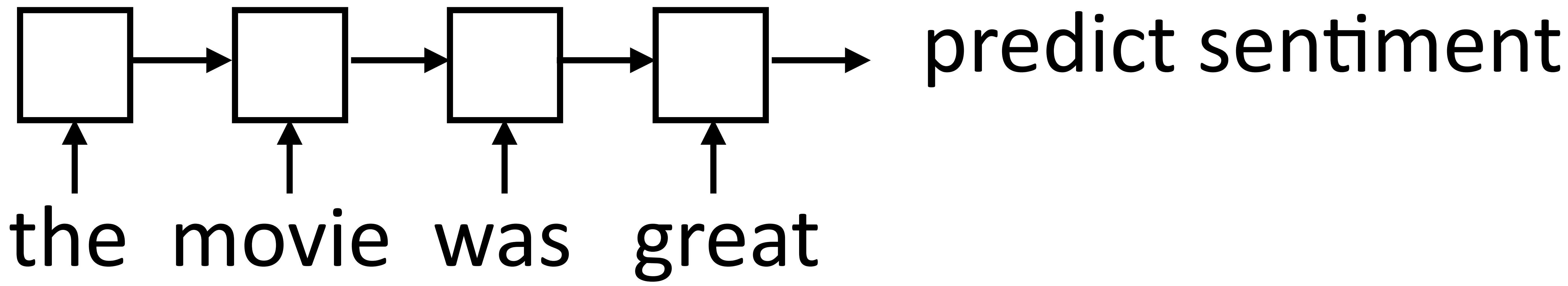
- Long history! (invented in the late 1980s)

Elman (1990)

Training Elman Networks

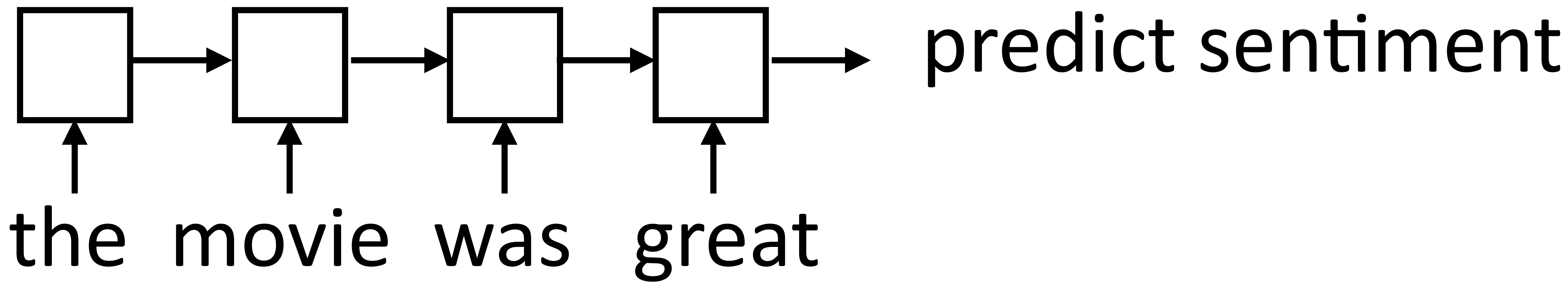


Training Elman Networks



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

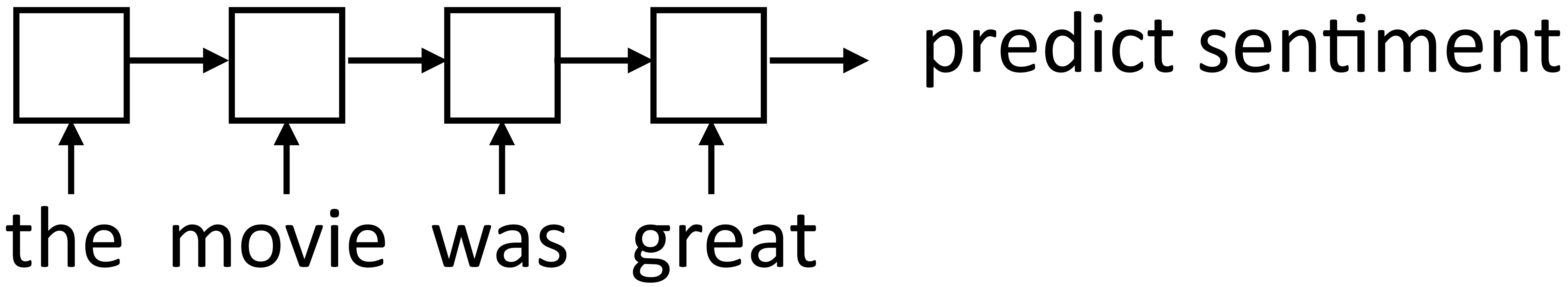
Training Elman Networks



- "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 -> **+**

Training Elman Networks



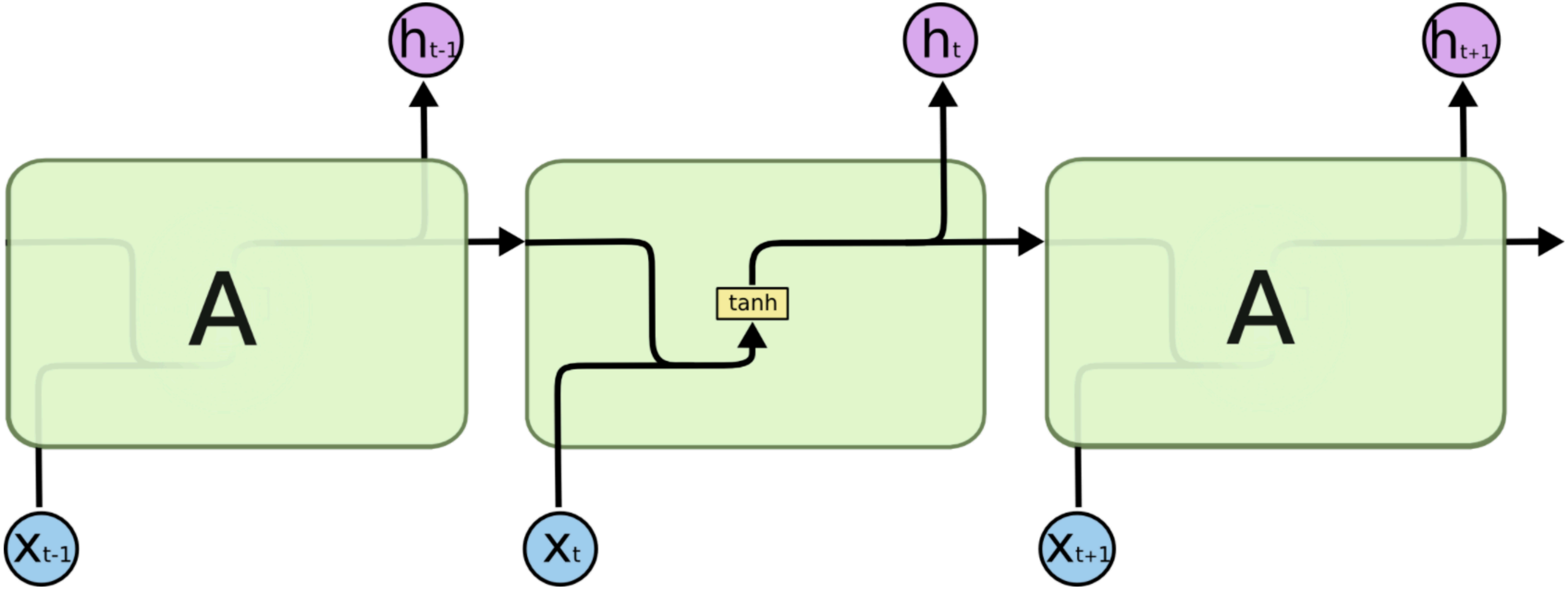
- "Backpropagation through time": build the network as one big computation graph, some parameters are shared

- 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 -> **+**

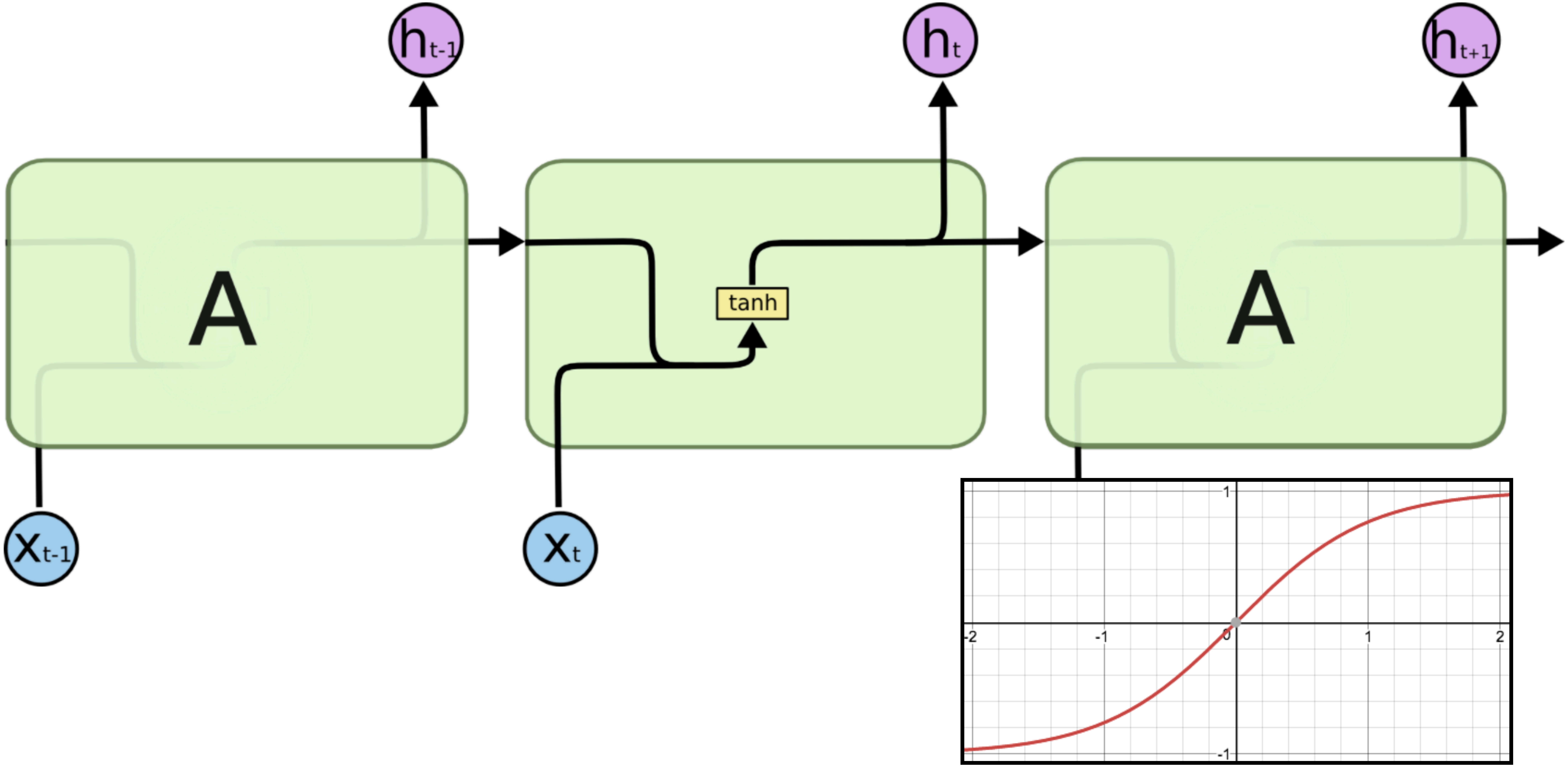
- "Correct" parameter update is to do a better job of remembering the sentiment of *favorite*

Vanishing Gradient



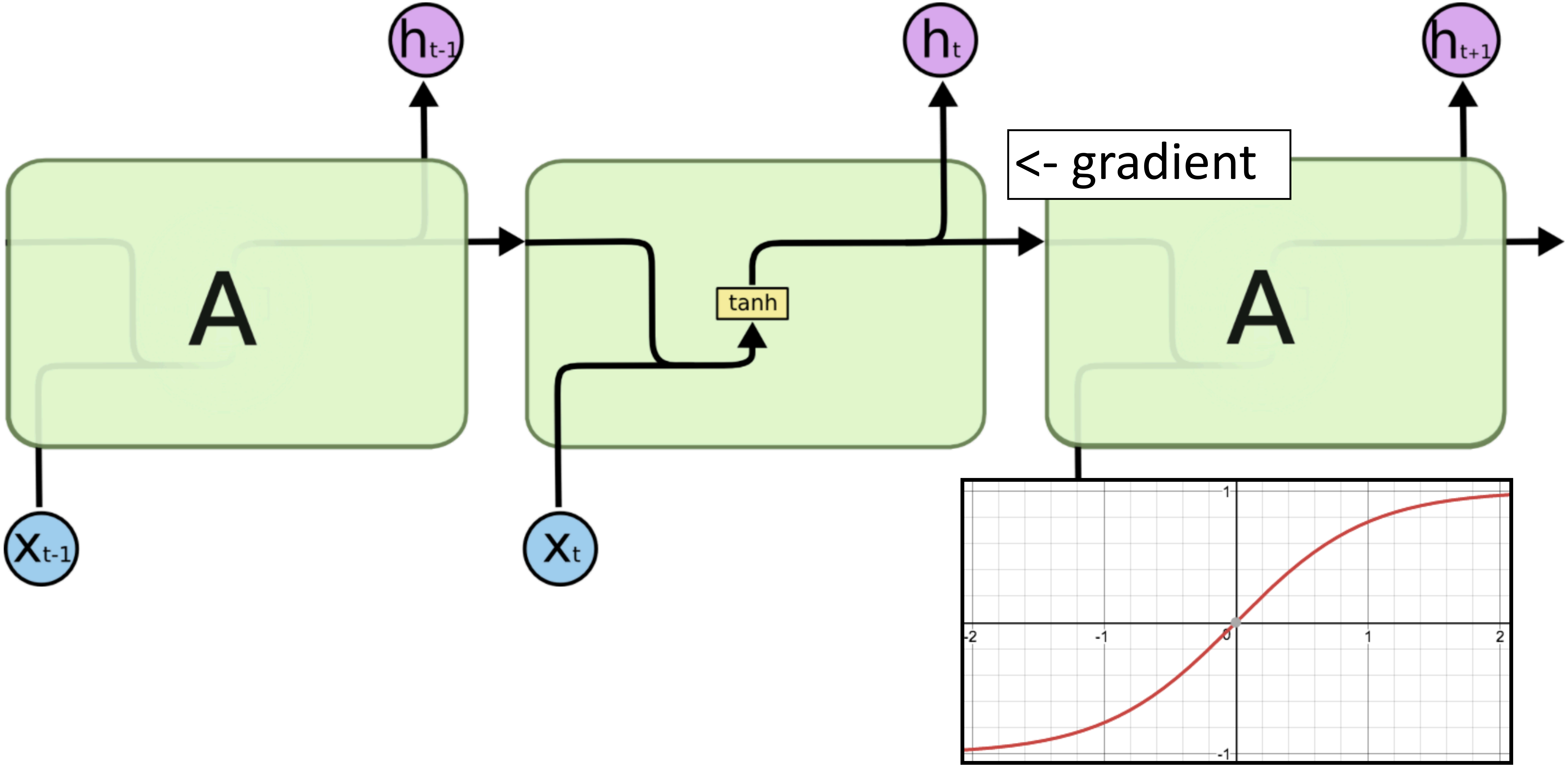
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient



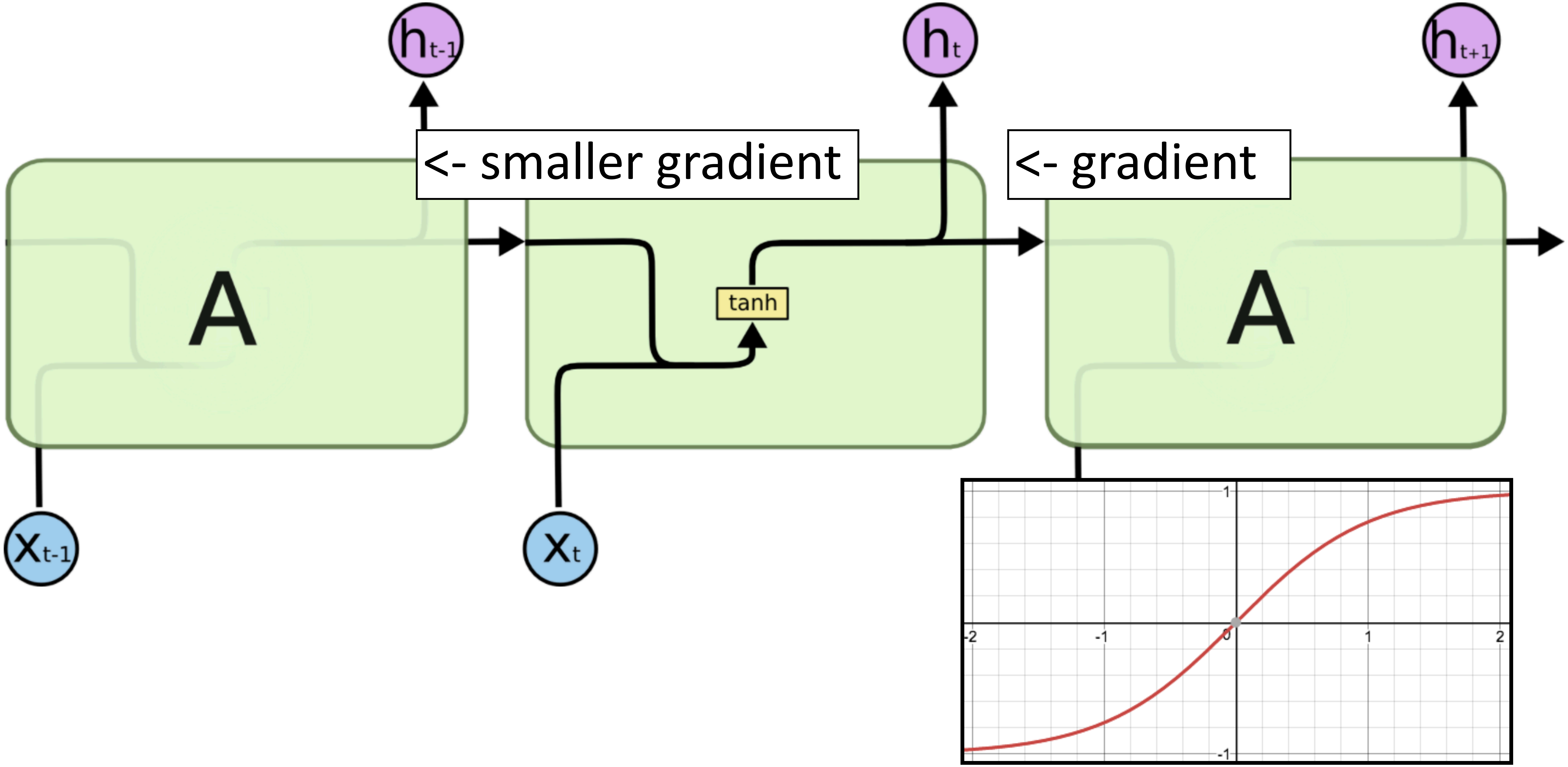
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient



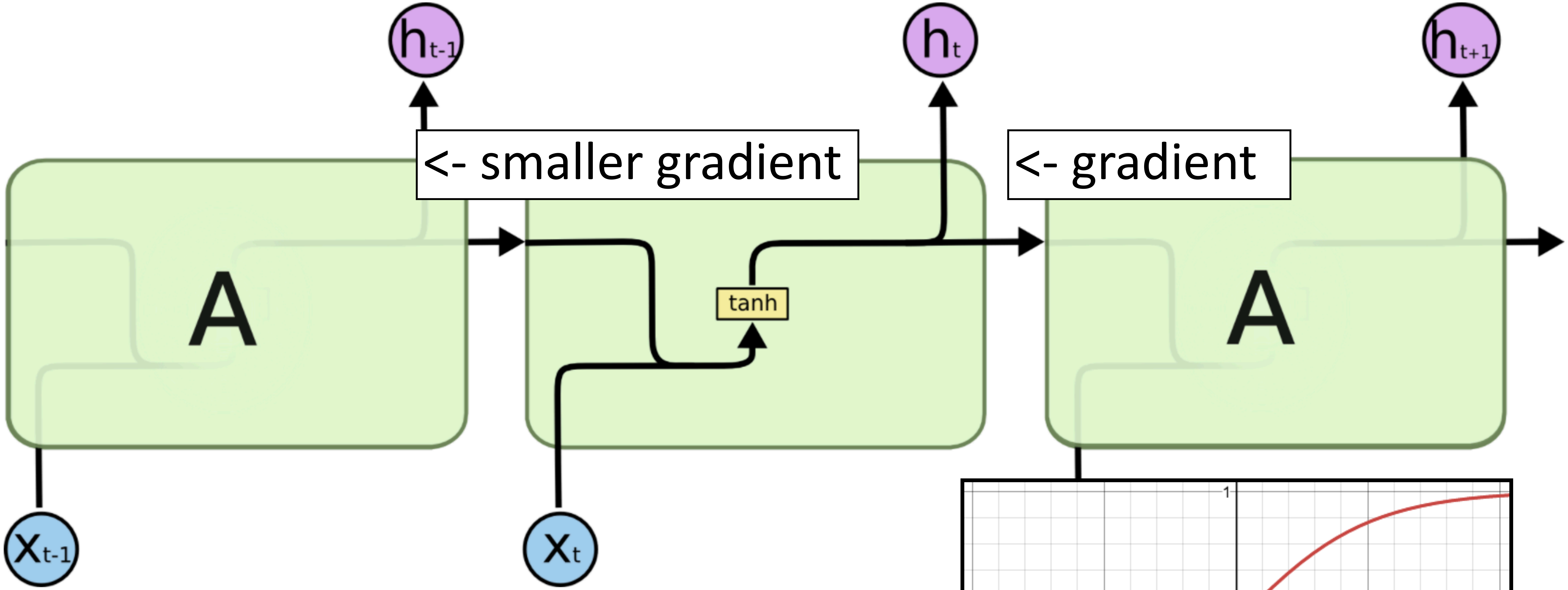
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient

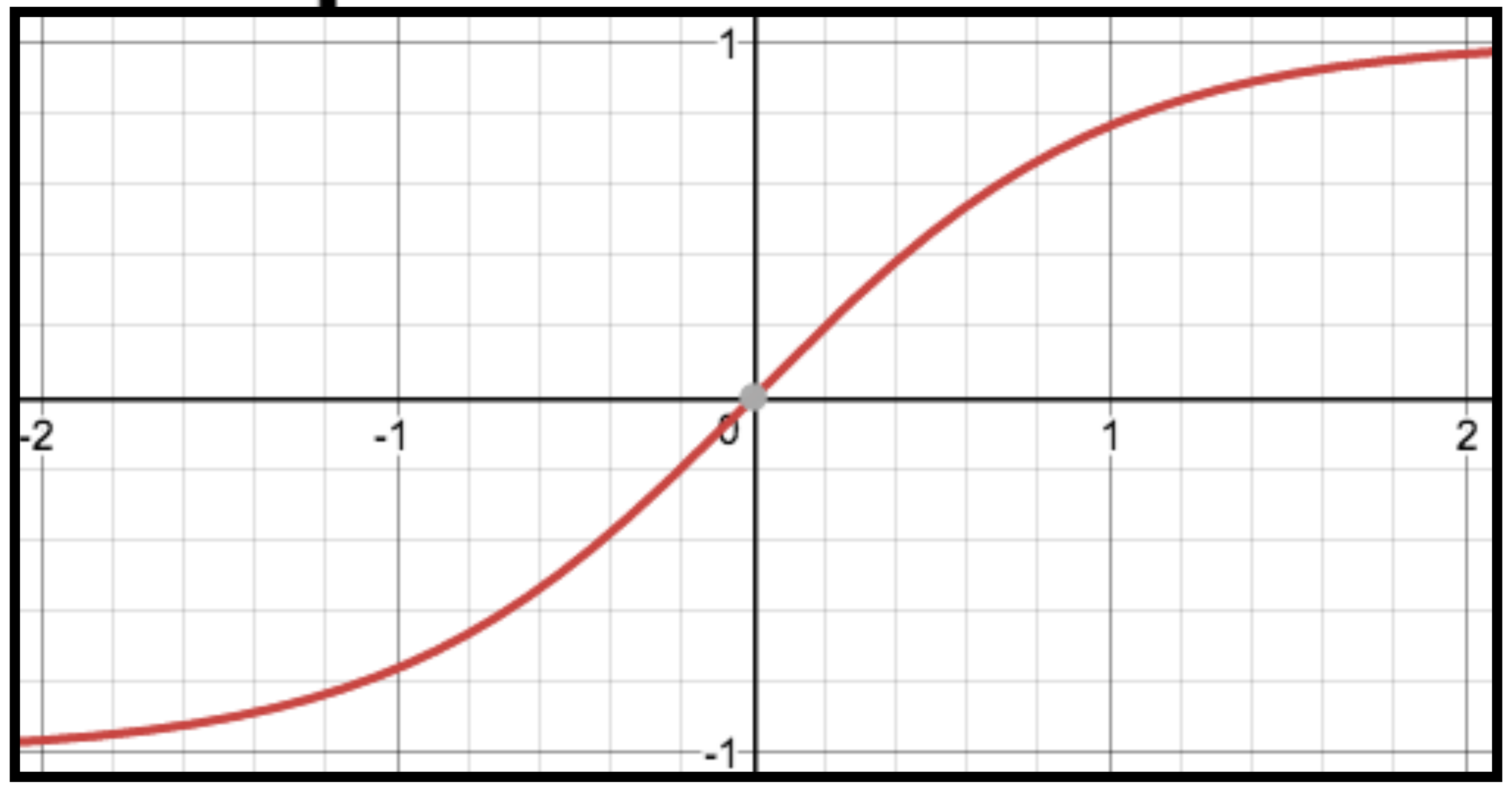


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient

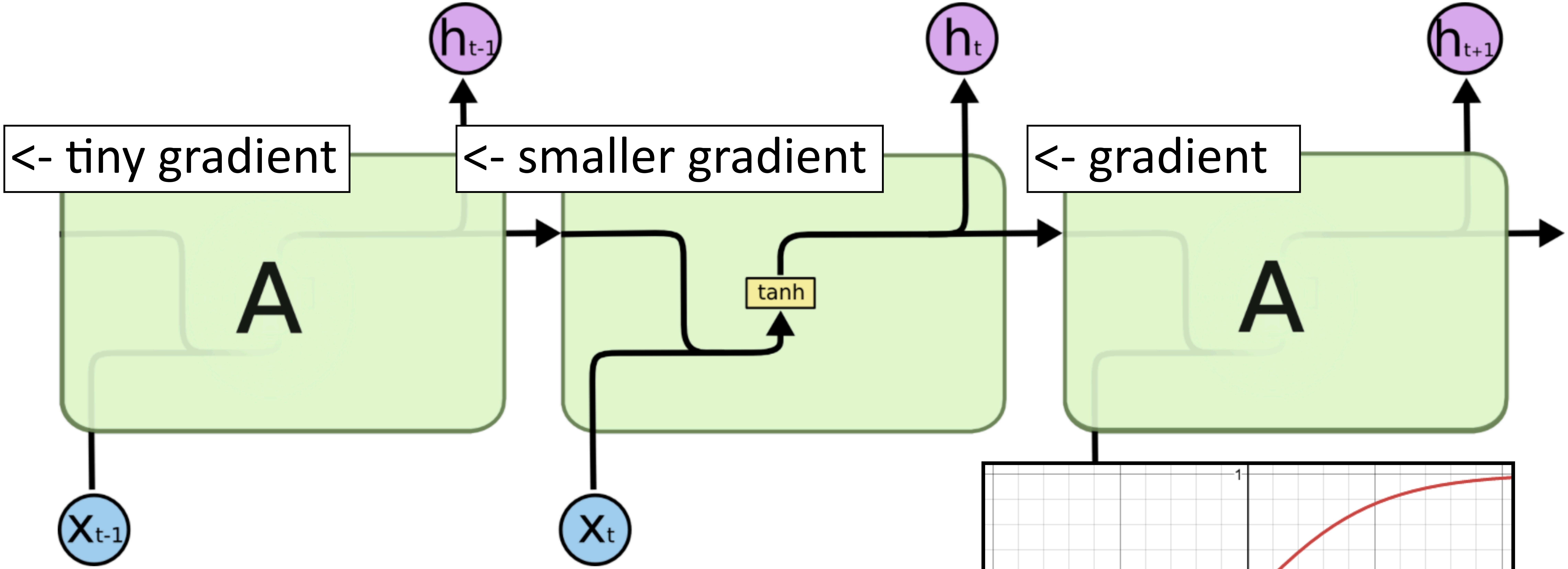


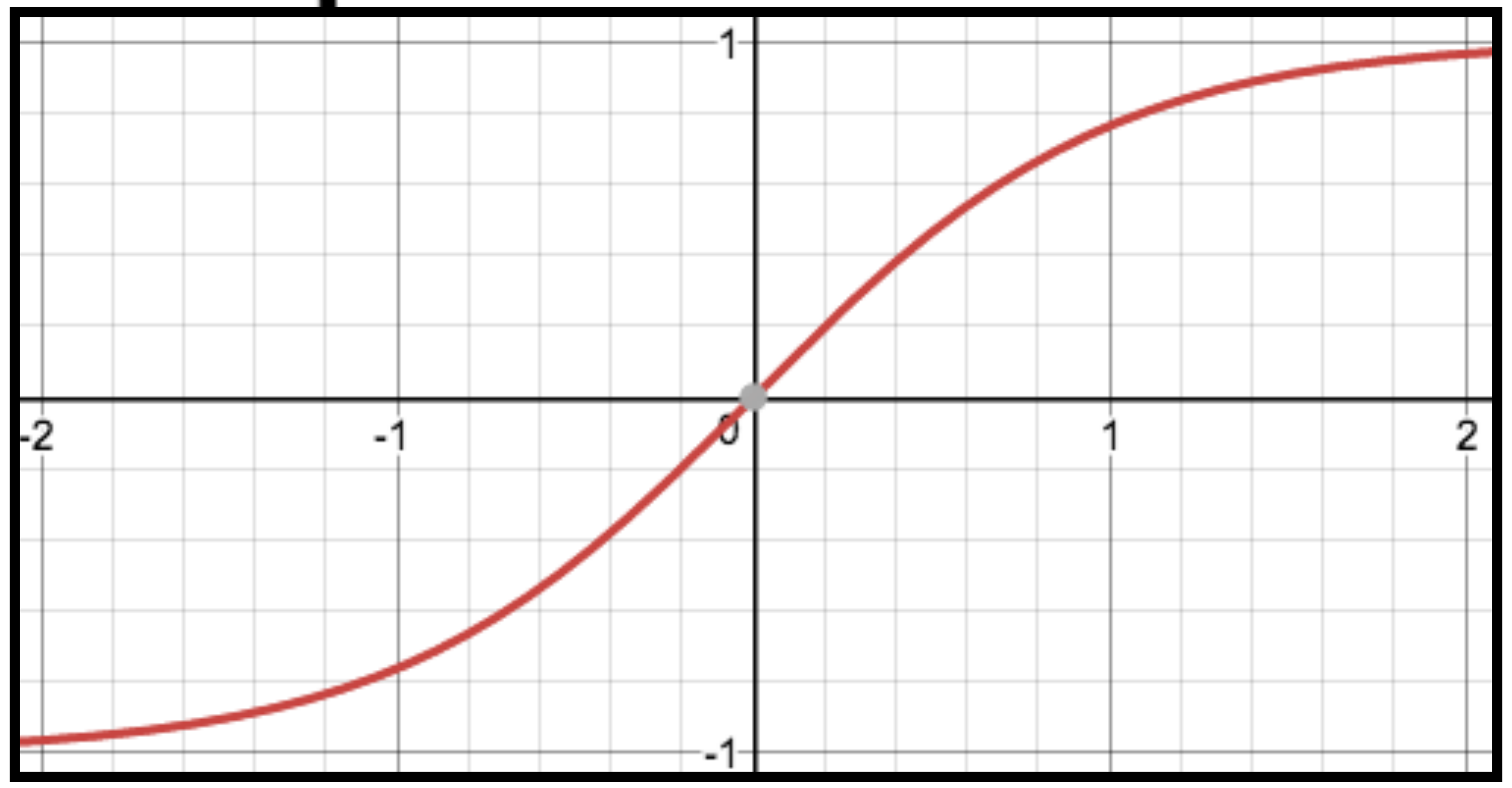
- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0



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

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*

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

gated

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

Elman

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)

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

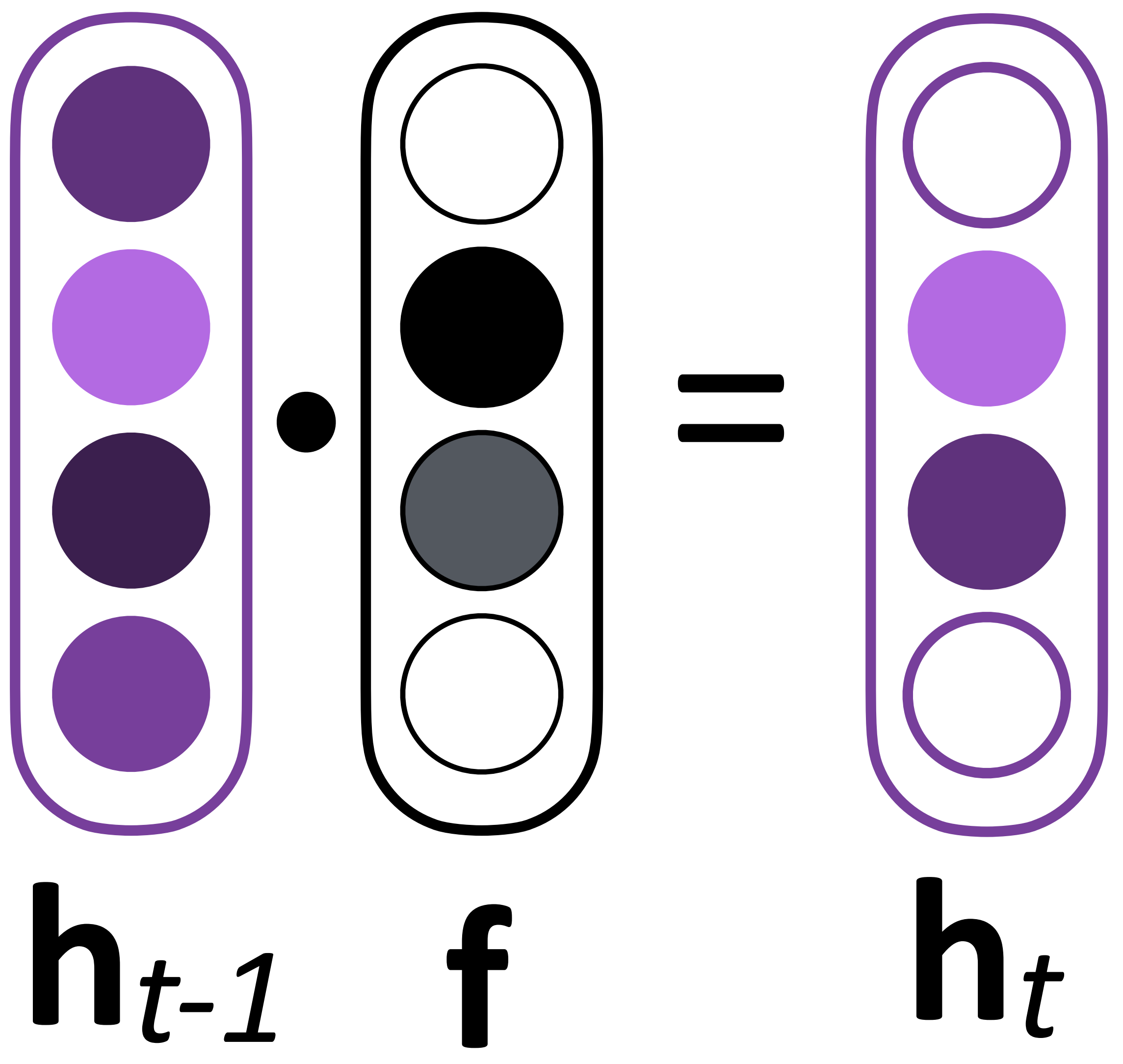
$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

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)



Gated Connections

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

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

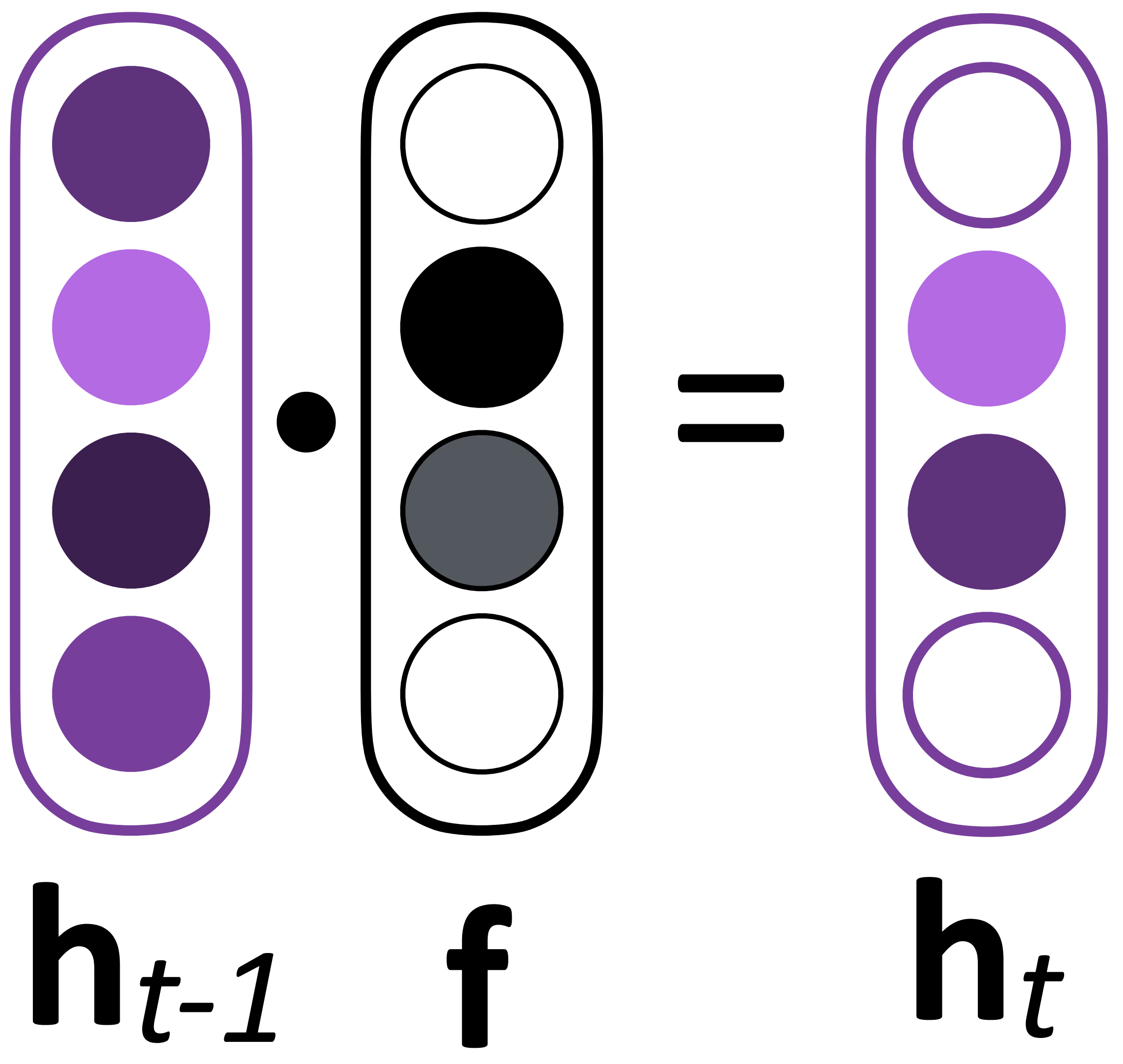
gated

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

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!

LSTMs

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

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

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

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

LSTMs

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

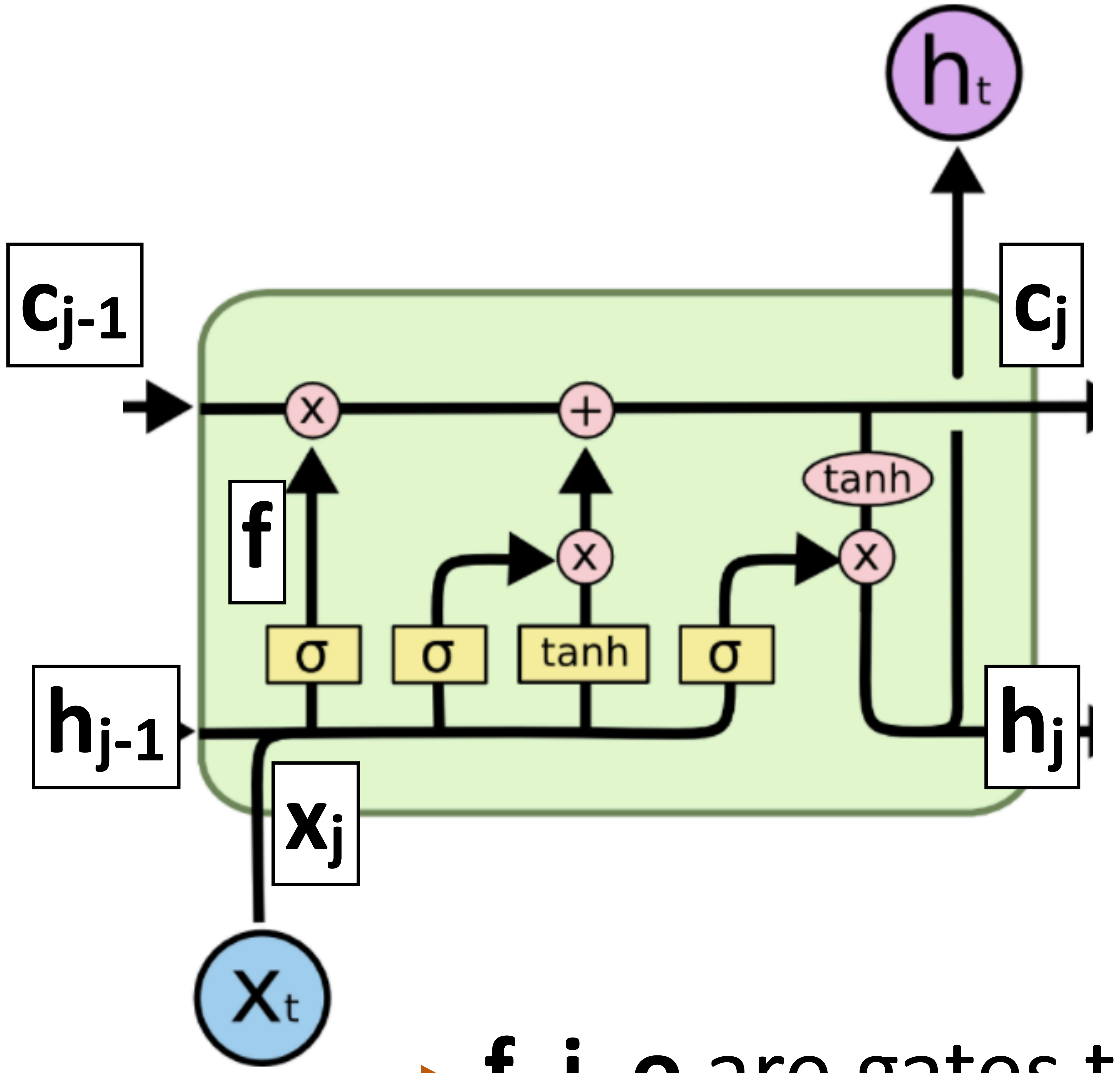
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f} + \boxed{\mathrm{func}(\mathbf{x}_t, \mathbf{h}_{t-1})}$$

- 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



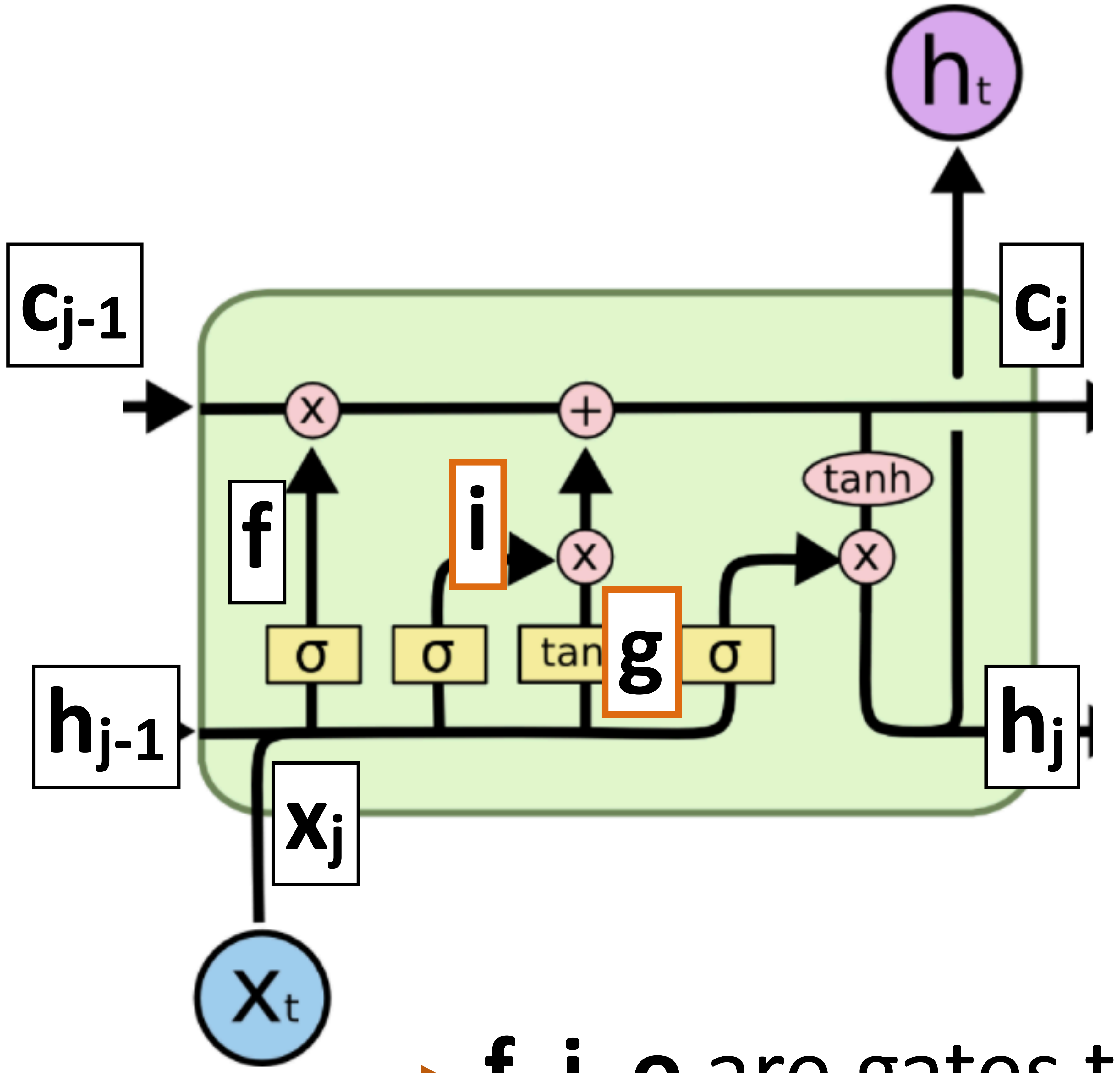
$$\mathbf{c_j} = \mathbf{c_{j-1}} \odot \mathbf{f} + \boxed{\mathbf{g} \odot \mathbf{i}}$$

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

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

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

LSTMs



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

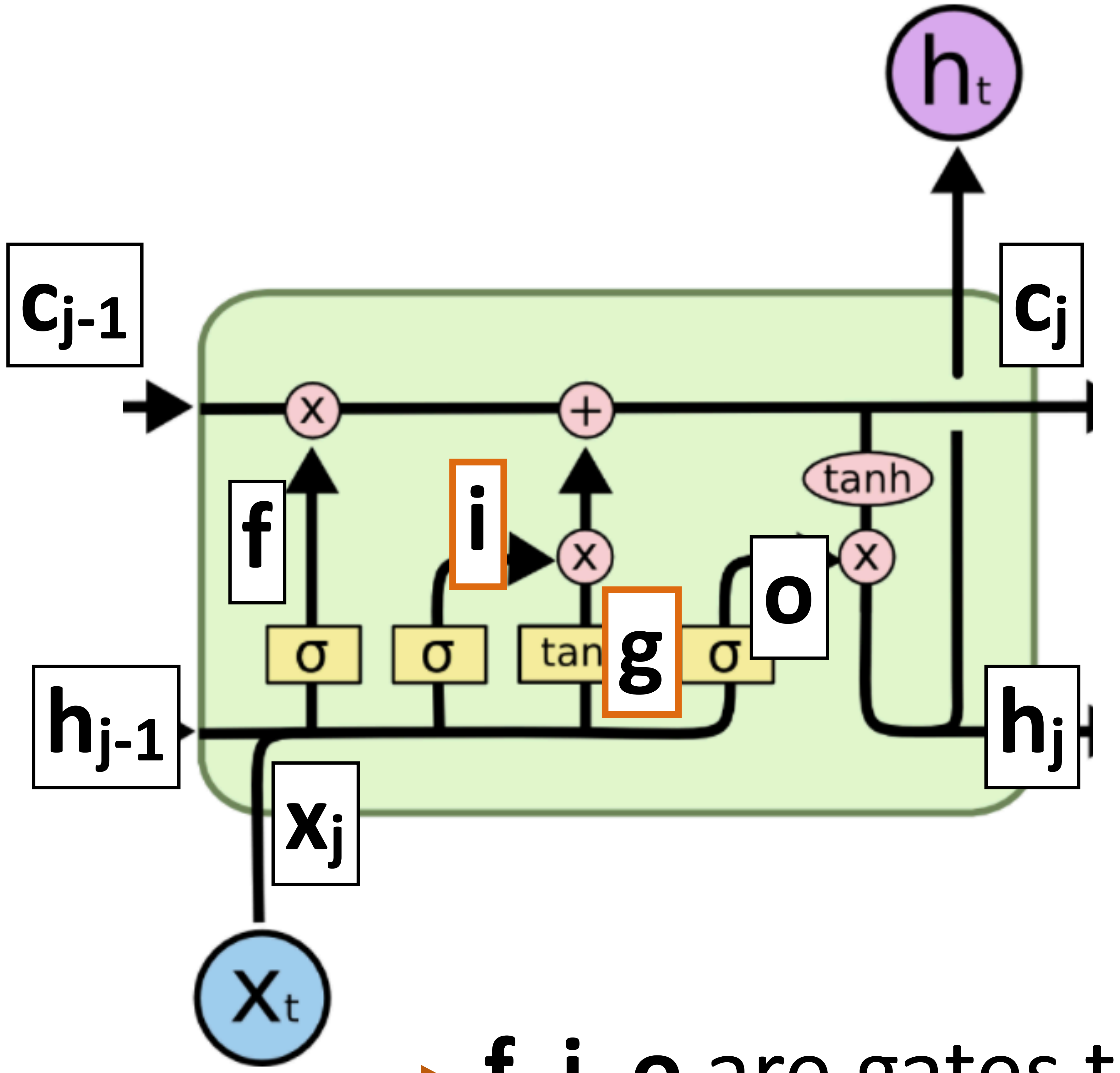
$$\mathbf{f} = \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}})$$

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

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

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

LSTMs



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

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

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

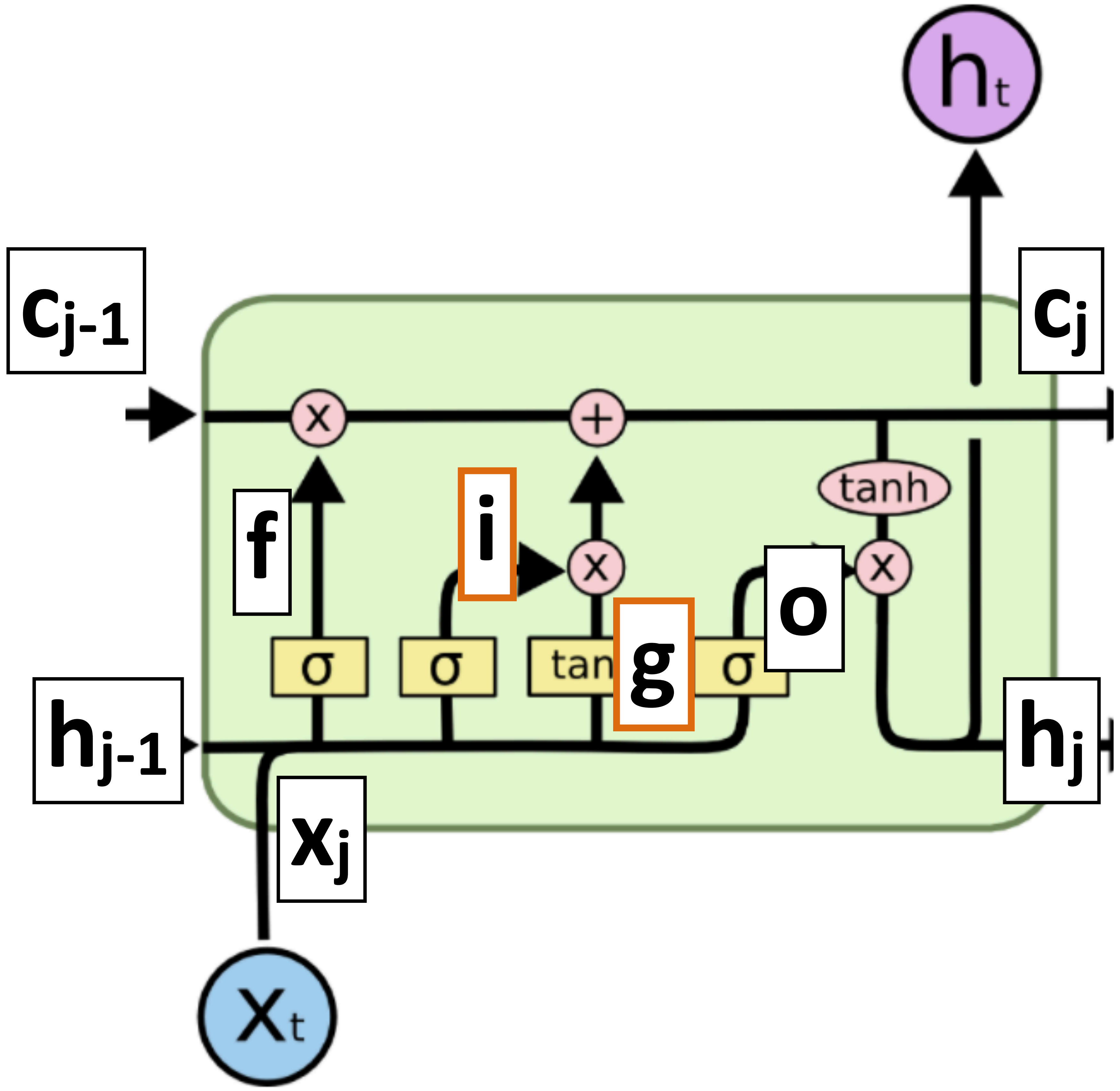
$$\mathbf{h_j} = \tanh(\mathbf{c_j}) \odot \mathbf{o}$$

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

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

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

LSTMs



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

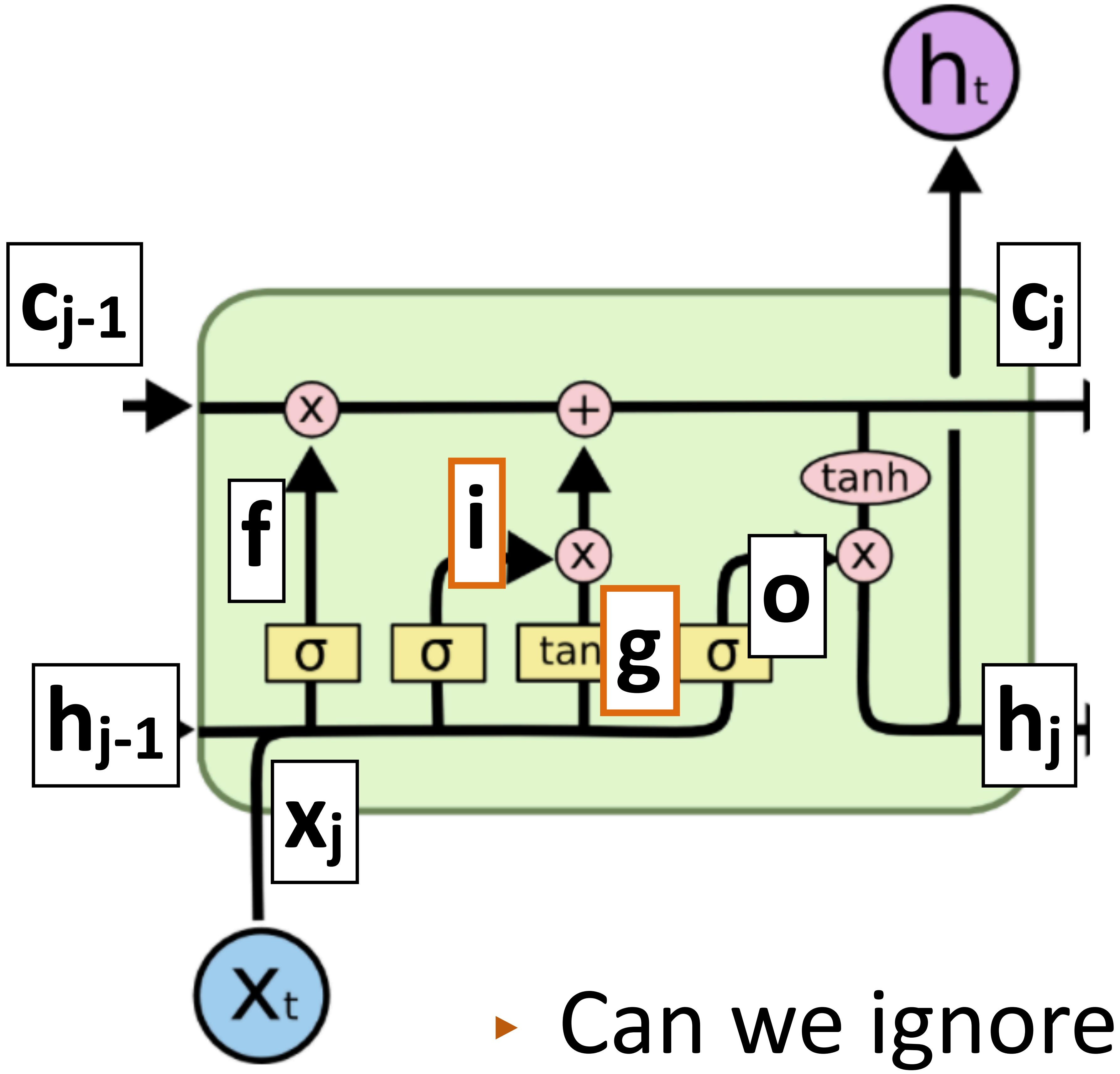
$$\mathbf{f} = \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}})$$

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

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

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

LSTMs



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

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

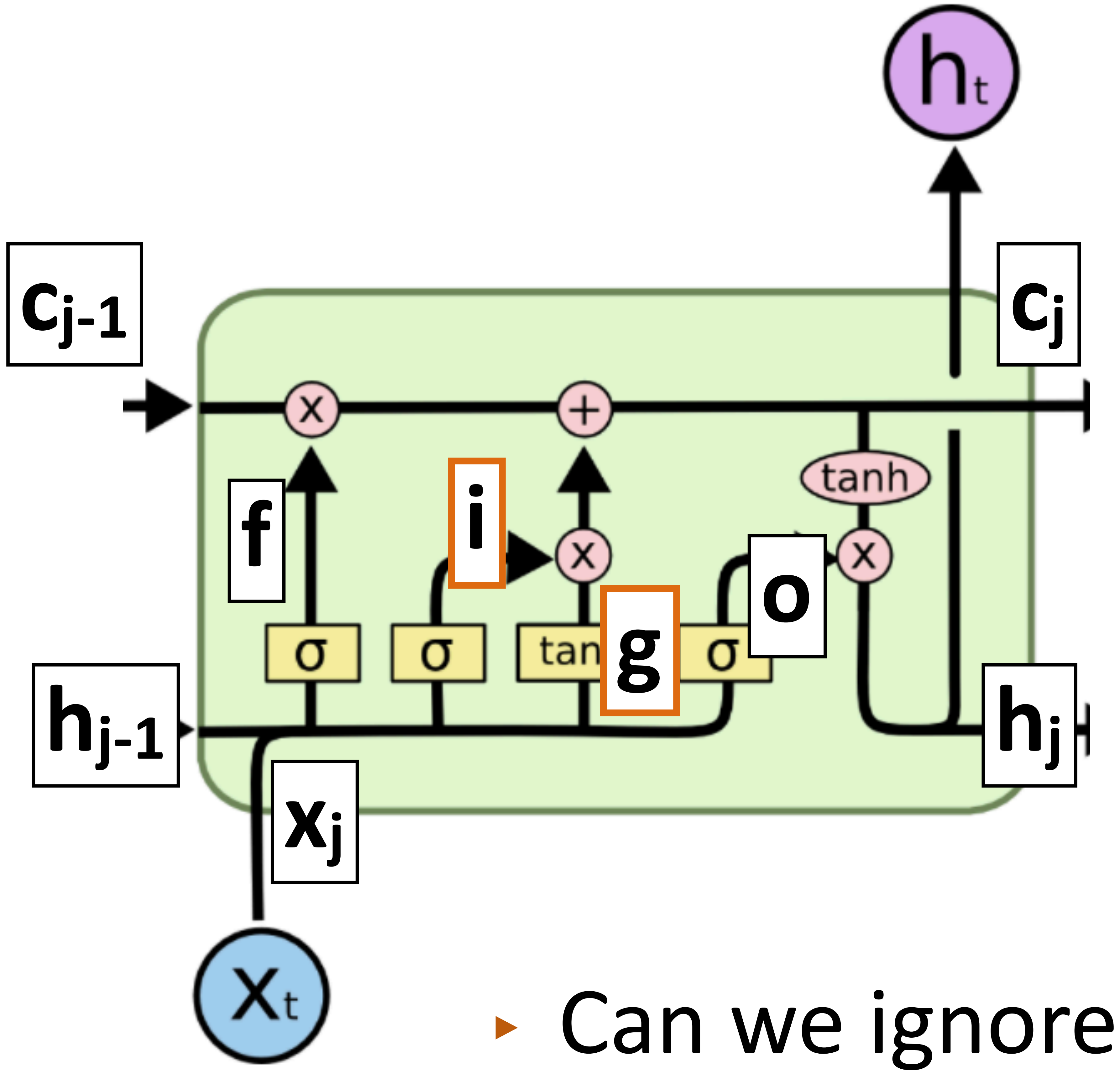
$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

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

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

- Can we ignore the old value of **c** for this timestep?

LSTMs



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

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

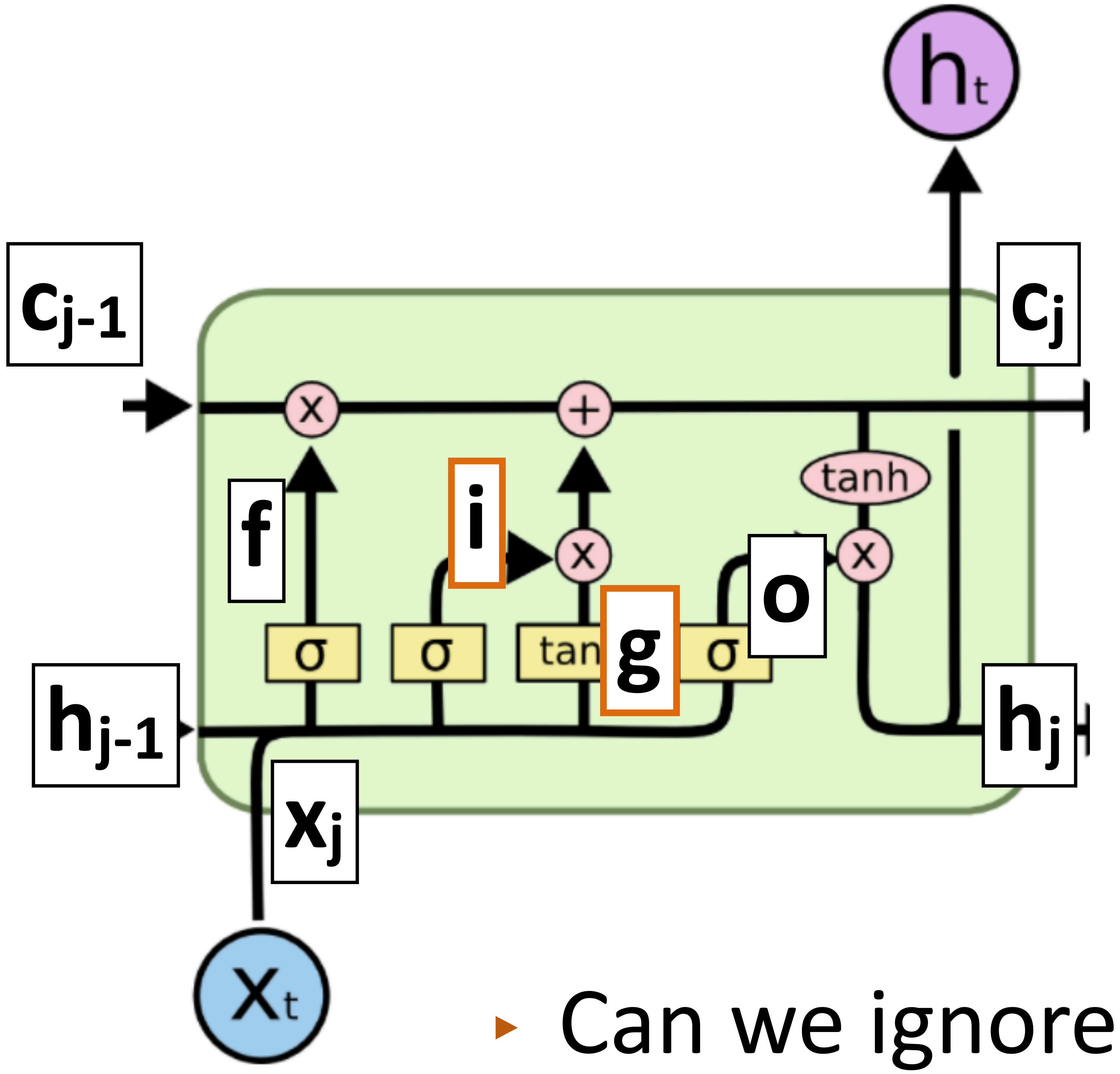
$$\boxed{\begin{aligned} \mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\ \mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \end{aligned}}$$

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

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

- Can we ignore the old value of **c** for this timestep?
- Can an LSTM sum up its inputs **x**?

LSTMs



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

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

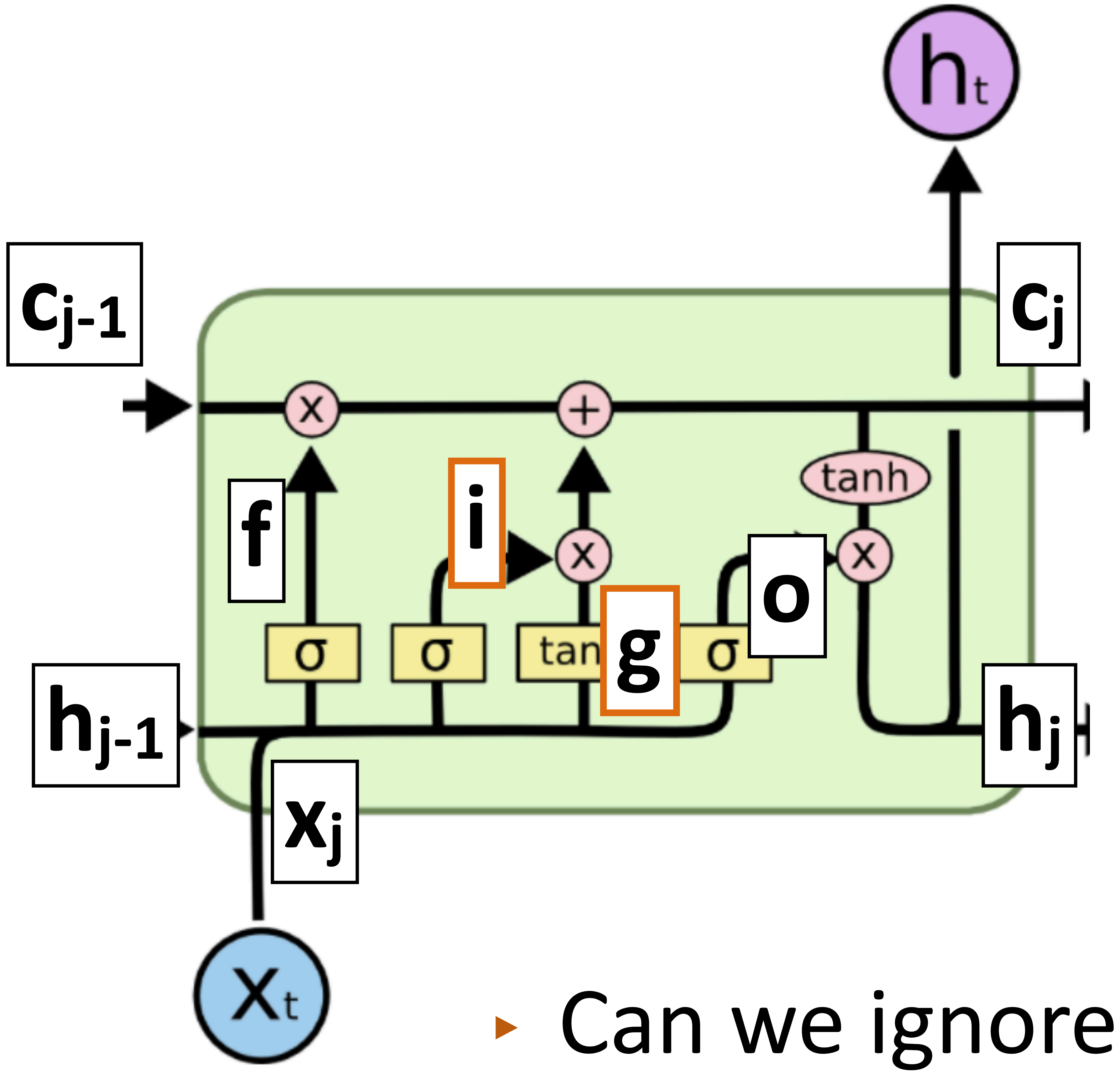
$$\boxed{\begin{aligned} g &= \tanh(x_j W^{xg} + h_{j-1} W^{hg}) \\ i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \end{aligned}}$$

$$h_j = \tanh(c_j) \odot o$$

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

- 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**?

LSTMs



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

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

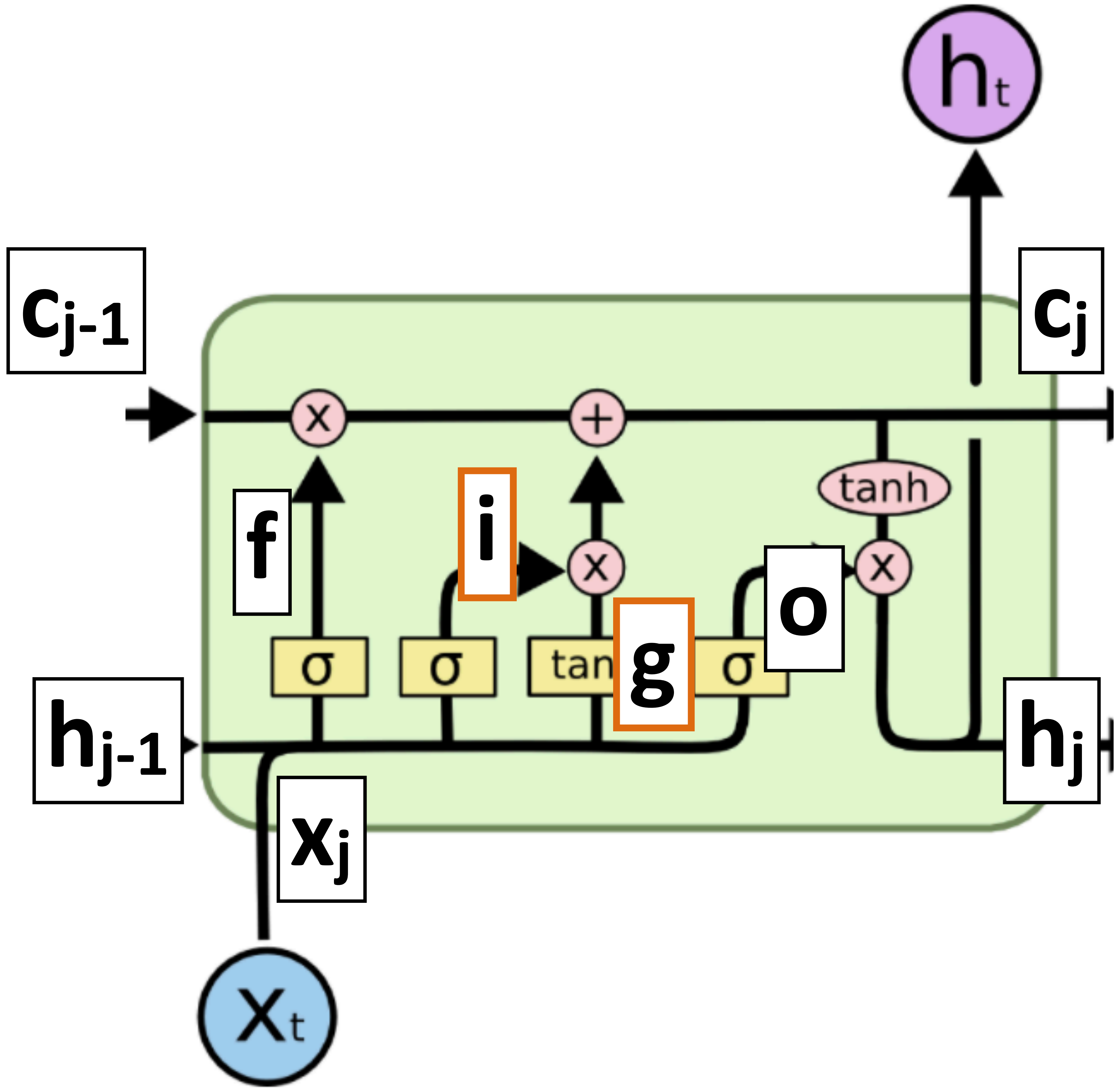
$$\boxed{\begin{aligned} g &= \tanh(x_j W^{xg} + h_{j-1} W^{hg}) \\ i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \end{aligned}}$$

$$h_j = \tanh(c_j) \odot o$$

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

- 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**?

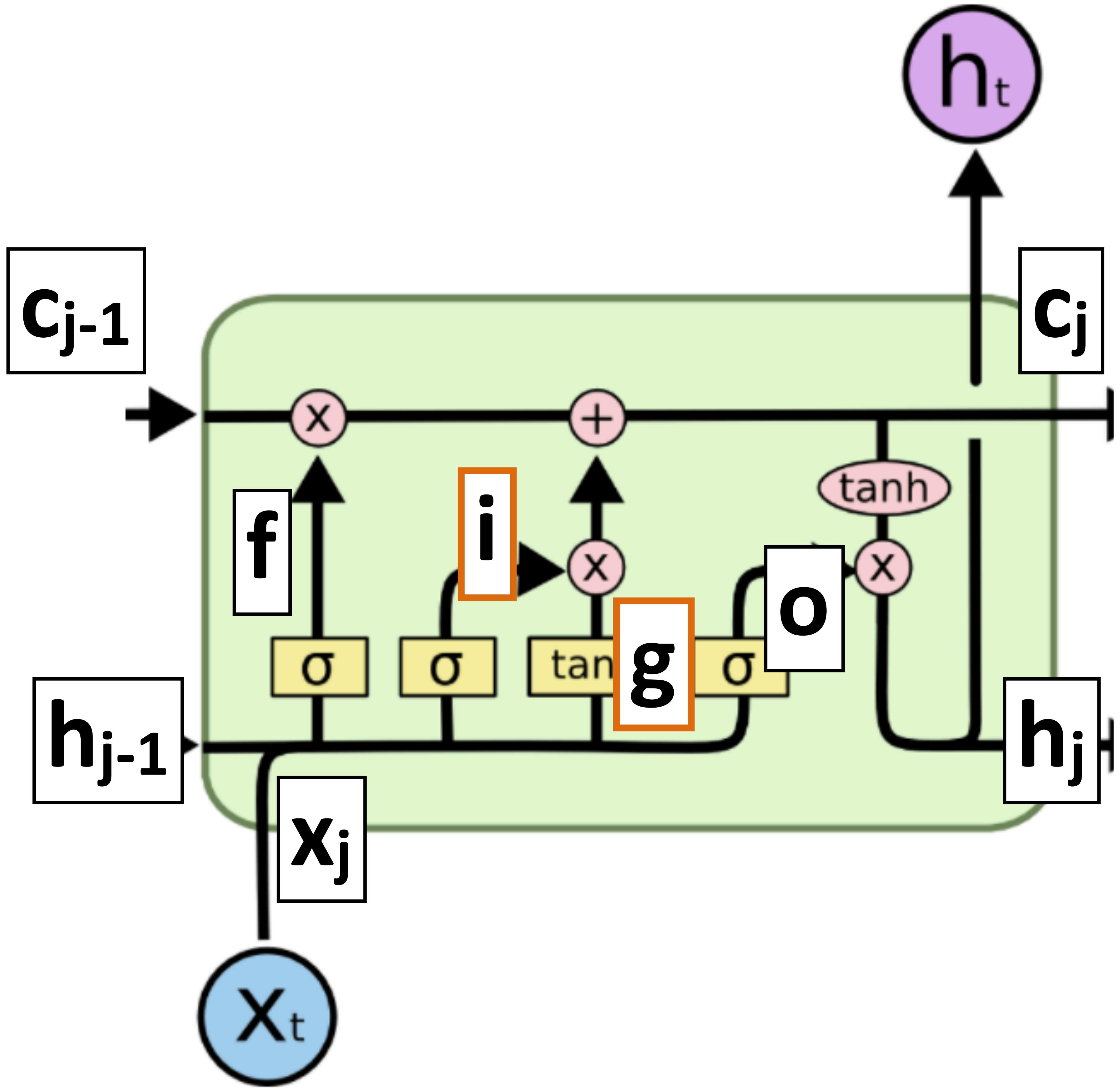
LSTMs



- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token

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

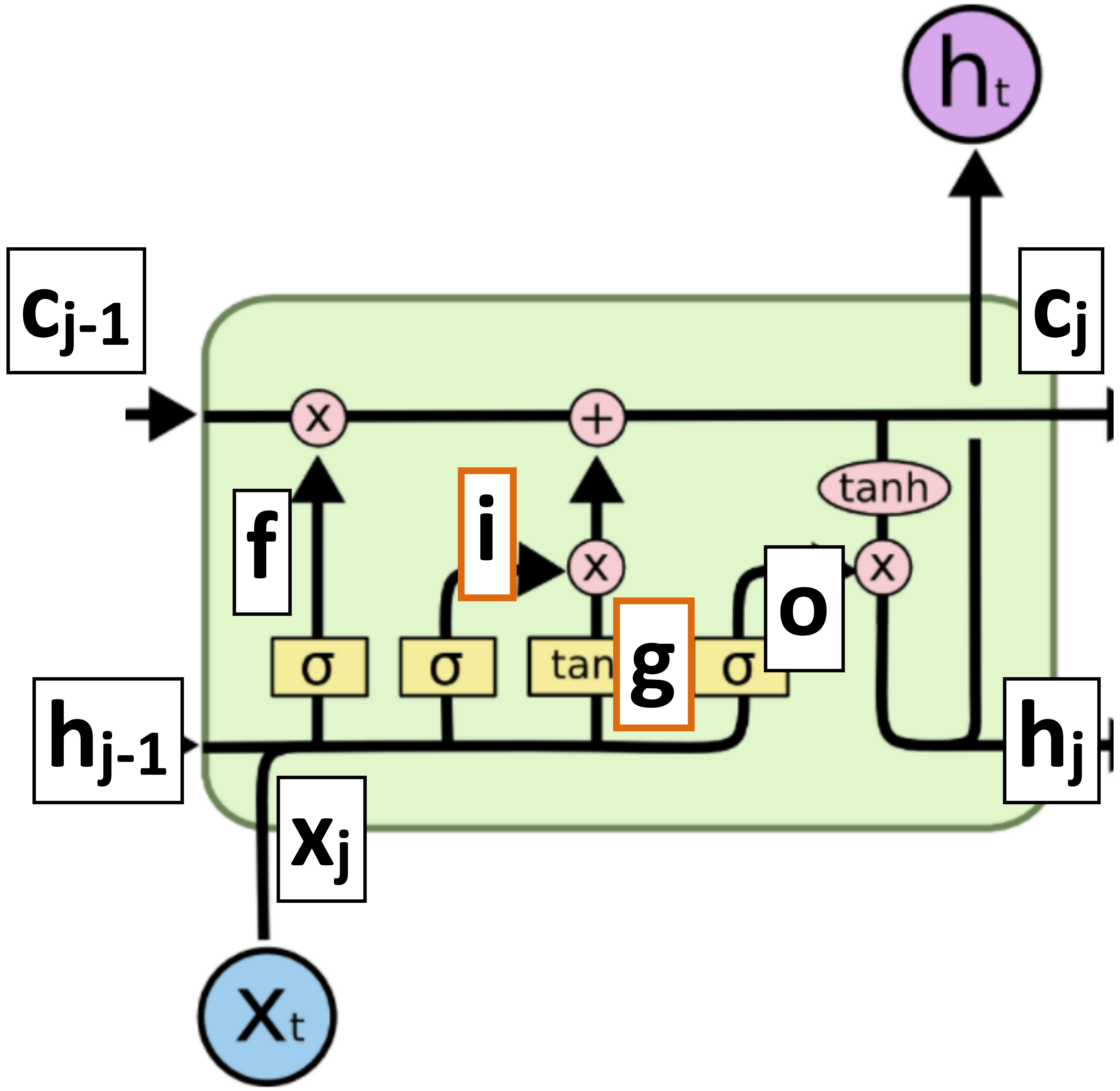
LSTMs



- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords

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

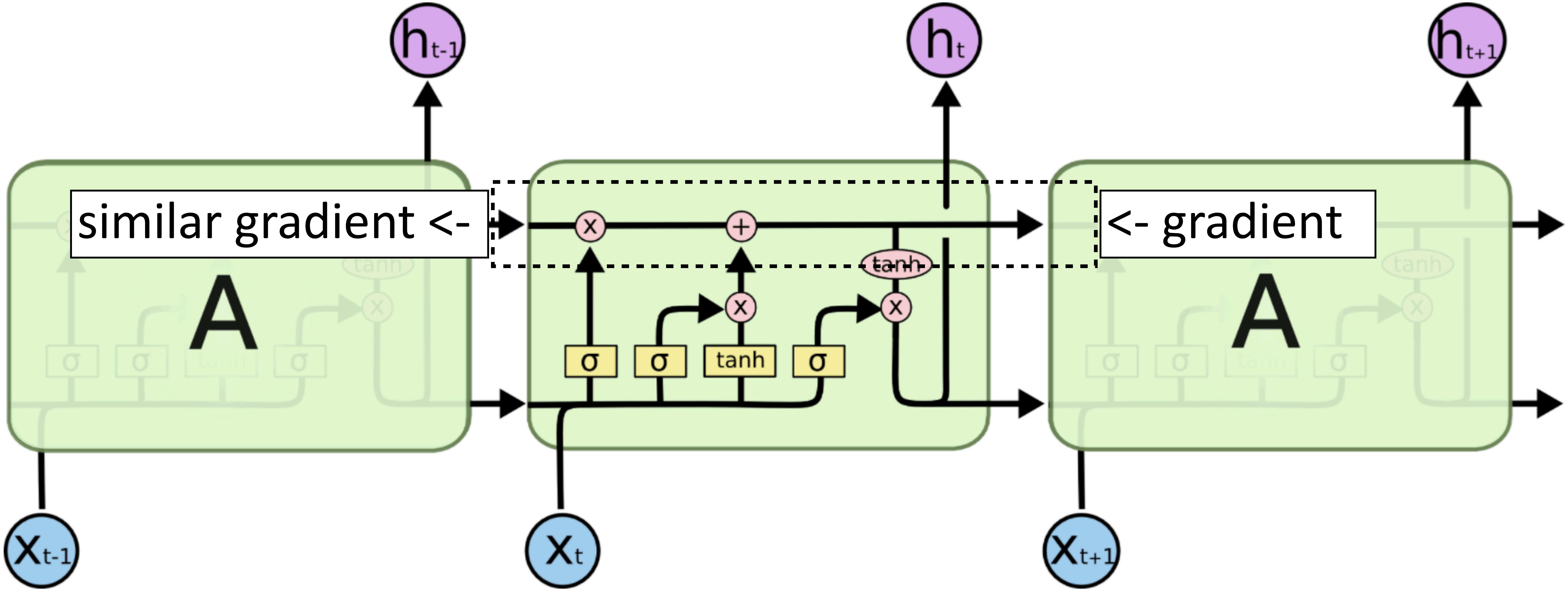
LSTMs



- 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

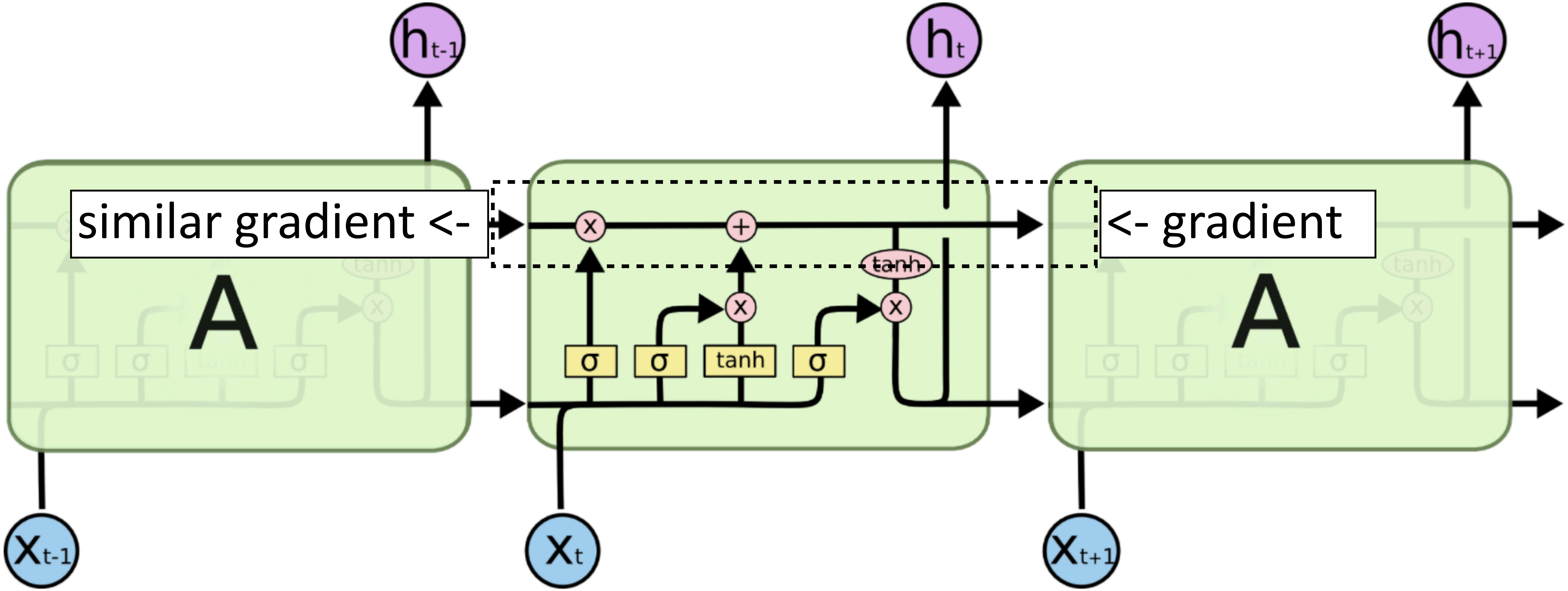
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs



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

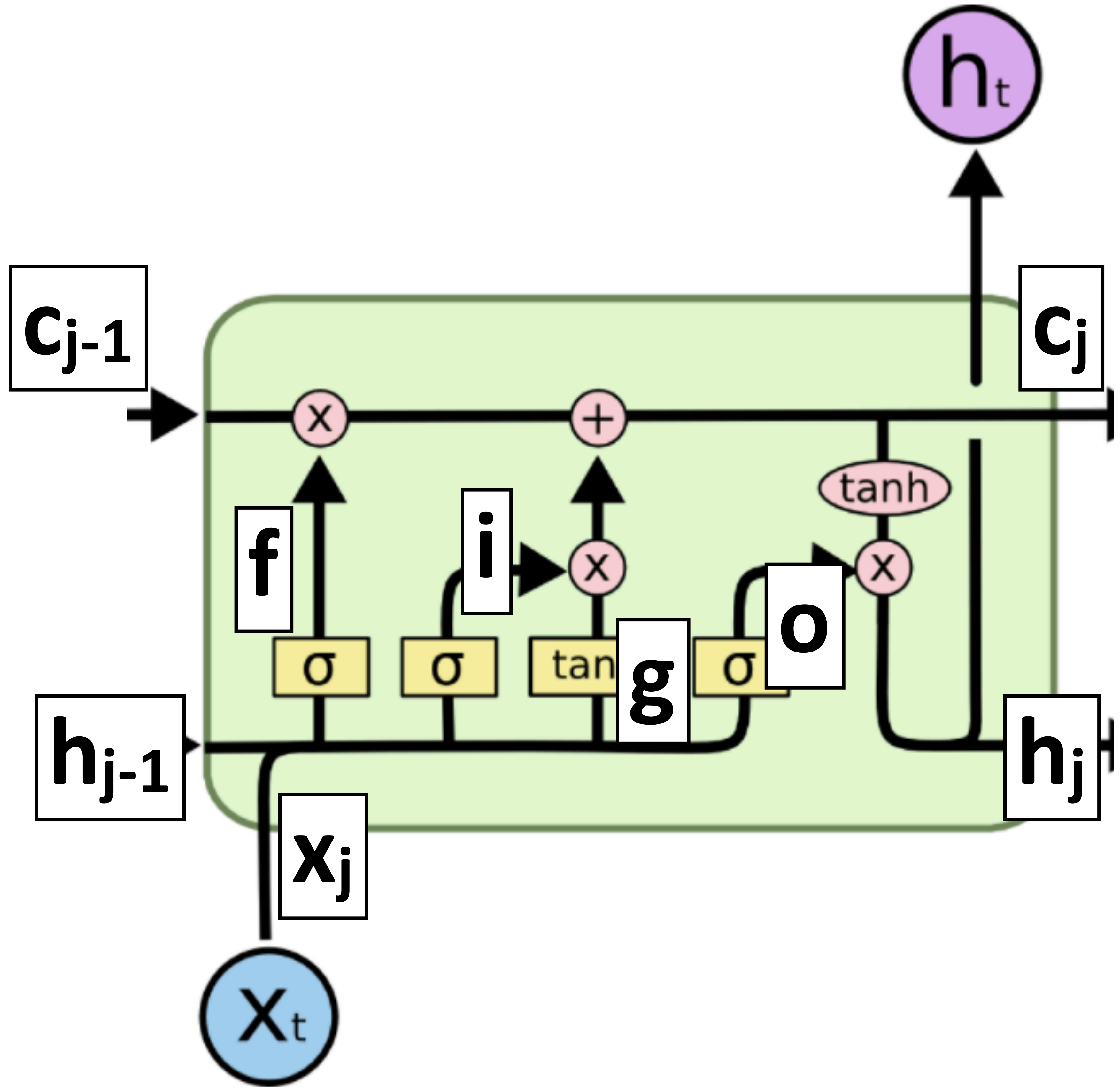
LSTMs

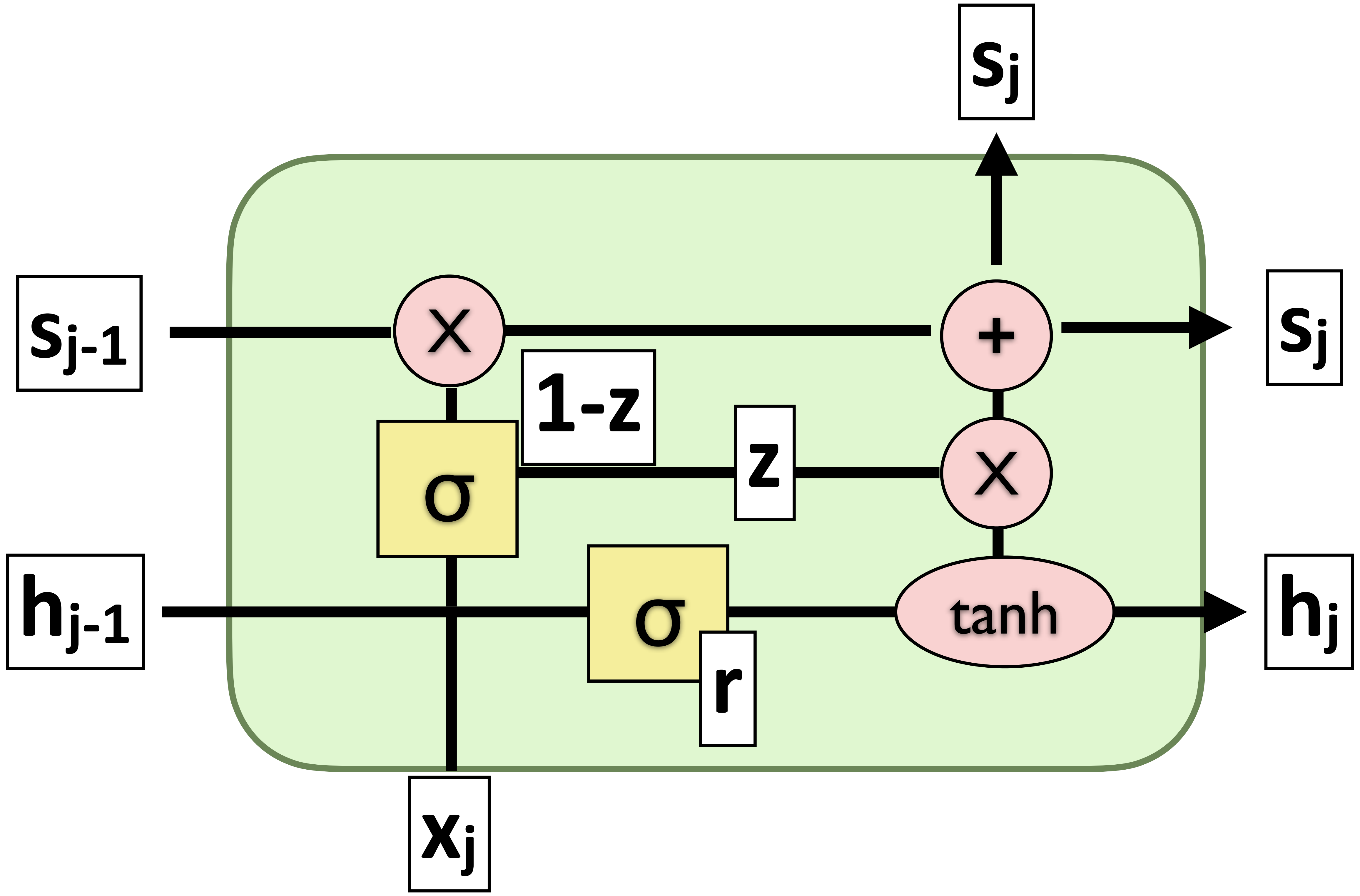


- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

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

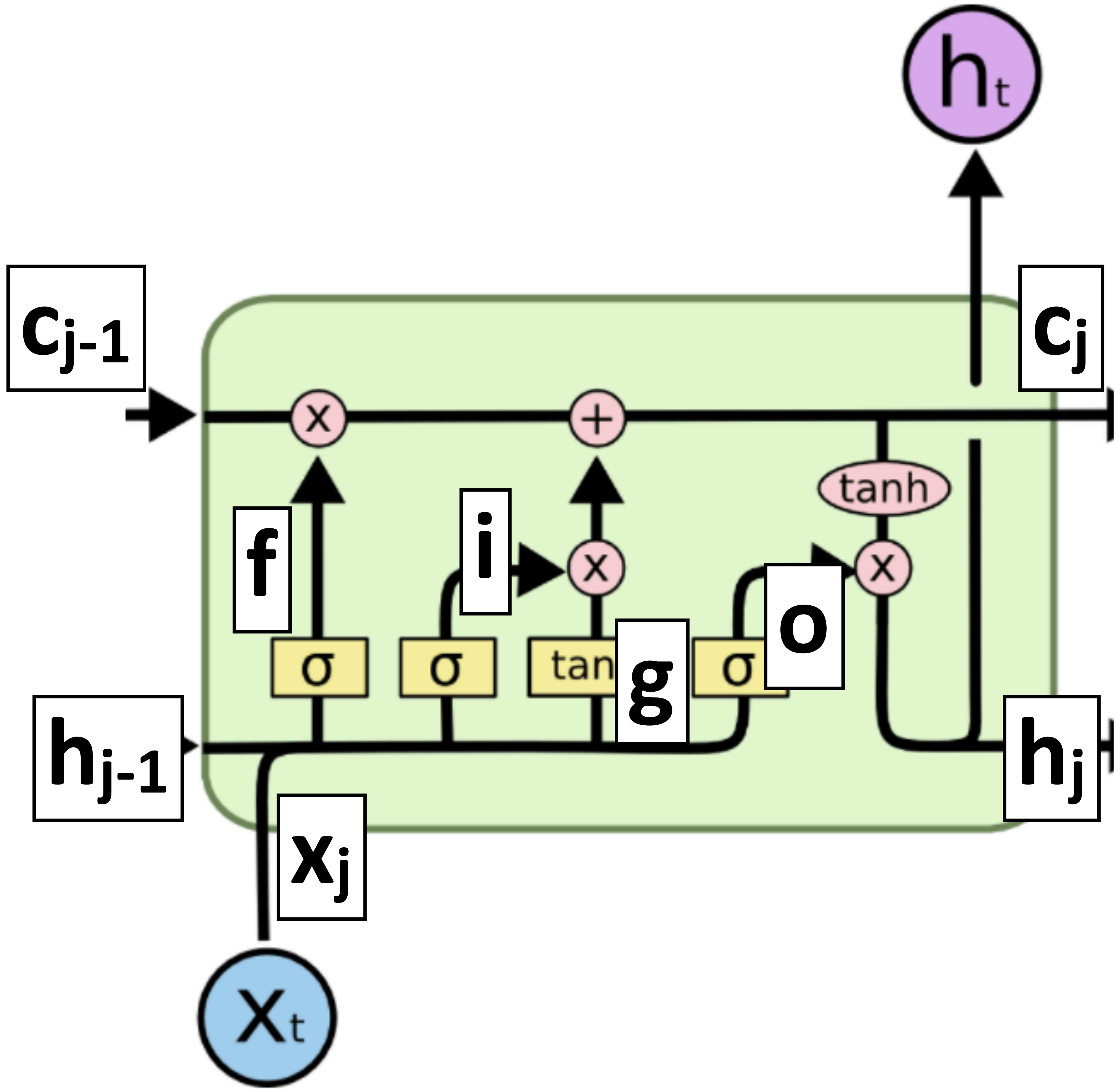
GRUs

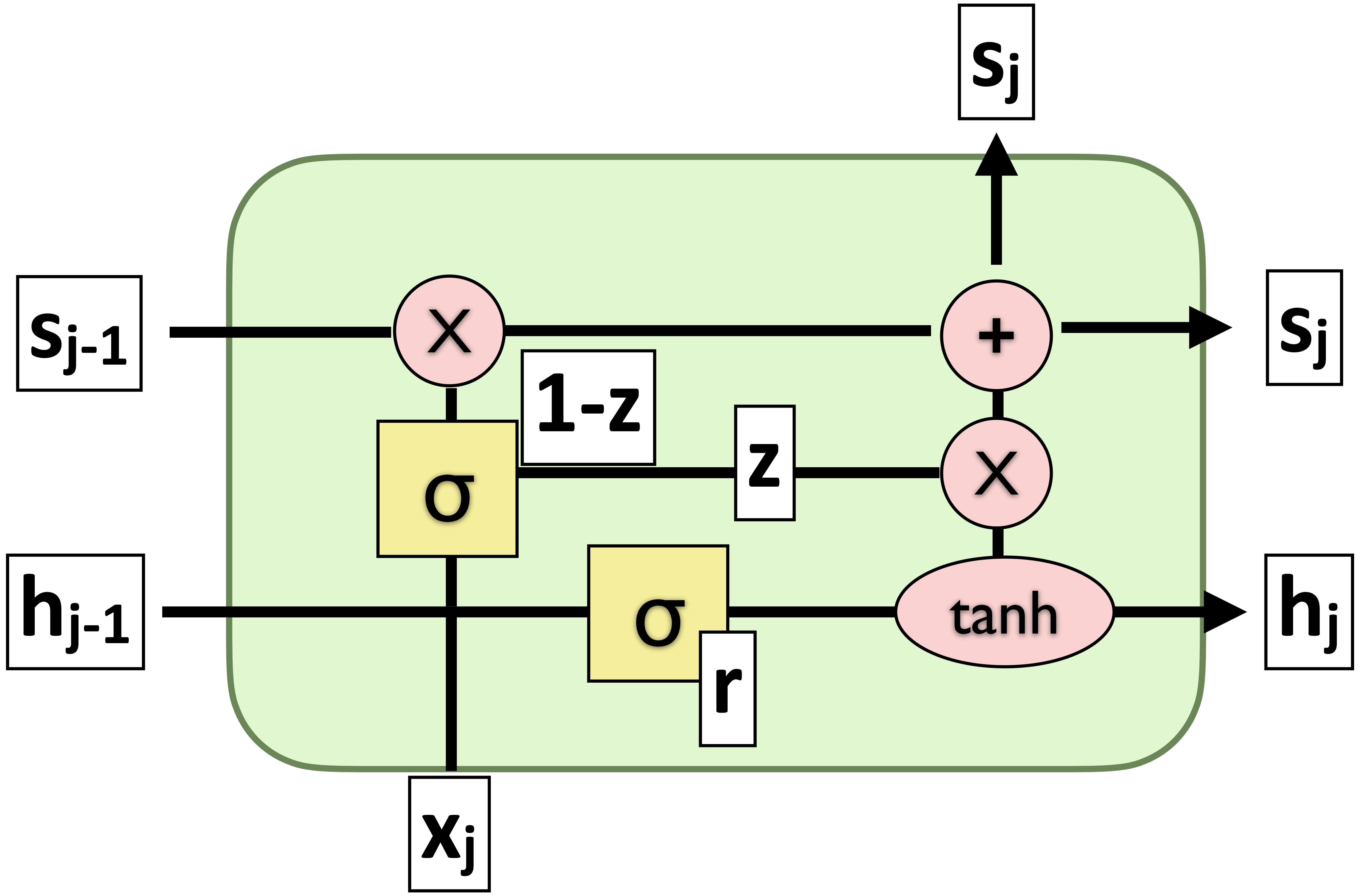




- LSTM: more complex and slower, may work a bit better

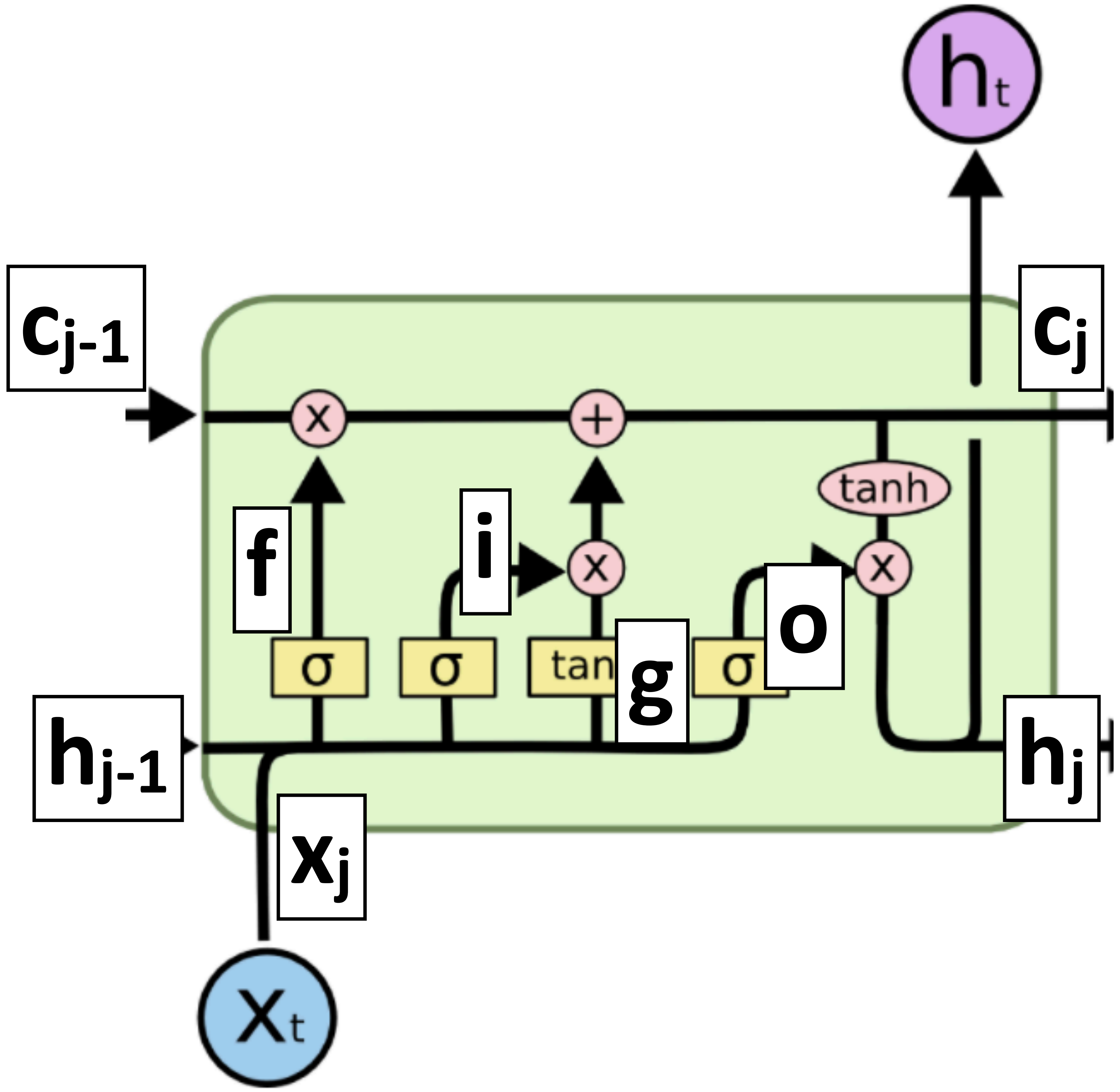
GRUs

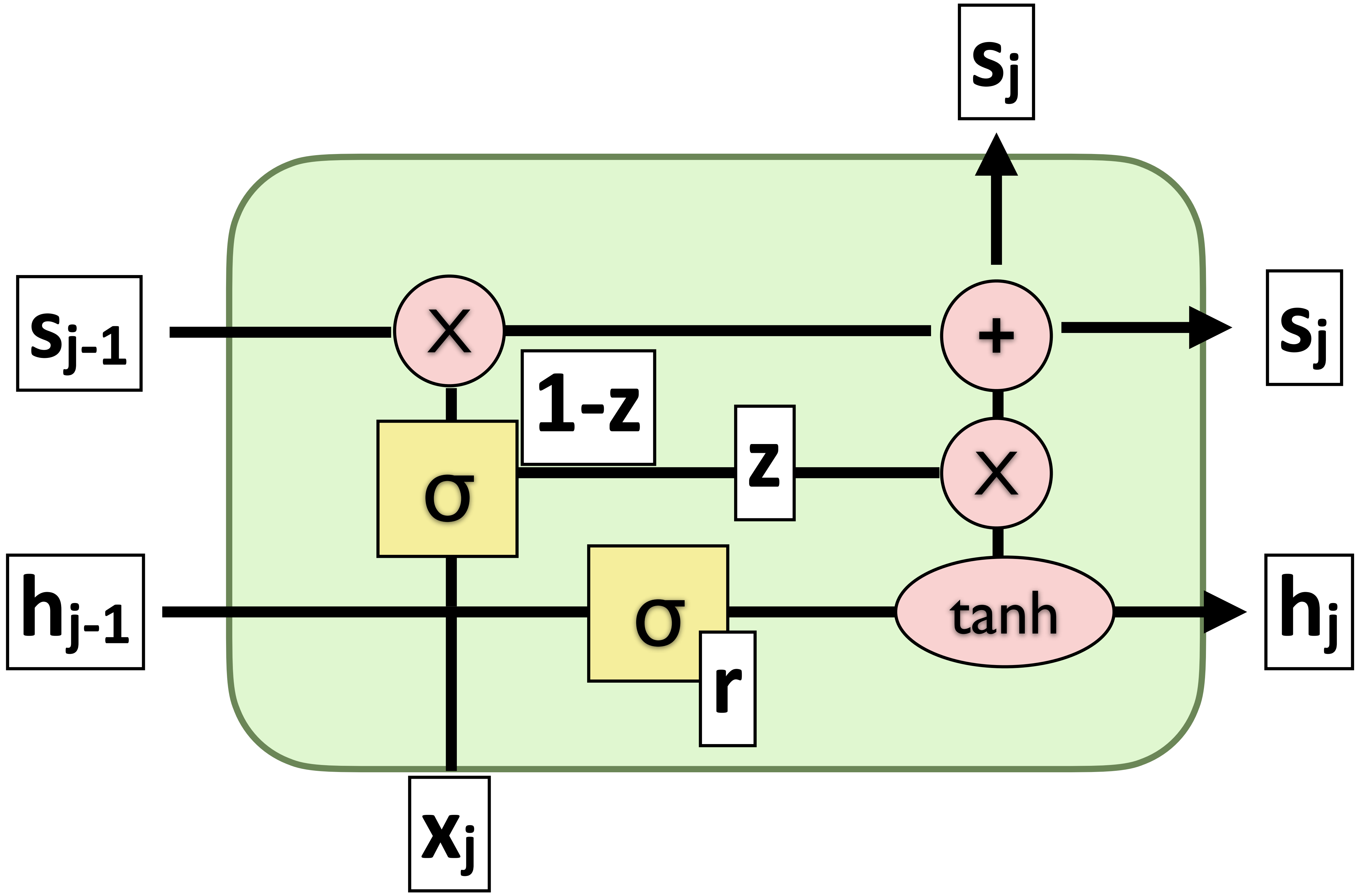




- LSTM: more complex and slower, may work a bit better
- GRU: faster, a bit simpler

GRUs

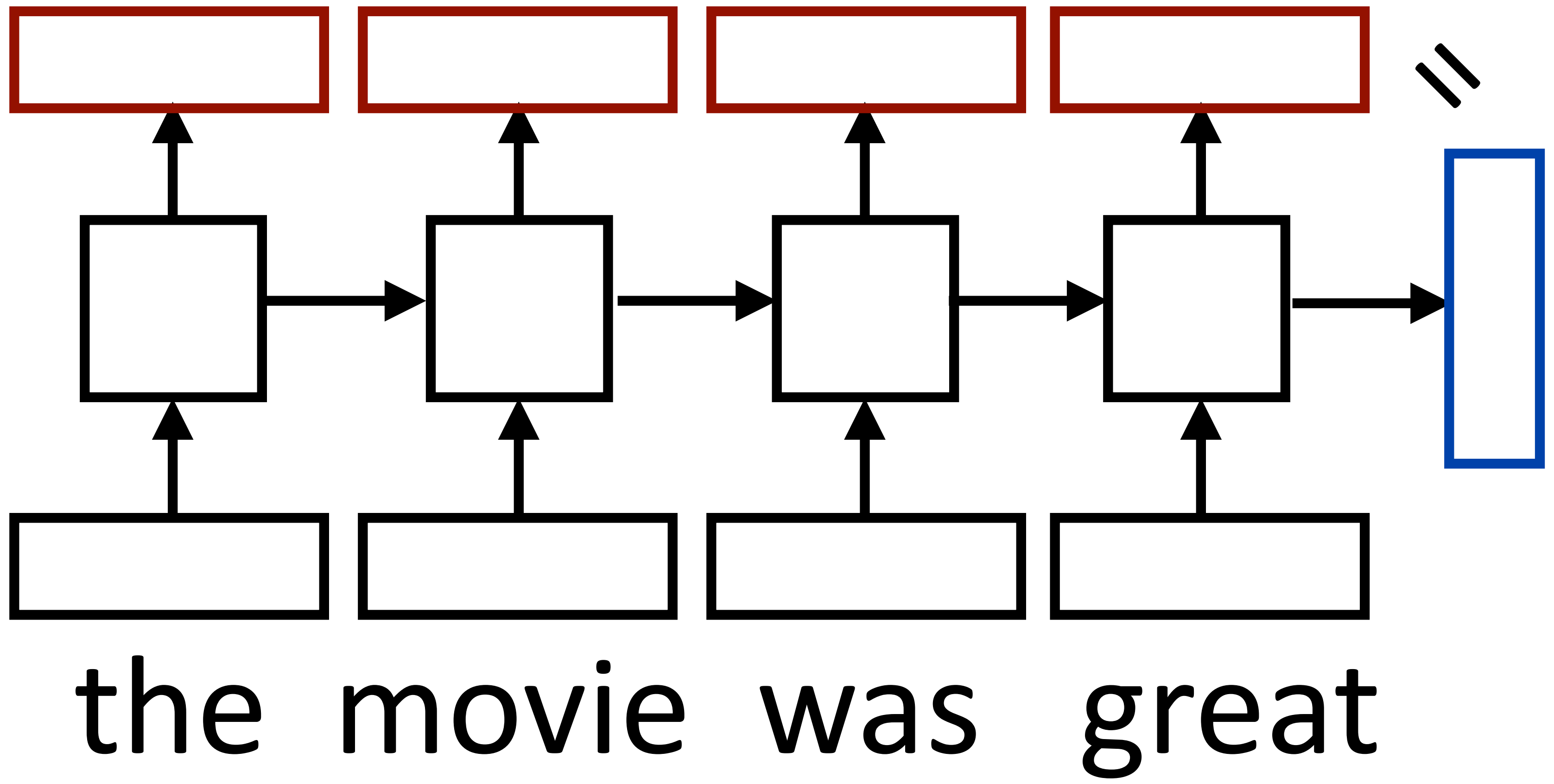




- LSTM: more complex and slower, may work a bit better

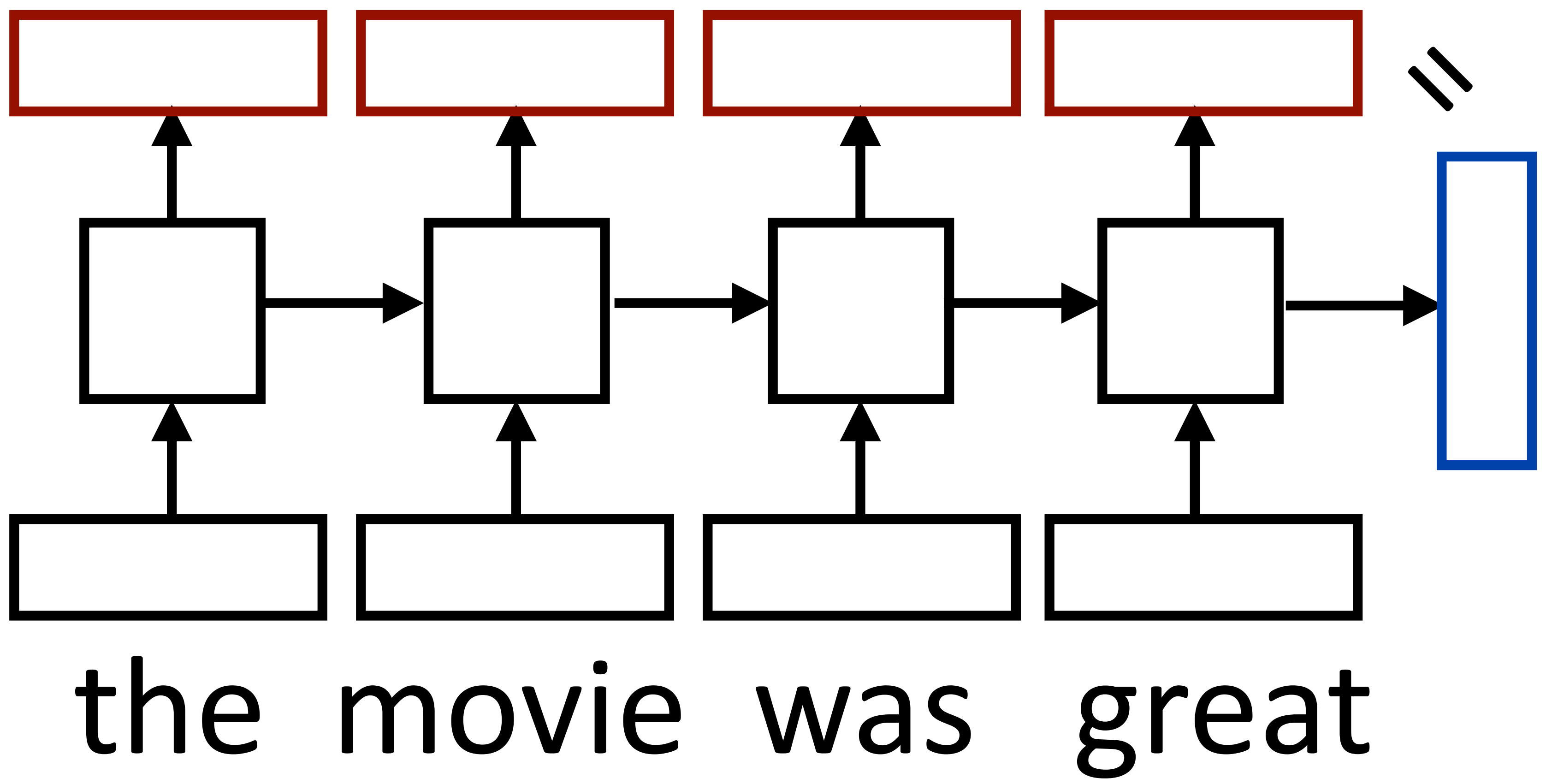
- GRU: faster, a bit simpler
- Two gates: **z** (forget, mixes **s** and **h**) and **r** (mixes **h** and **x**)

What do RNNs produce?



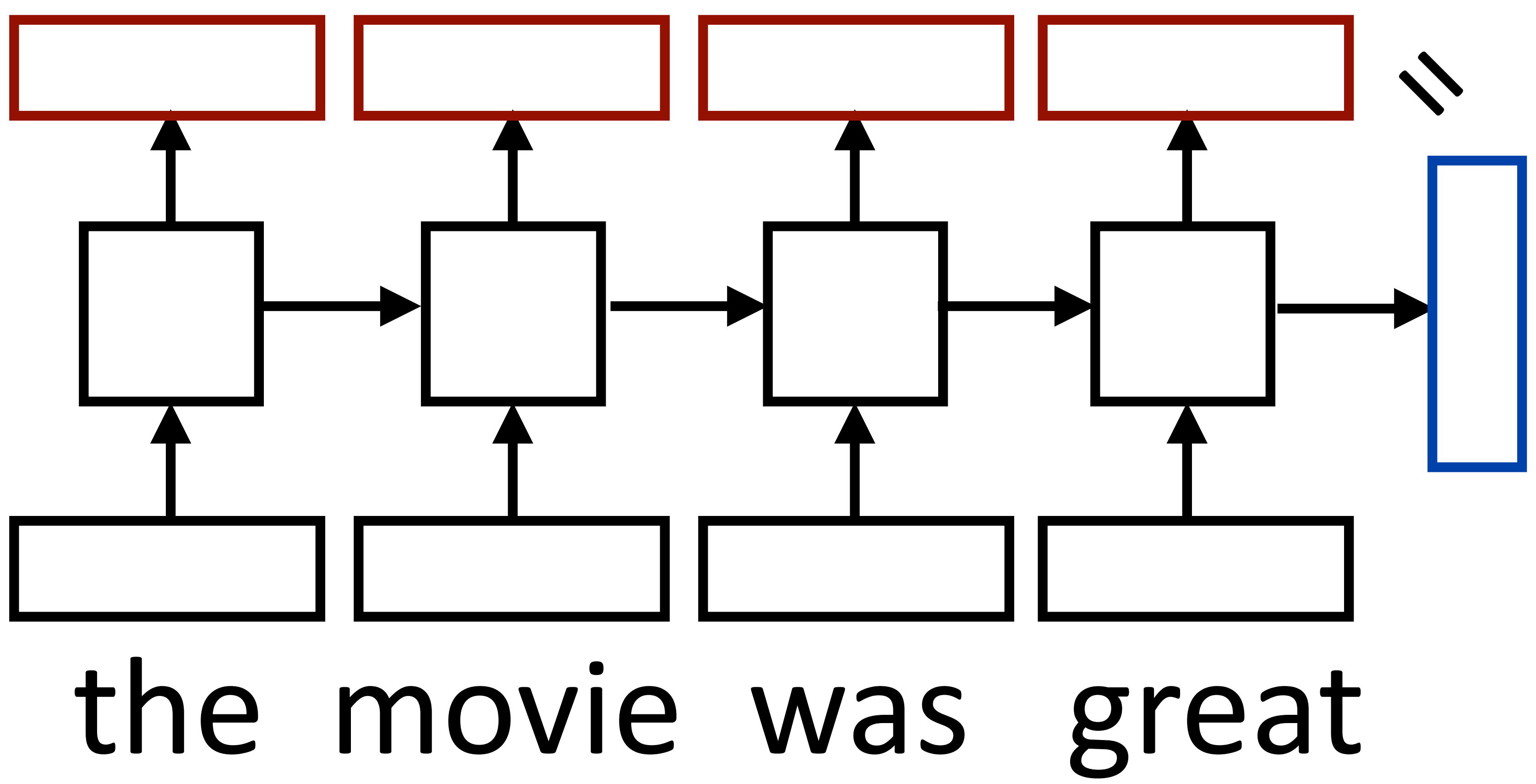
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence

What do RNNs produce?



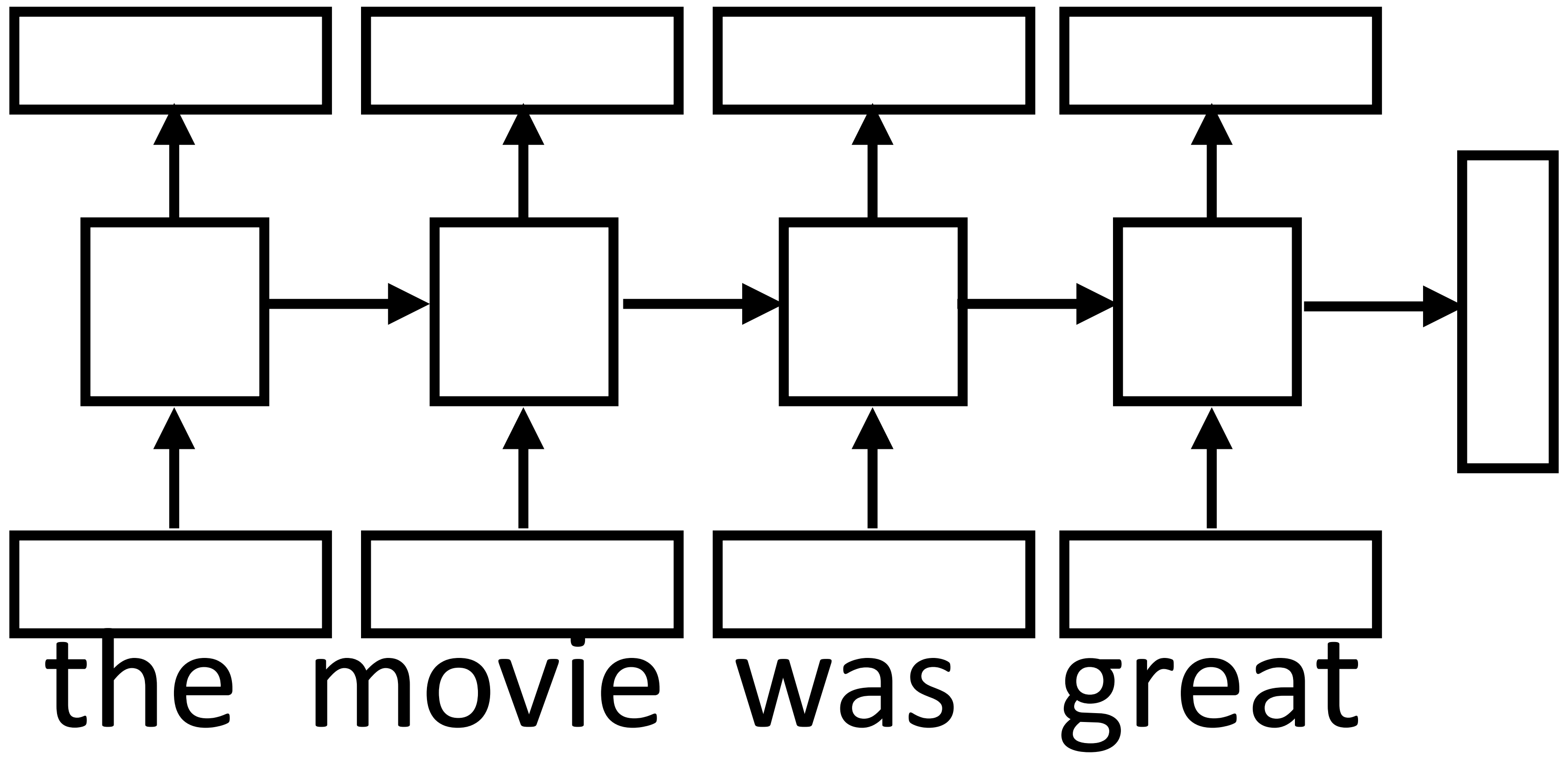
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)

What do RNNs produce?

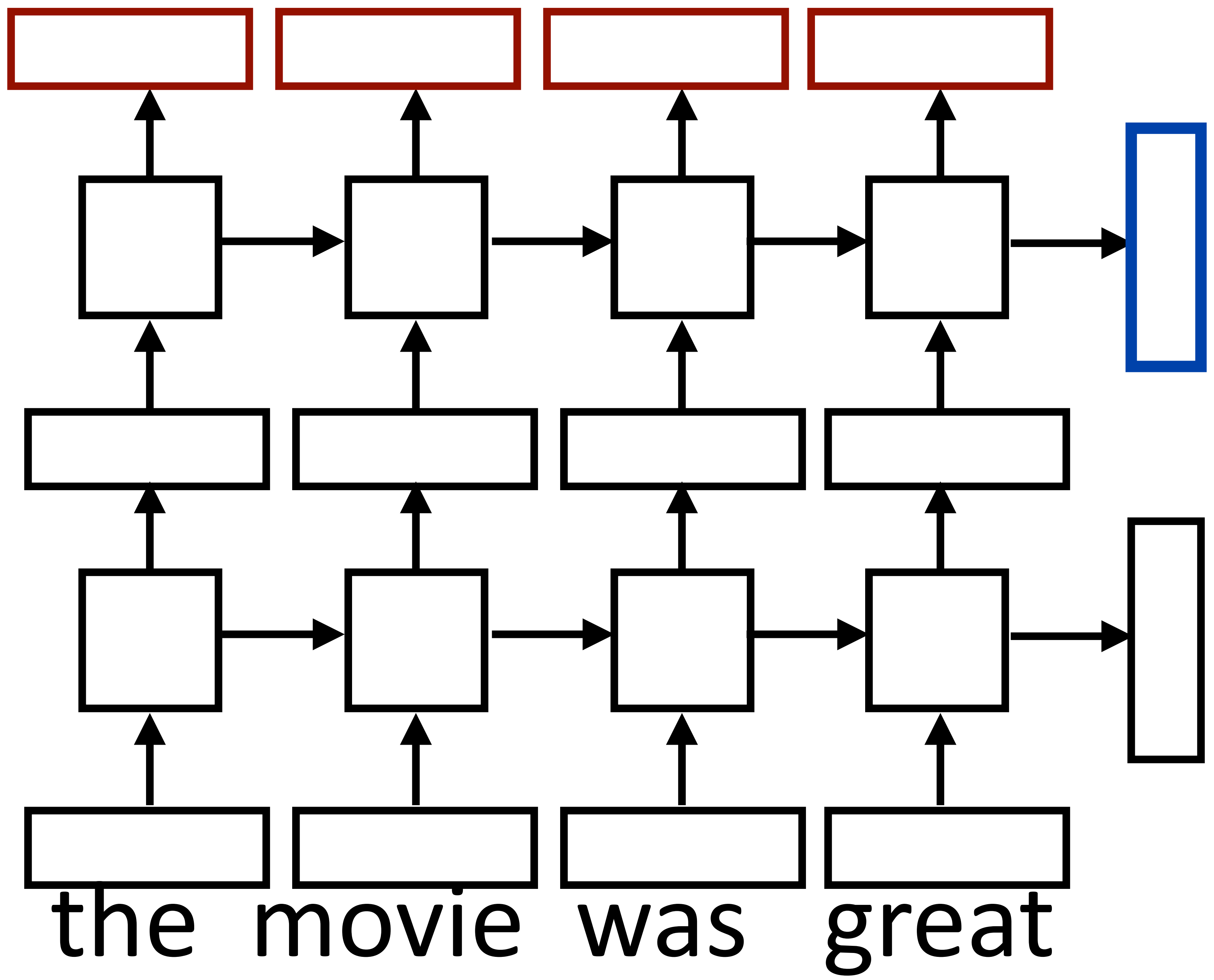


- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors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

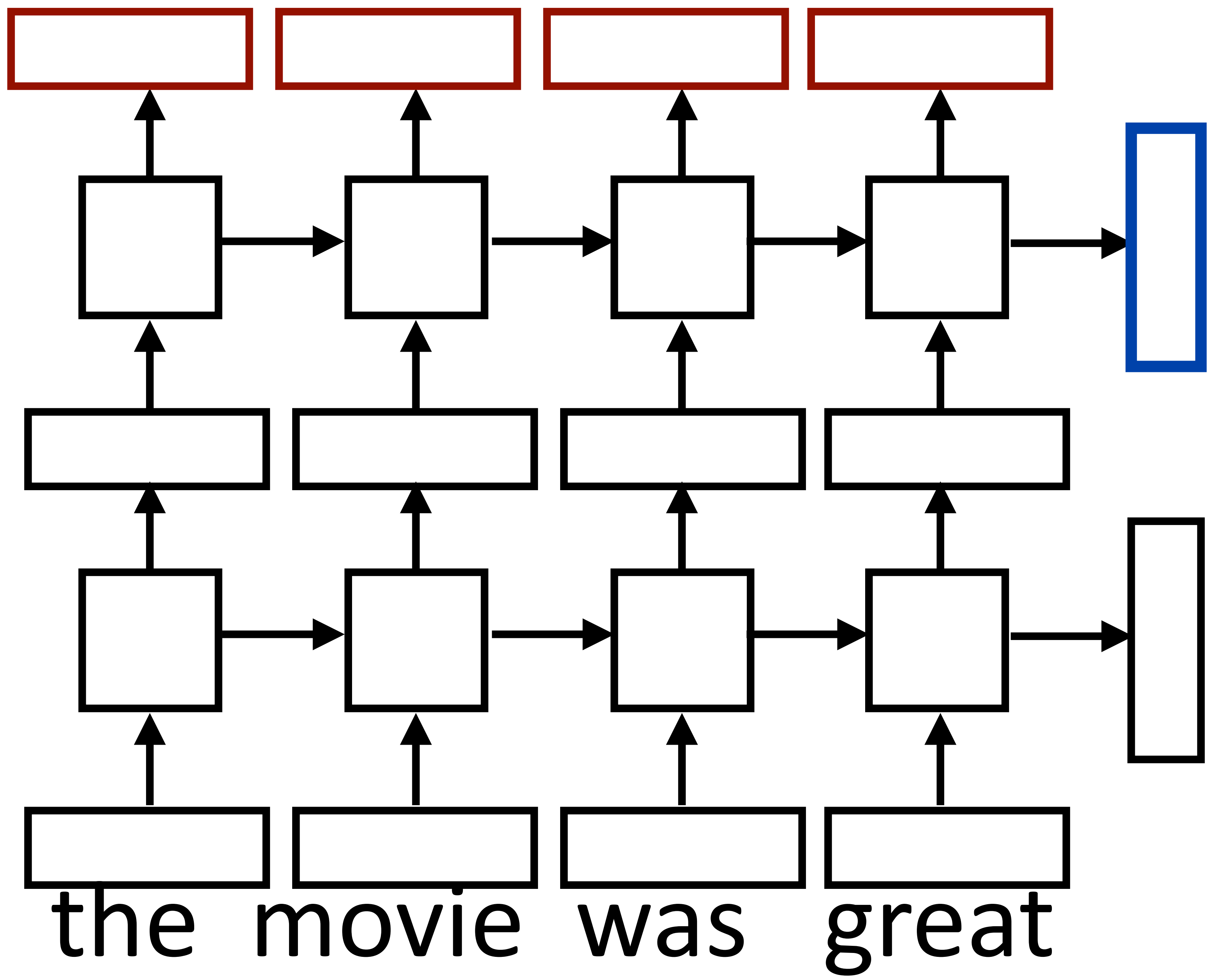
Multilayer Bidirectional RNN

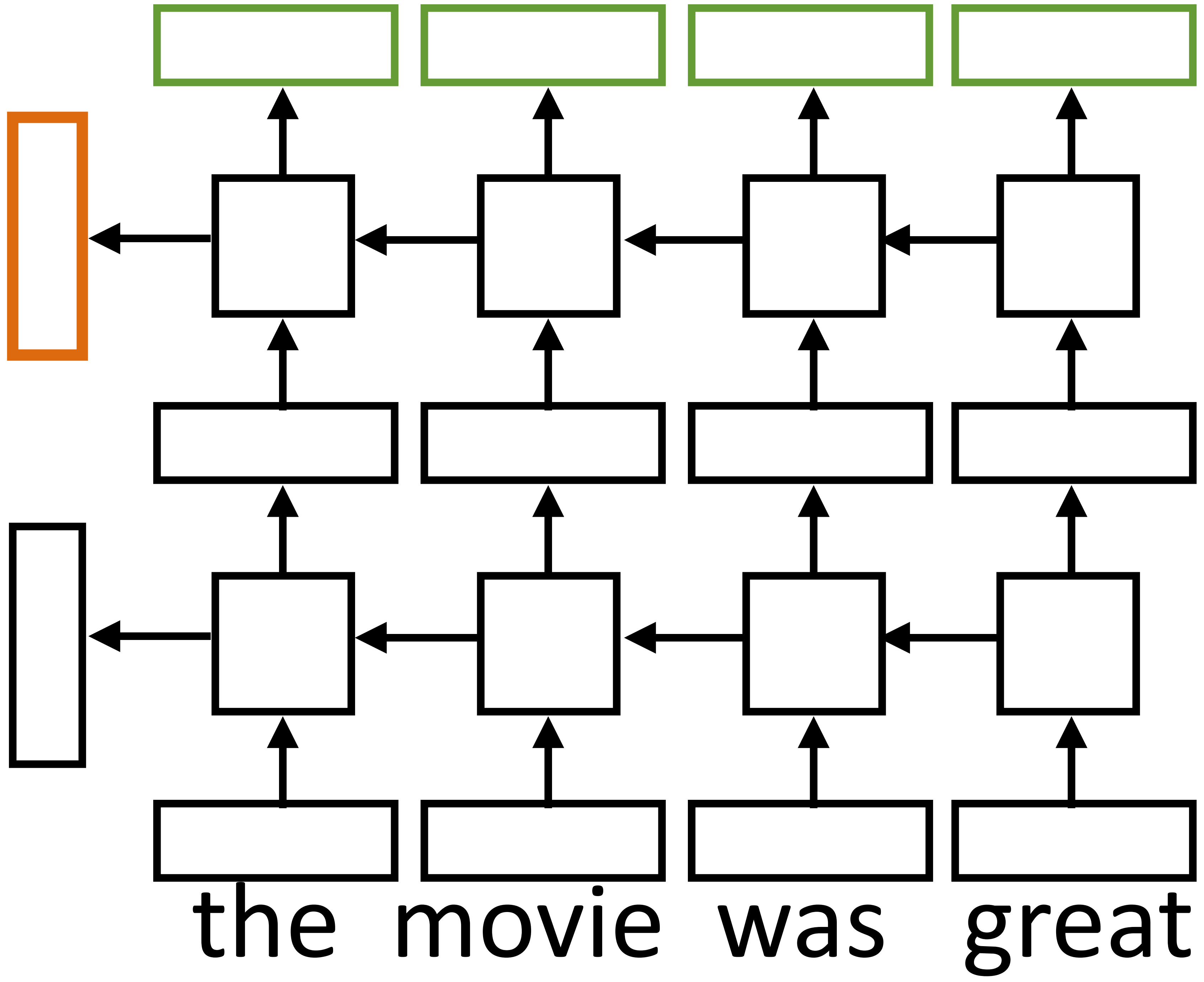


Multilayer Bidirectional RNN

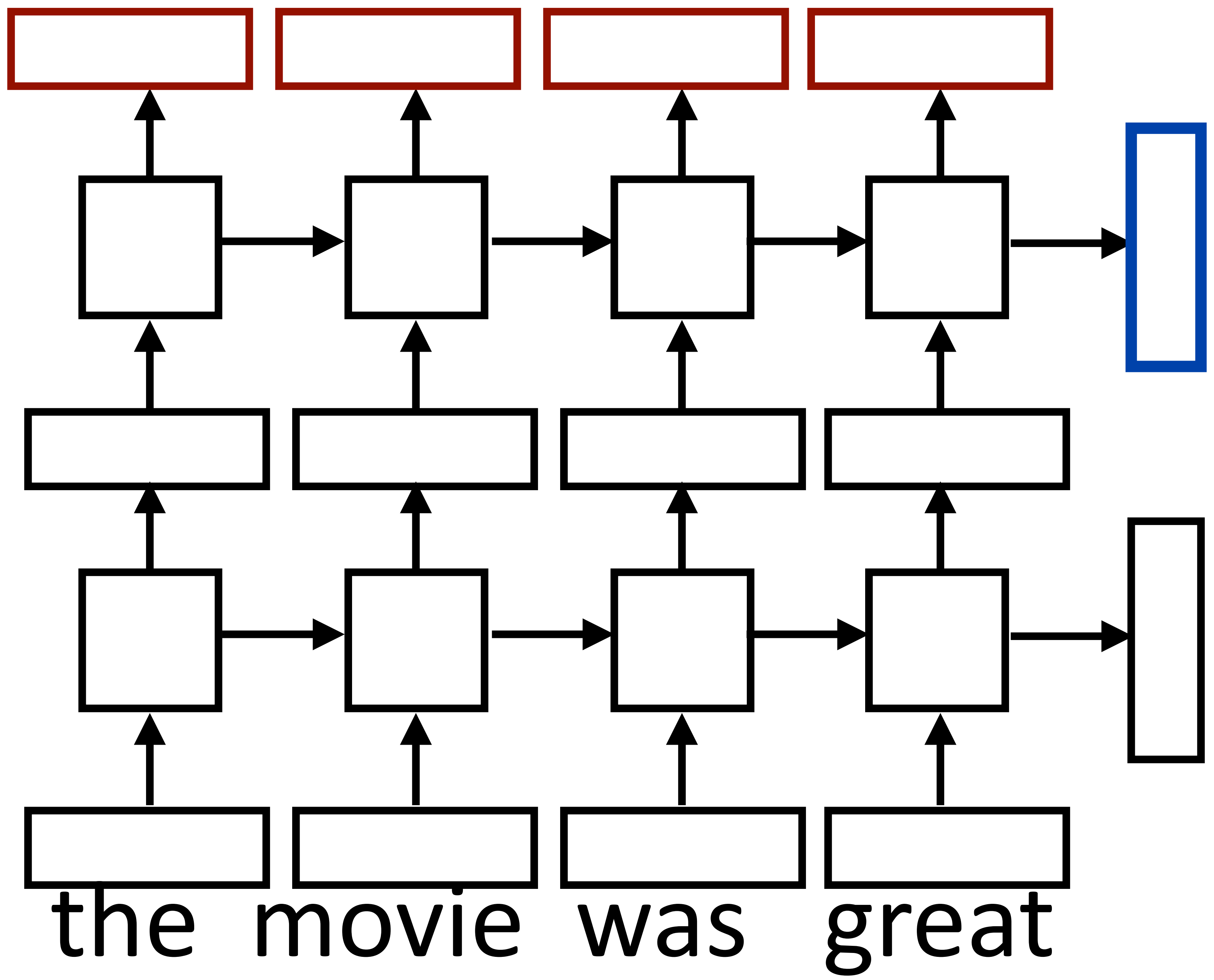


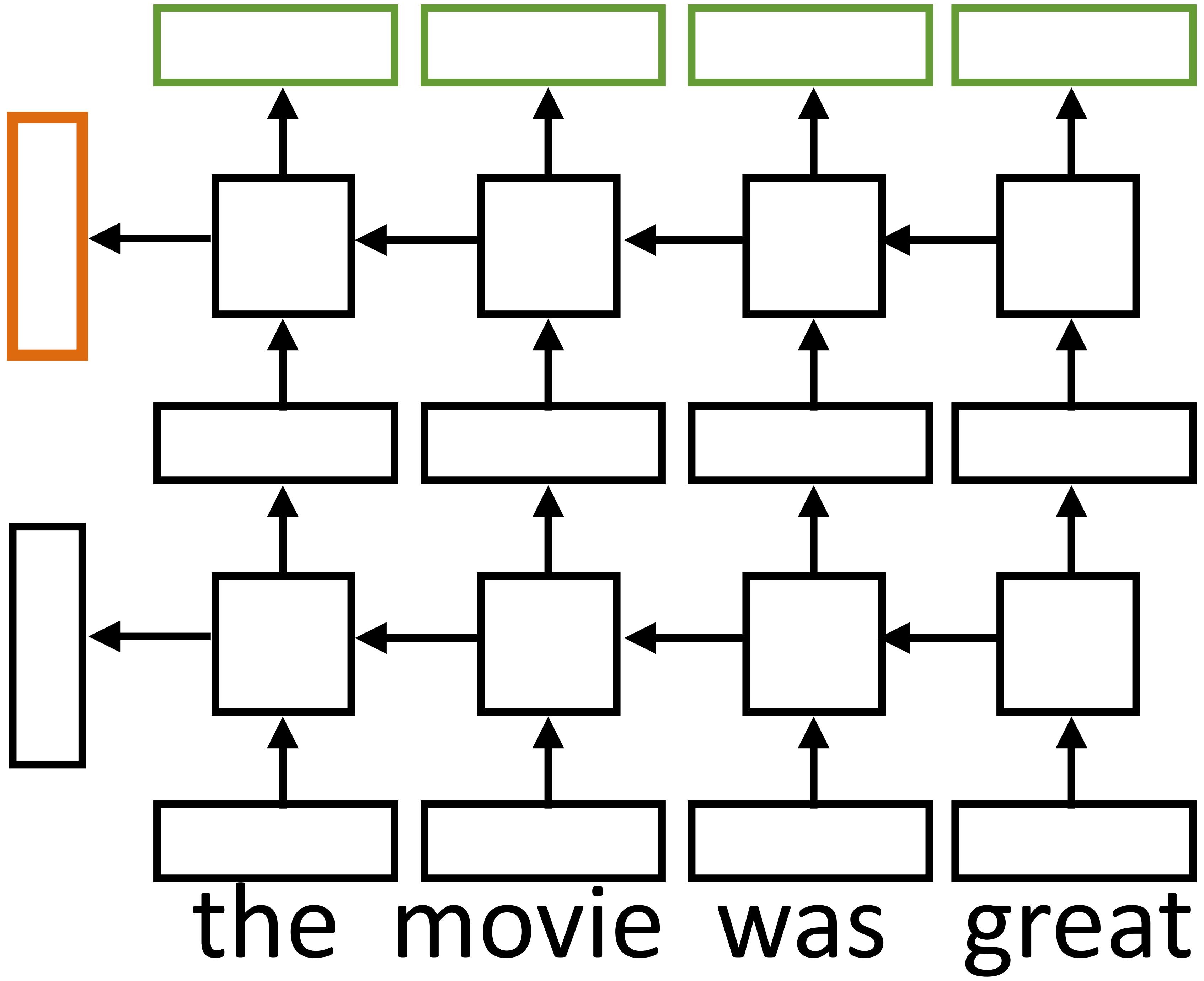
Multilayer Bidirectional RNN



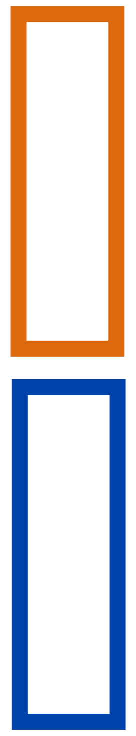


Multilayer Bidirectional RNN

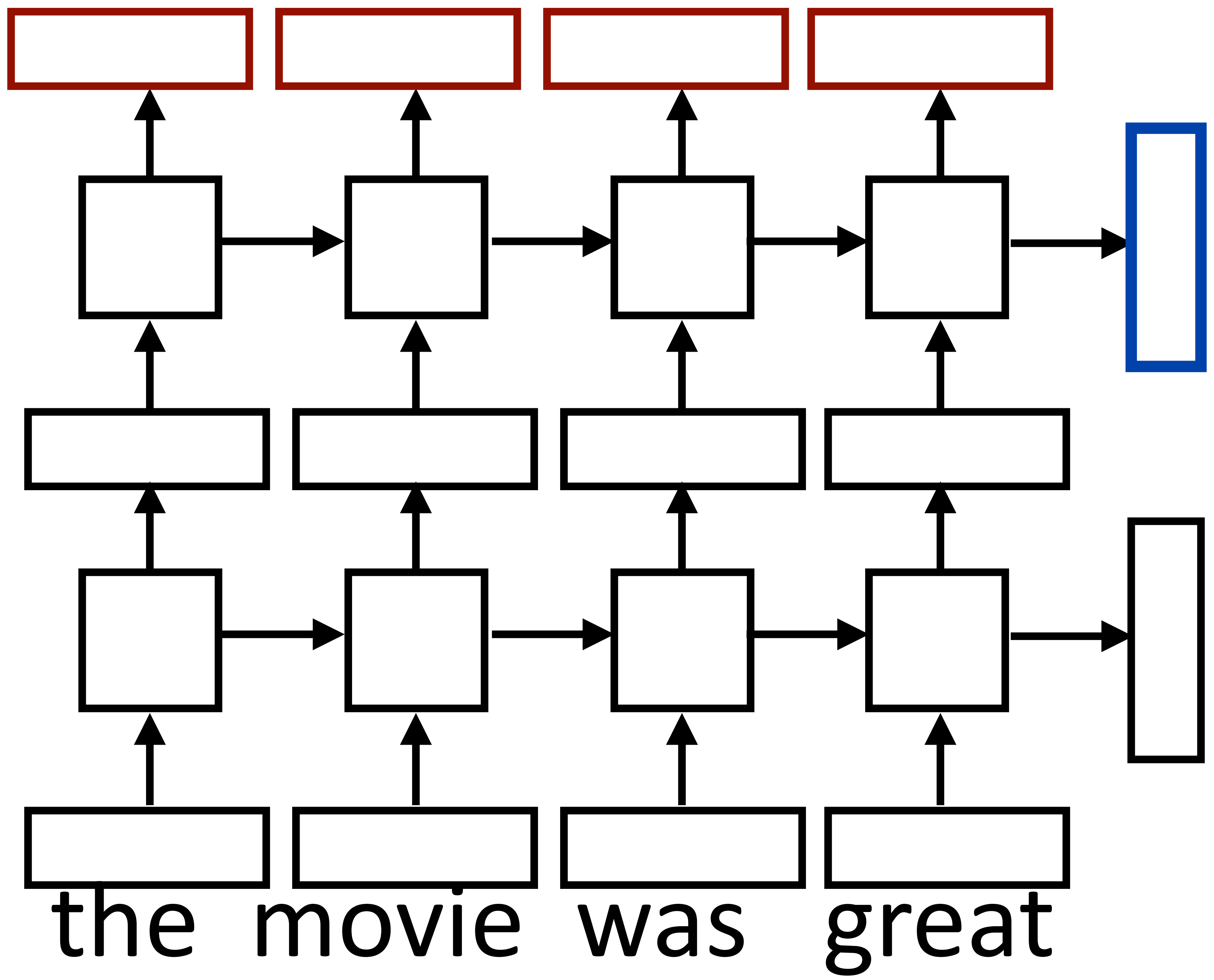


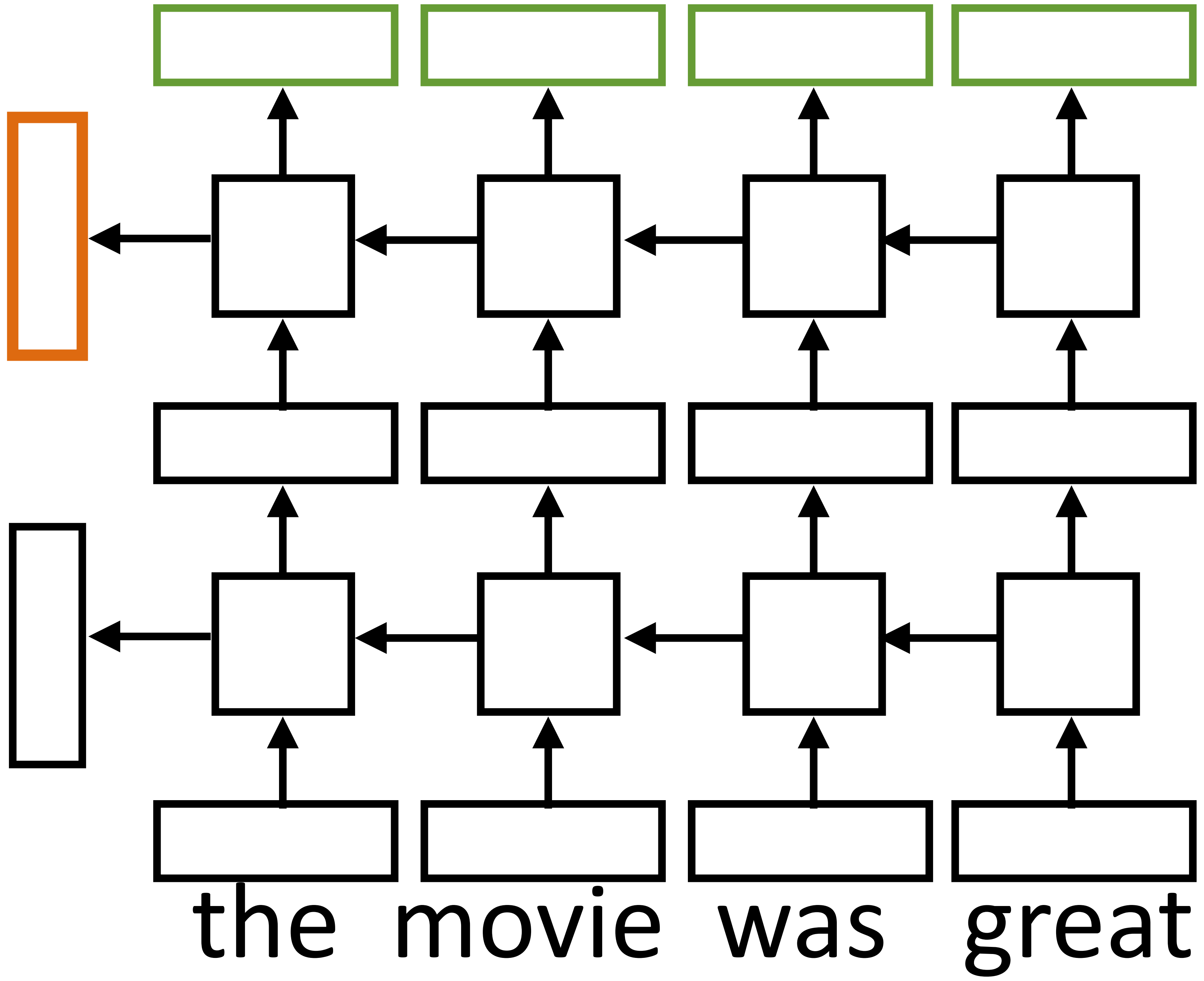


- Sentence classification based on concatenation of both final outputs

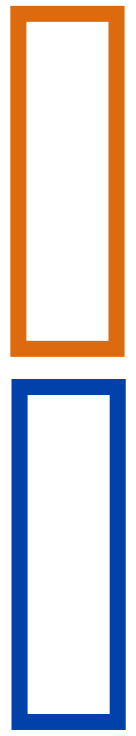


Multilayer Bidirectional RNN

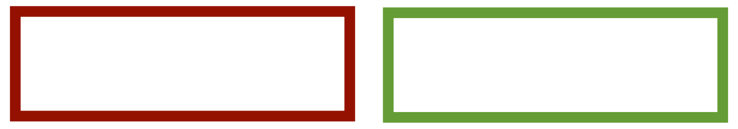




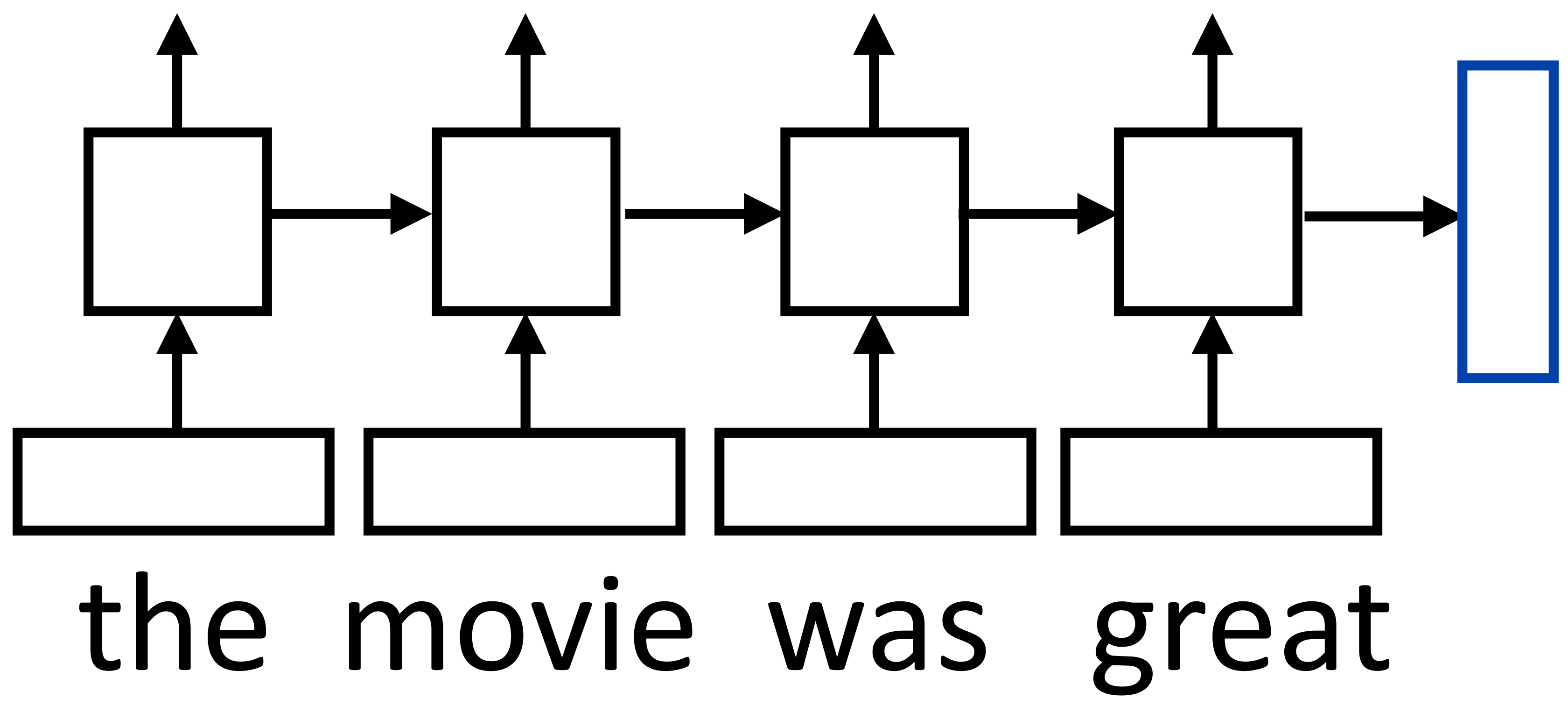
- Sentence classification based on concatenation of both final outputs



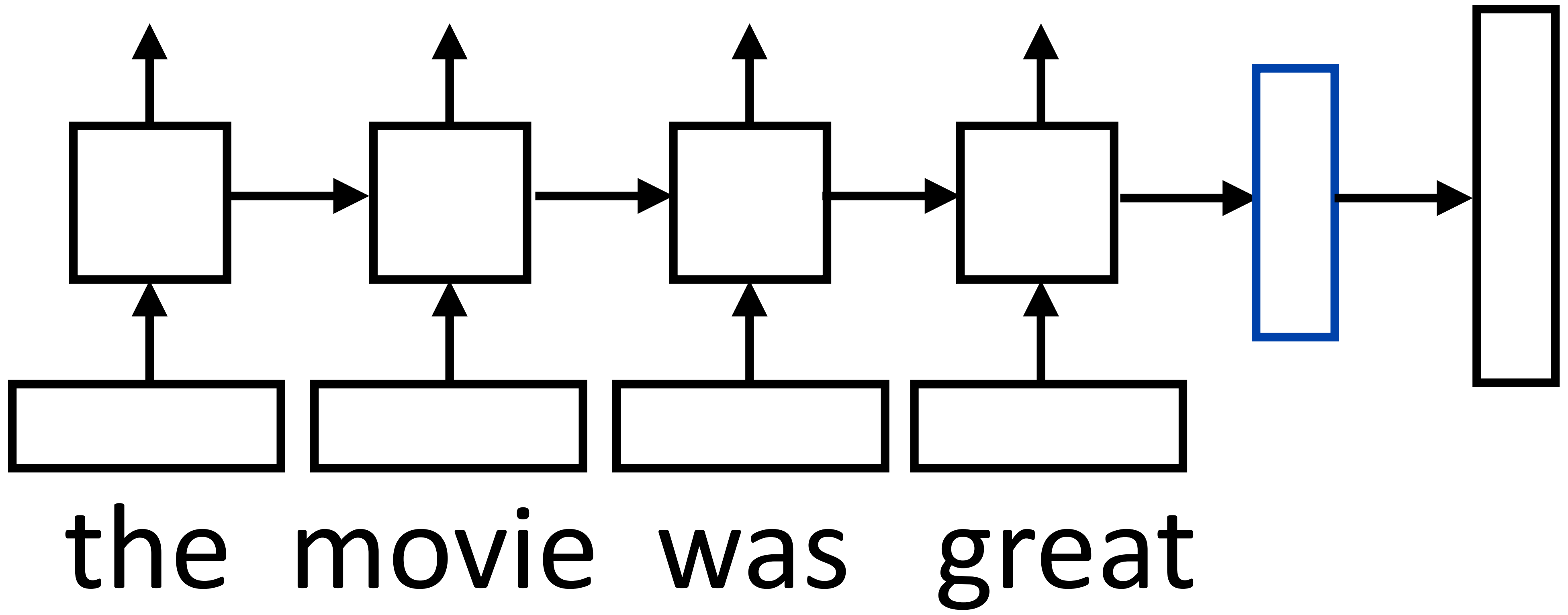
- Token classification based on concatenation of both directions' token representations



Training RNNs

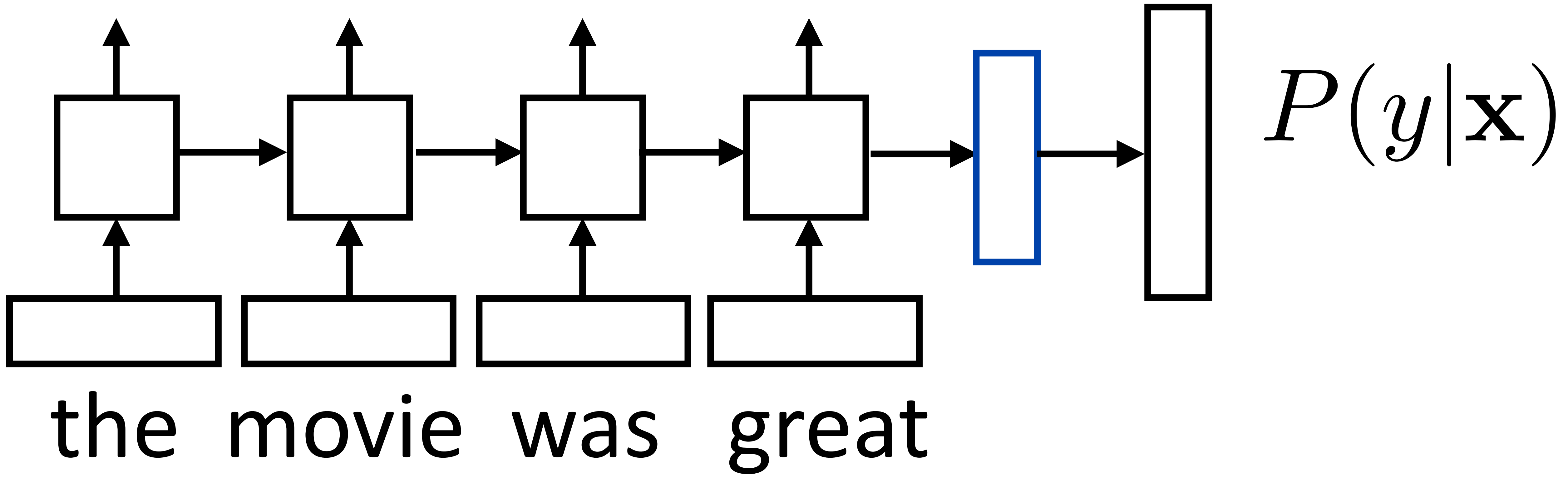


Training RNNs



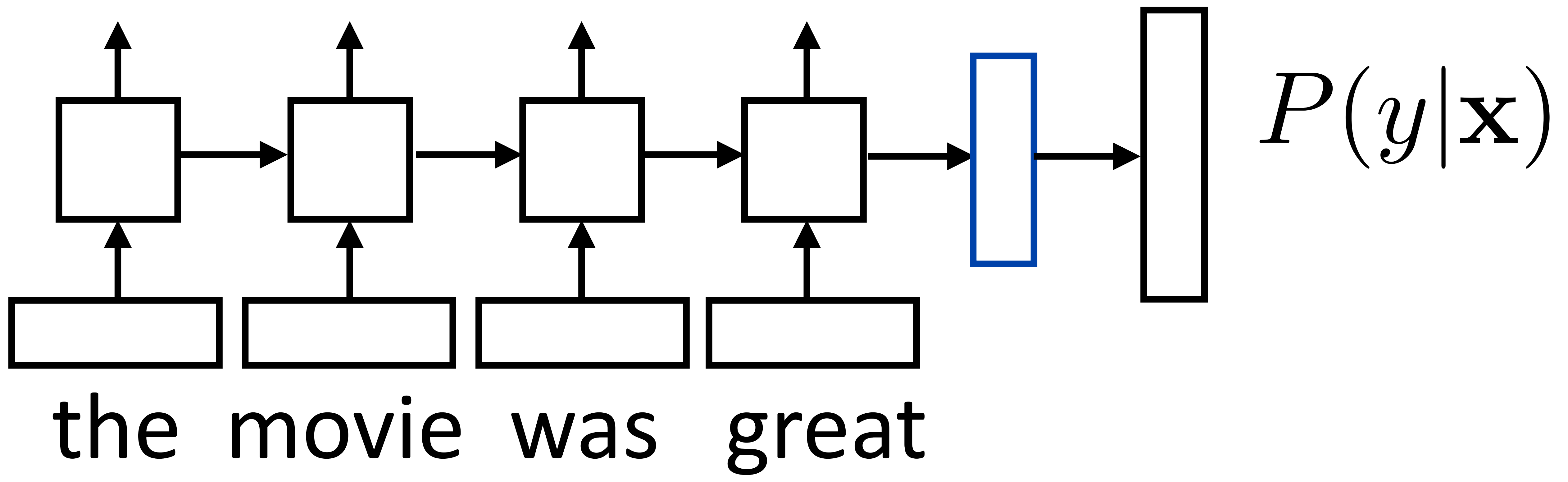
$P(y|\mathbf{x})$

Training RNNs



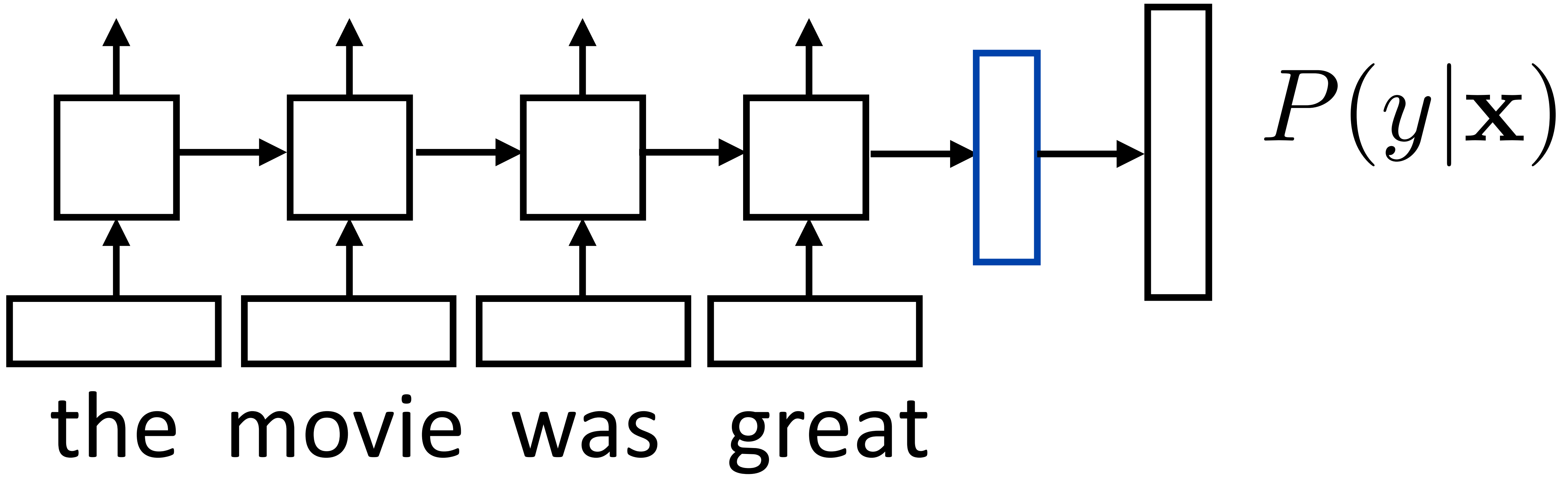
- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)

Training RNNs



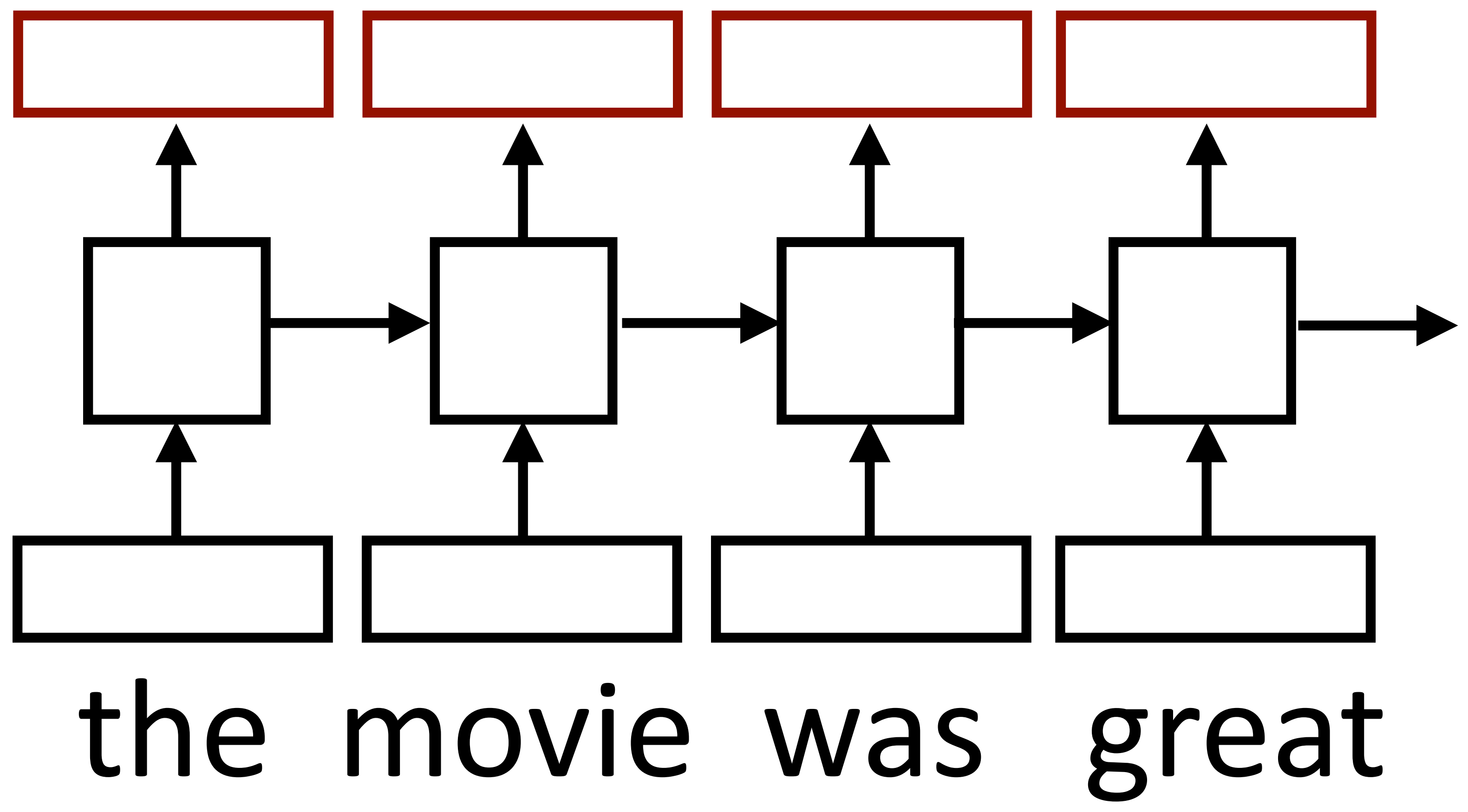
- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network

Training RNNs

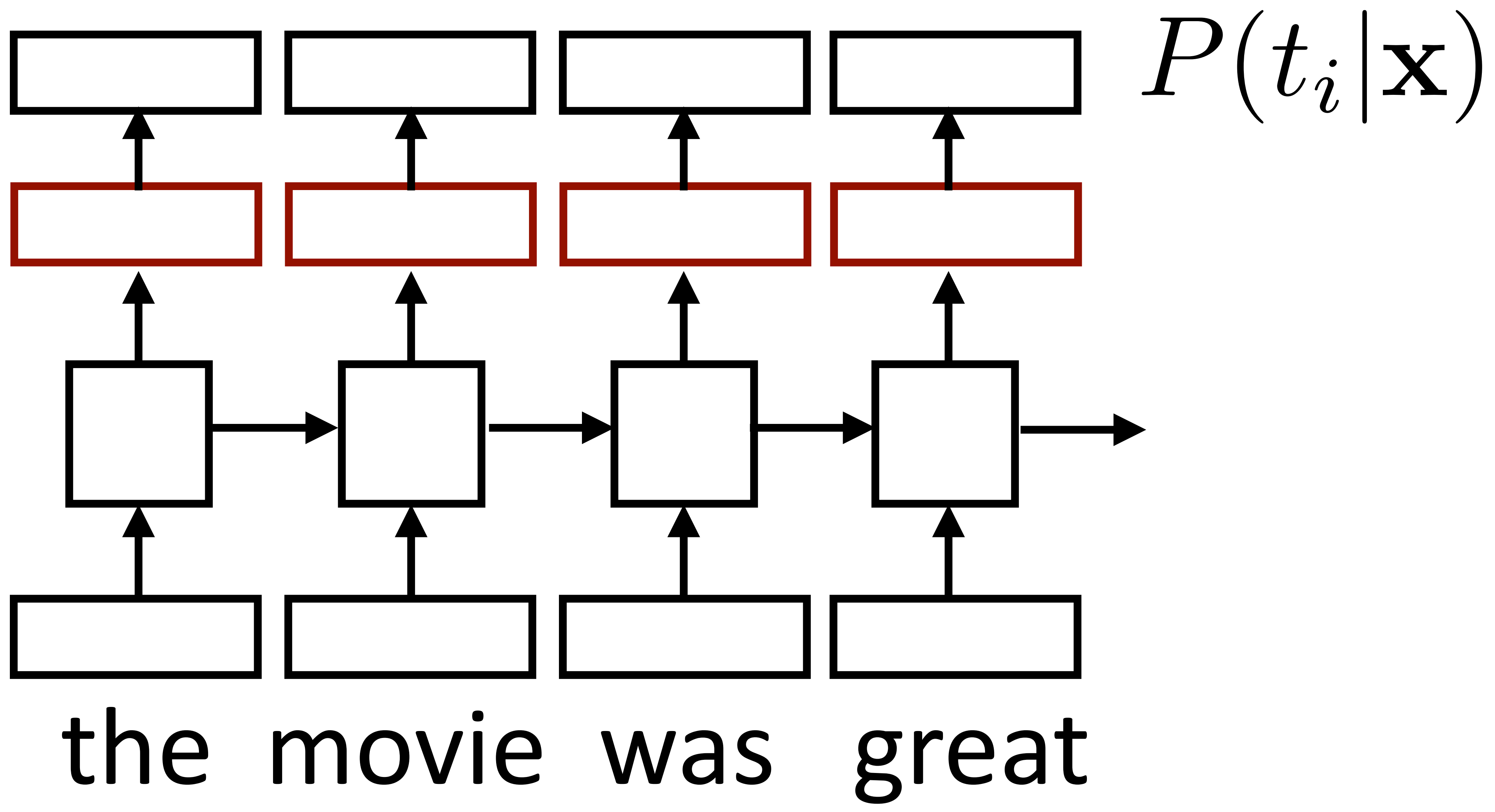


- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis

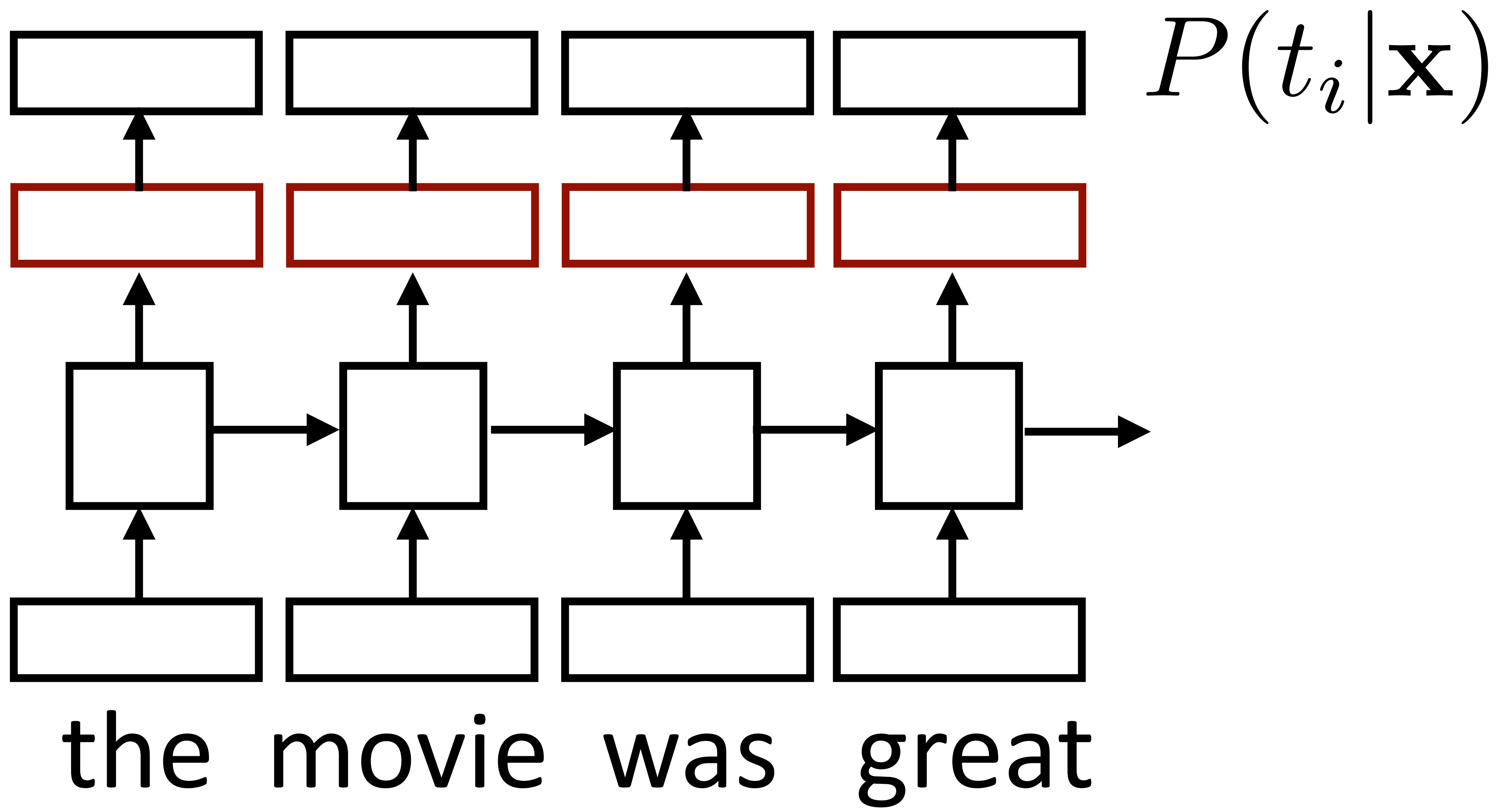
Training RNNs



Training RNNs

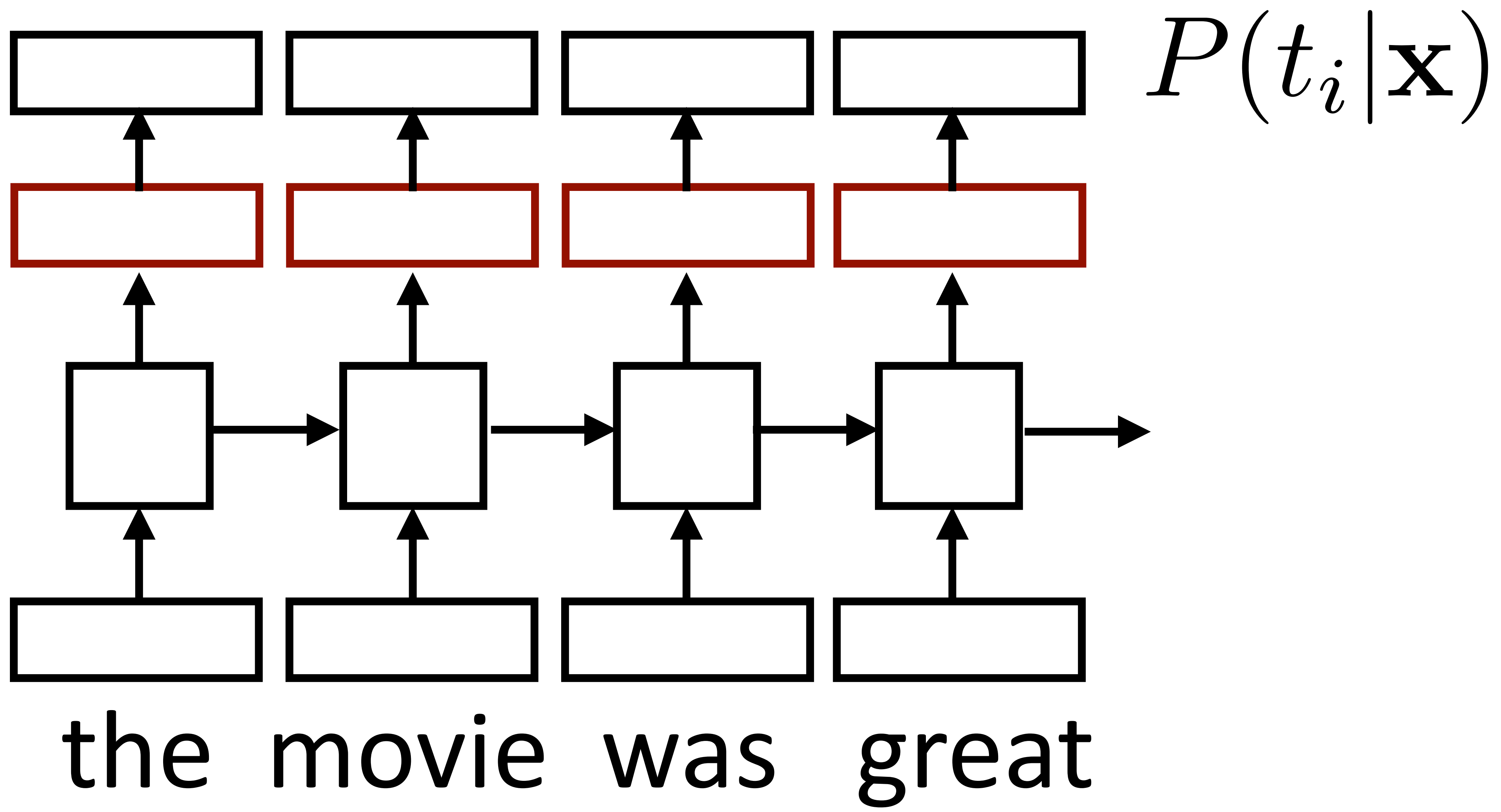


Training RNNs



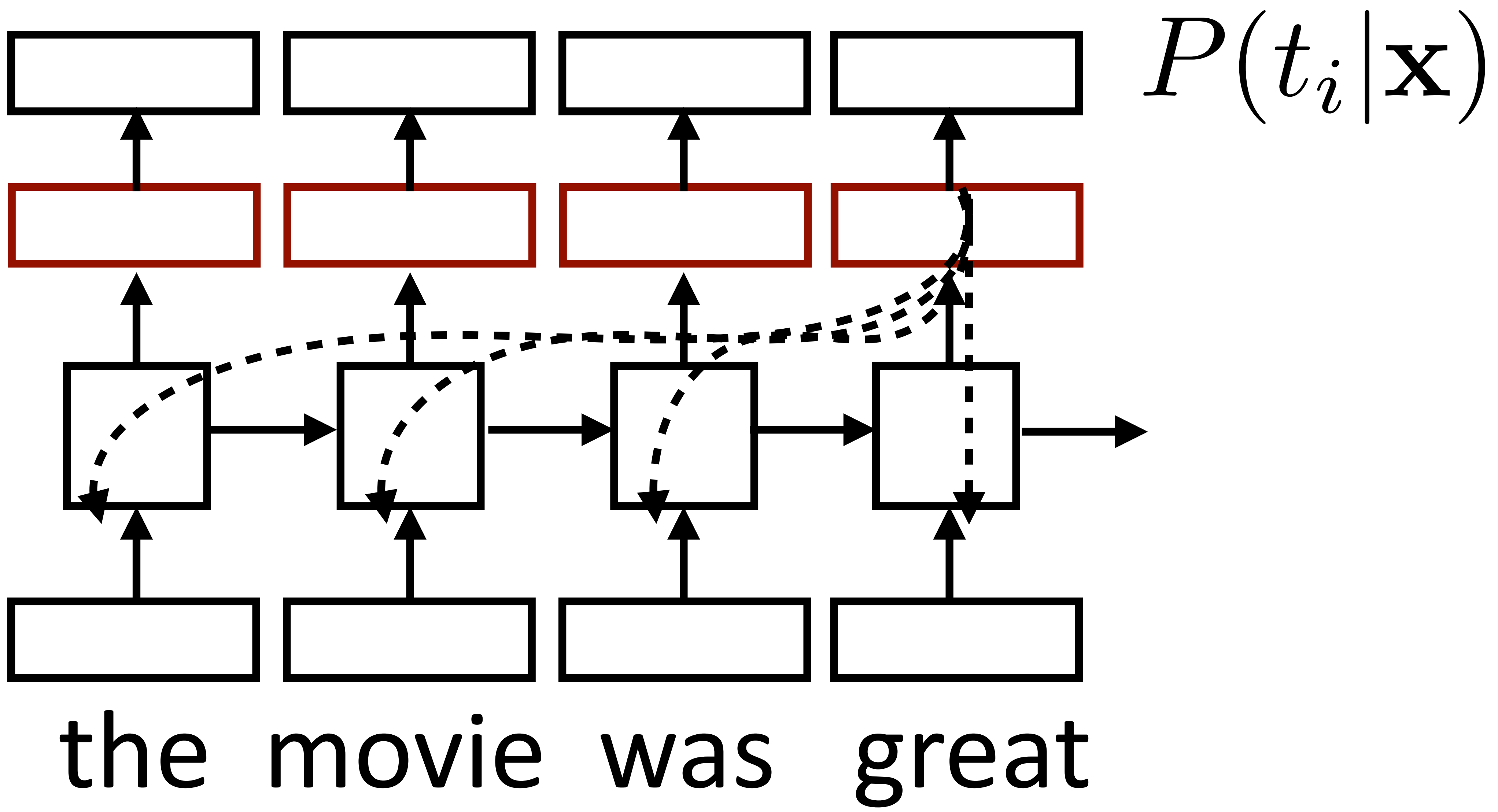
- Loss = negative log likelihood of probability of gold predictions, summed over the tags

Training RNNs



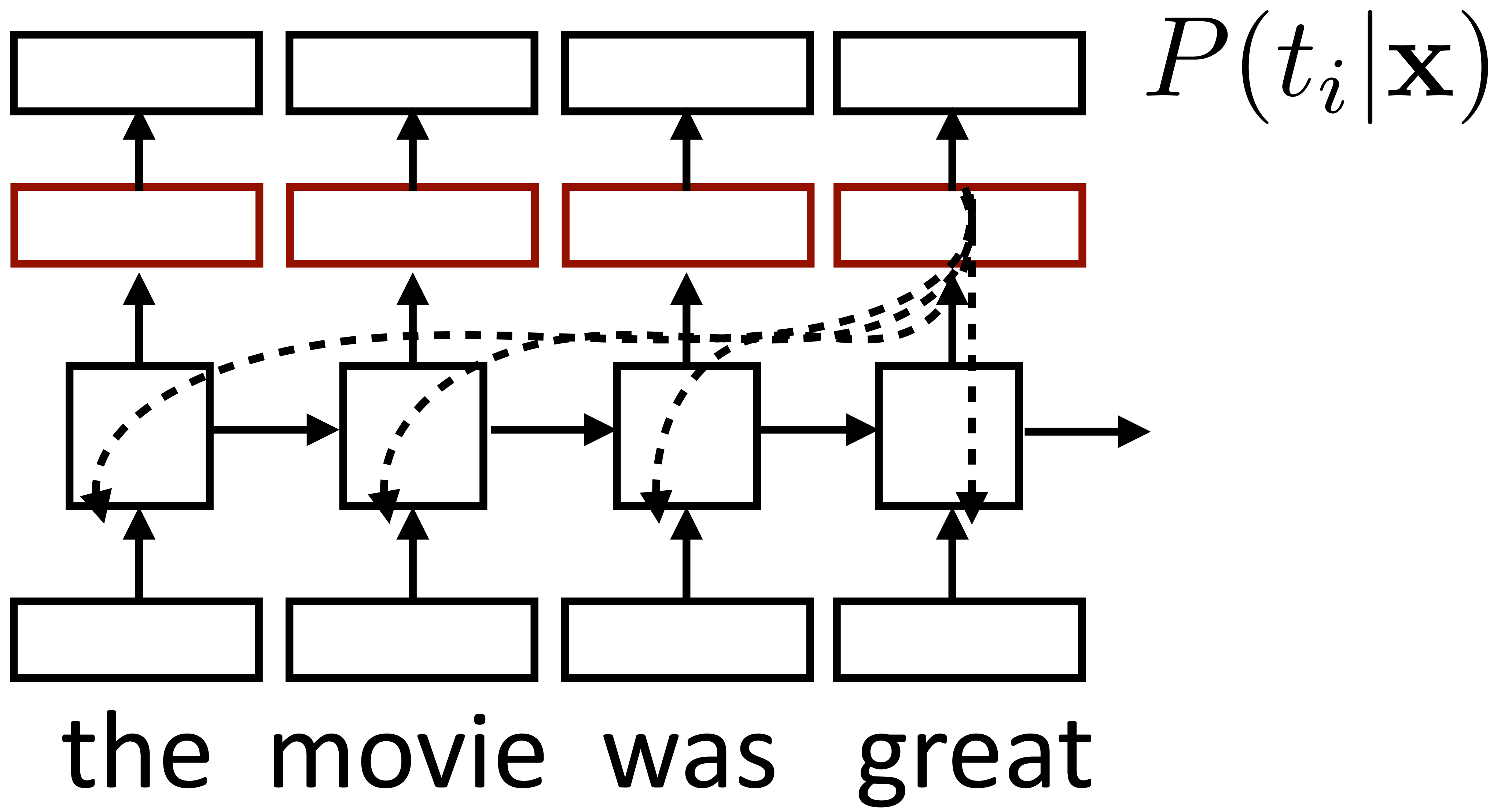
- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network

Training RNNs



- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network

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?

What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction

What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation

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

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

Karpathy et al. (2015)

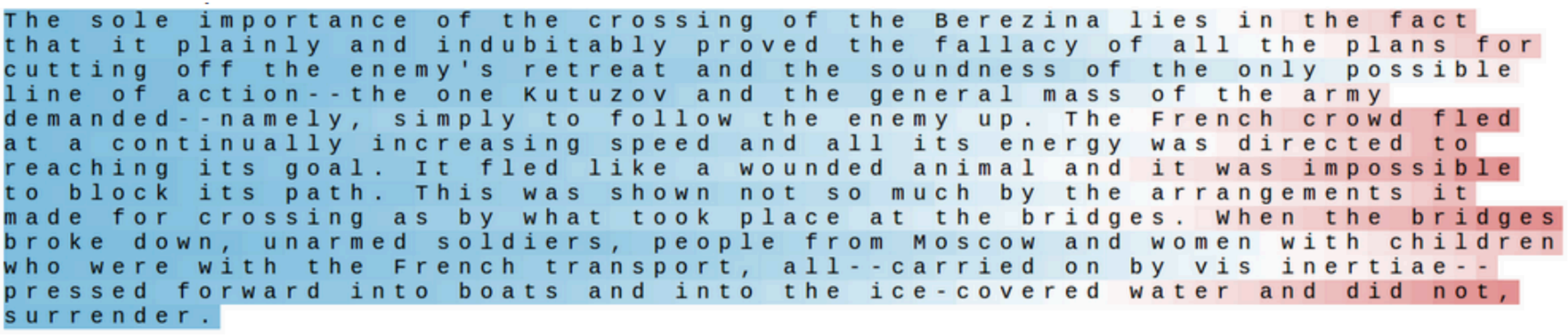
Visualizing LSTMs

- 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 (components of **c**) to understand them

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 (components of **c**) to understand them



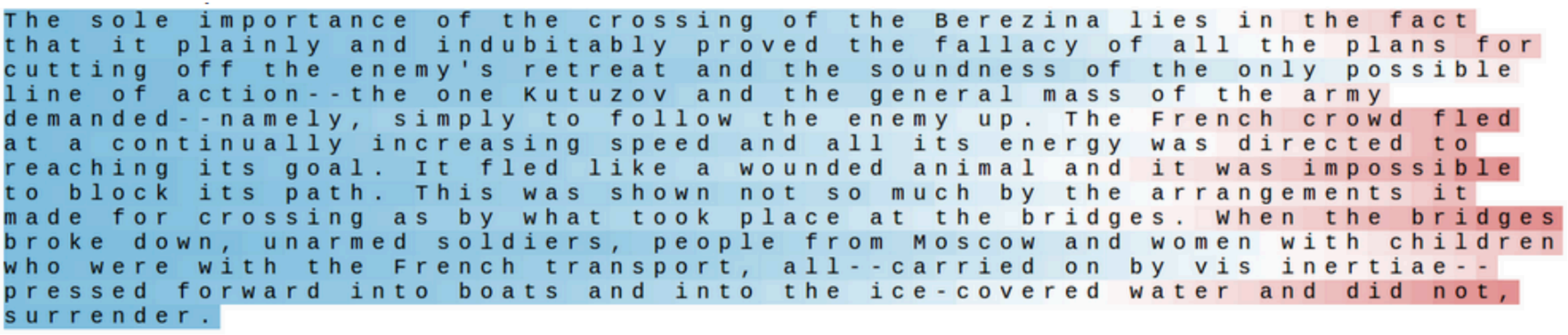
Karpathy et al. (2015)

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 (components of **c**) to understand them

- Counter: know when to generate \n

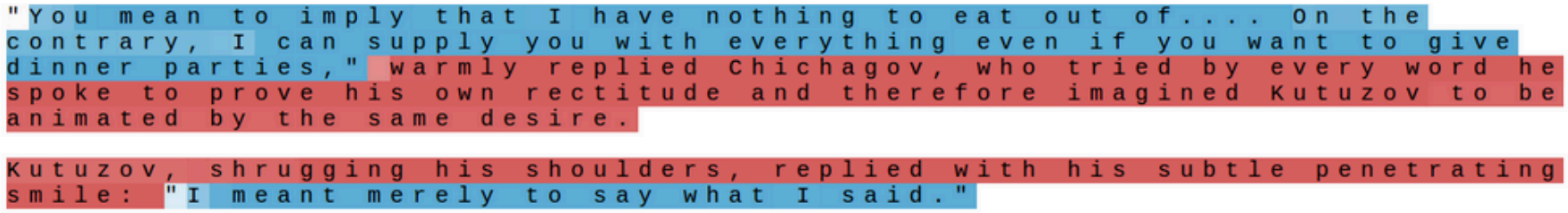


Karpathy et al. (2015)

Visualizing LSTMs

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

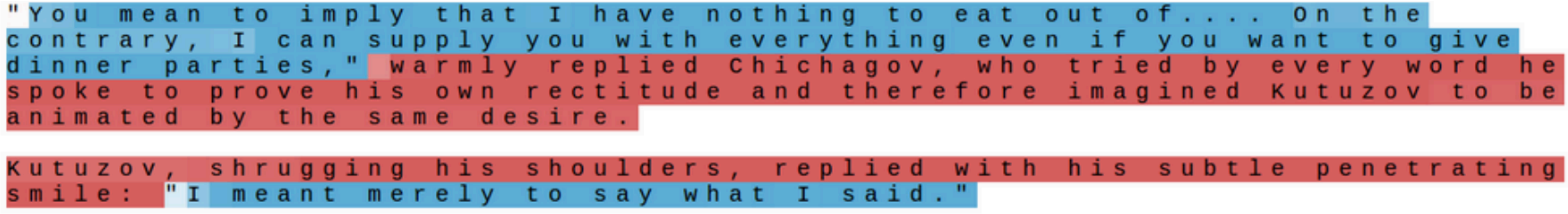
- Visualize activations of specific cells to see what they track



Karpathy et al. (2015)

Visualizing LSTMs

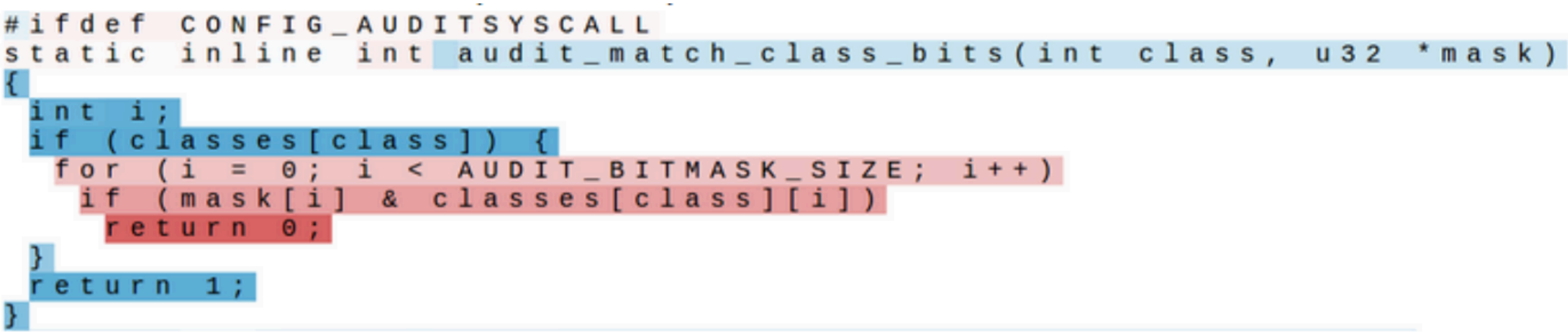
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: tells us if we're in a quote or not



Karpathy et al. (2015)

Visualizing LSTMs

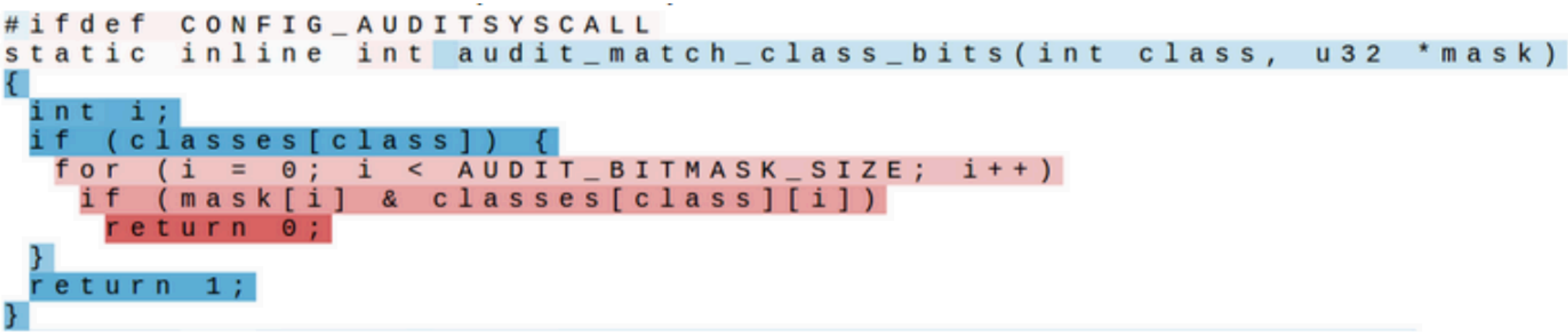
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track



Karpathy et al. (2015)

Visualizing LSTMs

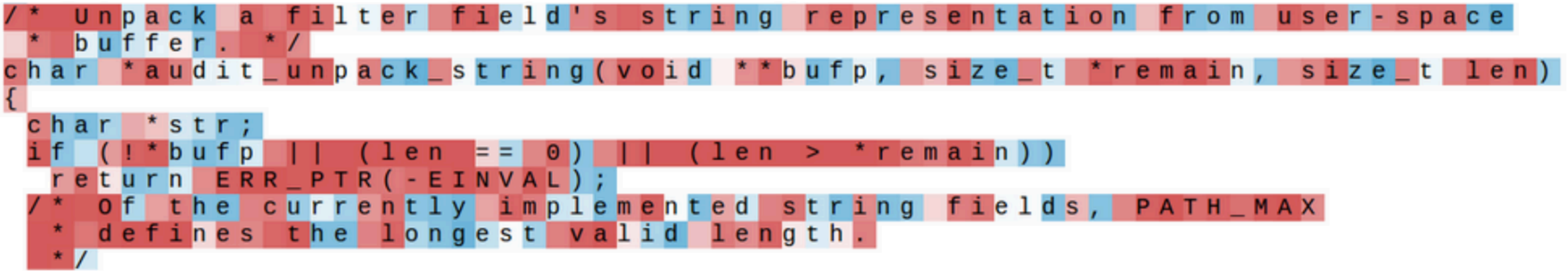
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation



Karpathy et al. (2015)

Visualizing LSTMs

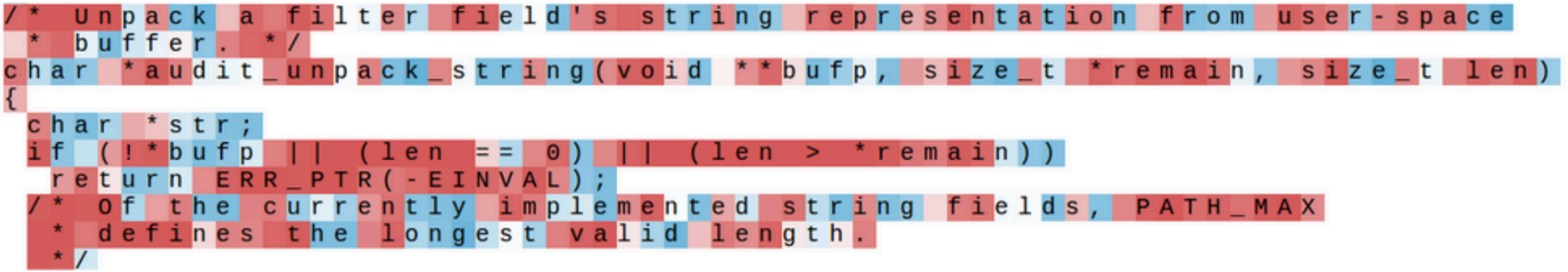
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track



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)

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

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	A boy is outside

Natural Language Inference

Premise		Hypothesis
A boy plays in the snow	*entails*	A boy is outside

Natural Language Inference

Premise		Hypothesis
A boy plays in the snow	*entails*	A boy is outside
A man inspects the uniform of a figure		The man is sleeping

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

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		Two men are smiling and laughing at cats playing

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

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)

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.)

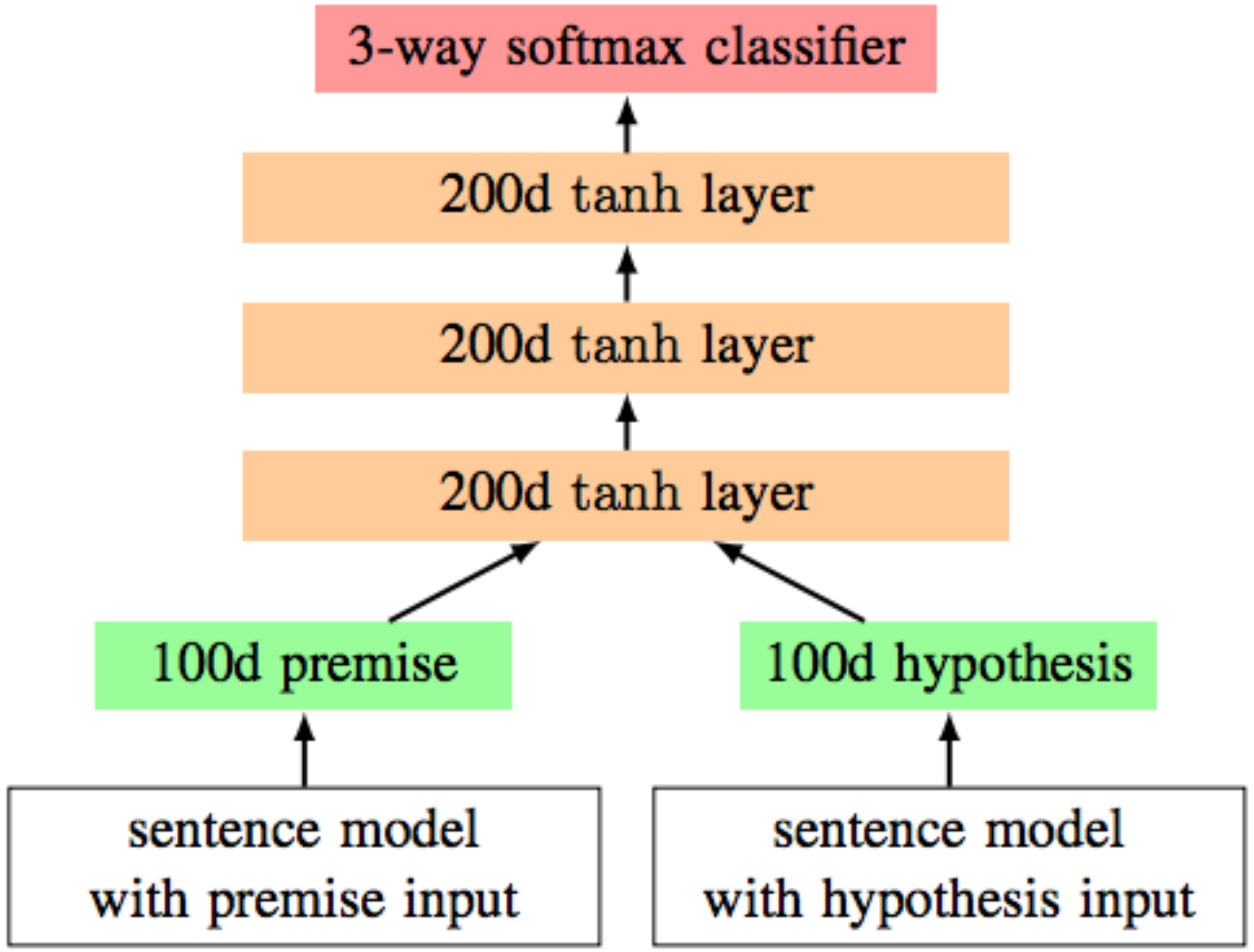
SNLI Dataset

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

Bowman et al. (2015)

SNLI Dataset

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

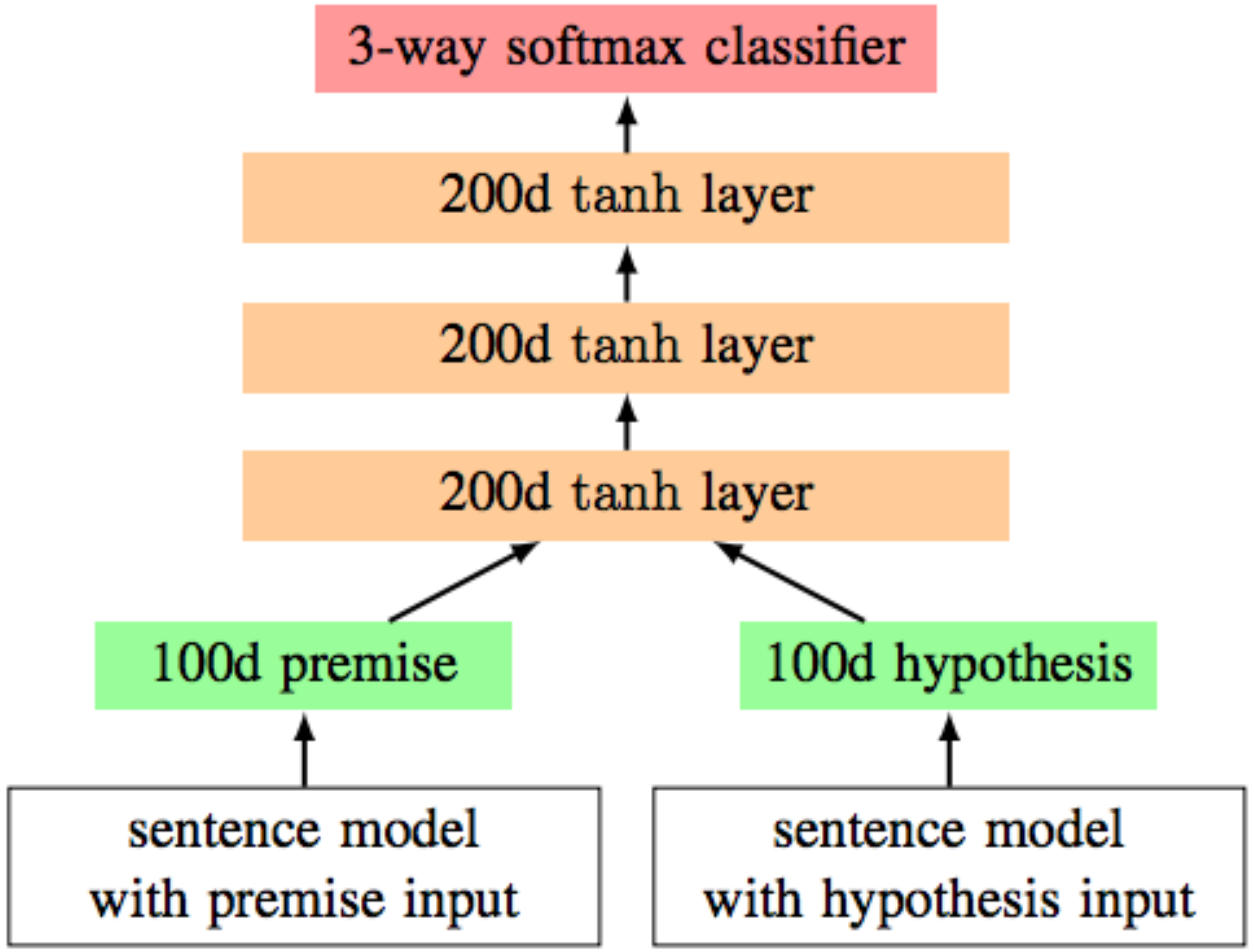


Bowman et al. (2015)

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



Bowman et al. (2015)

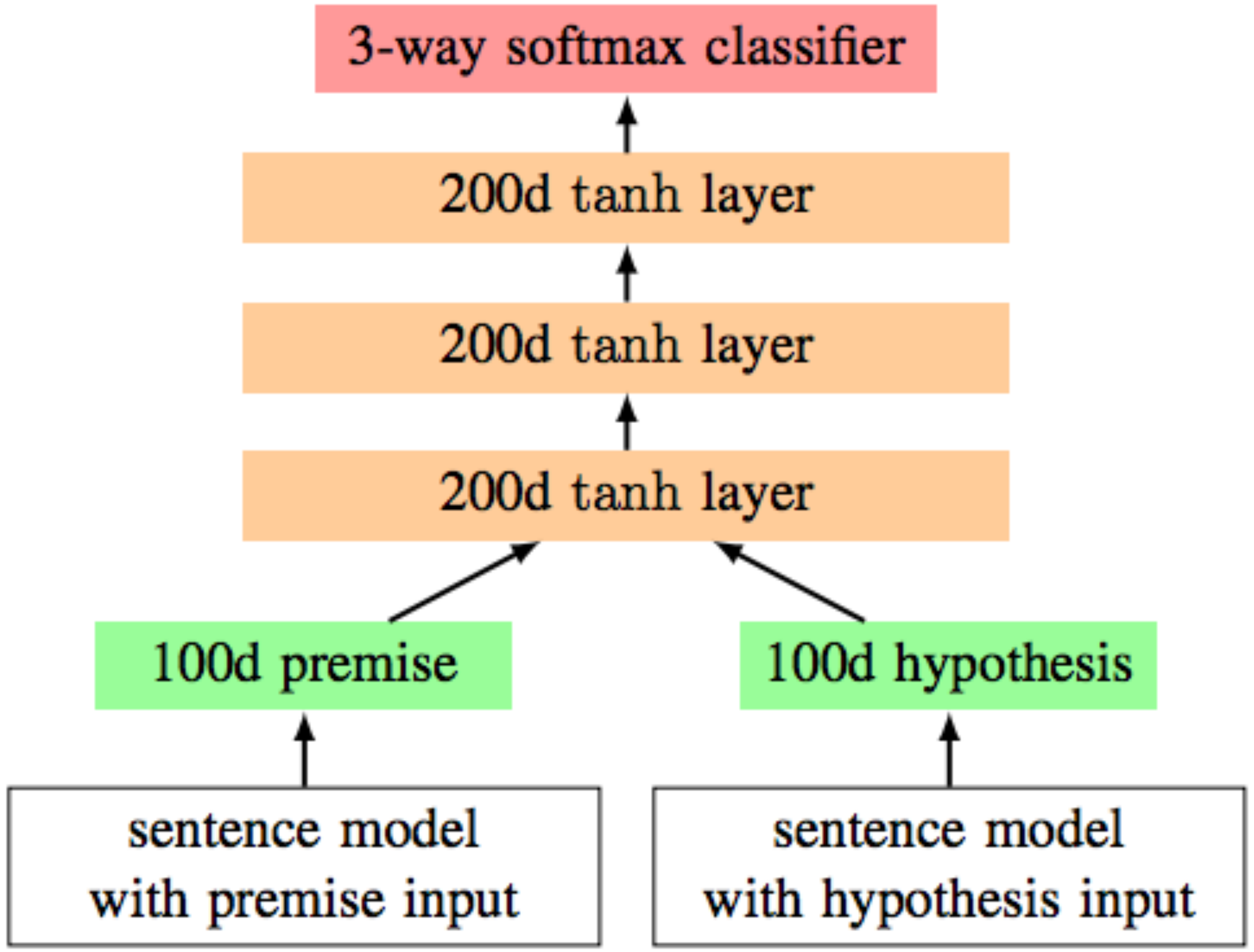
SNLI Dataset

- 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)



Bowman et al. (2015)

SNLI Dataset

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

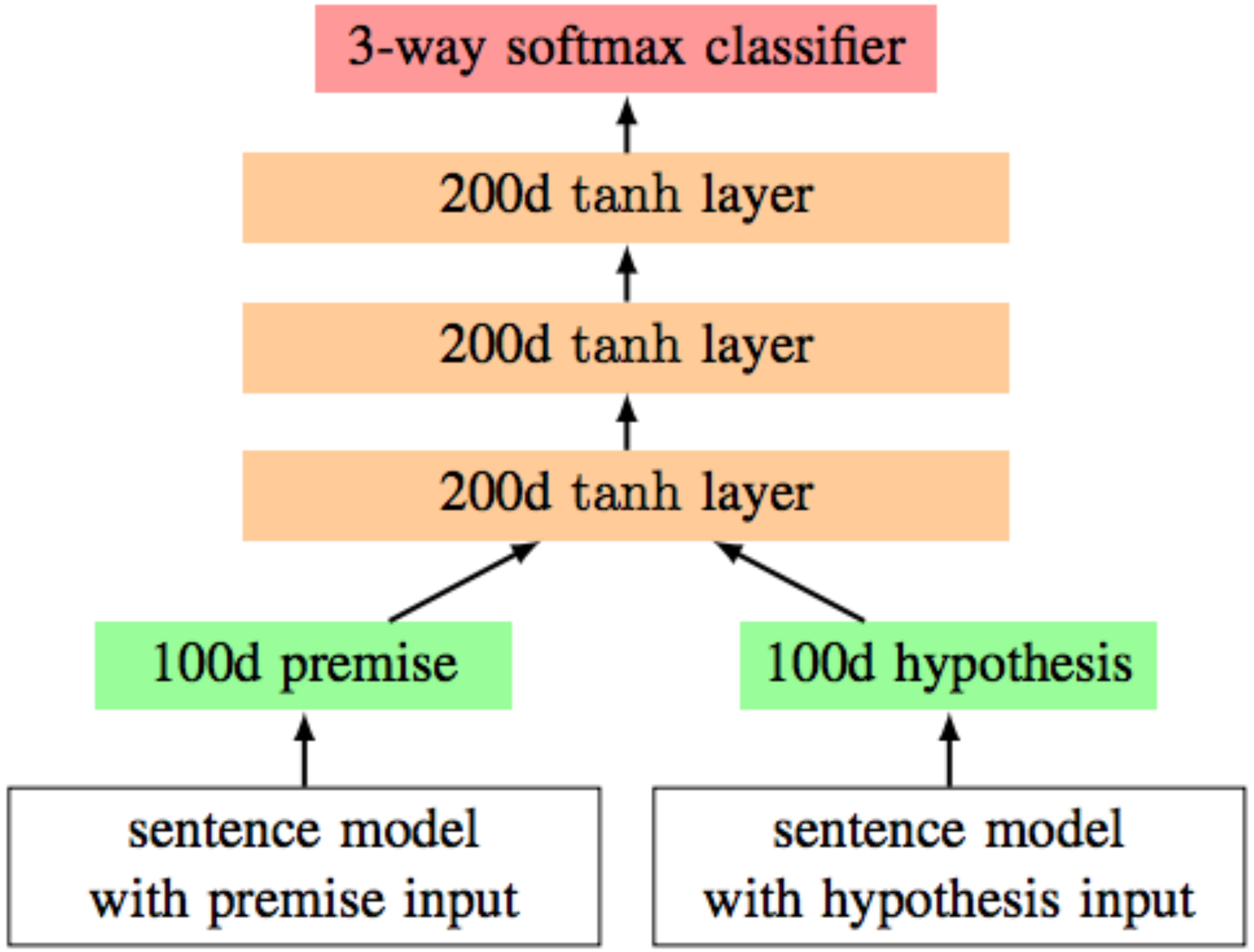
100D LSTM: 78% accuracy

300D LSTM: 80% accuracy

(Bowman et al., 2016)

300D BiLSTM: 83% accuracy

(Liu et al., 2016)



Bowman et al. (2015)

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

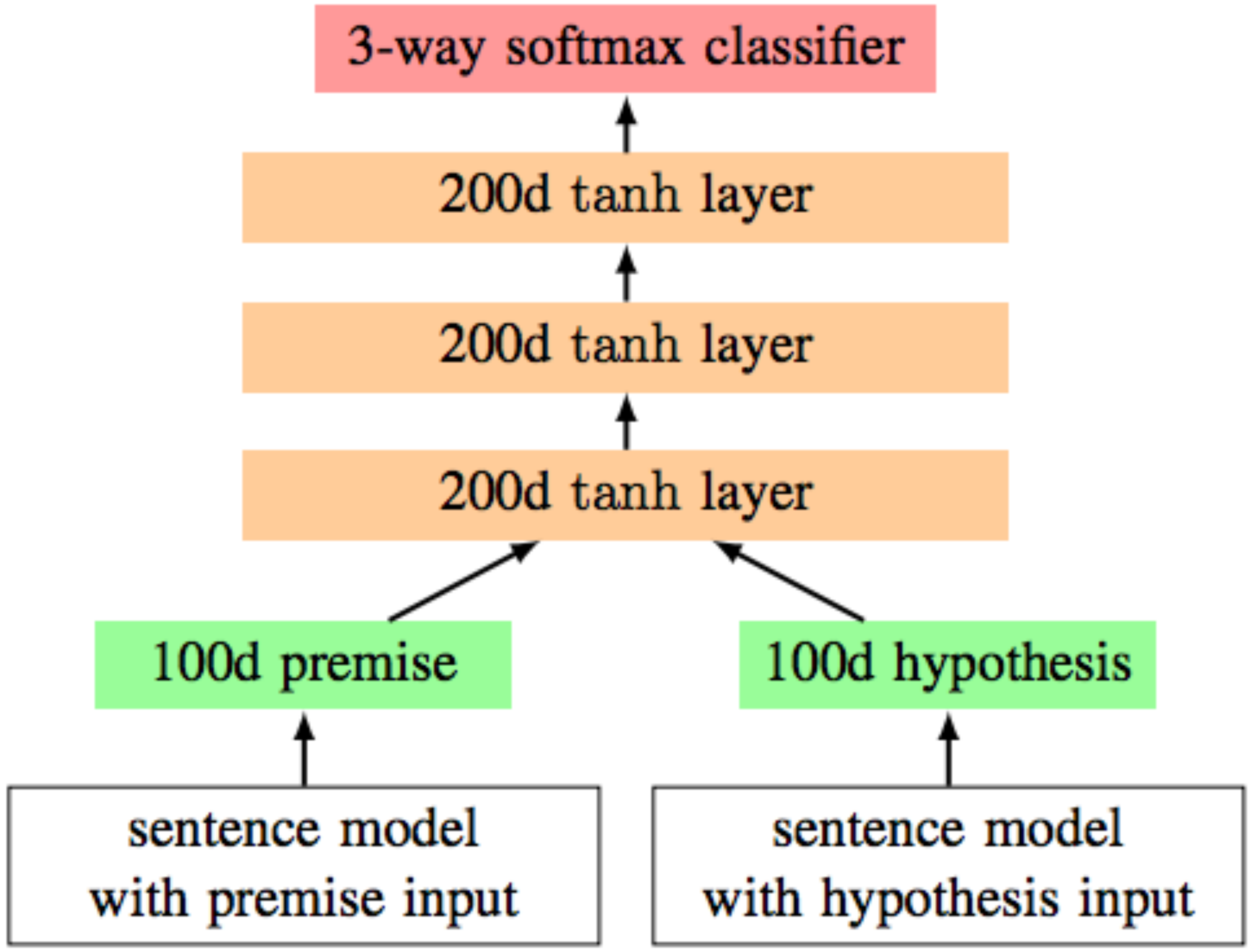
300D LSTM: 80% accuracy

(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