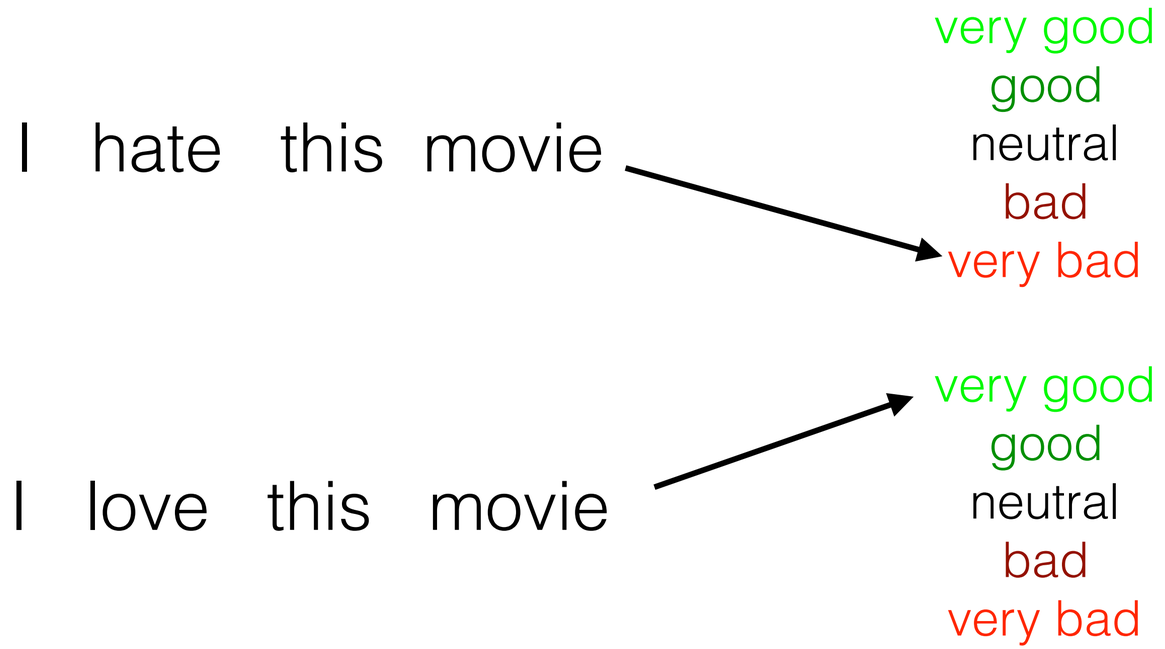
Neural Networks for Text

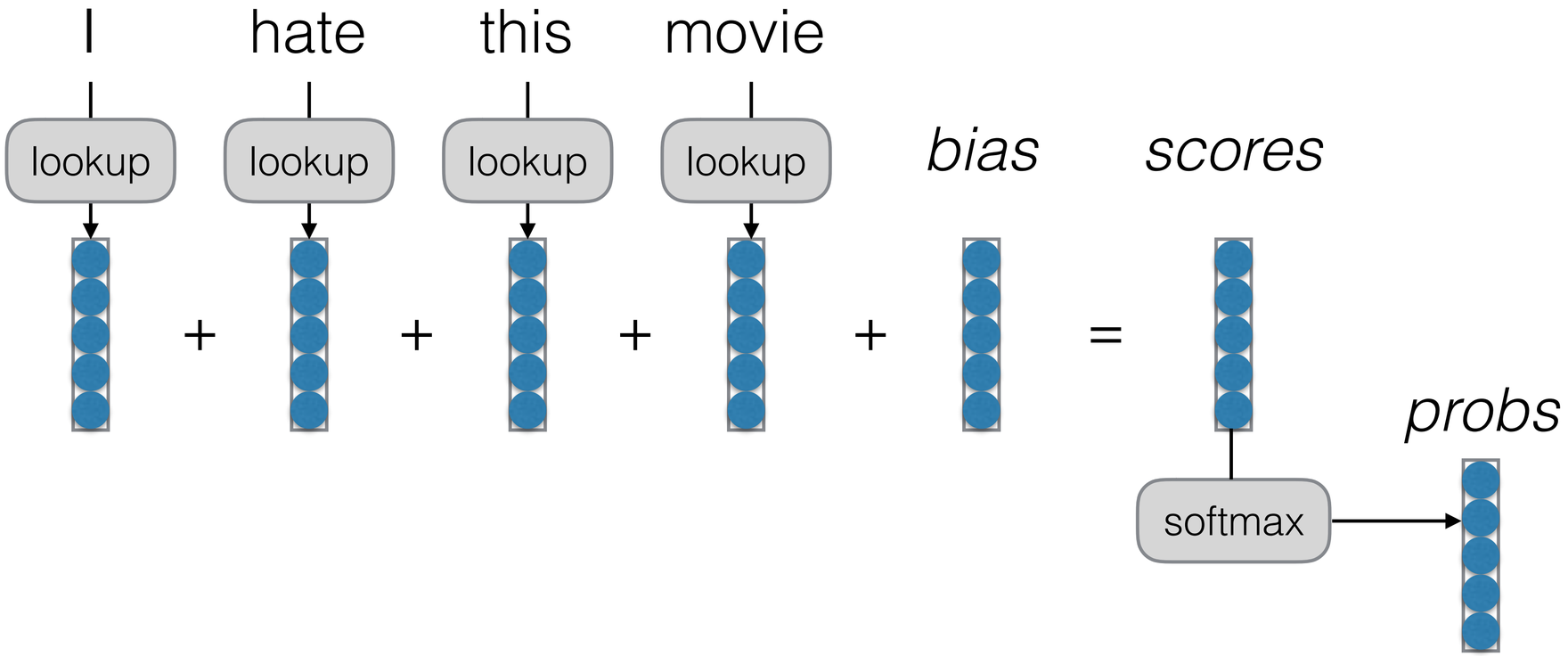
Instructor: Alan Ritter

Slides Adapted from Graham Neubig and Greg Durrett

An Example Prediction Problem: Sentence Classification



A First Try: Bag of Words (BOW)



What do Our Vectors Represent?

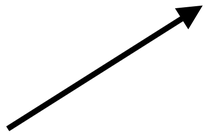
- Each word has its own 5 elements corresponding to [very good, good, neutral, bad, very bad]
- “hate” will have a high value for “very bad”, etc.

Build It, Break It

I don’t love this movie

very good
good
neutral
bad
very bad

There’s nothing I don’t love about this movie

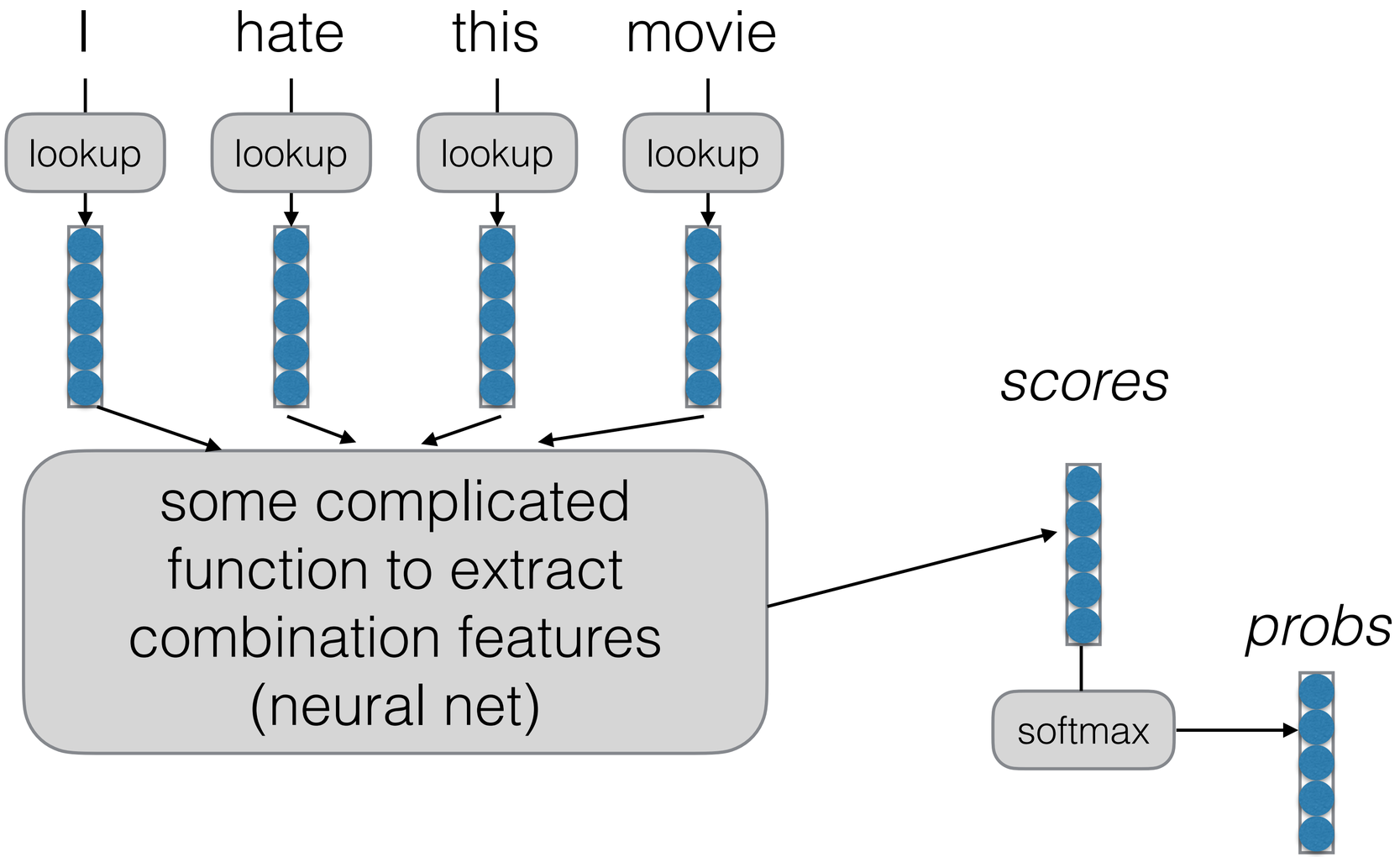


very good
good
neutral
bad
very bad

Combination Features

- Does it contain “don’t” and “love”?
- Does it contain “don’t”, “i”, “love”, and “nothing”?

Basic Idea of Neural Networks (for NLP Prediction Tasks)



EMBEDDING

CLASS `torch.nn.Embedding(`*`num_embeddings`*, *`embedding_dim`*, *`padding_idx=None`*, *`max_norm=None`*, *`norm_type=2.0`*, *`scale_grad_by_freq=False`*, *`sparse=False`*, *`_weight=None`*, *`device=None`*, *`dtype=None`*`)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int*, *optional*) – If specified, the entries at `padding_idx` do not contribute to the

1.10.0 ▼

Search Docs

Notes [+]

Language Bindings [+]

Python API [-]

torch
torch.nn
torch.nn.functional
torch.Tensor
Tensor Attributes
Tensor Views
torch.autograd
torch.cuda

EMBEDDING

Embedding

CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, device=None, dtype=None)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often use
list of indices, and the o

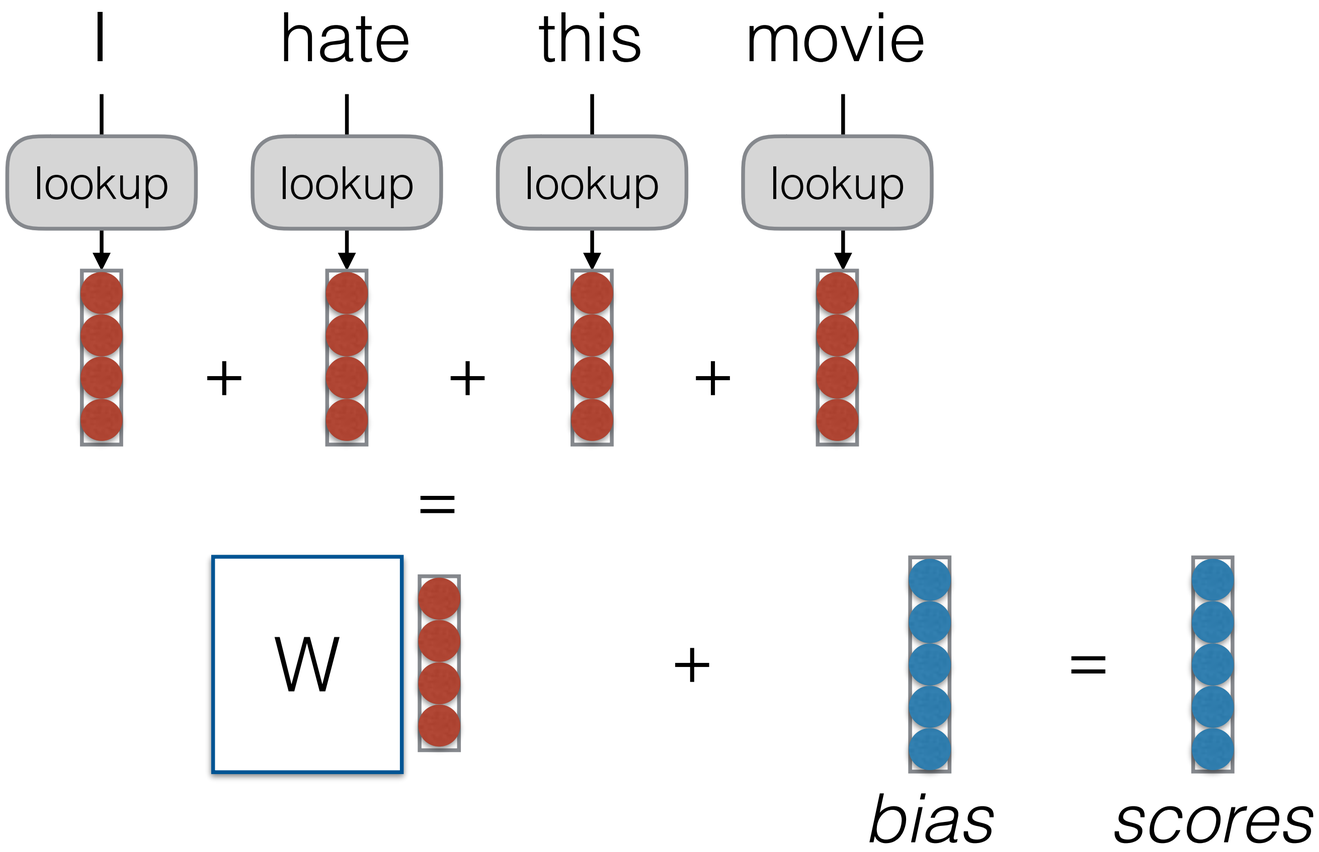
Parameters

- **num_emb**
- **embeddi**
- **padding_**

```
>>> # an Embedding module containing 10 tensors of size 3
>>> embedding = nn.Embedding(10, 3)
>>> # a batch of 2 samples of 4 indices each
>>> input = torch.LongTensor([[1,2,4,5],[4,3,2,9]])
>>> embedding(input)
tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
         [ 0.9124, -2.3616,  1.1151]]])
```

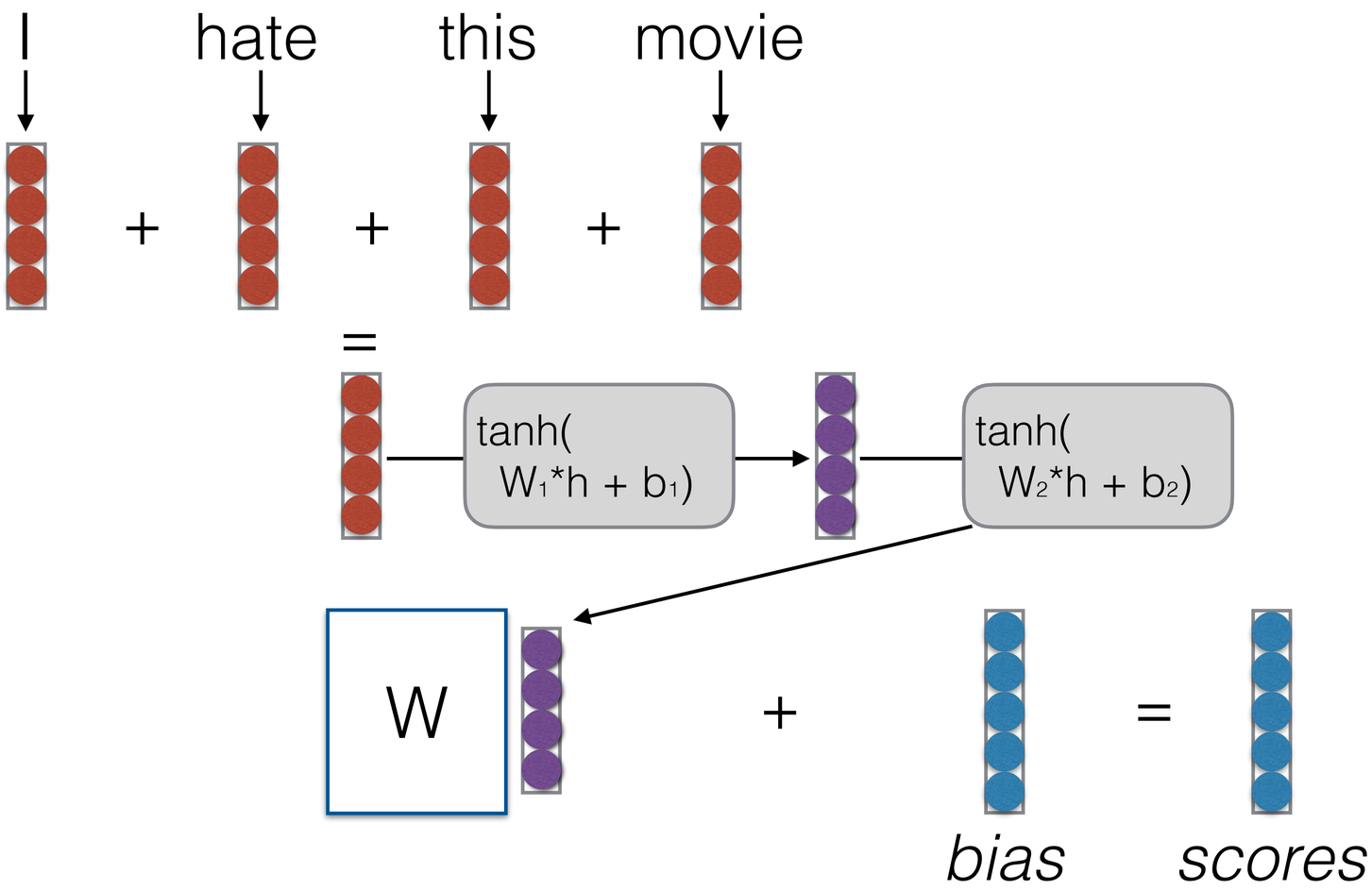
Continuous Bag of Words (CBOW)



What do Our Vectors Represent?

- Each vector has “features” (e.g. is this an animate object? is this a positive word, etc.)
- We sum these features, then use these to make predictions
- Still no combination features: only the expressive power of a linear model, but dimension reduced

Deep CBOW

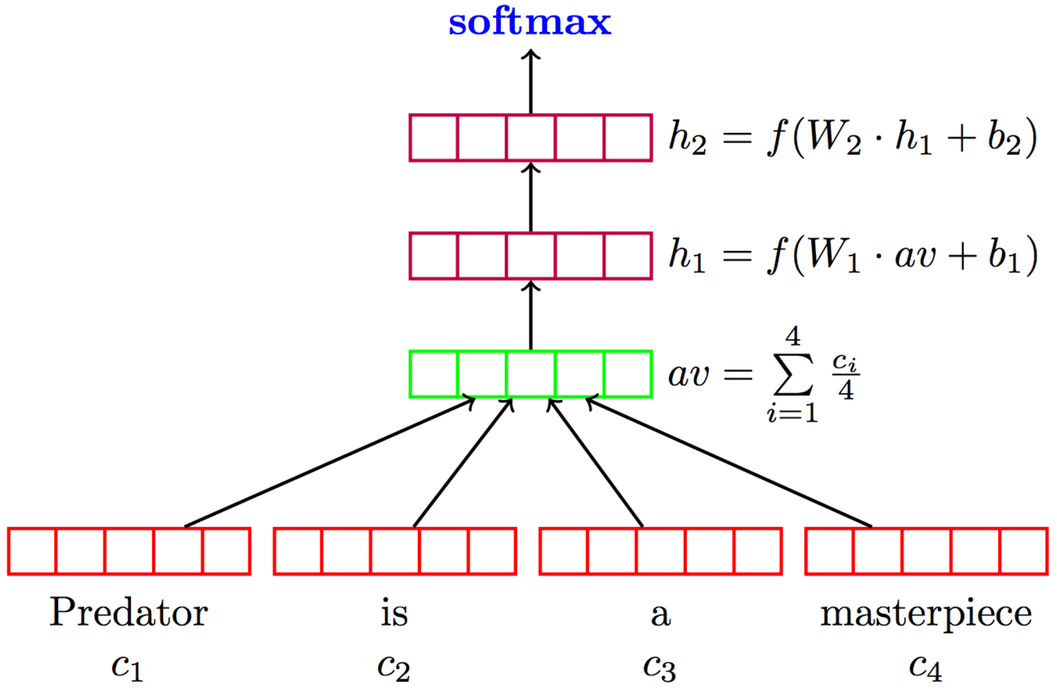


What do Our Vectors Represent?

- Now things are more interesting!
- We can learn feature combinations (a node in the second layer might be “feature 1 AND feature 5 are active”)
- e.g. capture things such as “not” AND “hate”

Sentiment Analysis

Deep Averaging Networks: feedforward neural network on average of word embeddings from input



Iyyer et al. (2015)

Sentiment Analysis

Model	RT	SST fine	SST bin	IMDB	Time (s)
DAN-ROOT	—	46.9	85.7	—	**31**
DAN-RAND	77.3	45.4	83.2	88.8	136
DAN	80.3	47.7	86.3	89.4	136
NBOW-RAND	76.2	42.3	81.4	88.9	91
NBOW	79.0	43.6	83.6	89.0	91
BiNB	—	41.9	83.1	—	—
NBSVM-bi	79.4	—	—	91.2	—
RecNN*	77.7	43.2	82.4	—	—
RecNTN*	—	45.7	85.4	—	—
DRecNN	—	49.8	86.6	—	431
TreeLSTM	—	**50.6**	86.9	—	—
DCNN*	—	48.5	86.9	89.4	—
PVEC*	—	48.7	87.8	**92.6**	—
CNN-MC	**81.1**	47.4	**88.1**	—	2,452
WRRBM*	—	—	—	89.2	—

Bag-of-words

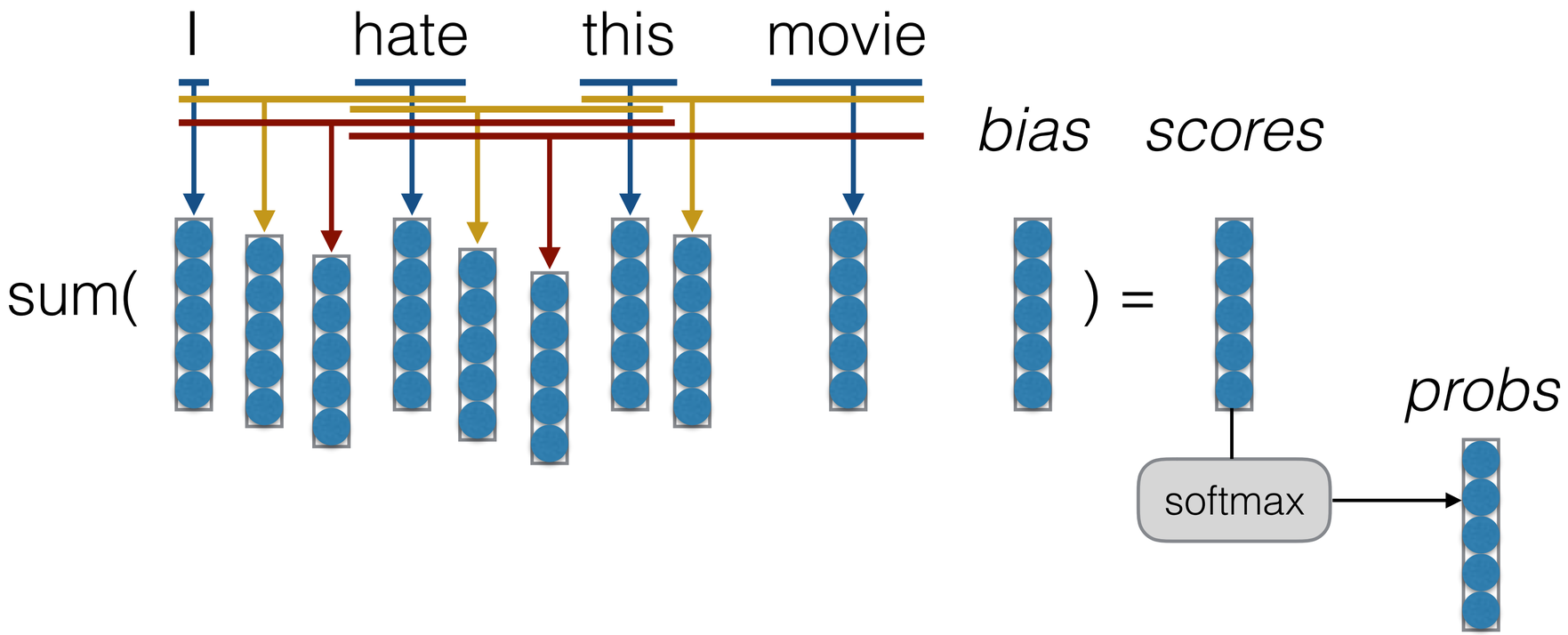
Tree RNNs / CNNS / LSTMS

Iyyer et al. (2015)

Wang and Manning (2012)

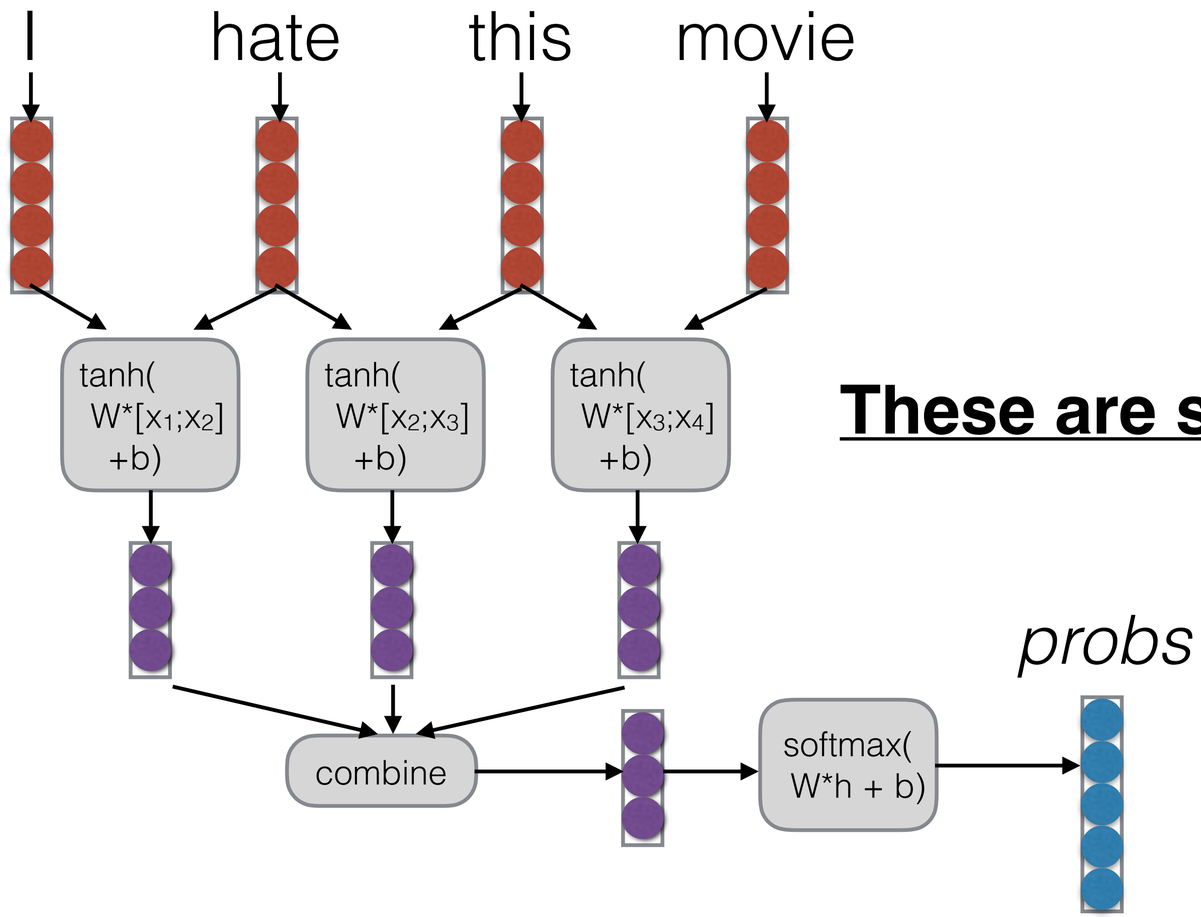
Kim (2014)

Bag of n-grams



Time Delay Neural Networks

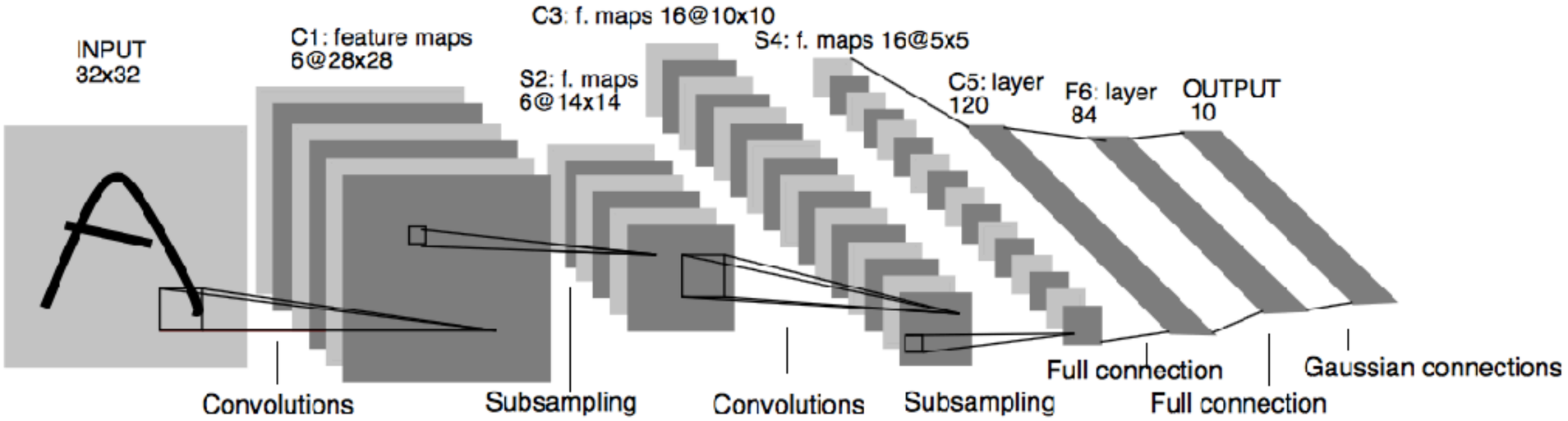
(Waibel et al. 1989)



These are soft 2-grams!

Convolutional Networks

(LeCun et al. 1997)



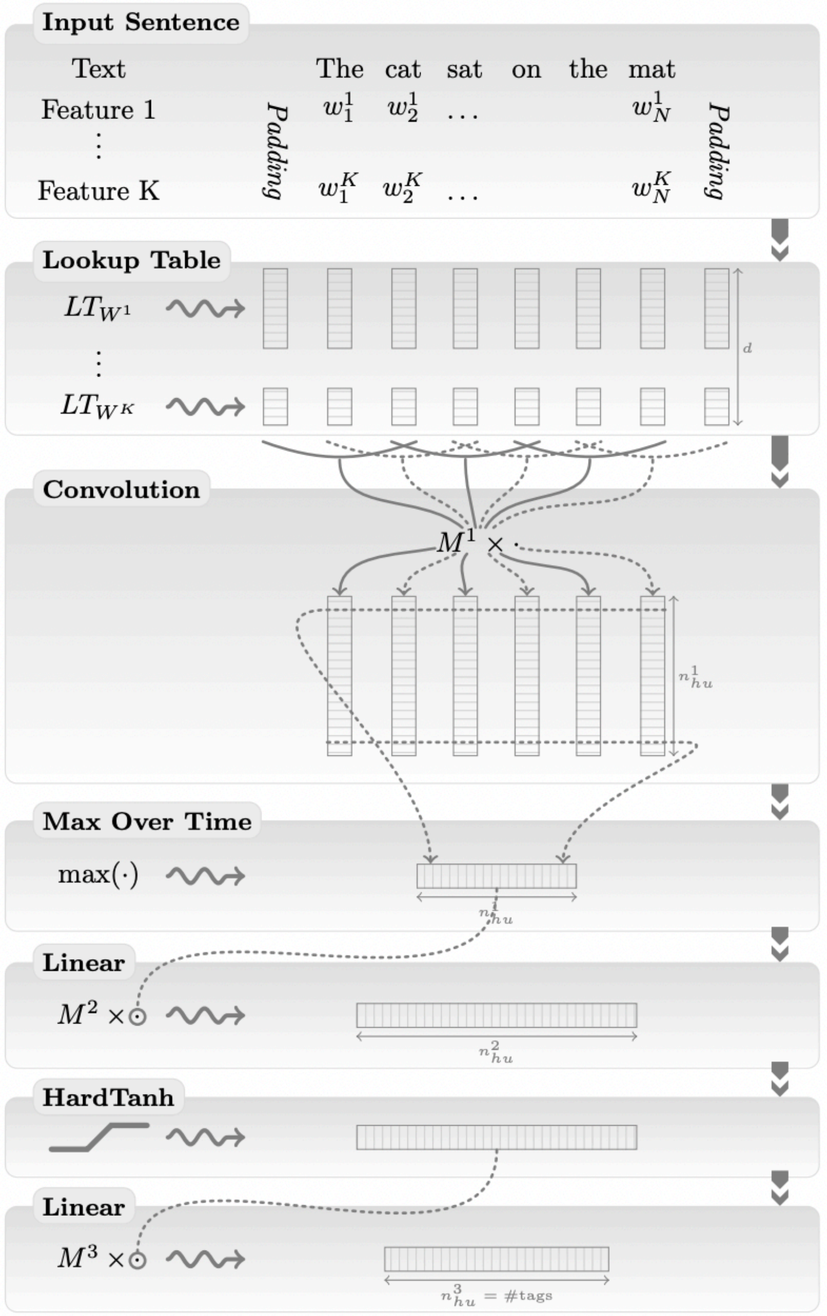
Parameter extraction performs a 2D sweep, not 1D

CNNs for Text
(Collobert and Weston 2011)

- 1D convolution ≈ Time Delay Neural Network
    - But often uses terminology/functions borrowed from image processing
- Two main paradigms:
    - **Context window modeling:** For tagging, etc. get the surrounding context before tagging
    - **Sentence modeling:** Do convolution to extract n-grams, pooling to combine over whole sentence

CNNs for Sentence Modeling

(Collobert and Weston 2011)



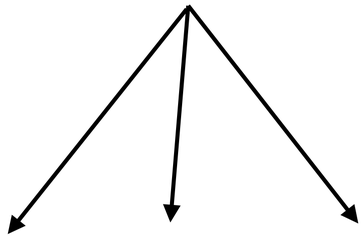
Sentence Classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

Kim (2014)

Sentence Classification

movie review
sentiment



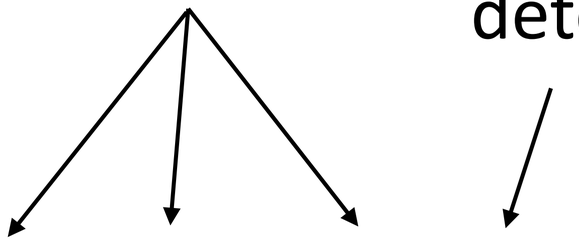
Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

Kim (2014)

Sentence Classification

movie review sentiment

subjectivity/objectivity detection



Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

Kim (2014)

Sentence Classification

movie review sentiment

subjectivity/objectivity detection

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

question type classification

Kim (2014)

Sentence Classification

movie review sentiment

subjectivity/objectivity detection

product reviews

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

question type classification

Kim (2014)

Sentence Classification

movie review sentiment

subjectivity/objectivity detection

product reviews

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3

question type classification

- Also effective at document-level text classification

Kim (2014)

RNN Basics

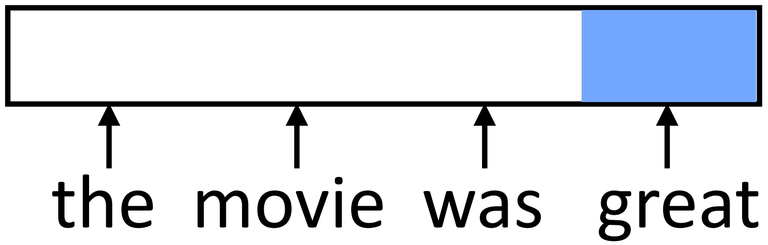
RNN Motivation

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

the movie was great

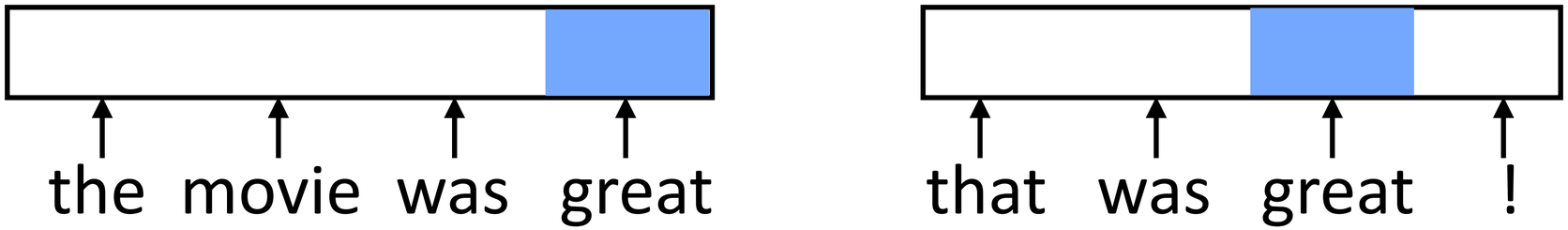
RNN Motivation

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



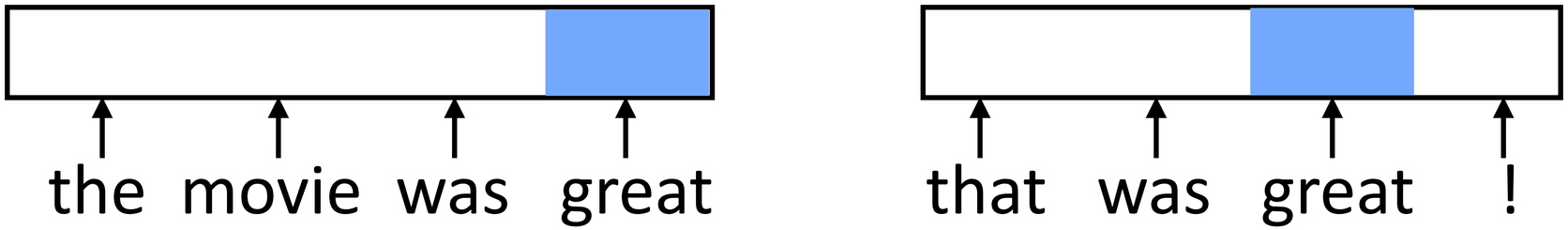
RNN Motivation

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



RNN Motivation

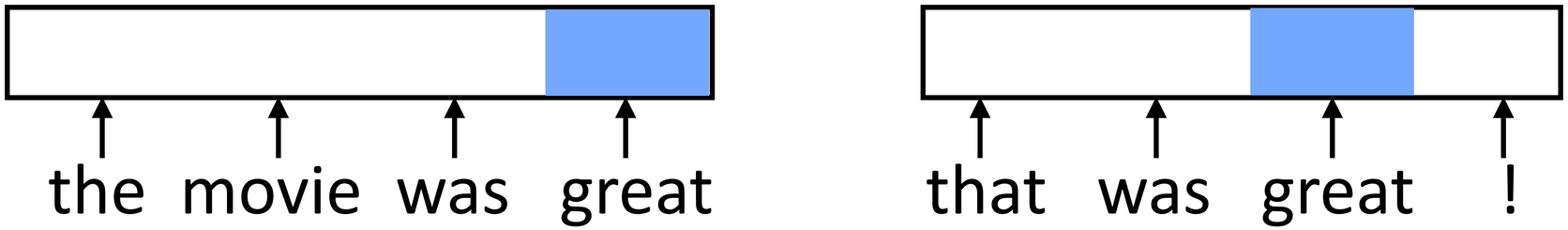
- Feedforward NNs can't handle variable length input: each position in the feature vector has fixed semantics



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

RNN Motivation

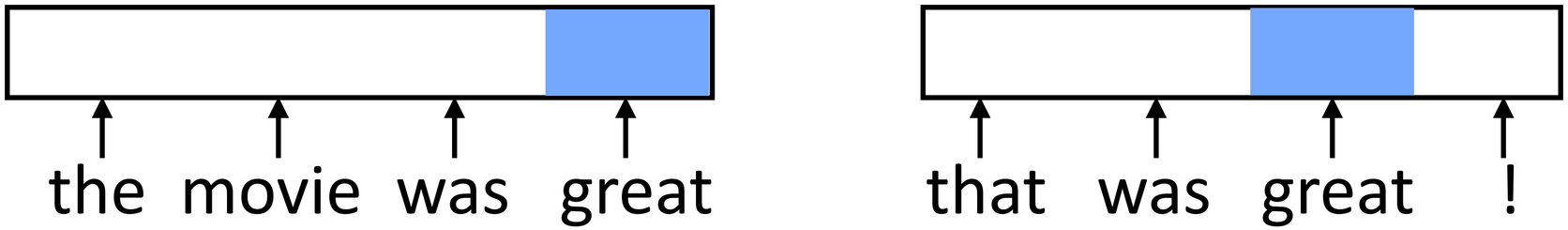
- Feedforward NNs can't handle variable length input: each position in the feature vector has fixed semantics



- 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



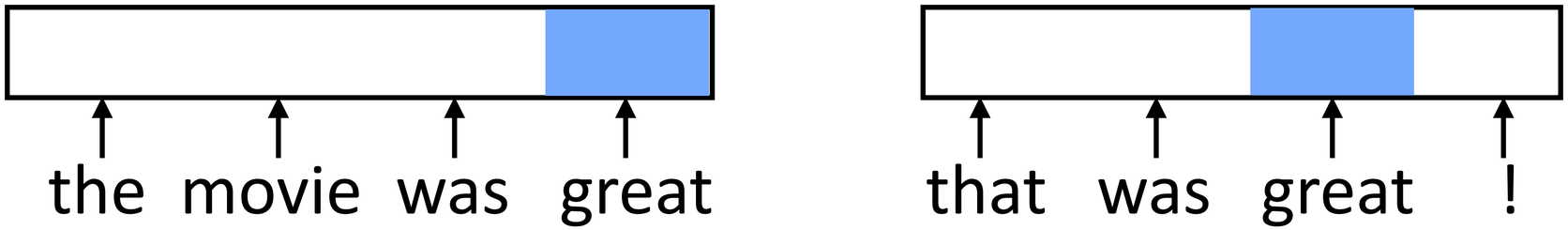
- These don't look related (*great* is in two different orthogonal subspaces)

- 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)

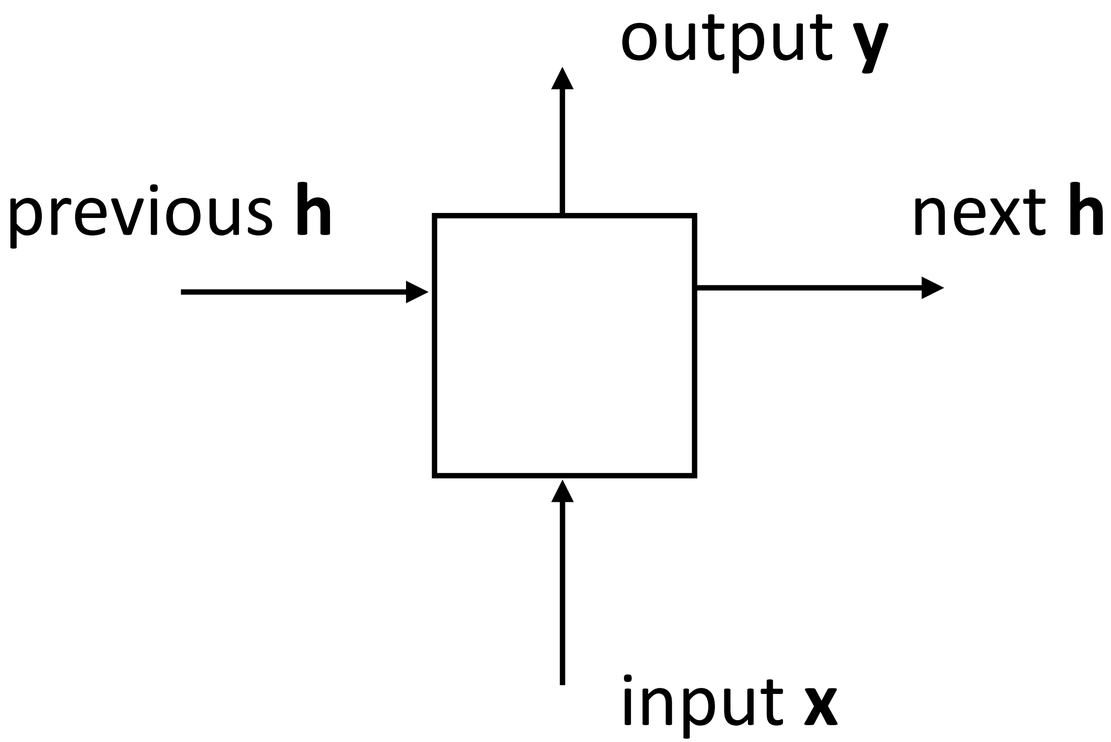
- Instead, we need to:

1) Process each word in a uniform way

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

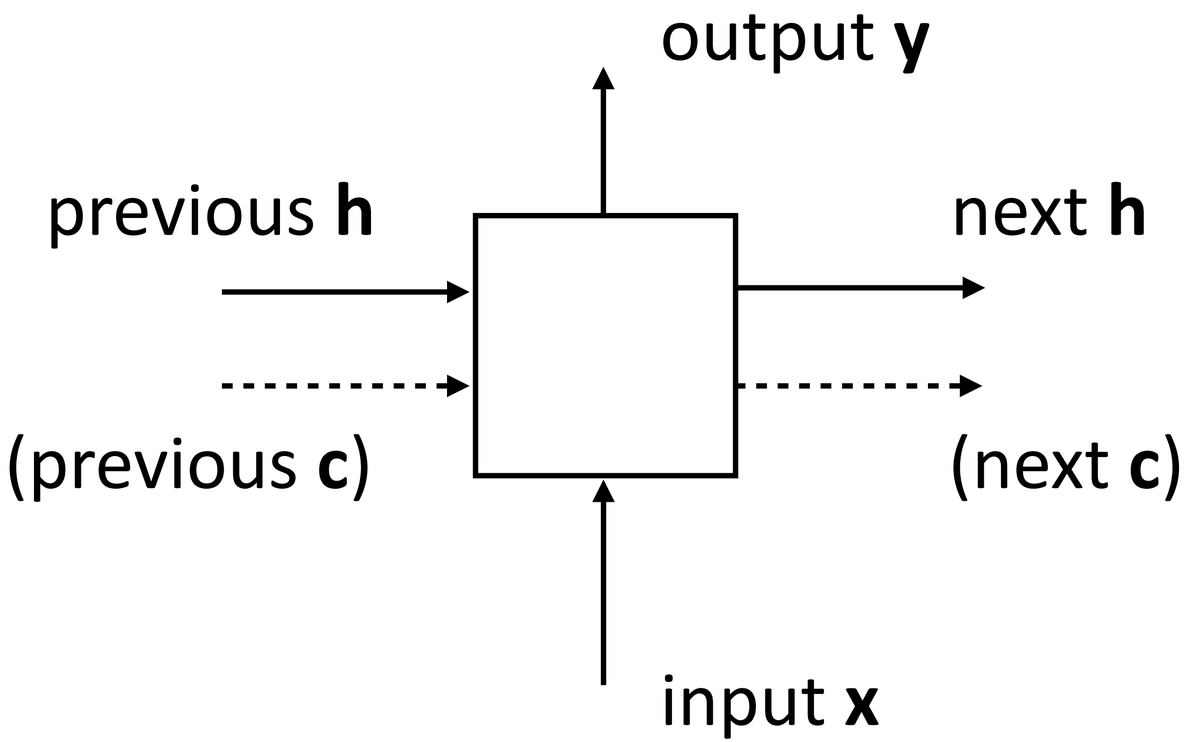
RNN Abstraction

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



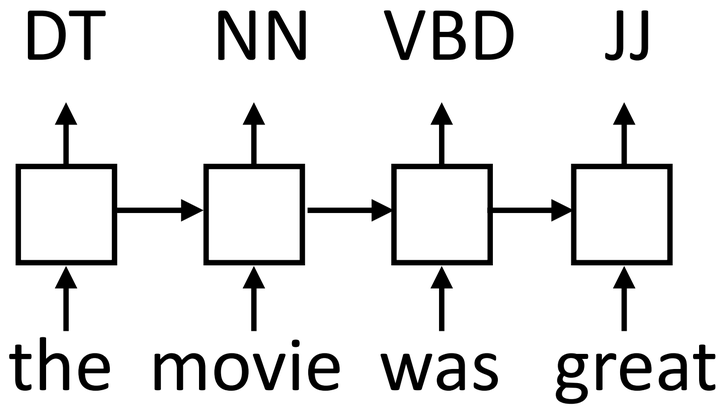
RNN Abstraction

- 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

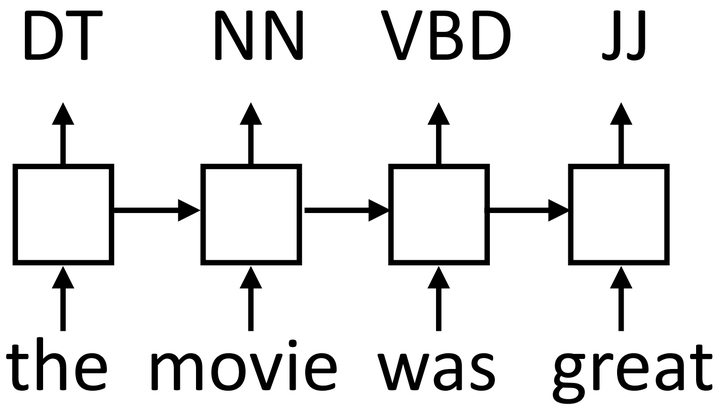
- Transducer: make some prediction for each element in a sequence



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

RNN Uses

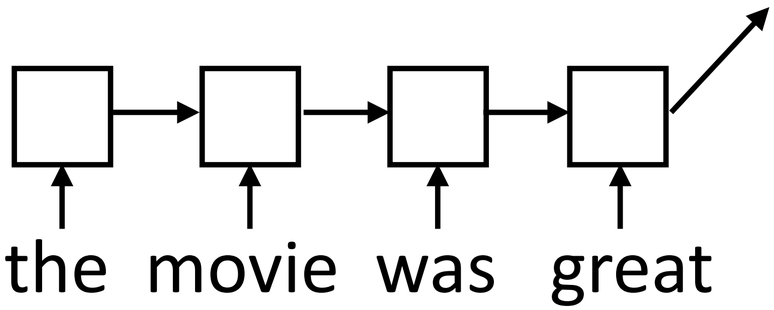
- Transducer: make some prediction for each element in a sequence



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

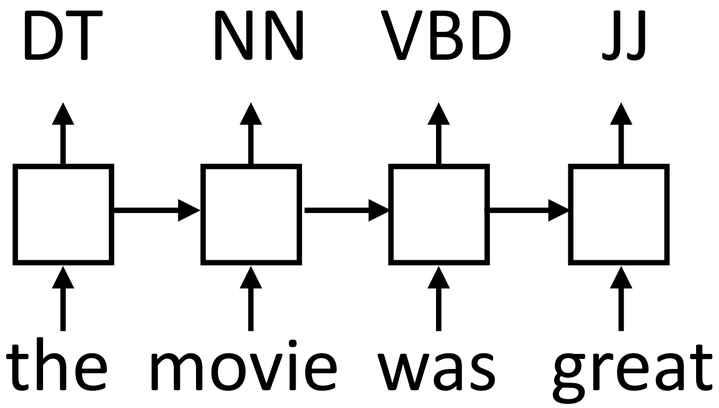
- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

predict sentiment (matmul + softmax)



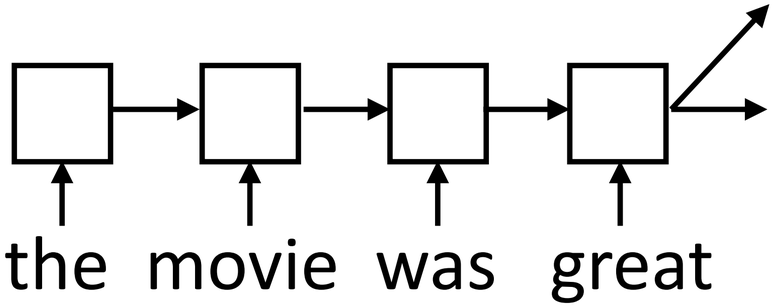
RNN Uses

- 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

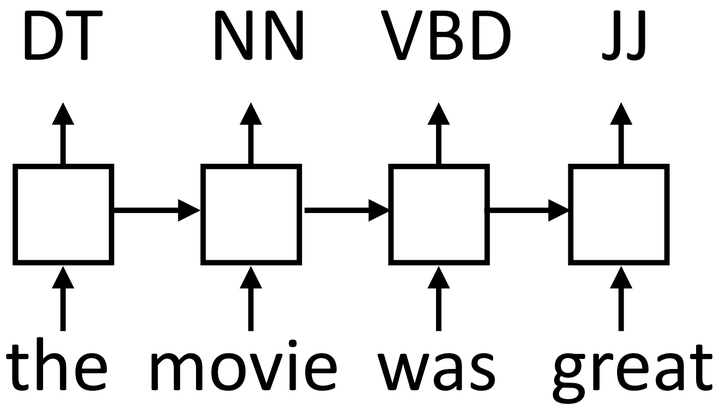


predict sentiment (matmul + softmax)

translate

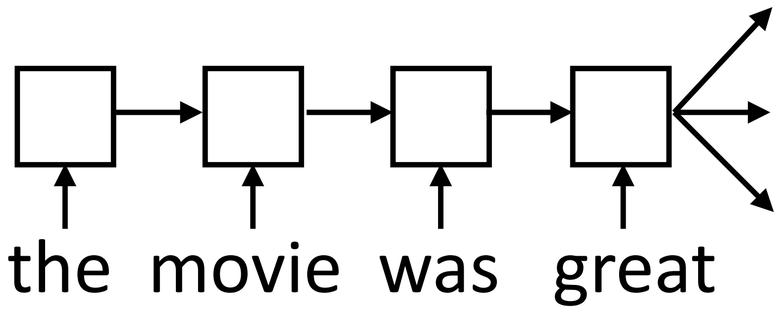
RNN Uses

- 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

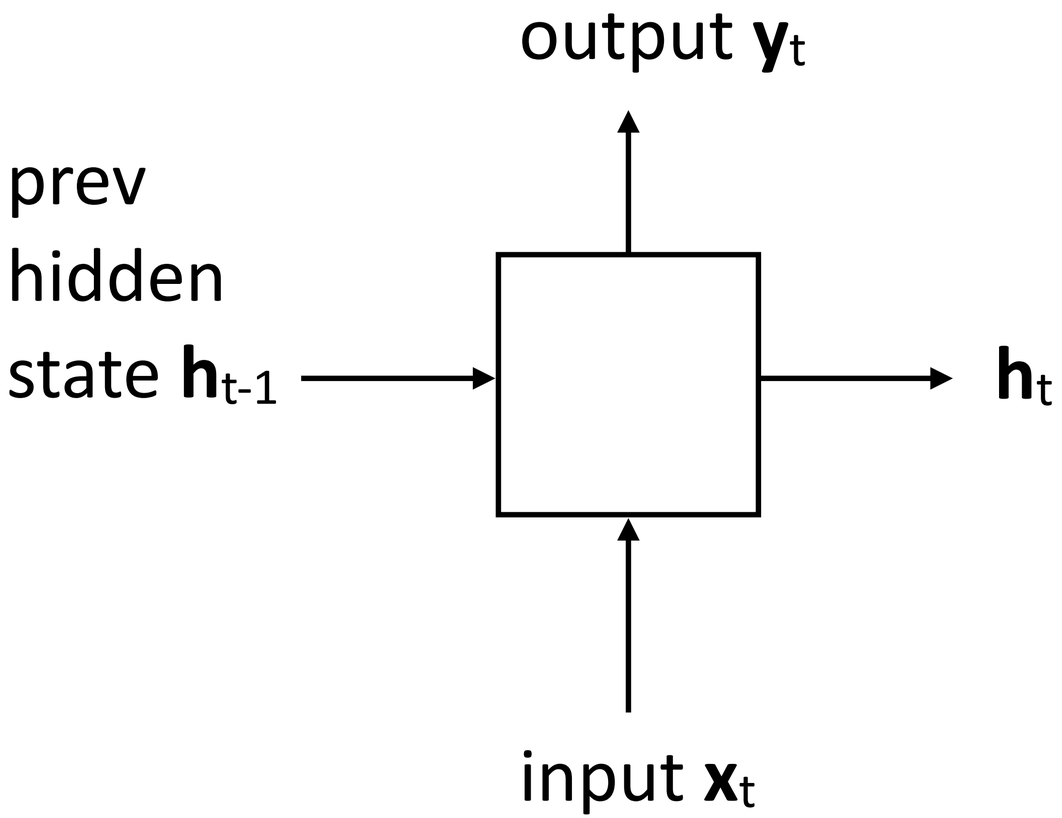


predict sentiment (matmul + softmax)

translate

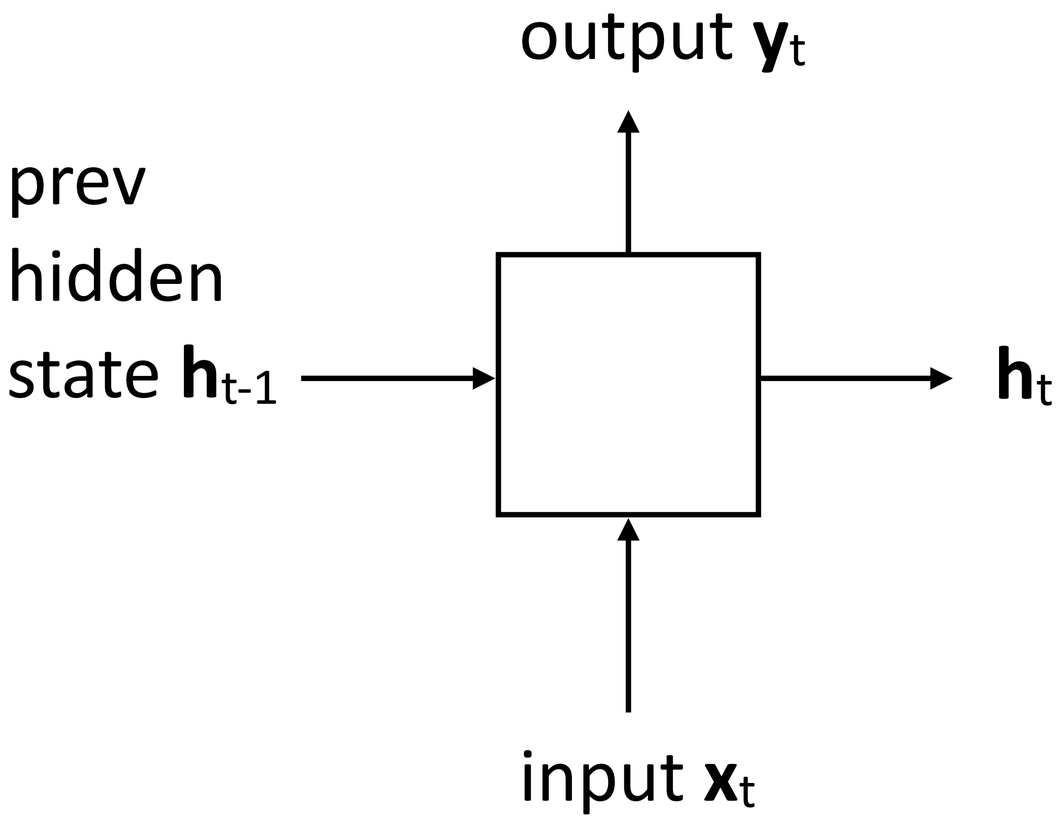
paraphrase/compress

Elman Networks



Elman (1990)

Elman Networks

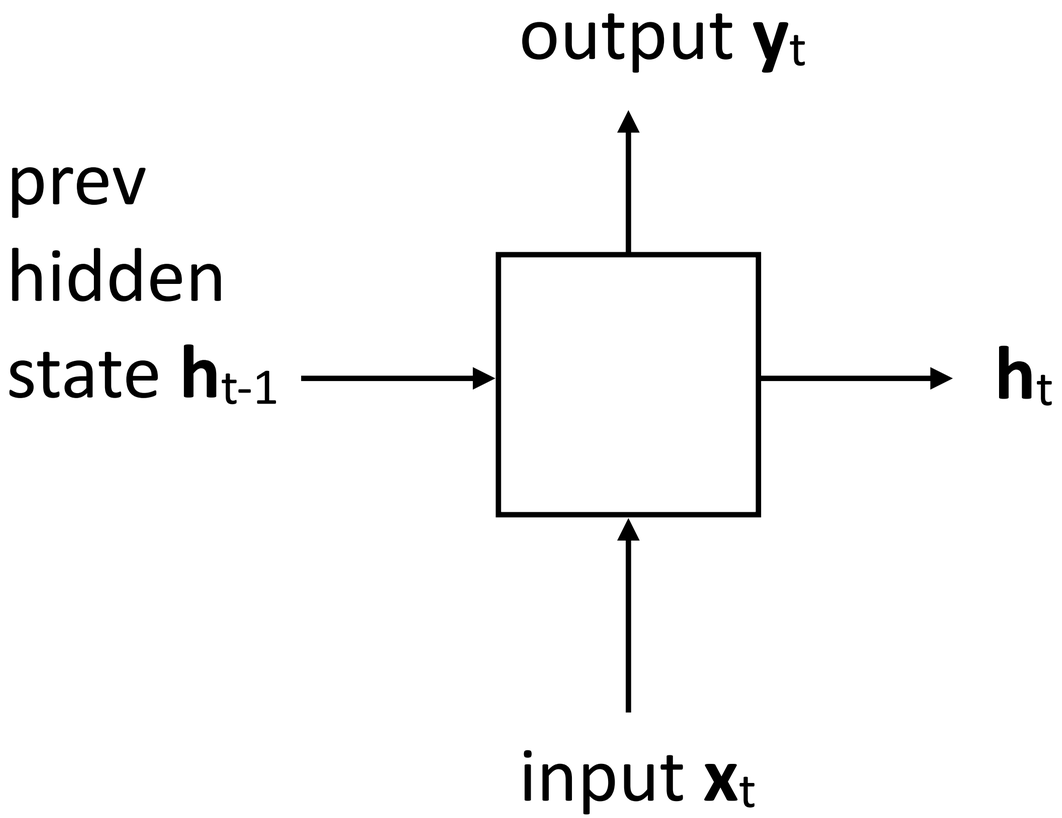


$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

- Updates hidden state based on input and current hidden state

Elman (1990)

Elman Networks



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

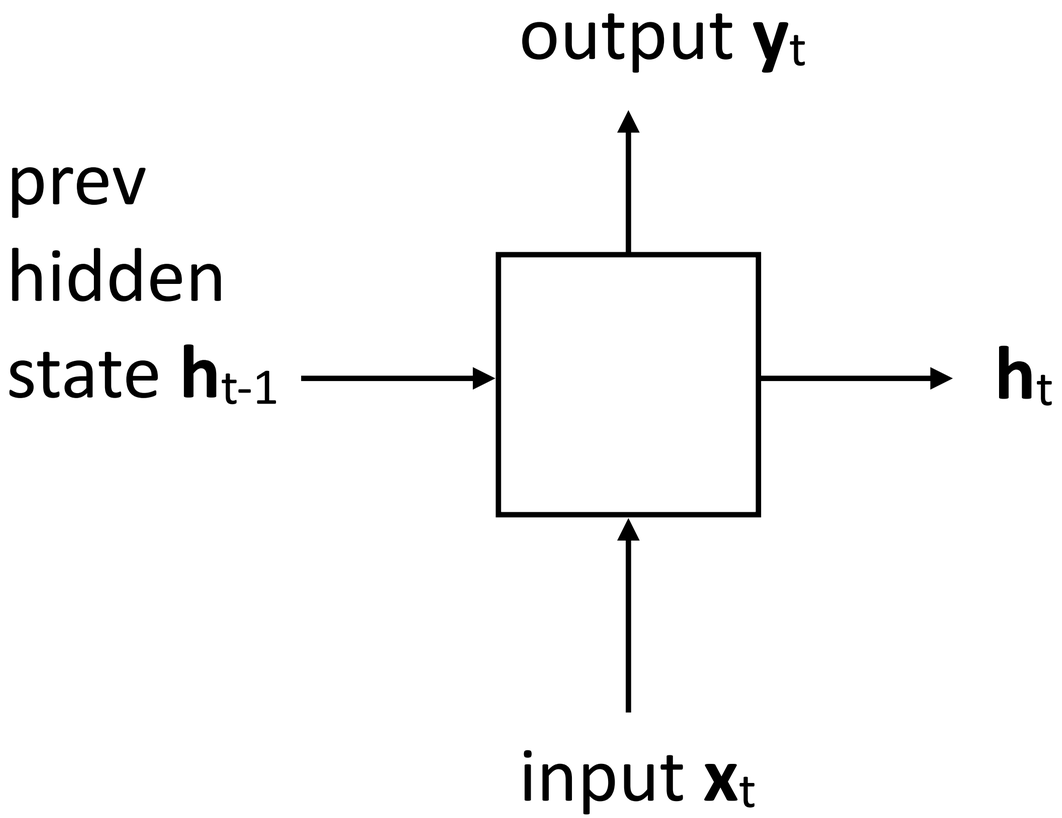
- Updates hidden state based on input and current hidden state

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

- 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

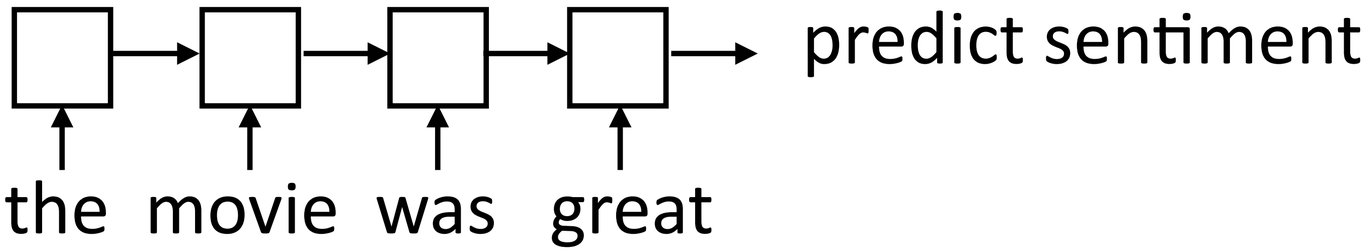
$$\mathbf{y}_t = \tanh(U\mathbf{h_t} + \mathbf{b}_y)$$

- Computes output from hidden state

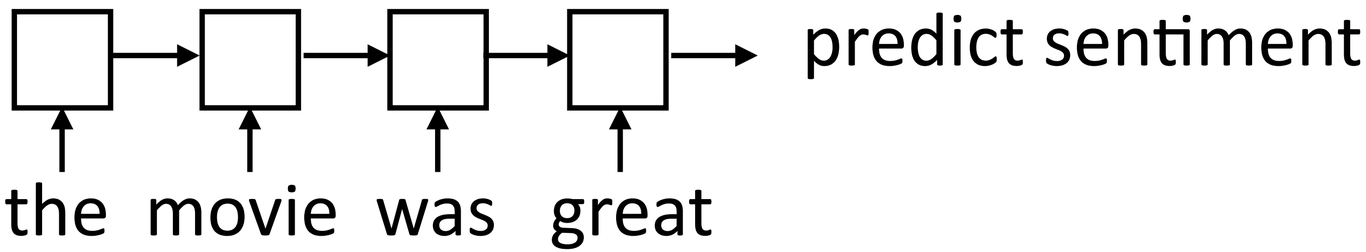
- Long history! (invented in the late 1980s)

Elman (1990)

Training Elman Networks

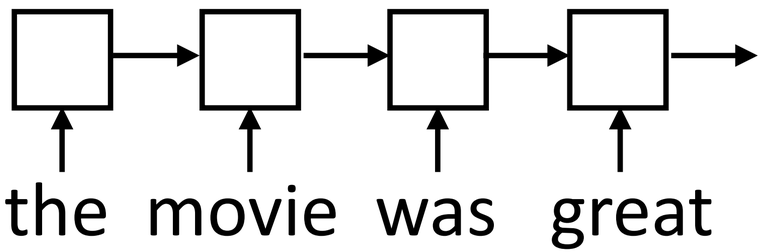


Training Elman Networks



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

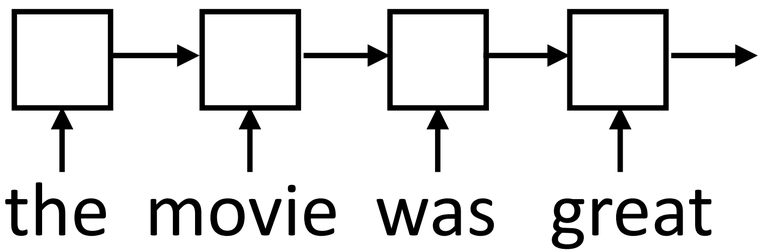
Training Elman Networks



- "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

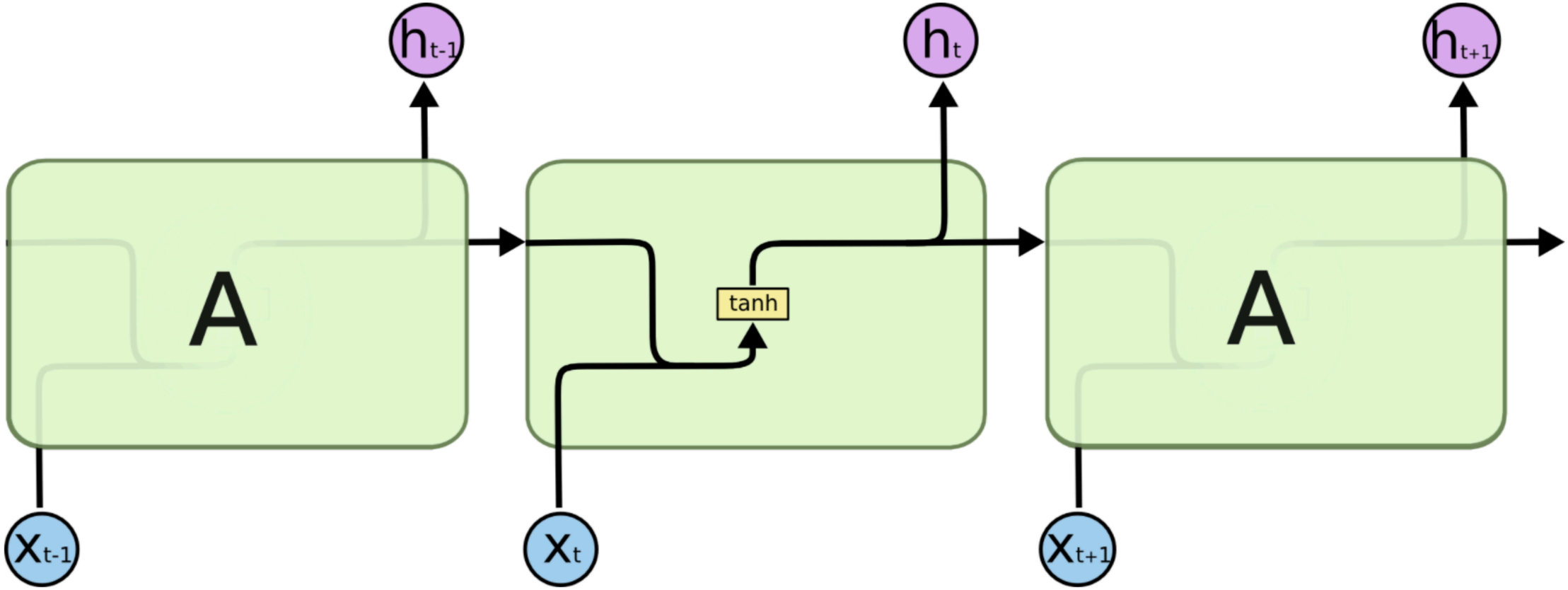


- "Backpropagation through time": build the network as one big computation graph, some parameters are shared
- RNN potentially needs to learn how to "remember" information for a long time!

it was my favorite movie of 2016, though it wasn't without problems -> +

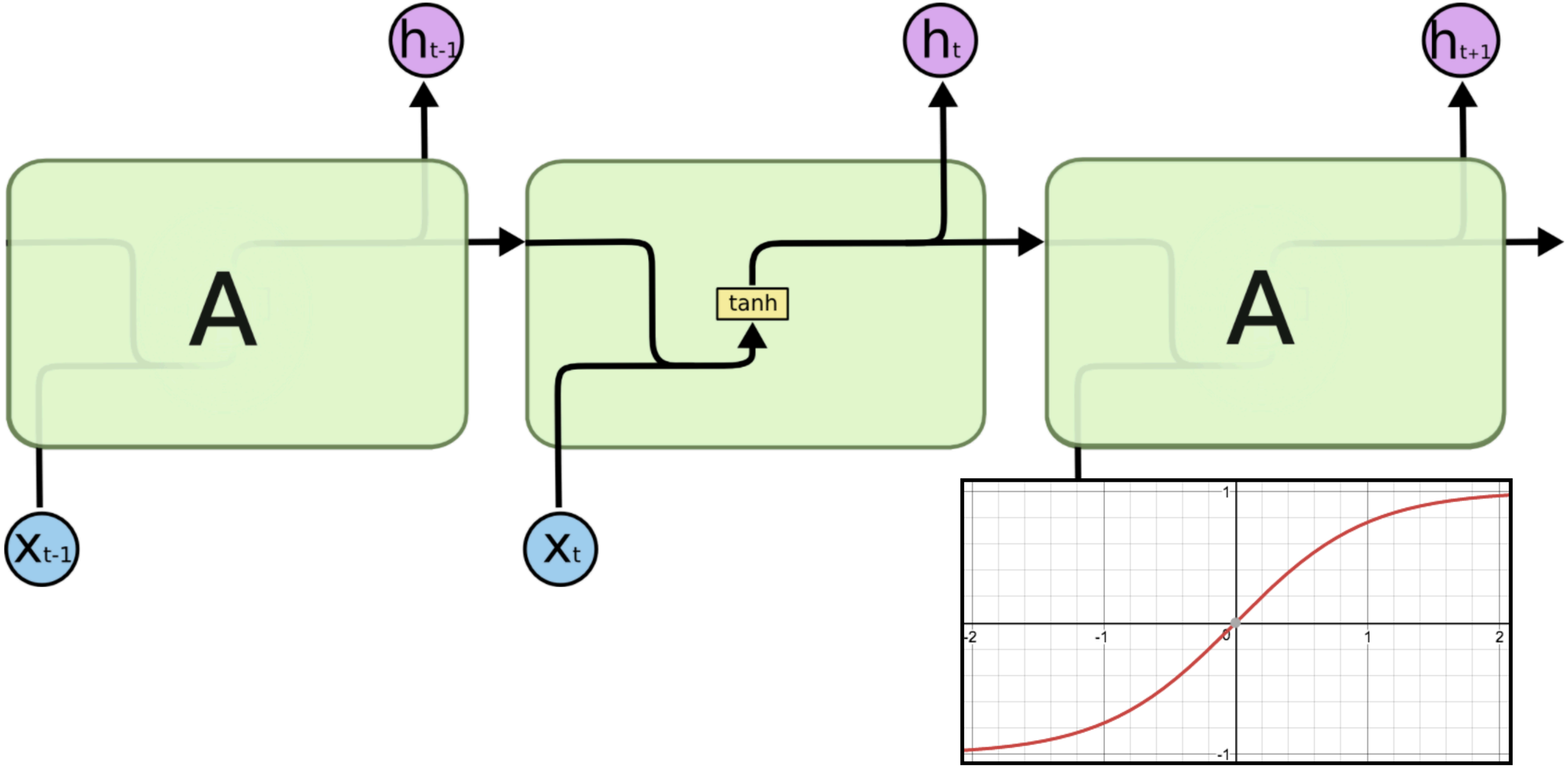
- "Correct" parameter update is to do a better job of remembering the sentiment of *favorite*

Vanishing Gradient



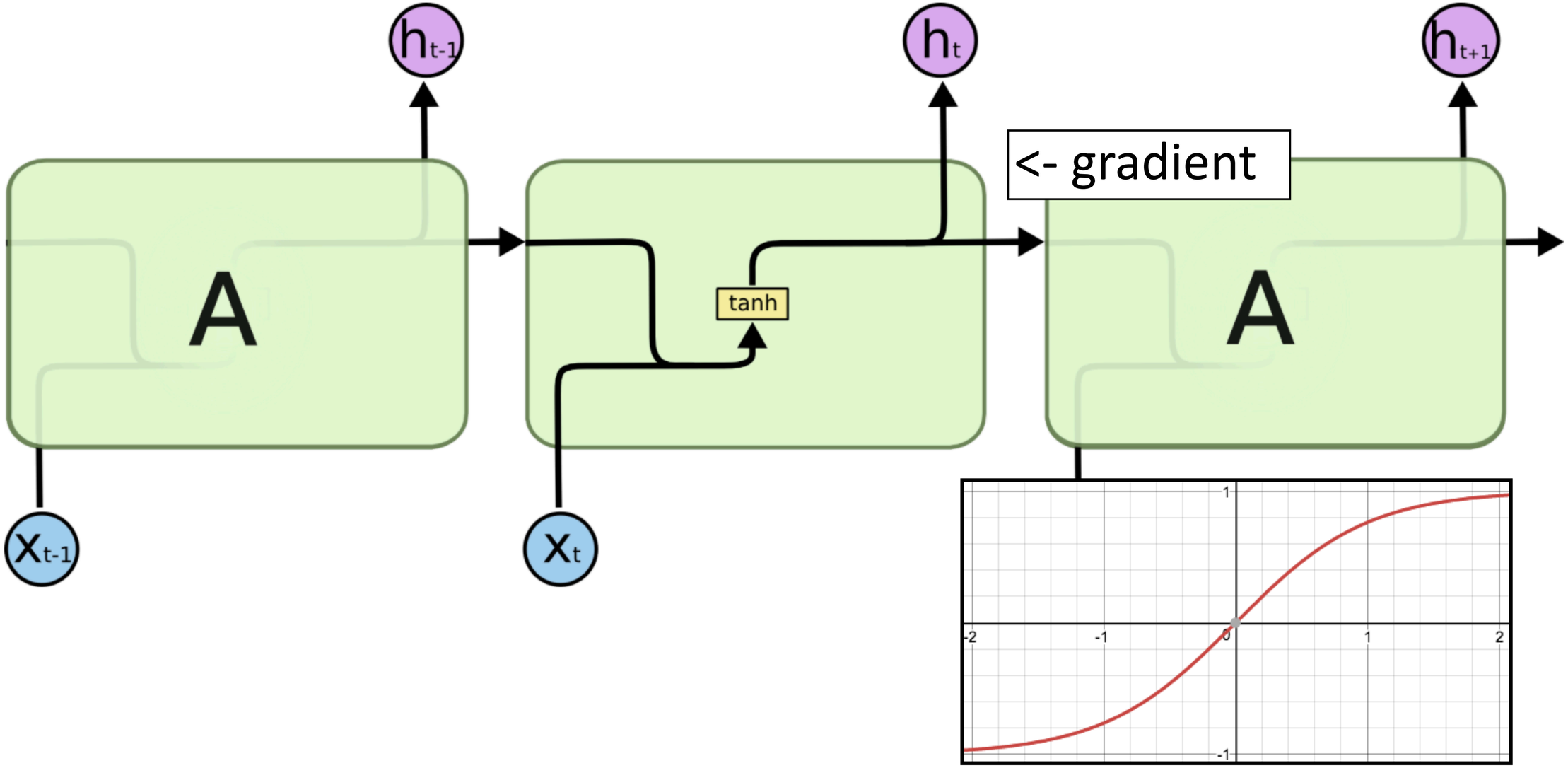
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient



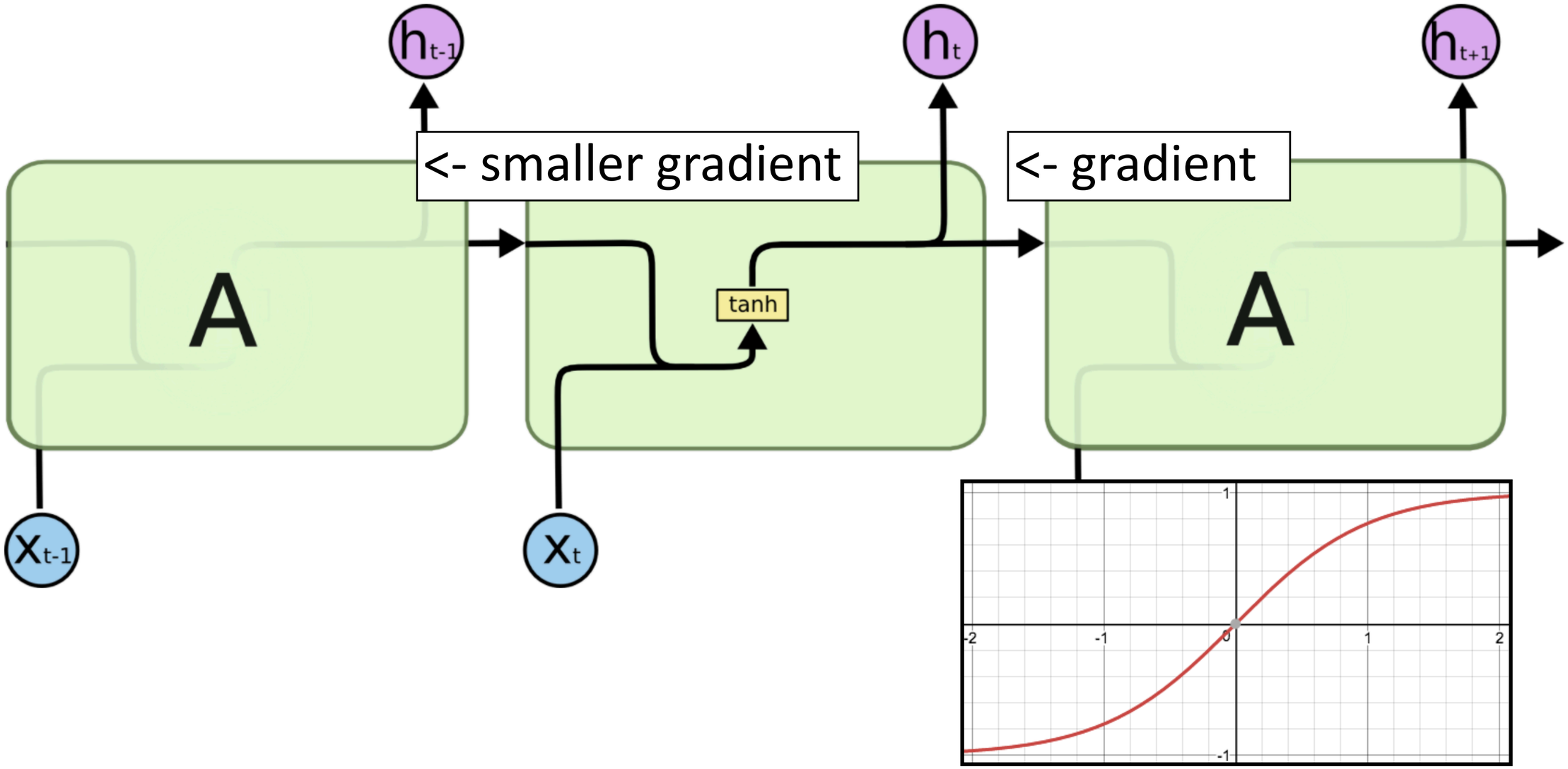
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing Gradient



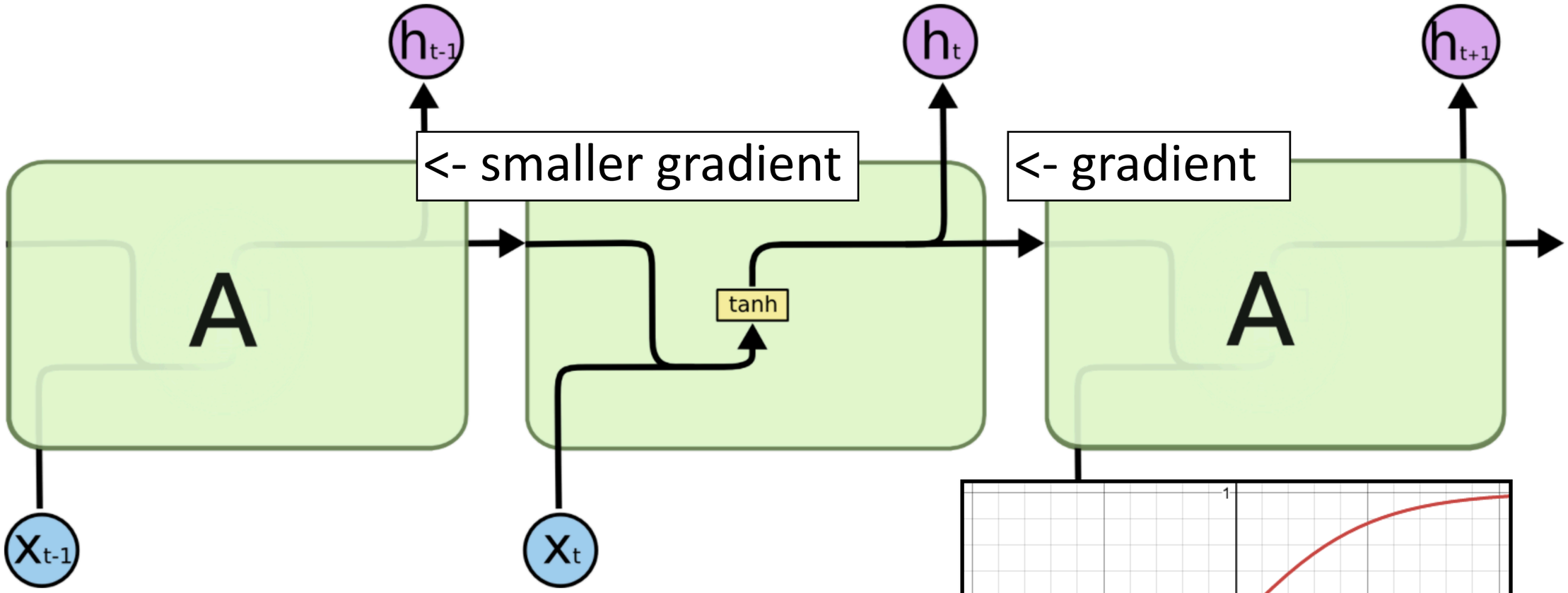
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

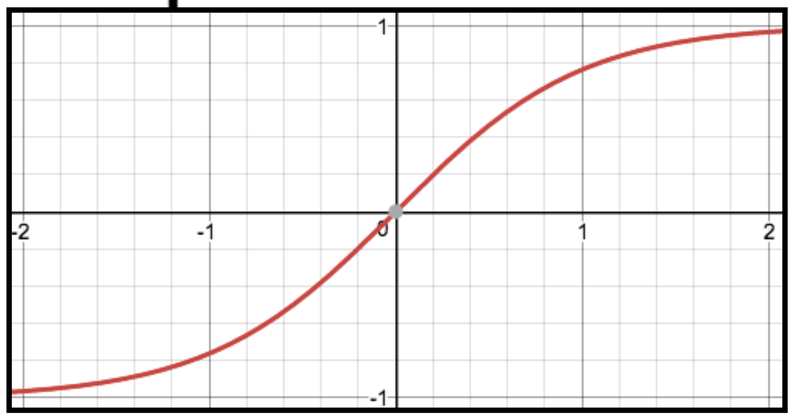
Vanishing Gradient



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

Vanishing Gradient

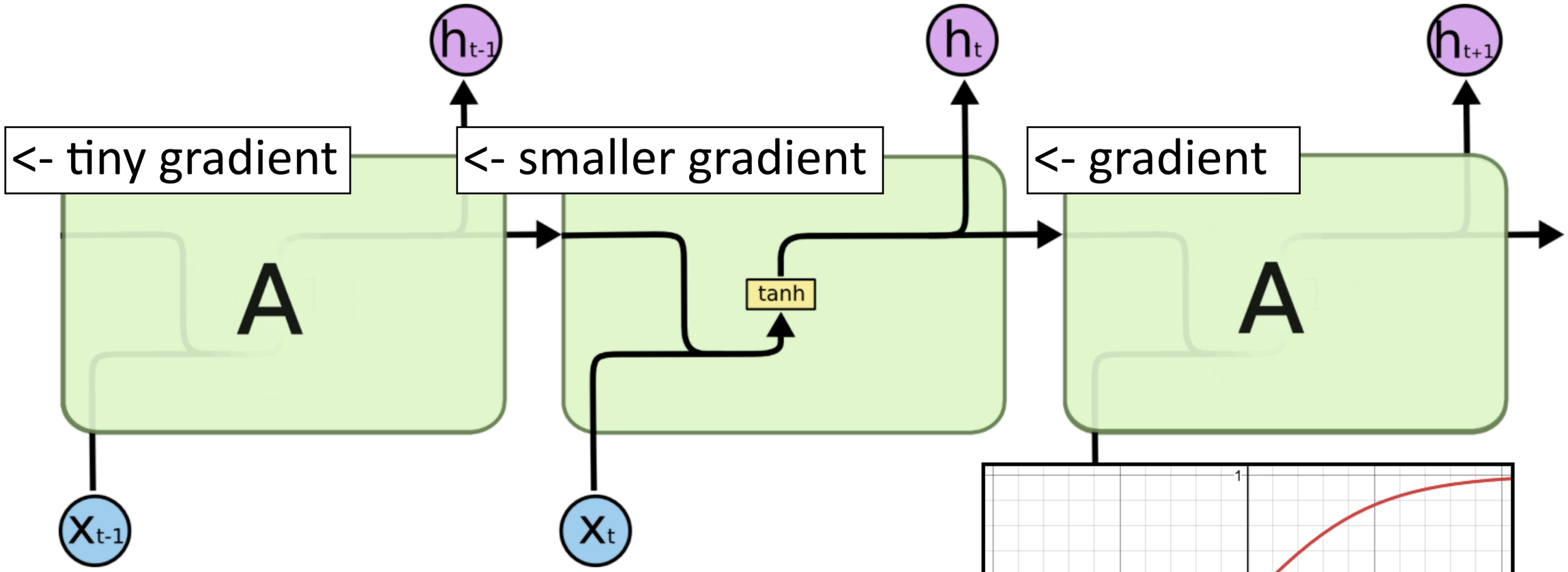




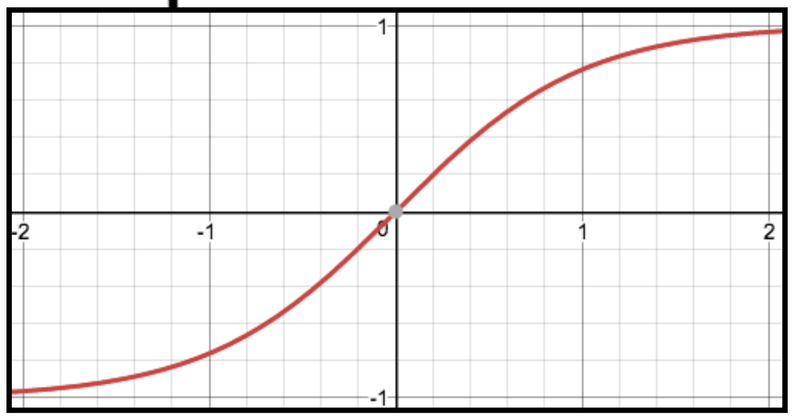
- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0

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

Vanishing Gradient

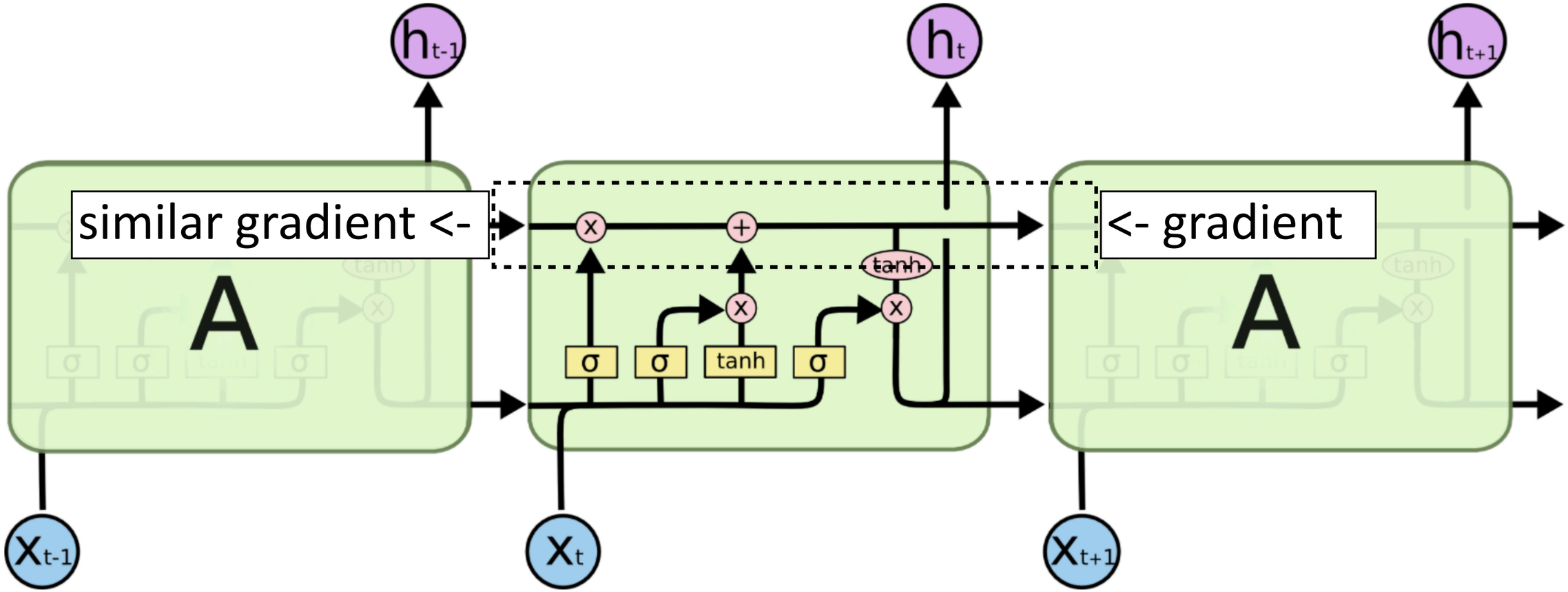


- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0



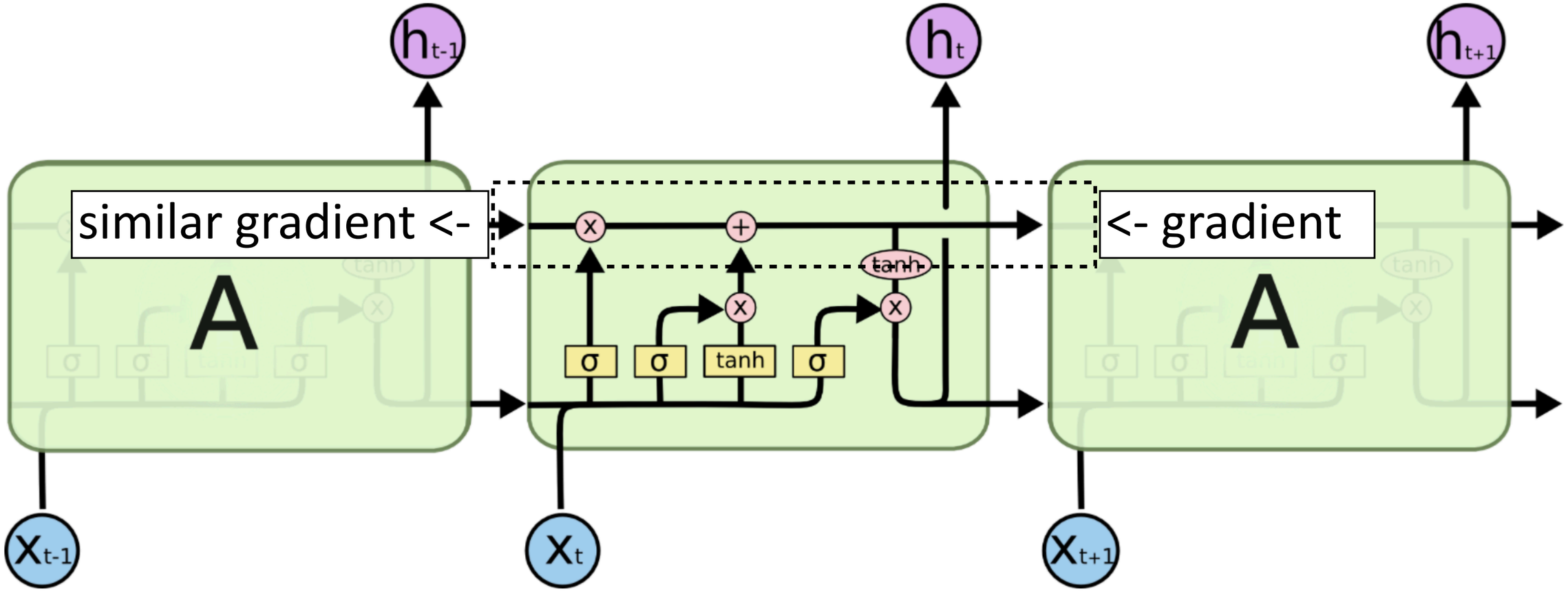
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs



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

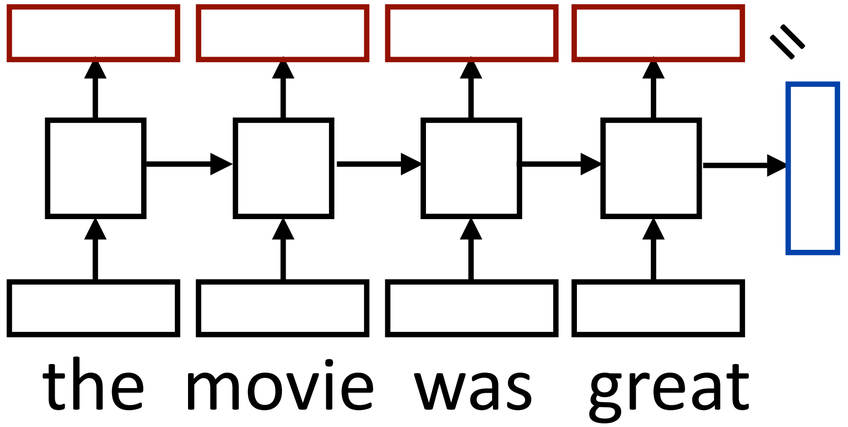
LSTMs



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

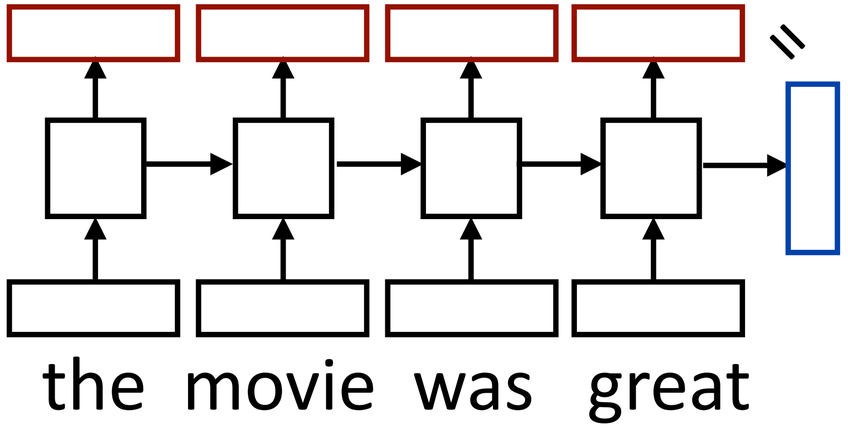
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

What do RNNs produce?



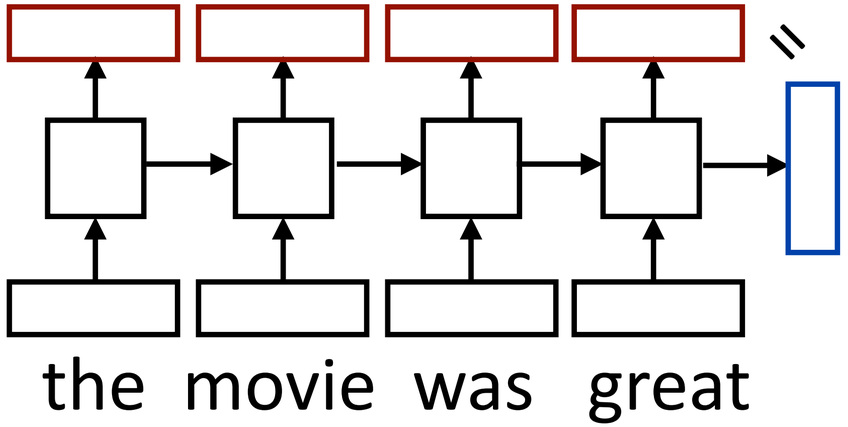
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence

What do RNNs produce?



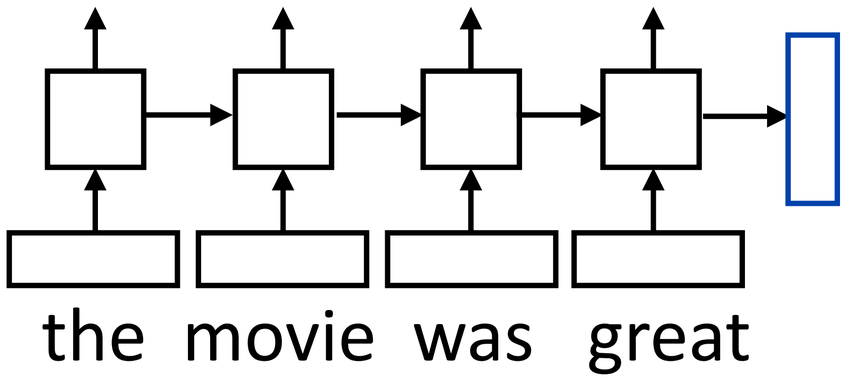
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)

What do RNNs produce?

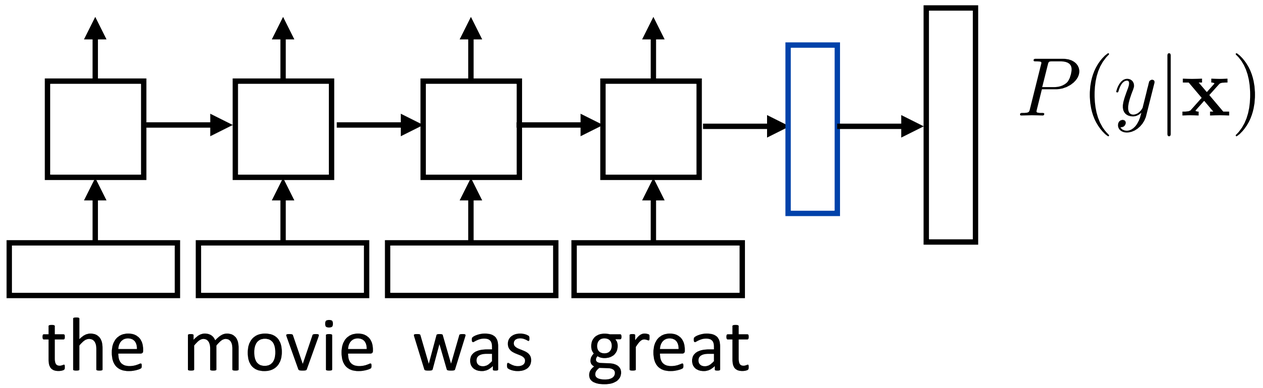


- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors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

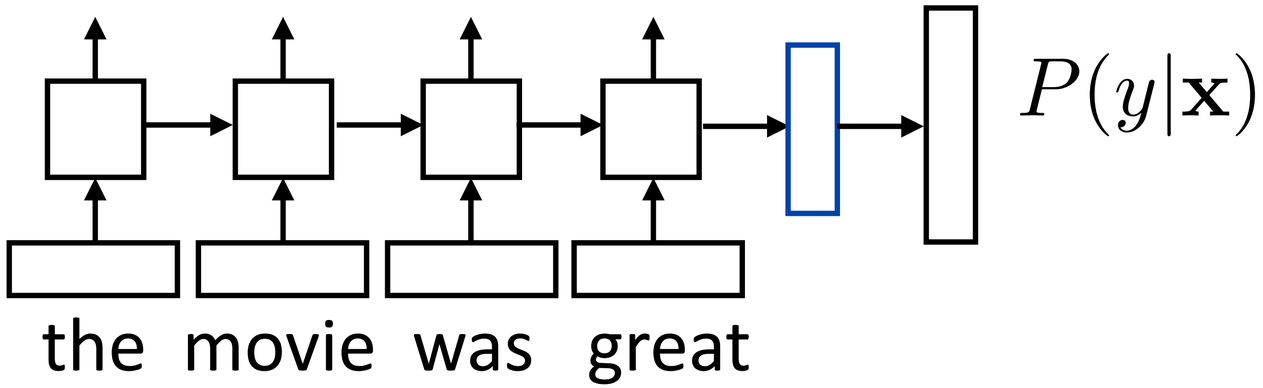
Training RNNs



Training RNNs

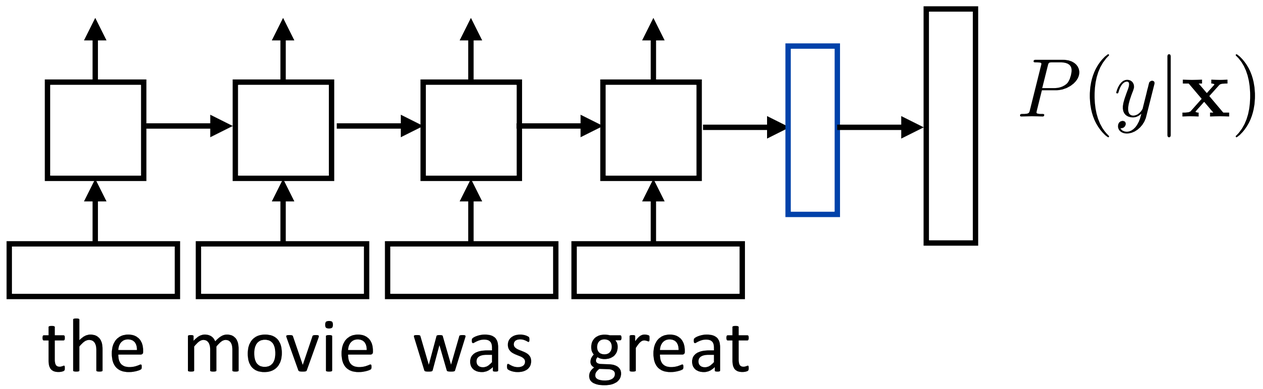


Training RNNs



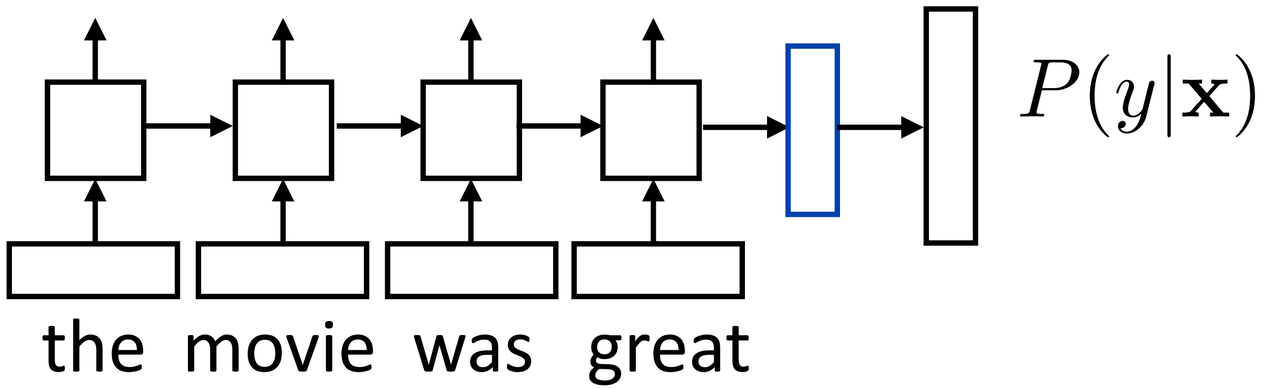
- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)

Training RNNs



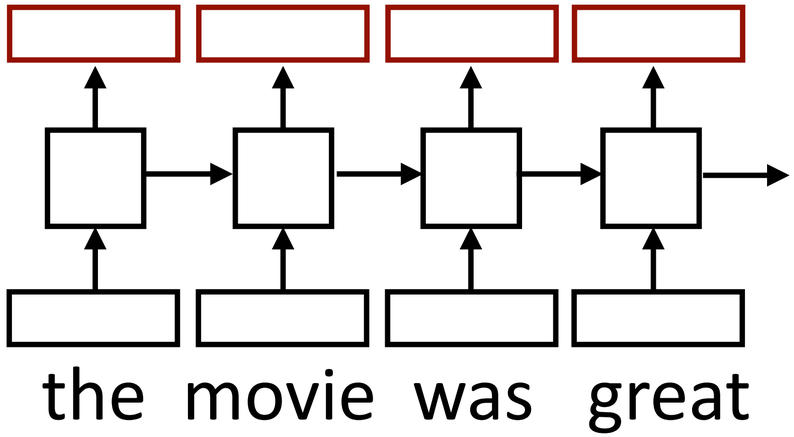
- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network

Training RNNs

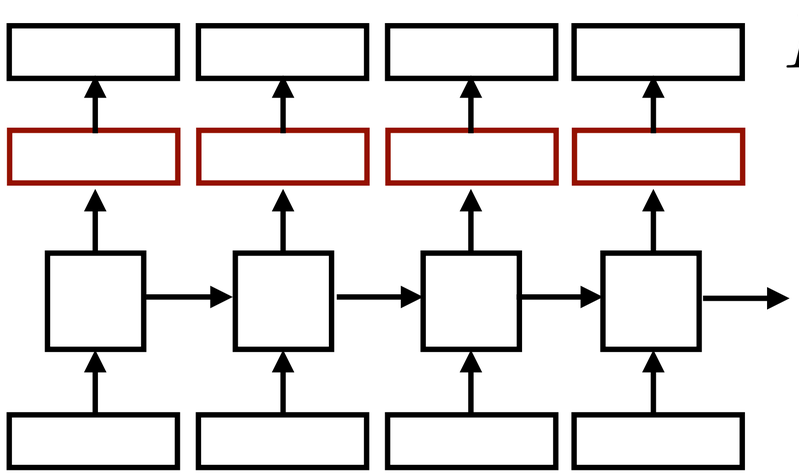


- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis

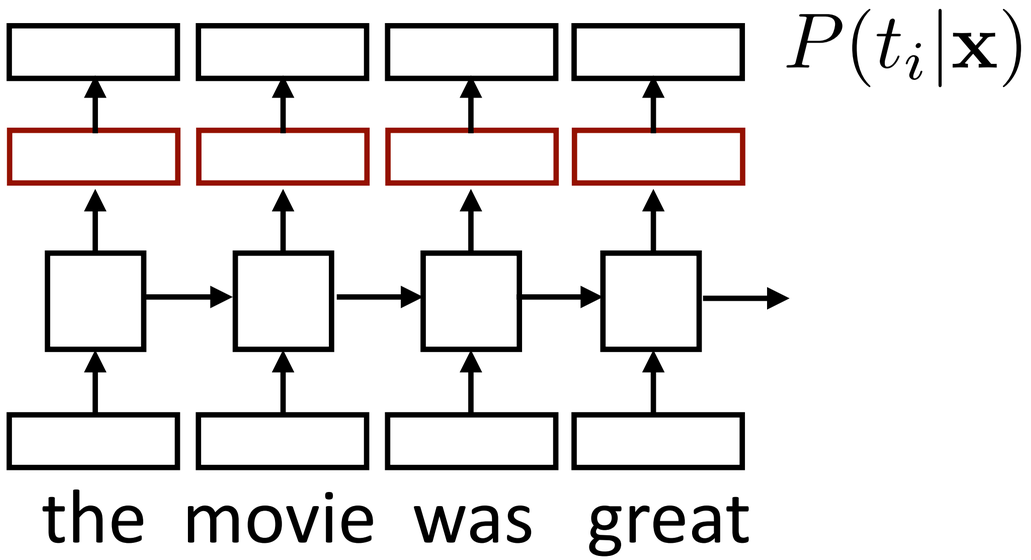
Training RNNs



Training RNNs

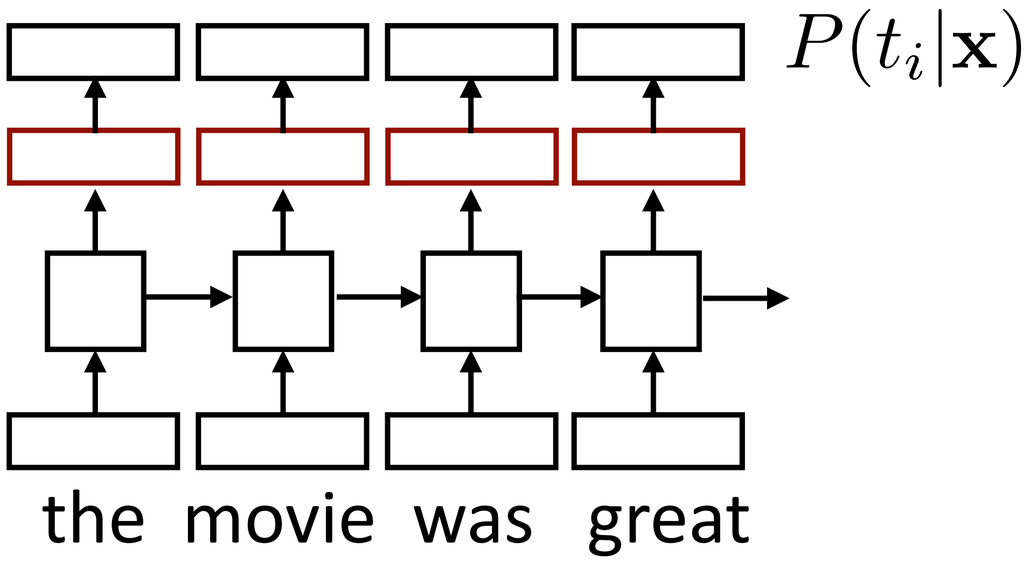


Training RNNs



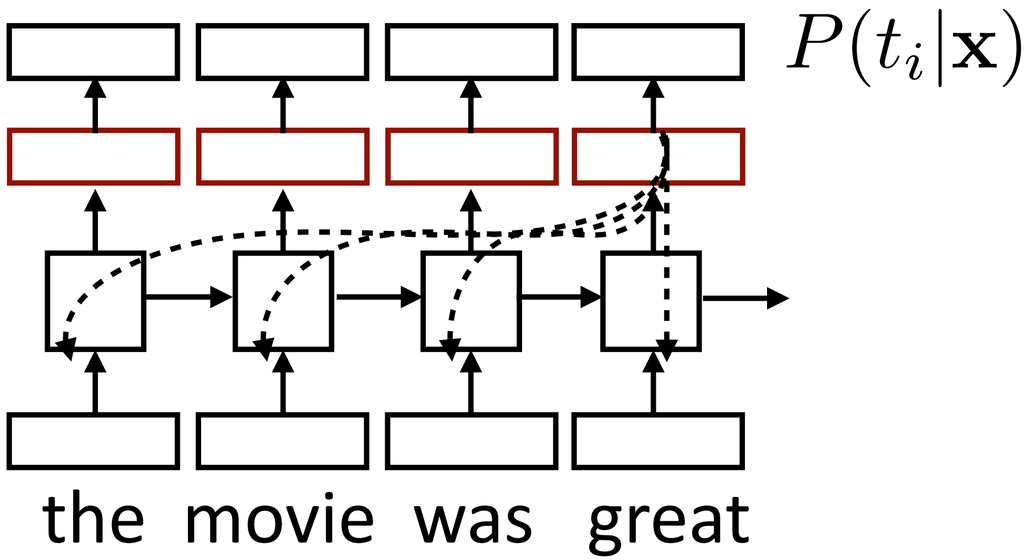
- Loss = negative log likelihood of probability of gold predictions, summed over the tags

Training RNNs



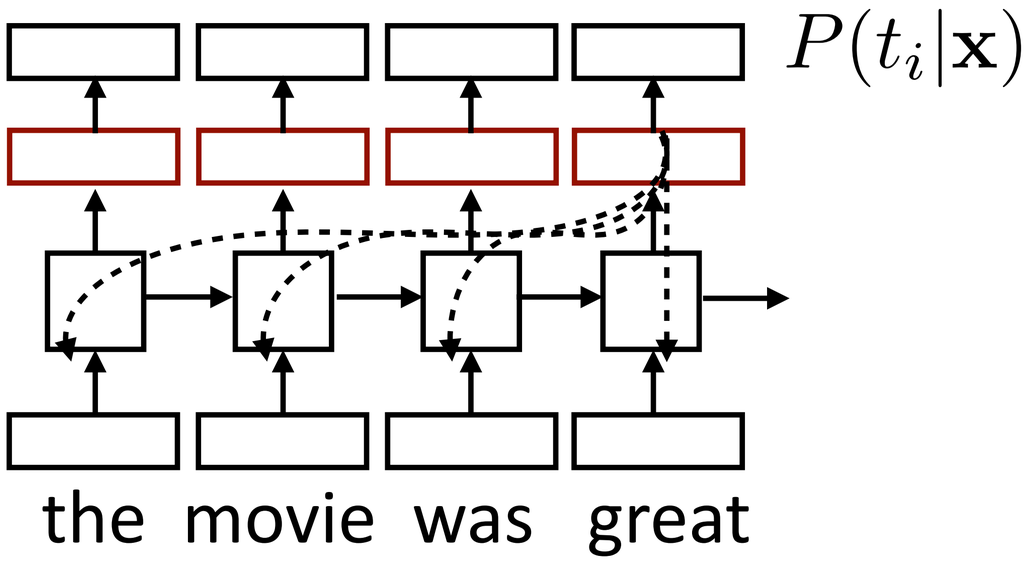
- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network

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)

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

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)

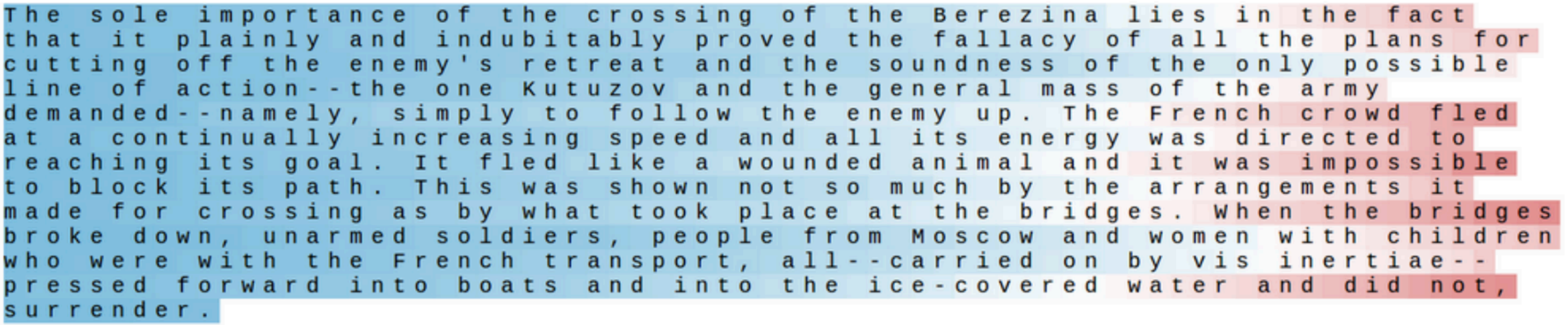
Visualizing LSTMs

- 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

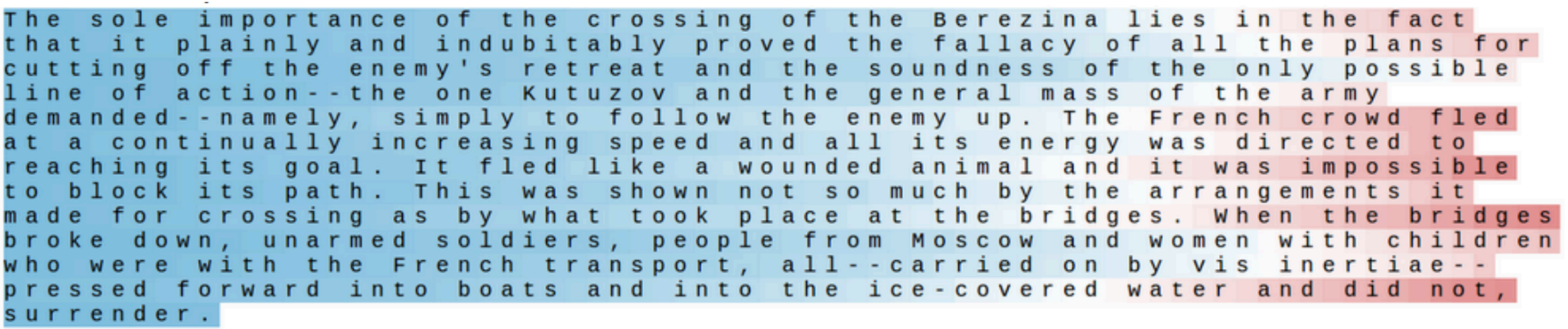
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them



Karpathy et al. (2015)

Visualizing LSTMs

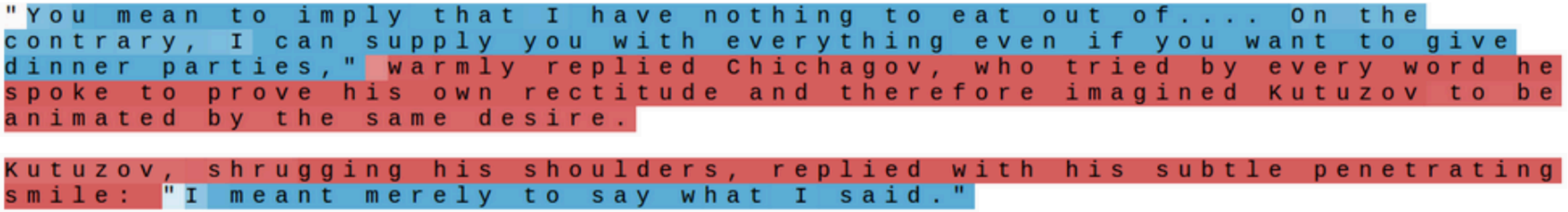
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n



Karpathy et al. (2015)

Visualizing LSTMs

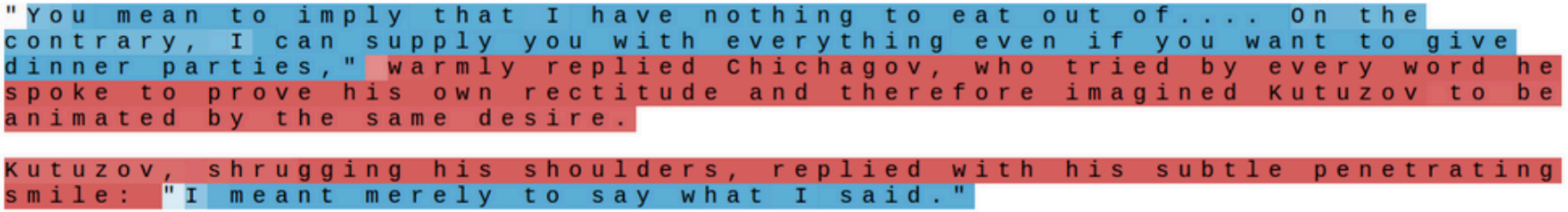
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track



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

```
#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

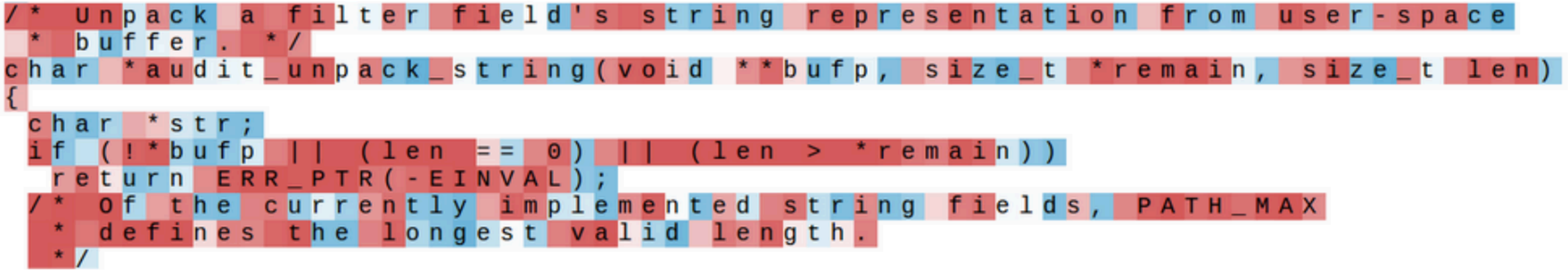
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#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

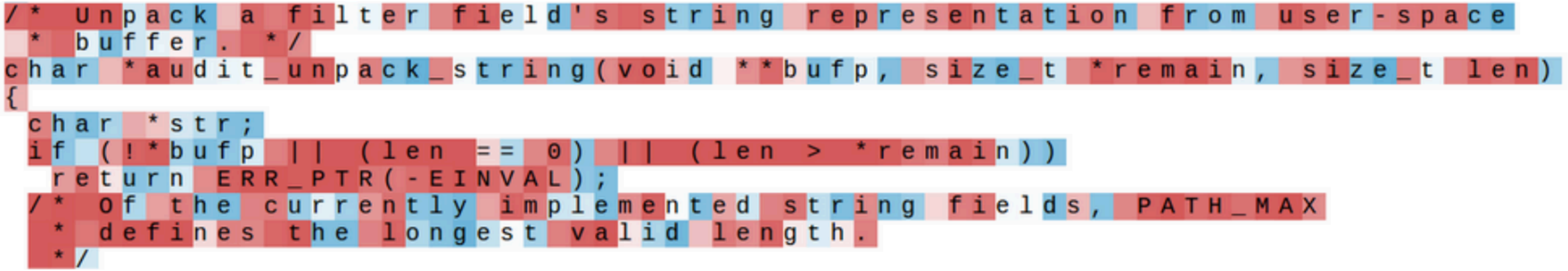
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track



Karpathy et al. (2015)

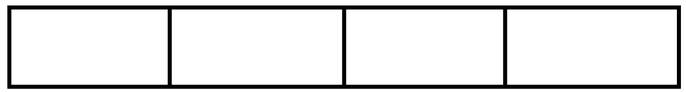
Visualizing LSTMs

- 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)

CNNs vs. LSTMs

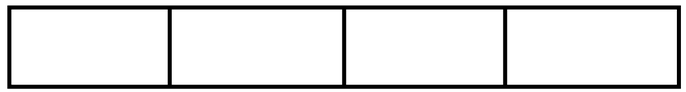


n x k

the movie was good

CNNs vs. LSTMs

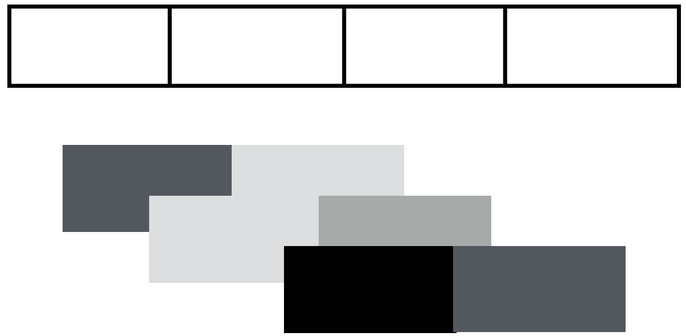
c filters,
m x k each



n x k

the movie was good

CNNs vs. LSTMs



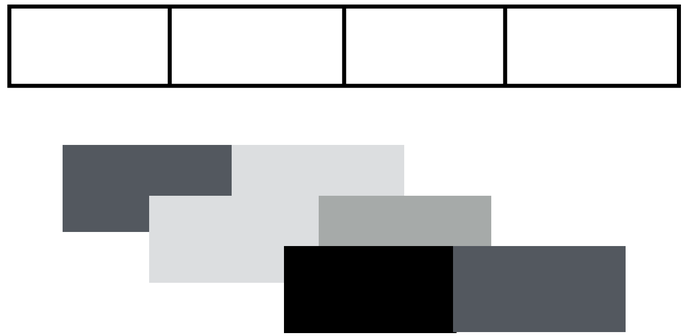
O(n) x c

c filters,
m x k each

n x k

the movie was good

CNNs vs. LSTMs



O(n) x c

c filters,
m x k each

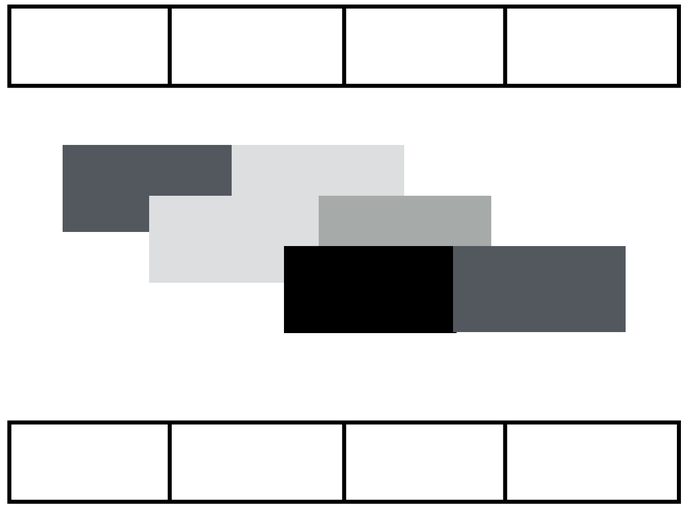
n x k

the movie was good

n x k

the movie was good

CNNs vs. LSTMs

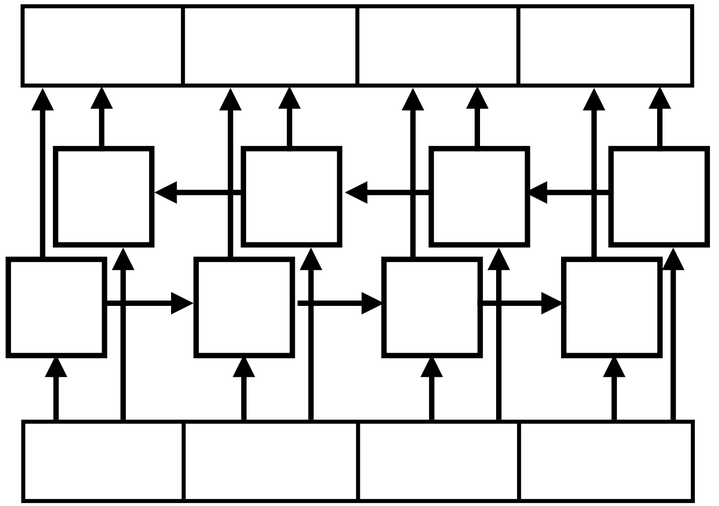


O(n) x c

c filters,
m x k each

n x k

the movie was good



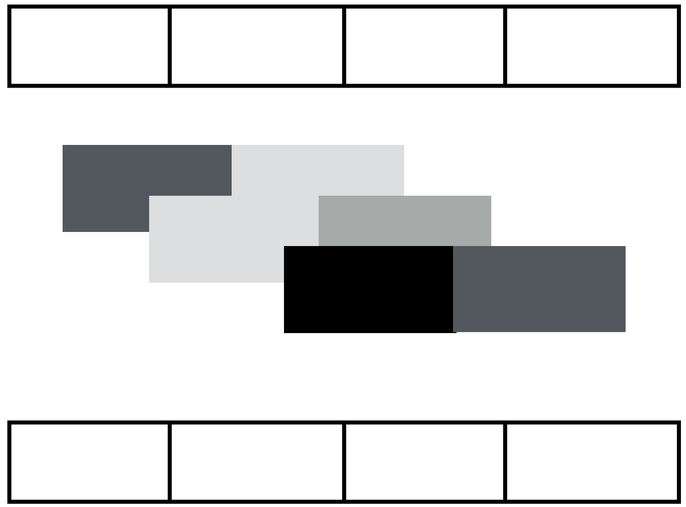
n x 2c

BiLSTM with
hidden size c

n x k

the movie was good

CNNs vs. LSTMs

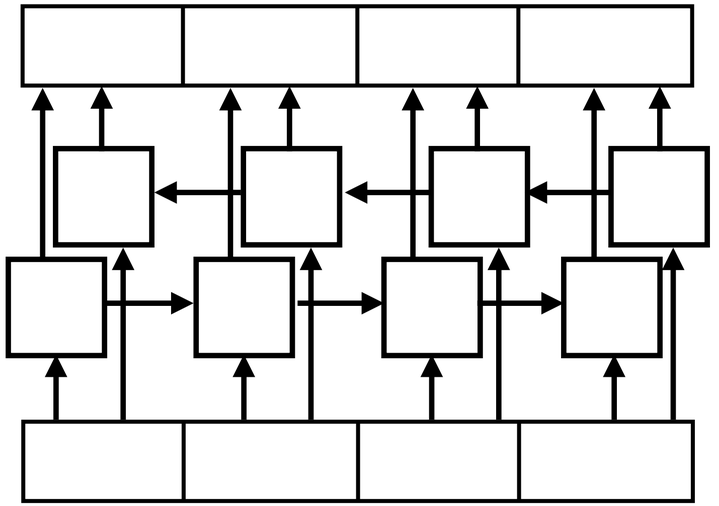


O(n) x c

c filters,
m x k each

n x k

the movie was good



n x 2c

BiLSTM with
hidden size c

n x k

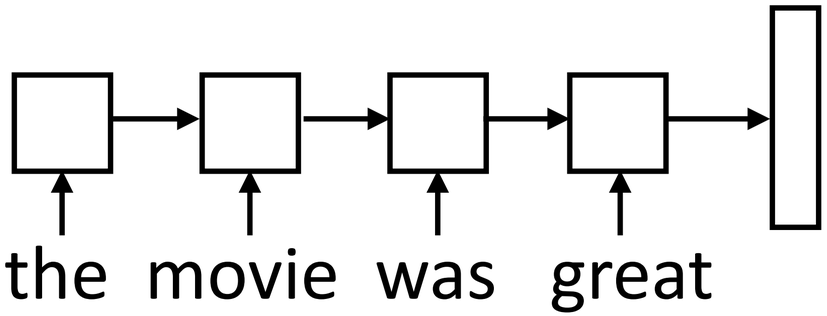
the movie was good

- Both LSTMs and convolutional layers transform the input using context
- LSTM: “globally” looks at the entire sentence (but local for many problems)
- CNN: local depending on filter width + number of layers

Text-to-Text Generation

Encoder-Decoder

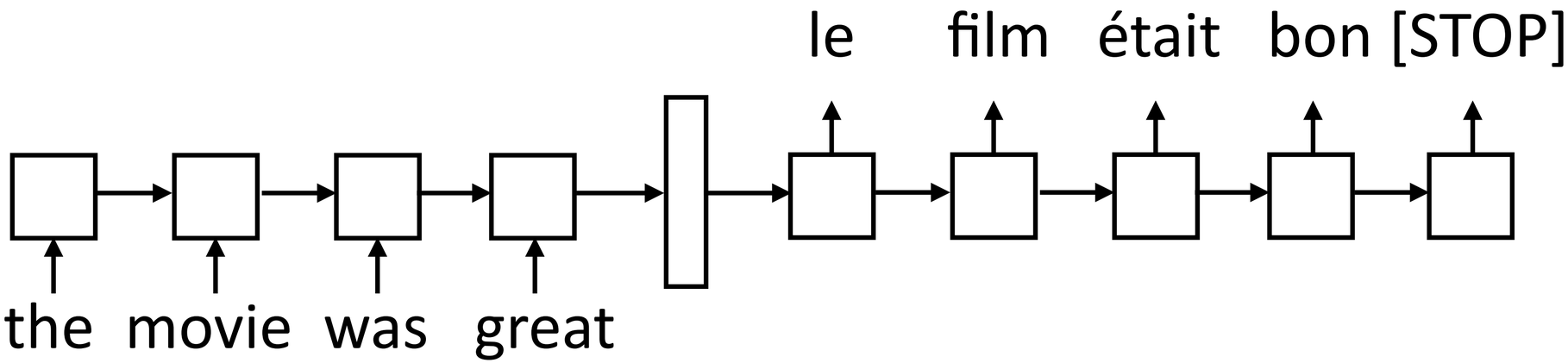
- Encode a sequence into a fixed-sized vector



Sutskever et al. (2014)

Encoder-Decoder

- Encode a sequence into a fixed-sized vector

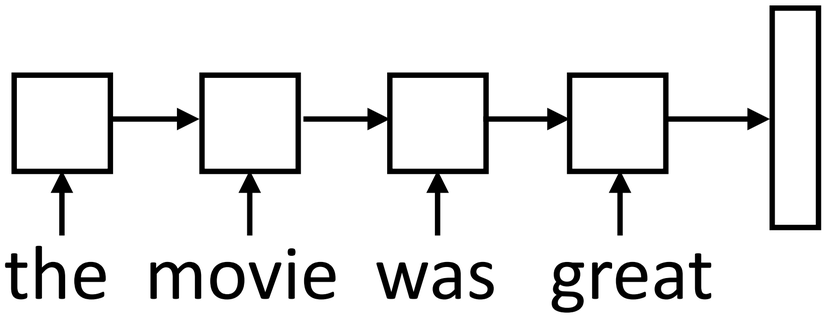


- Now use that vector to produce a series of tokens as output from a separate LSTM *decoder*

Sutskever et al. (2014)

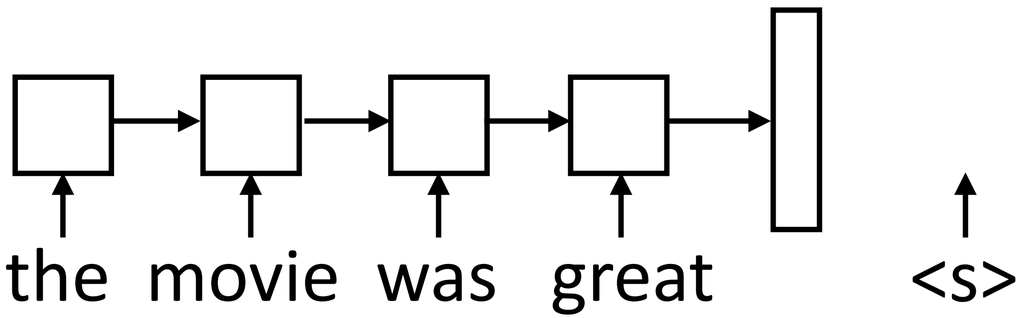
Model

- Generate next word conditioned on previous word as well as hidden state



Model

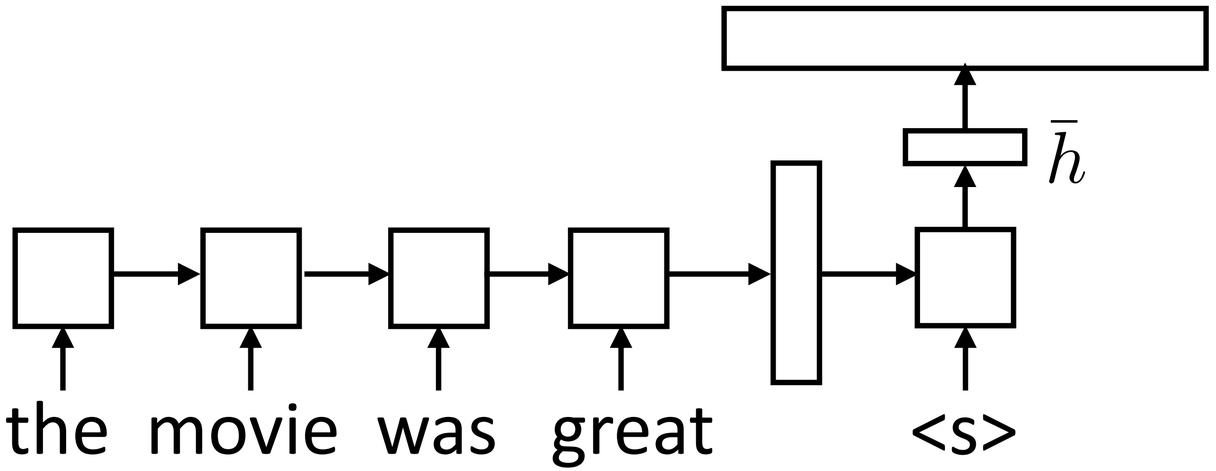
- Generate next word conditioned on previous word as well as hidden state



Model

- Generate next word conditioned on previous word as well as hidden state
- W size is |vocab| x |hidden state|, softmax over entire vocabulary

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

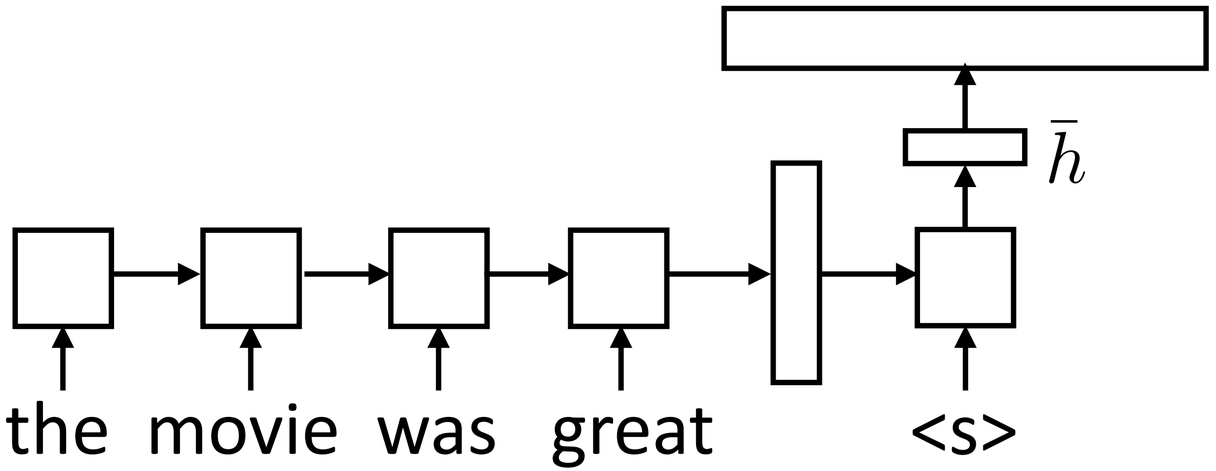


Model

- Generate next word conditioned on previous word as well as hidden state
- W size is |vocab| x |hidden state|, softmax over entire vocabulary

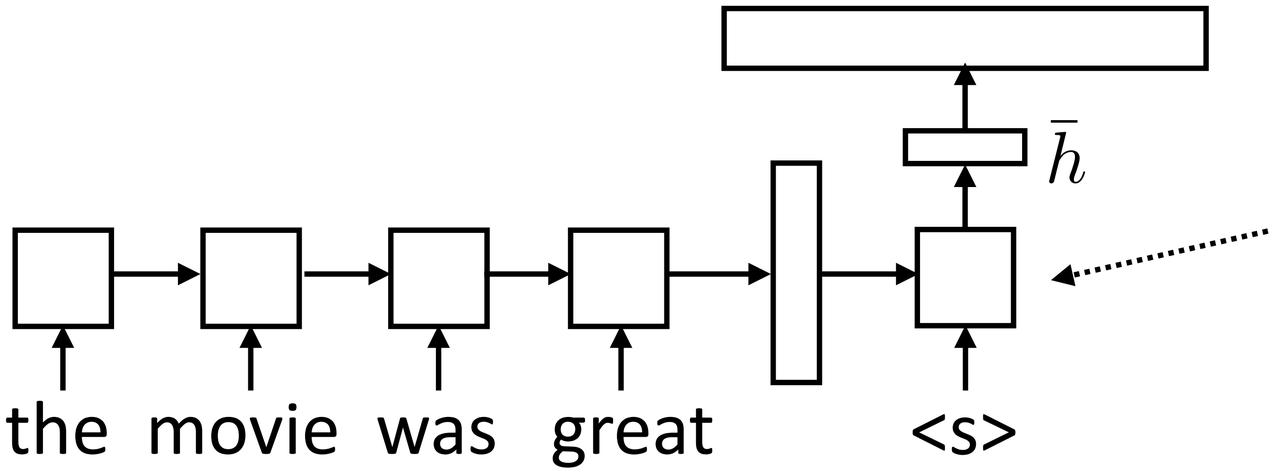
$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$



Model

- Generate next word conditioned on previous word as well as hidden state
- W size is |vocab| x |hidden state|, softmax over entire vocabulary



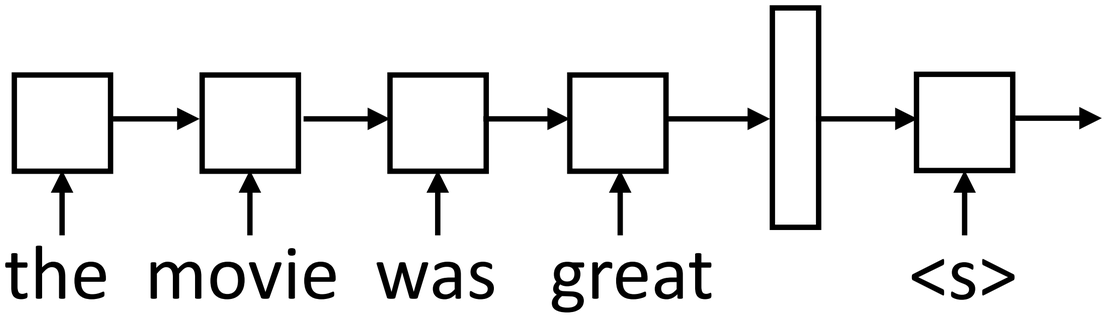
$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$

Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)

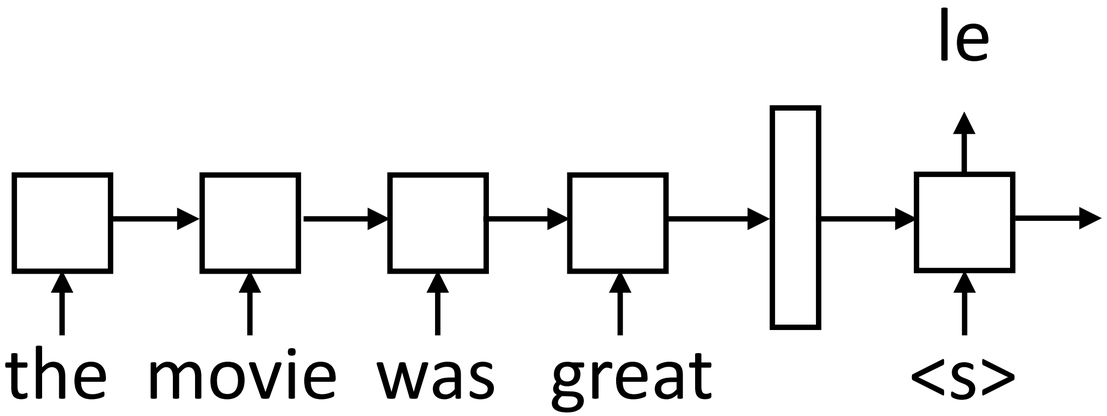
Inference

- Generate next word conditioned on previous word as well as hidden state



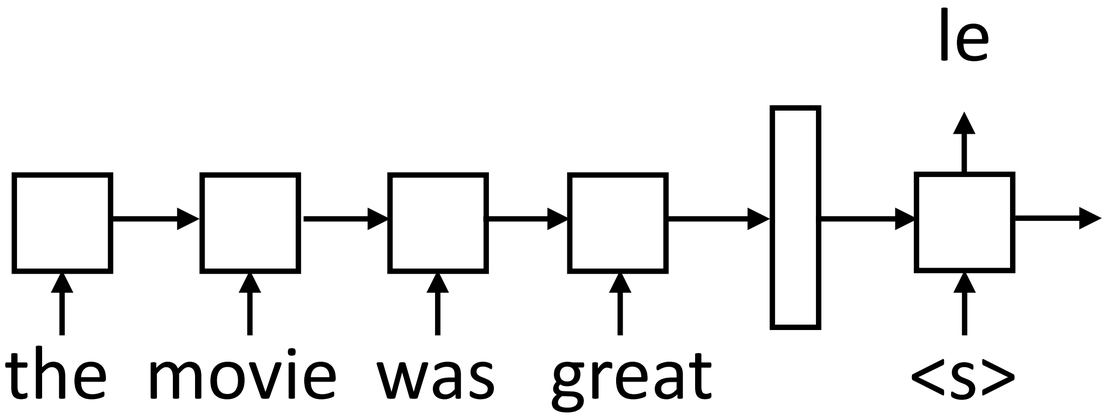
Inference

- Generate next word conditioned on previous word as well as hidden state



Inference

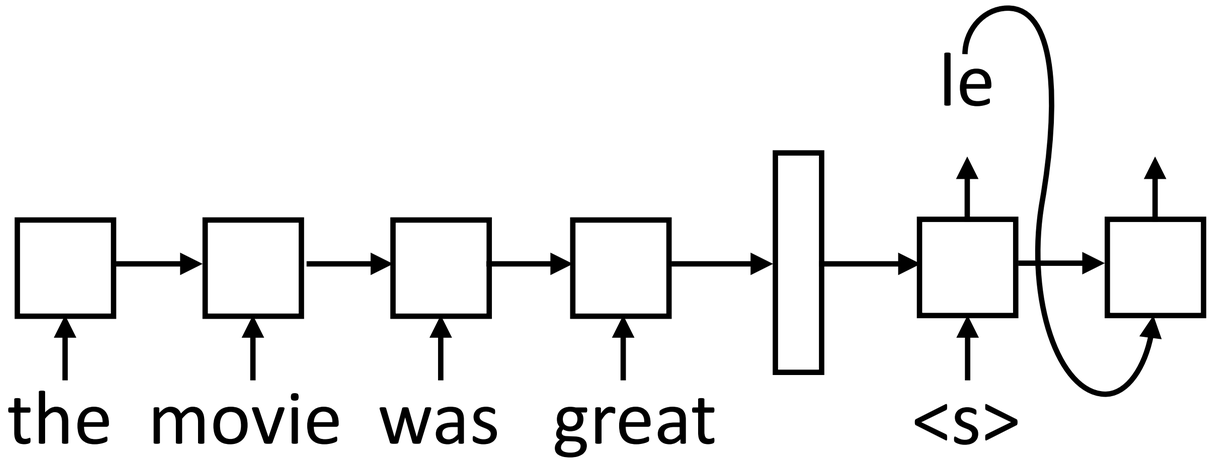
- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

Inference

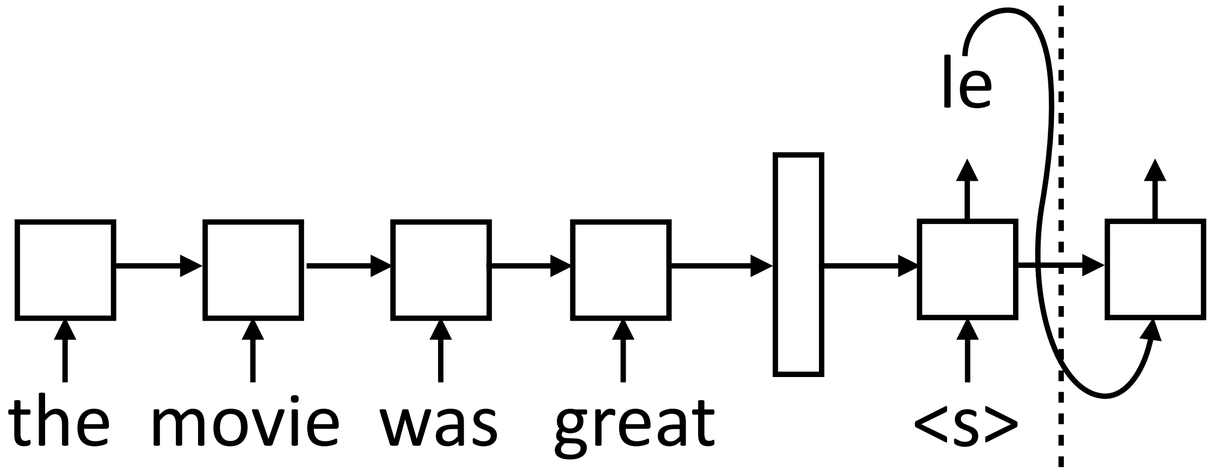
- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

Inference

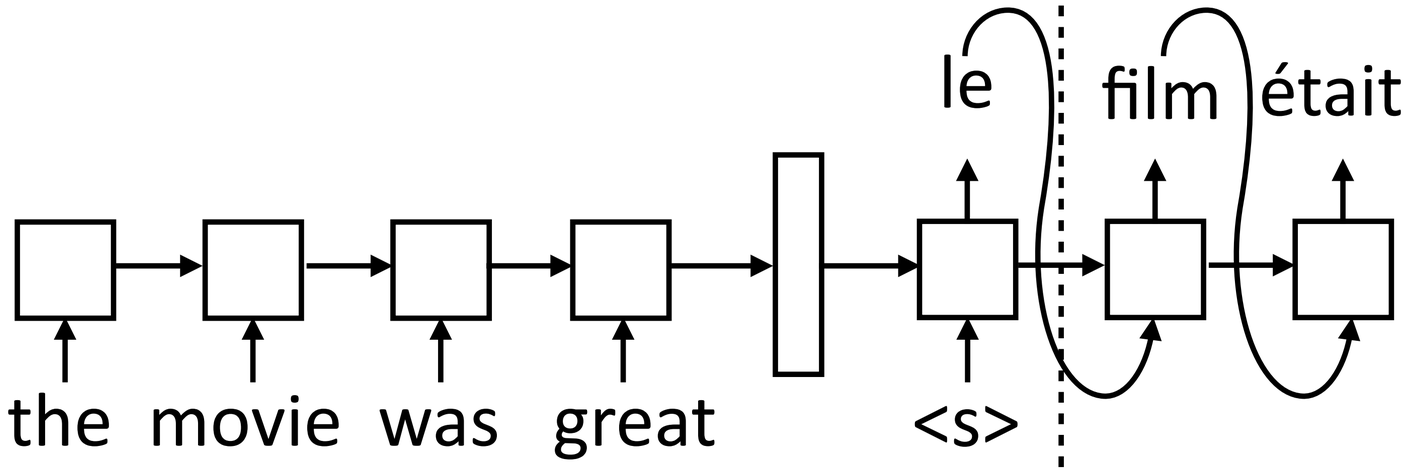
- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state

Inference

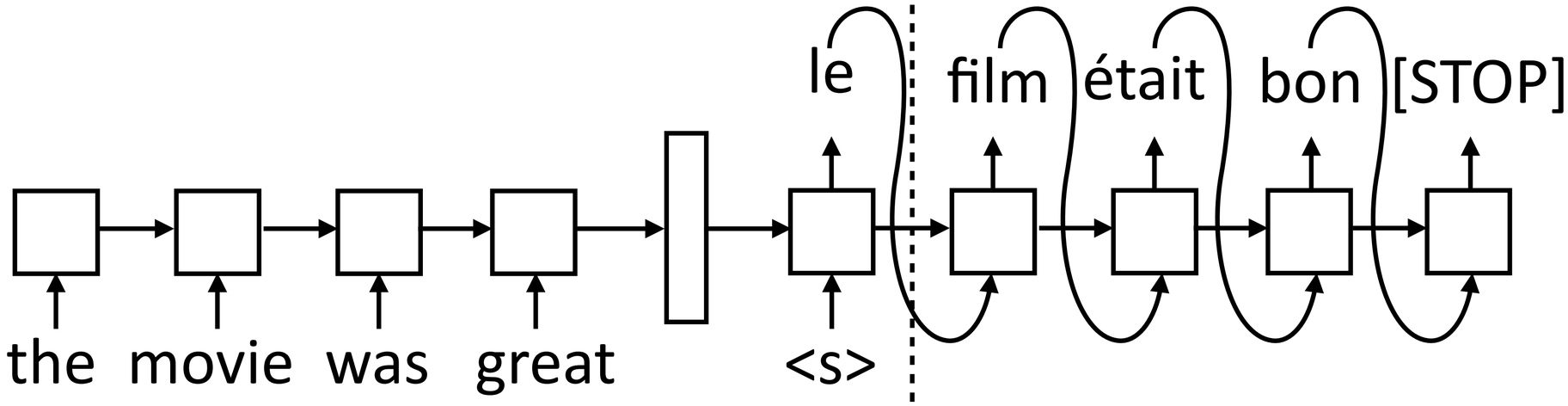
- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state

Inference

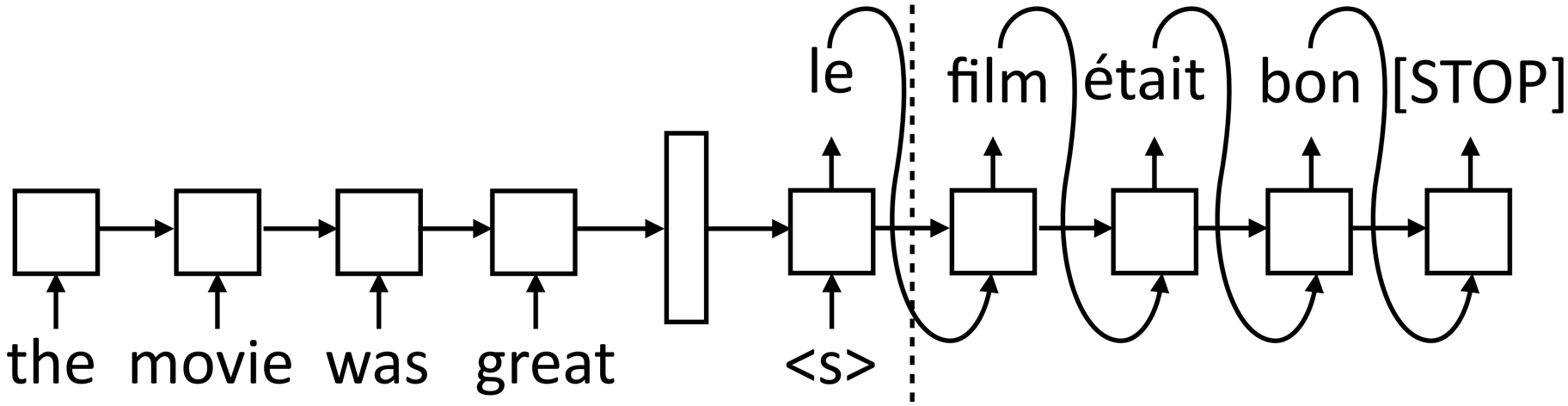
- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state

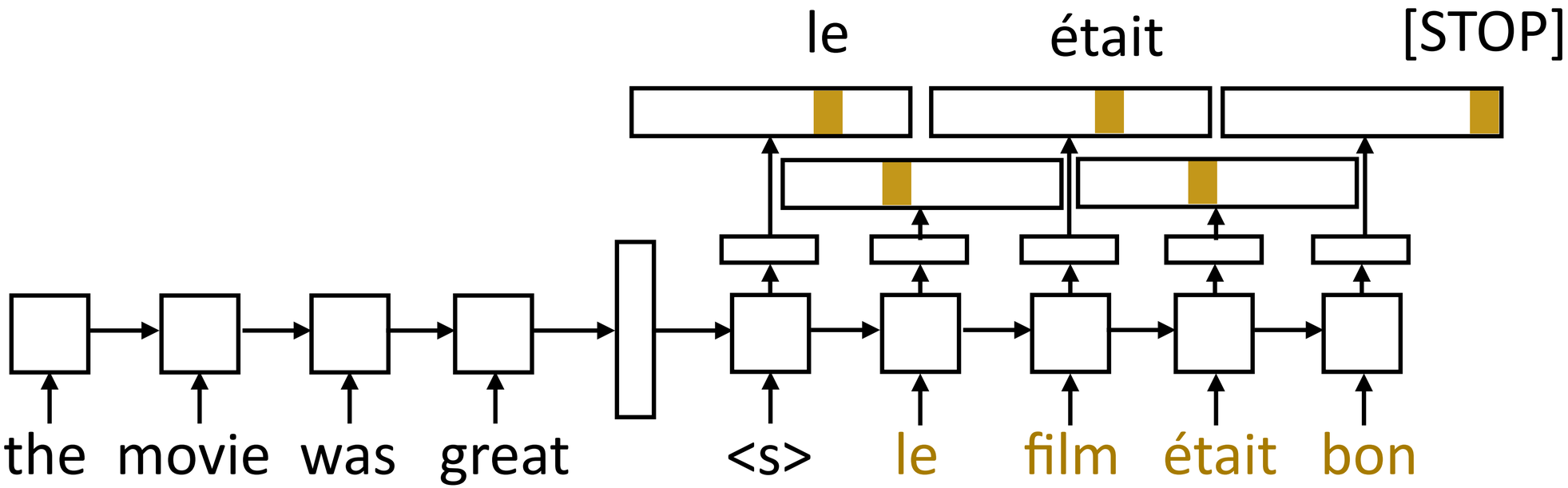
Inference

- Generate next word conditioned on previous word as well as hidden state



- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state
- Decoder is advanced one state at a time until [STOP] is reached

Training



- Objective: maximize $\sum_{(\mathbf{x},\mathbf{y})} \sum_{i=1}^{n} \log P(y_i^* | \mathbf{x}, y_1^*, \ldots, y_{i-1}^*)$
- One loss term for each target-sentence word, feed the correct word regardless of model's prediction

Attention

Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

- Often a byproduct of training these models poorly

Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

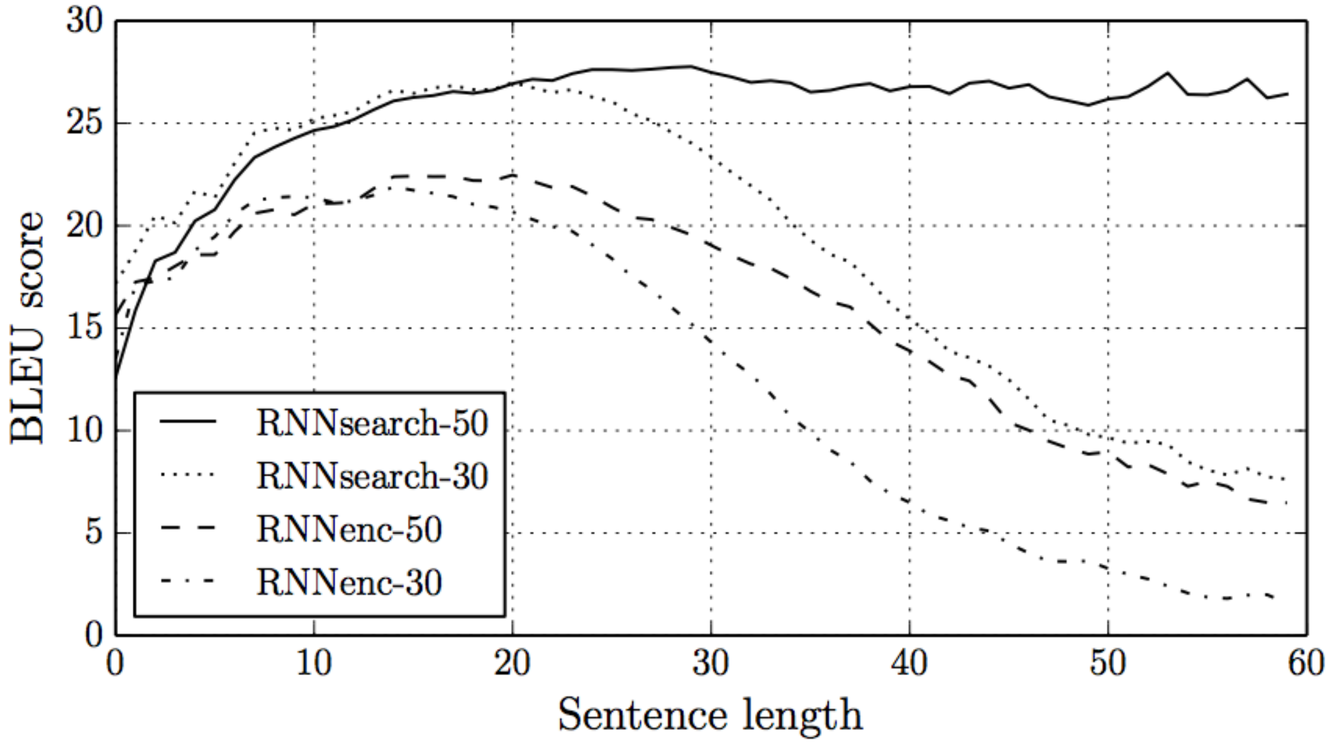
Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

- Often a byproduct of training these models poorly

- Need some notion of input coverage or what input words we've translated

Problems with Seq2seq Models

- Bad at long sentences: 1) a fixed-size representation doesn't scale; 2) LSTMs still have a hard time remembering for really long periods of time



RNNsearch: introduces attention mechanism to give "variable-sized" representation

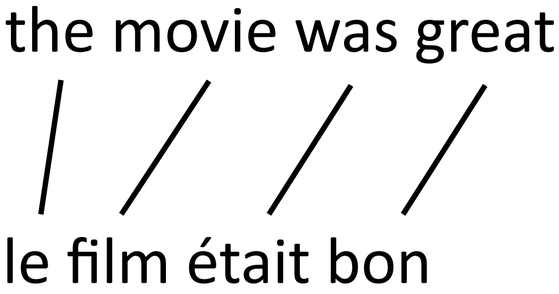
Bahdanau et al. (2014)

Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

Aligned Inputs

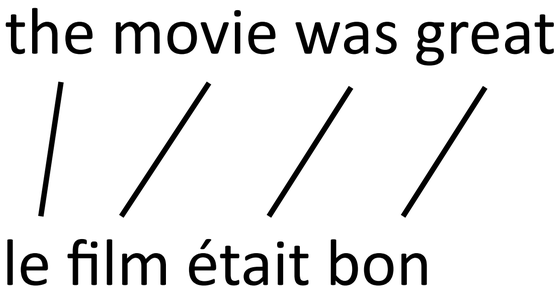
- Suppose we knew the source and target would be word-by-word translated



Aligned Inputs

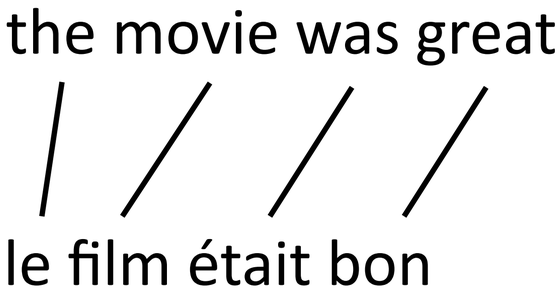
- Suppose we knew the source and target would be word-by-word translated

- Can look at the corresponding input word when translating — this could scale!

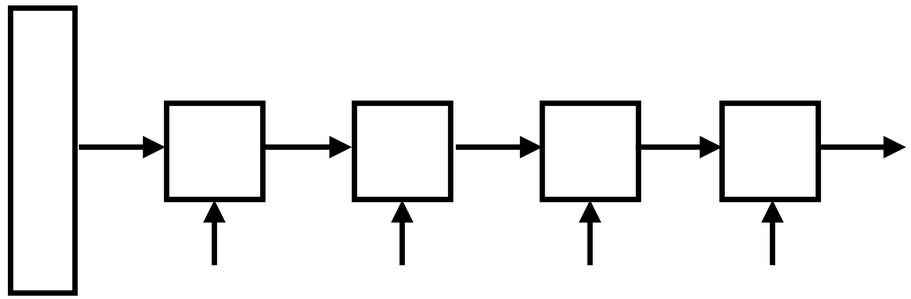


Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

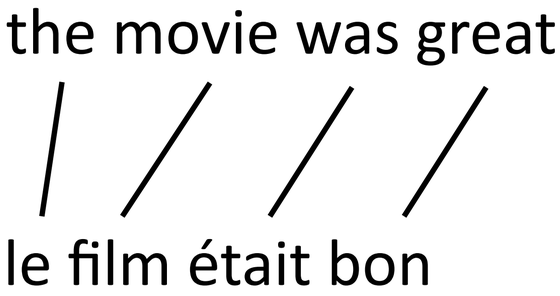


- Can look at the corresponding input word when translating — this could scale!

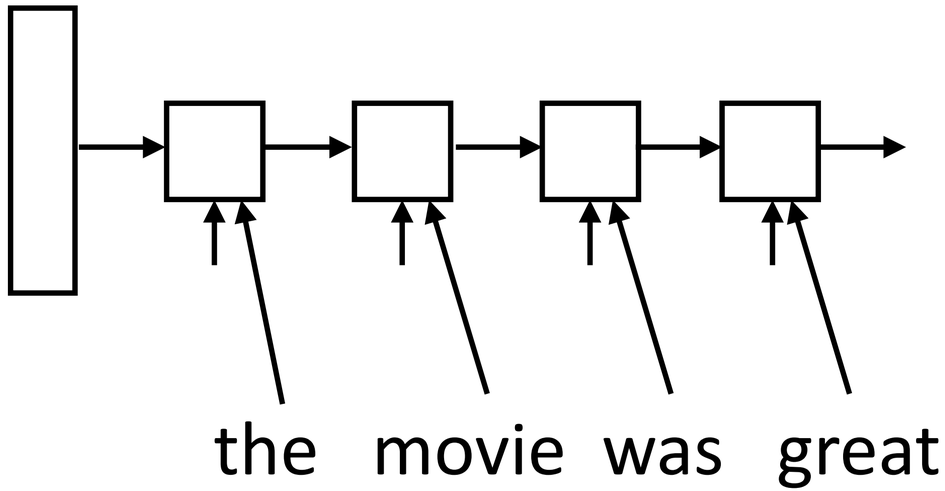


Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

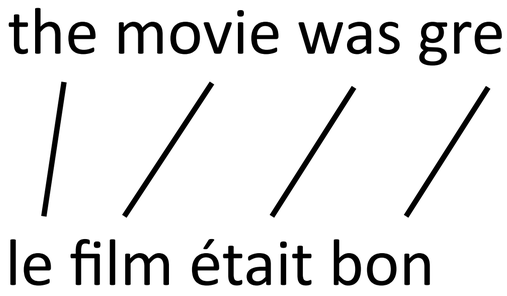


- Can look at the corresponding input word when translating — this could scale!

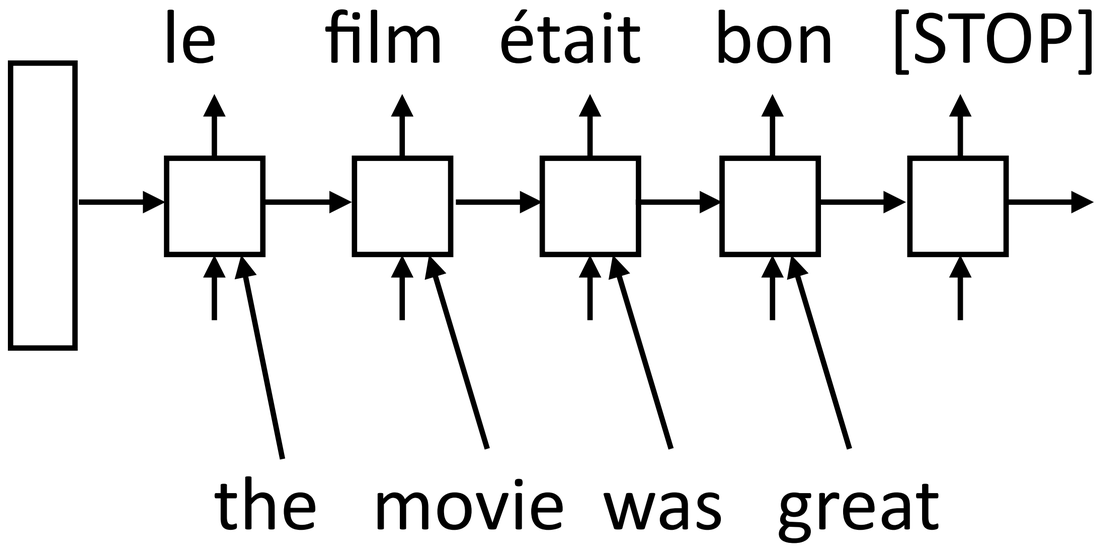


Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

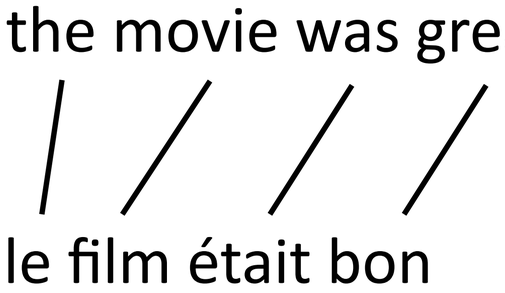


- Can look at the corresponding input word when translating — this could scale!

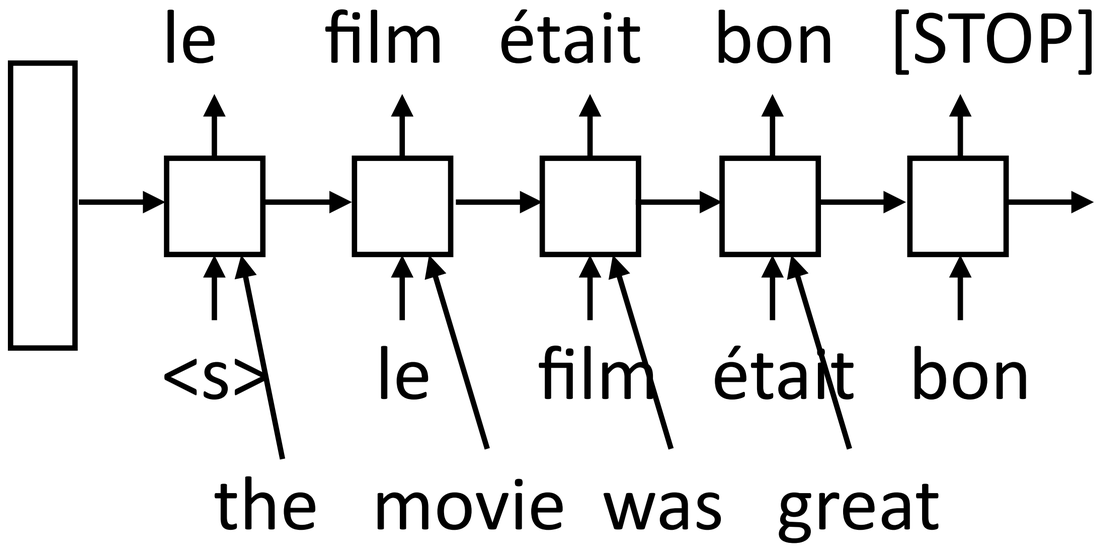


Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated

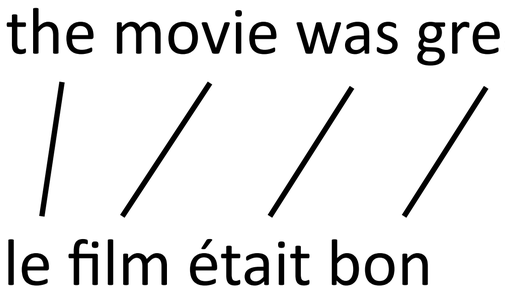


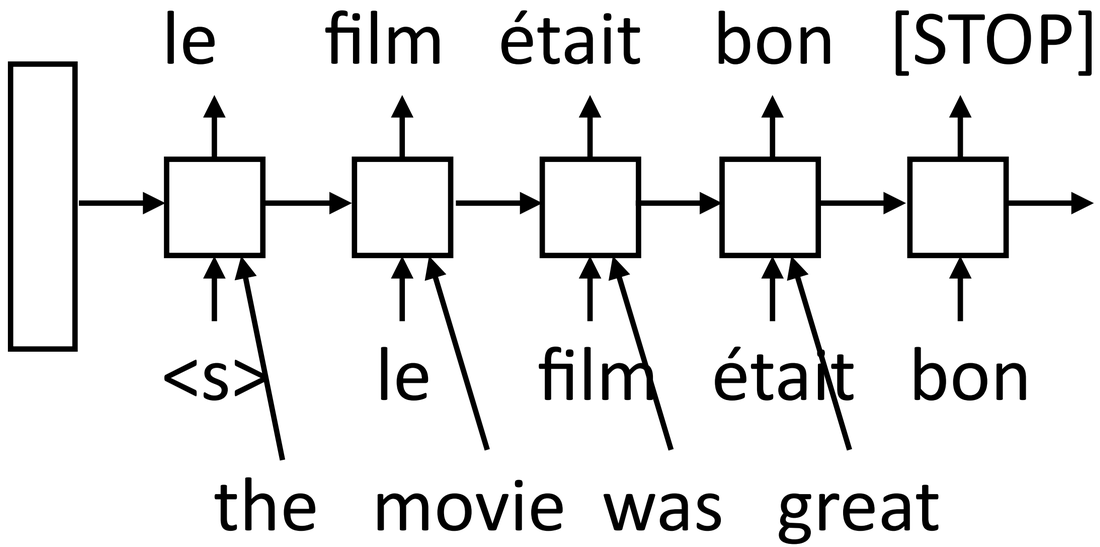
- Can look at the corresponding input word when translating — this could scale!



Aligned Inputs

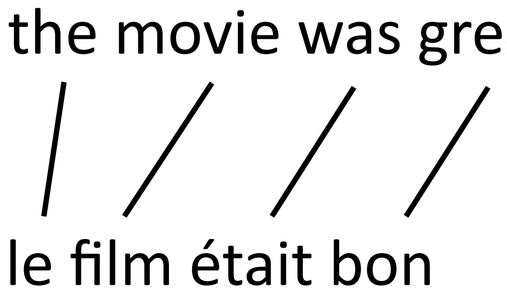
- Suppose we knew the source and target would be word-by-word translated
- Can look at the corresponding input word when translating — this could scale!
- Much less burden on the hidden state



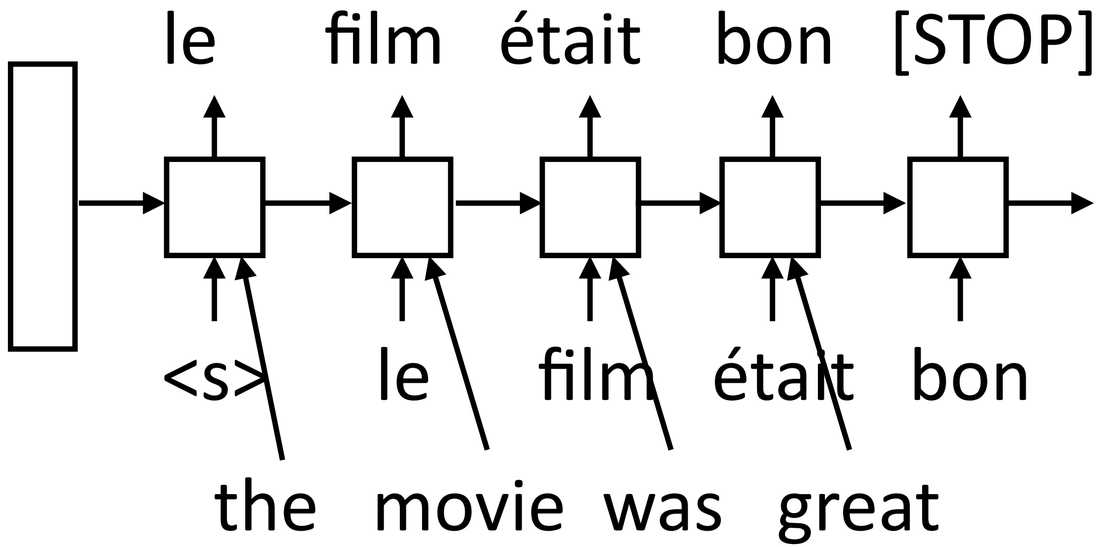


Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated



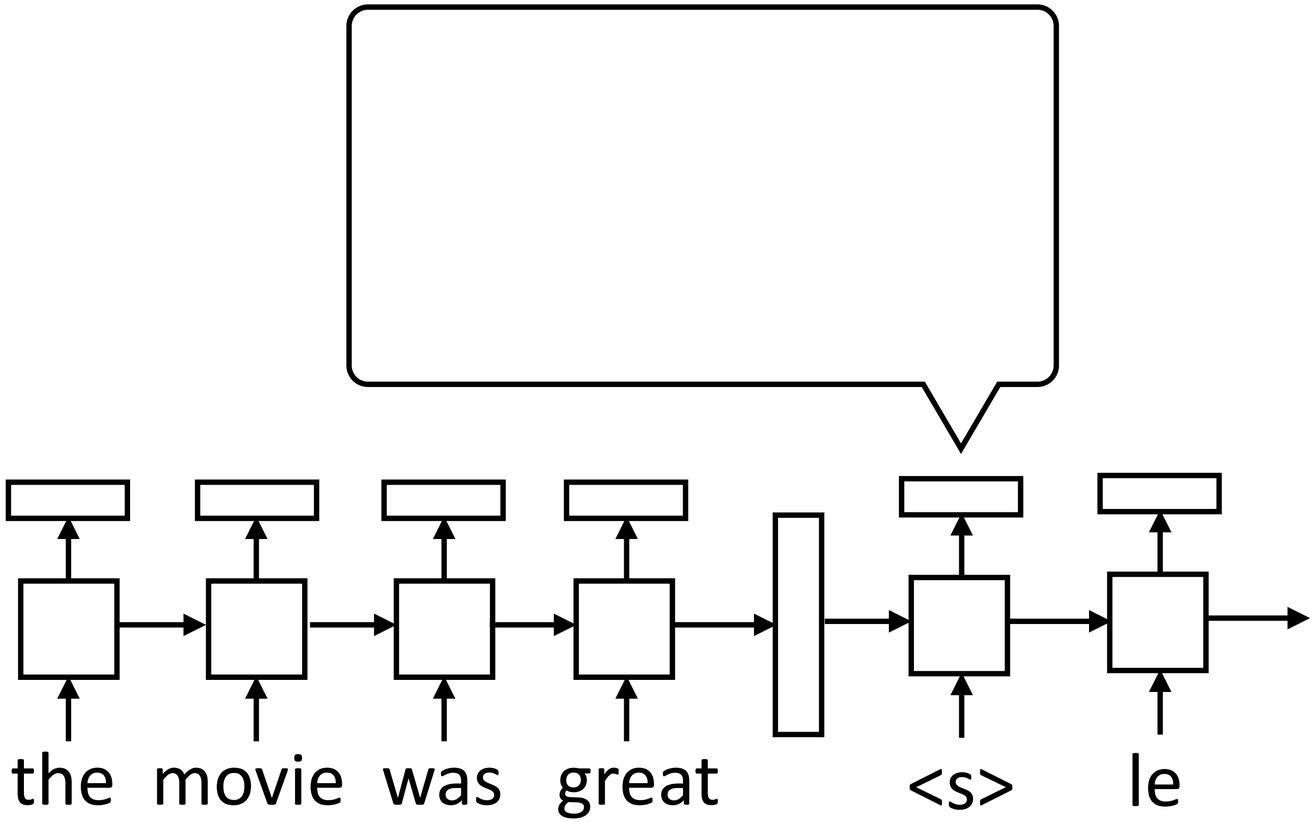
- Can look at the corresponding input word when translating — this could scale!



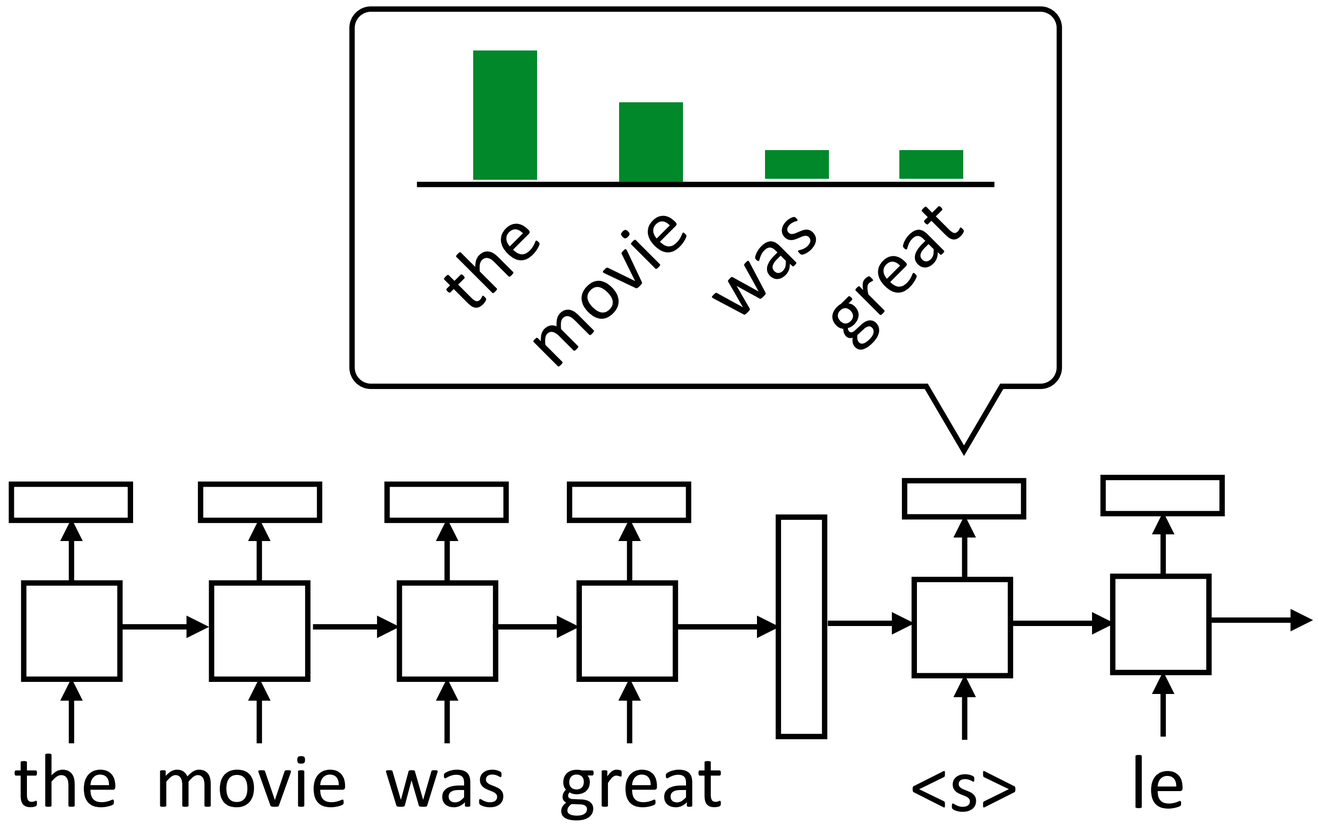
- Much less burden on the hidden state

- How can we achieve this without hardcoding it?

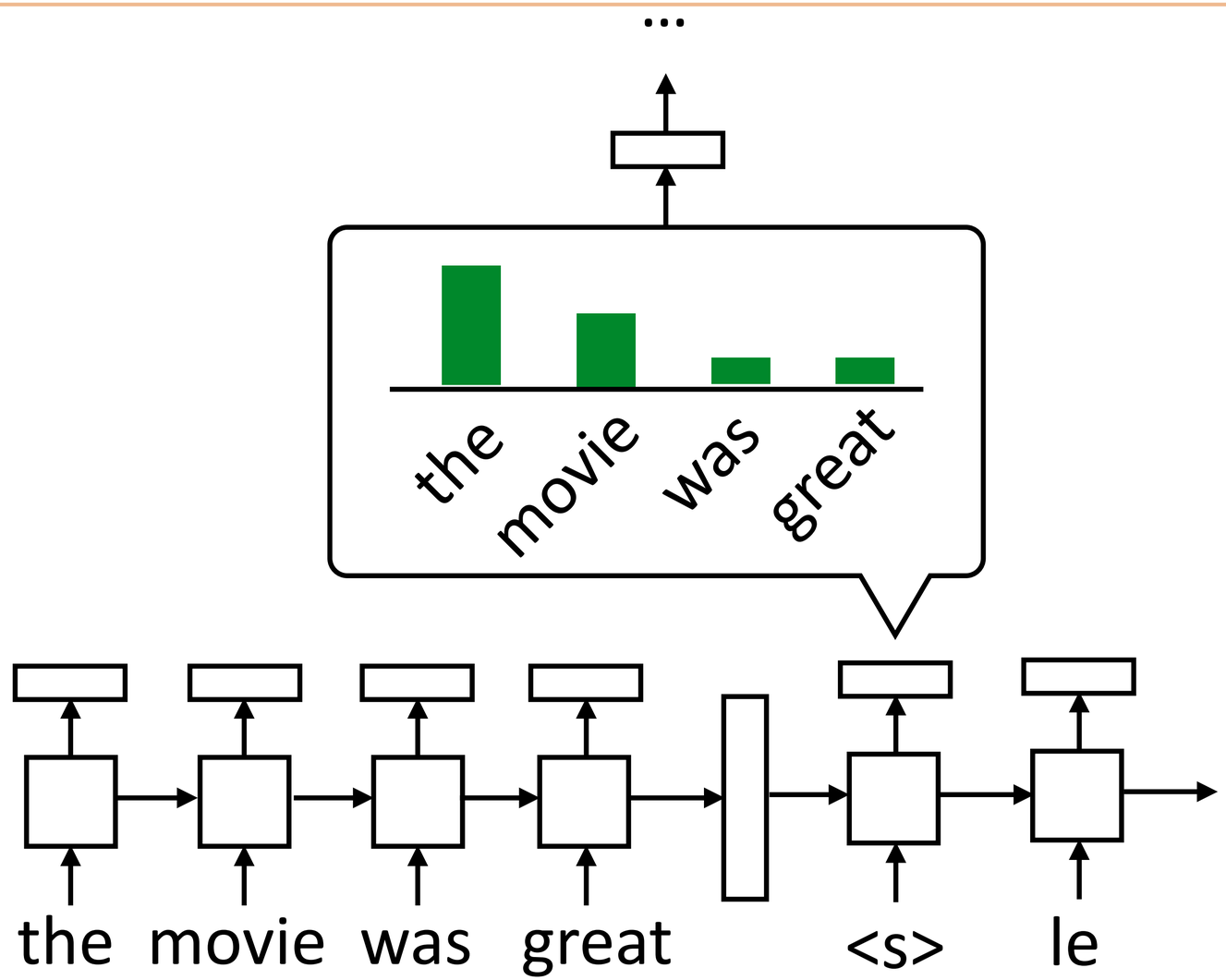
Attention



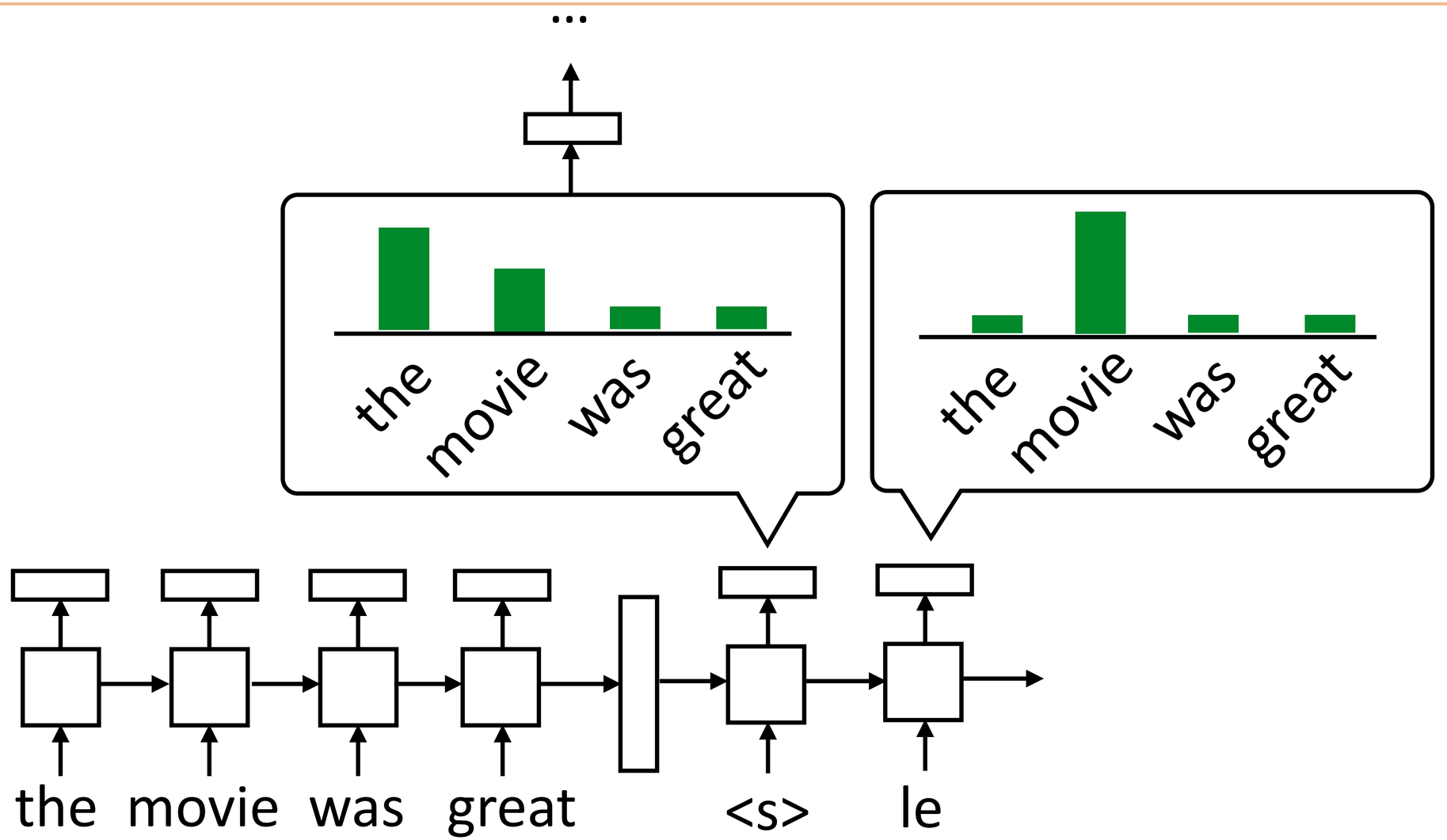
Attention



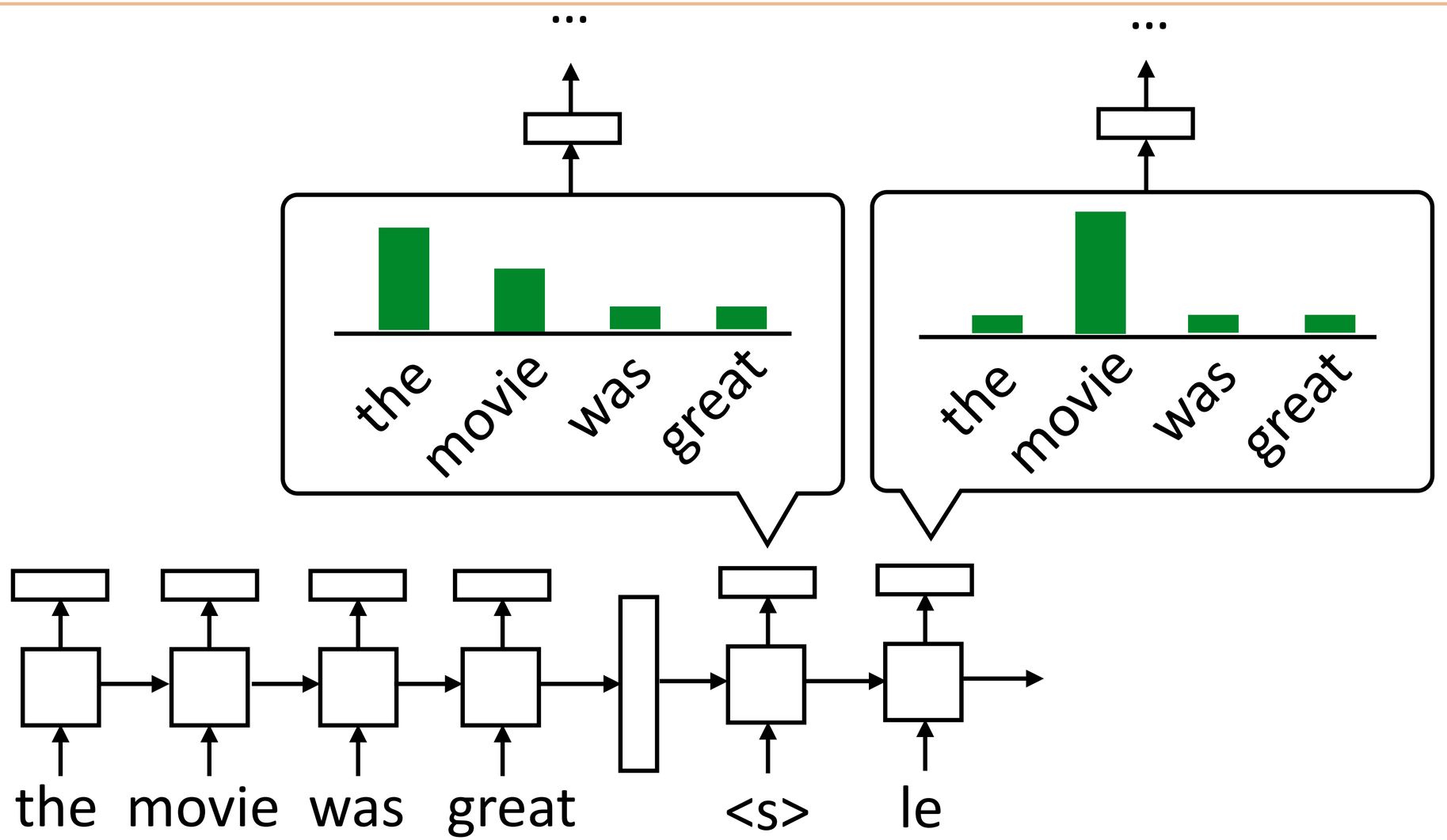
Attention



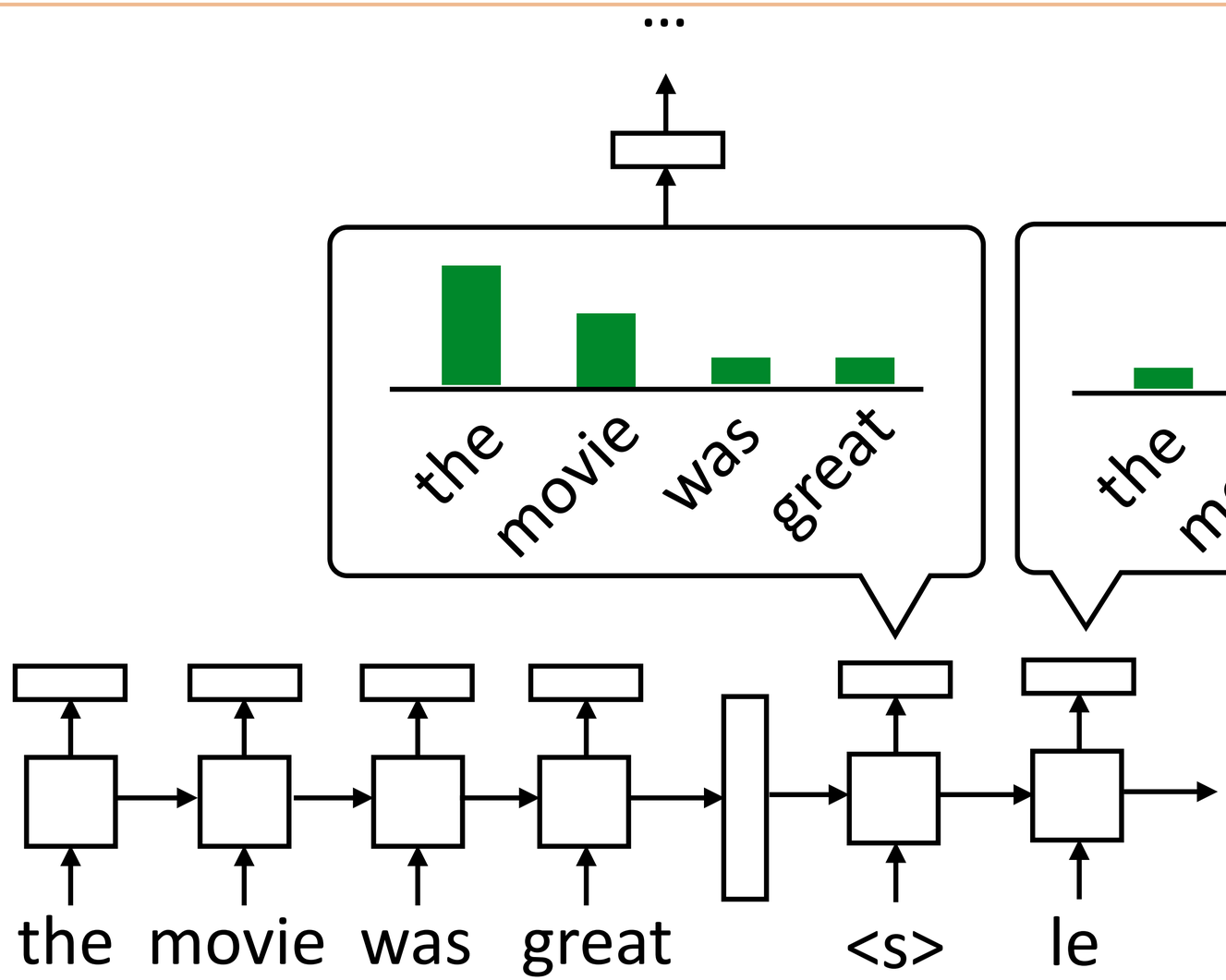
Attention


...
the
movie
was
great
the
movie
was
great
the
movie
was
great
<s>
le

Attention

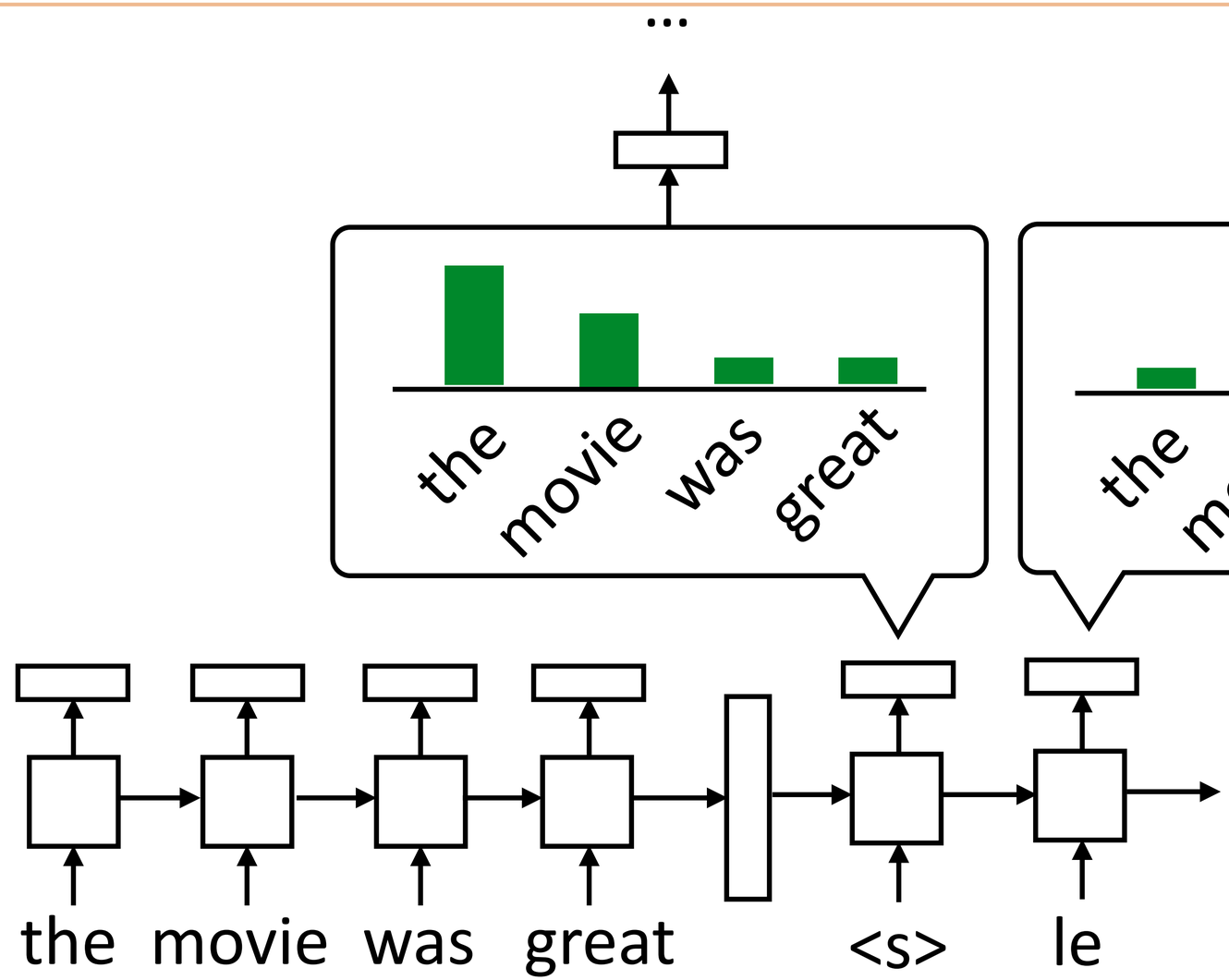

...
...
the
movie
was
great
the
movie
was
great
the
movie
was
great
<s>
le

Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state

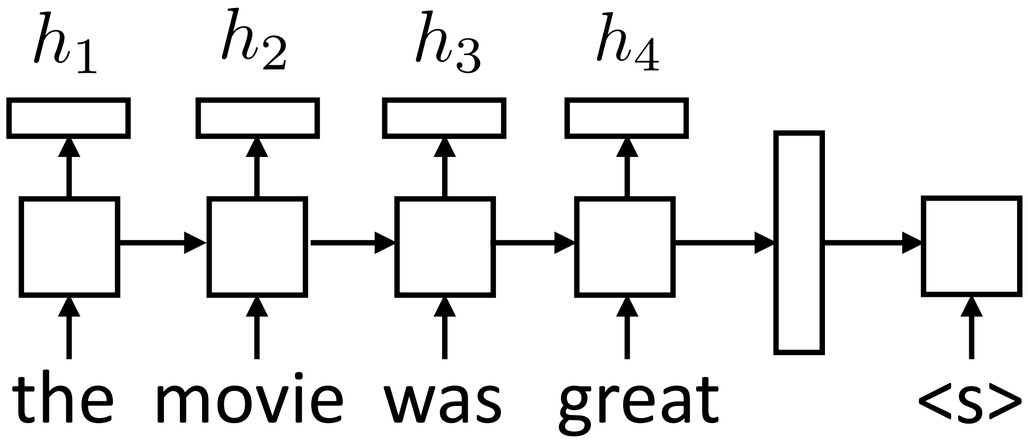
Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state
- Use that in output layer

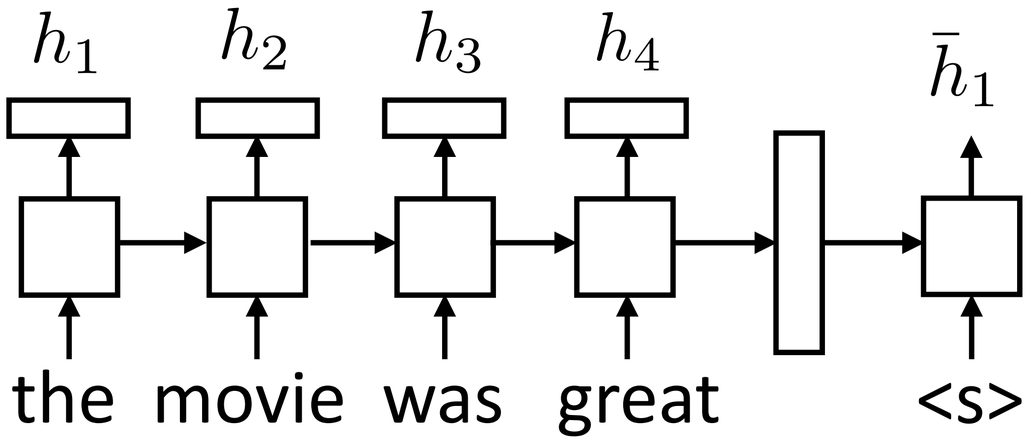
Attention

- For each decoder state, compute weighted sum of input states



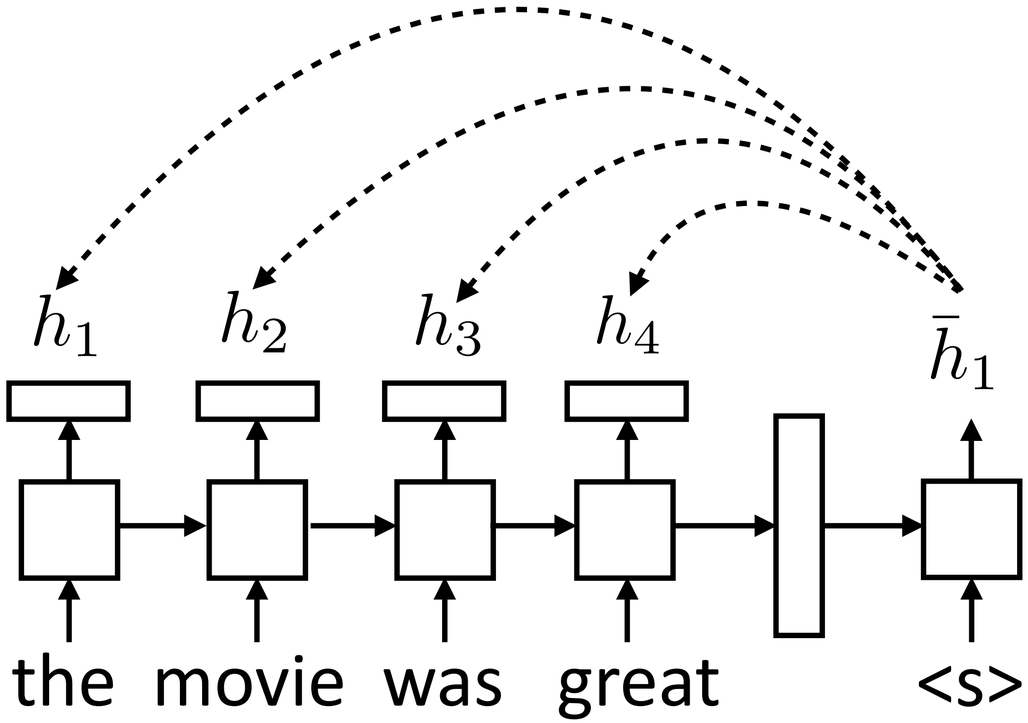
Attention

- For each decoder state, compute weighted sum of input states



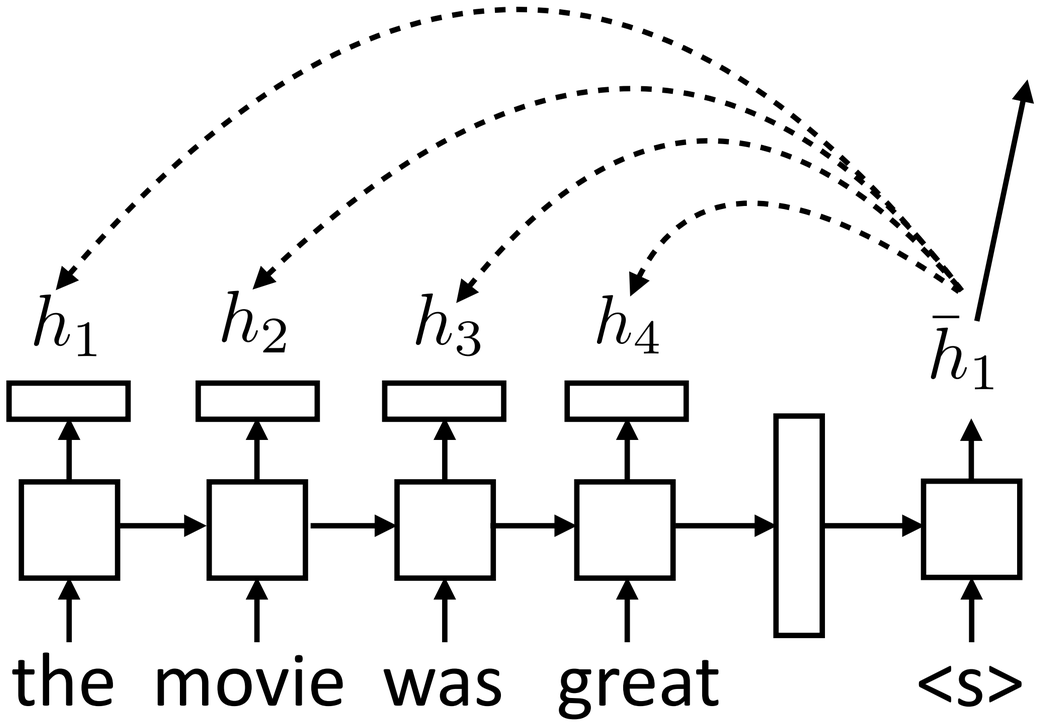
Attention

- For each decoder state, compute weighted sum of input states



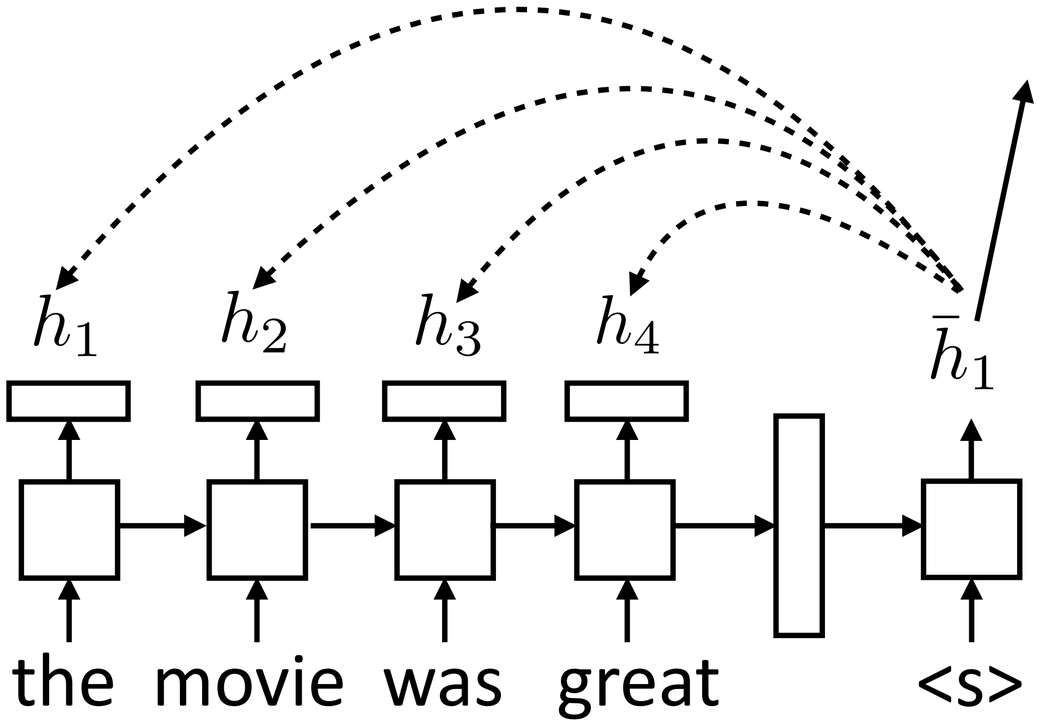
Attention

- For each decoder state, compute weighted sum of input states



Attention

- For each decoder state, compute weighted sum of input states

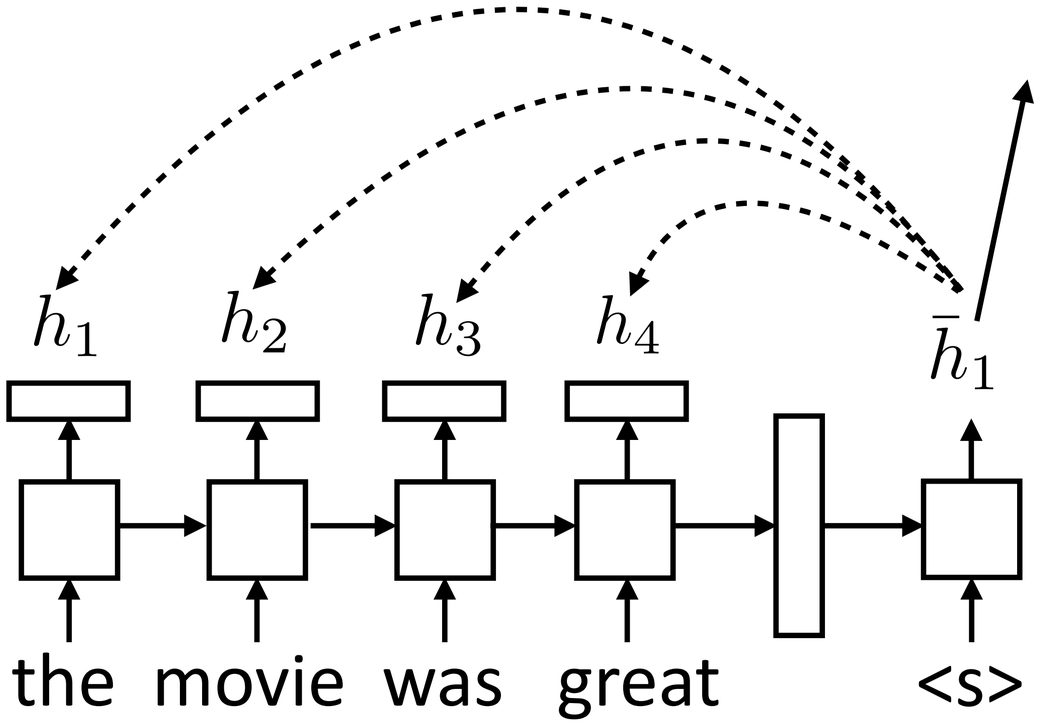


$$e_{ij} = f(\bar{h}_i, h_j)$$

- Unnormalized scalar weight

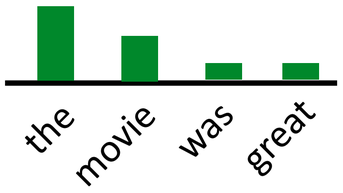
Attention

- For each decoder state, compute weighted sum of input states



$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

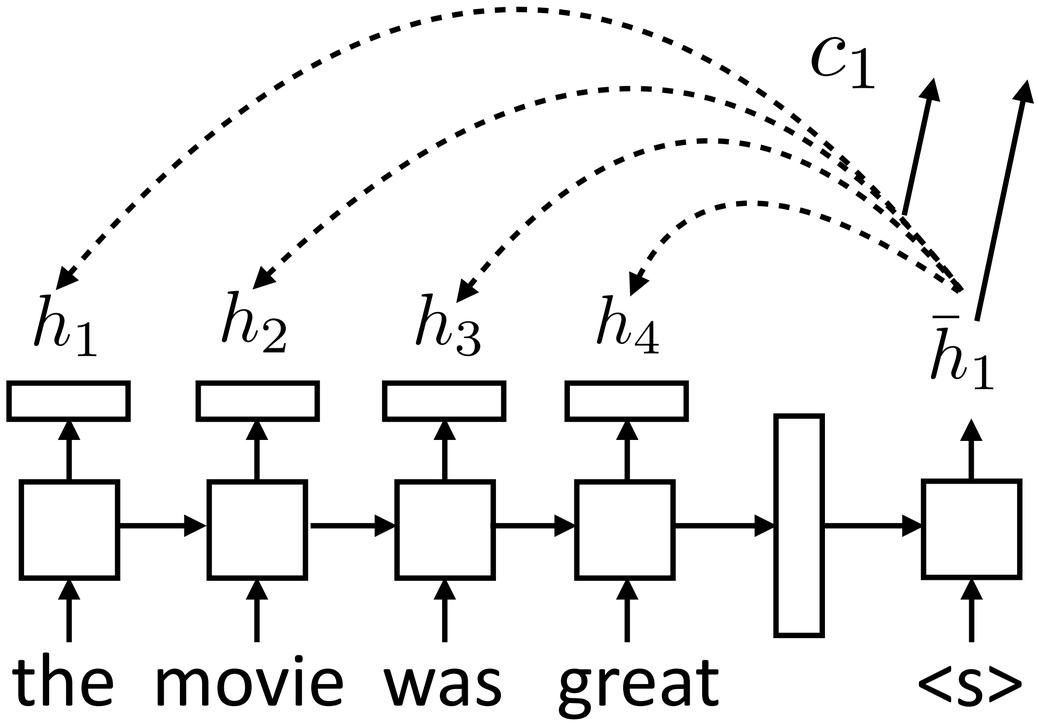
$$e_{ij} = f(\bar{h}_i, h_j)$$



- Unnormalized scalar weight

Attention

- For each decoder state, compute weighted sum of input states

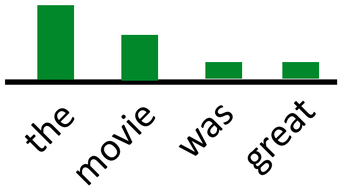


$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

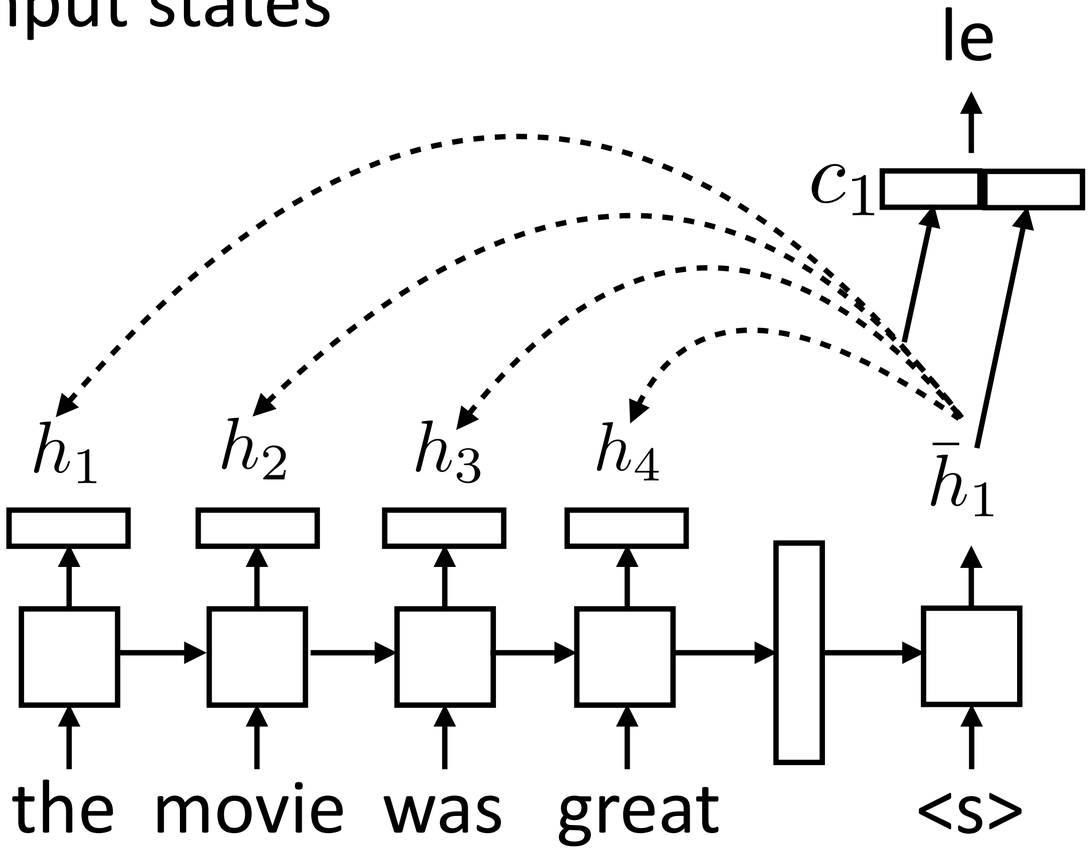
- Weighted sum of input hidden states (vector)



- Unnormalized scalar weight

Attention

- For each decoder state, compute weighted sum of input states

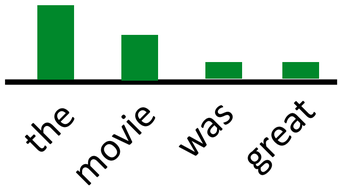


$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

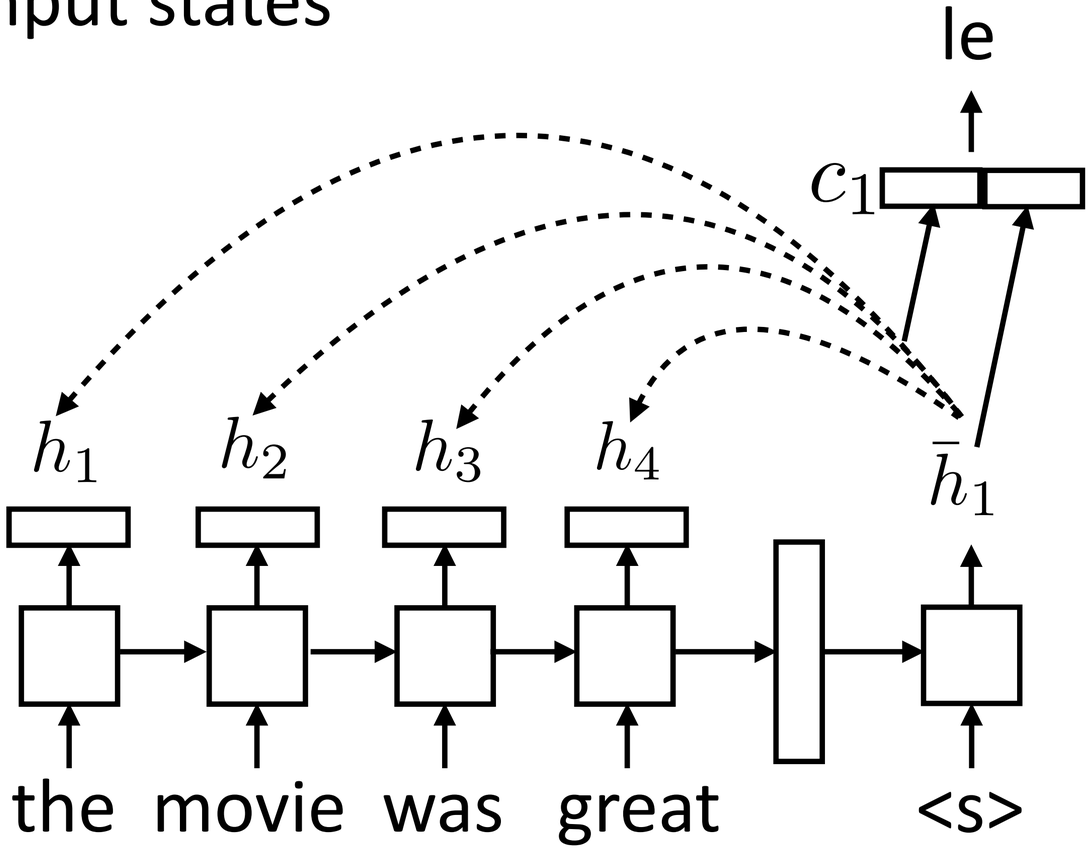
- Weighted sum of input hidden states (vector)



- Unnormalized scalar weight

Attention

- For each decoder state, compute weighted sum of input states

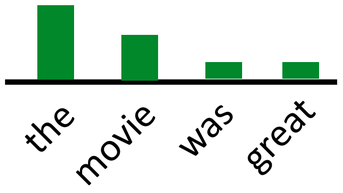


$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

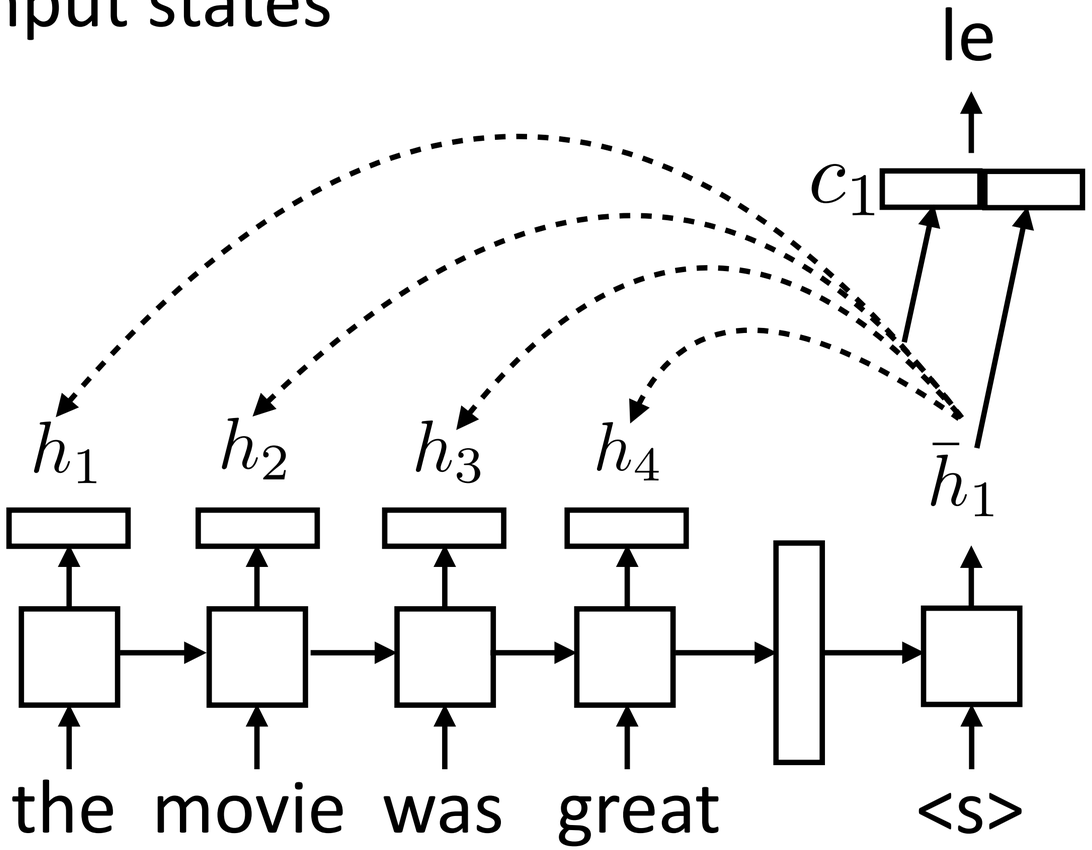


$$e_{ij} = f(\bar{h}_i, h_j)$$

- Unnormalized scalar weight

Attention

- For each decoder state, compute weighted sum of input states



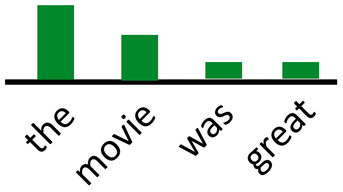
- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

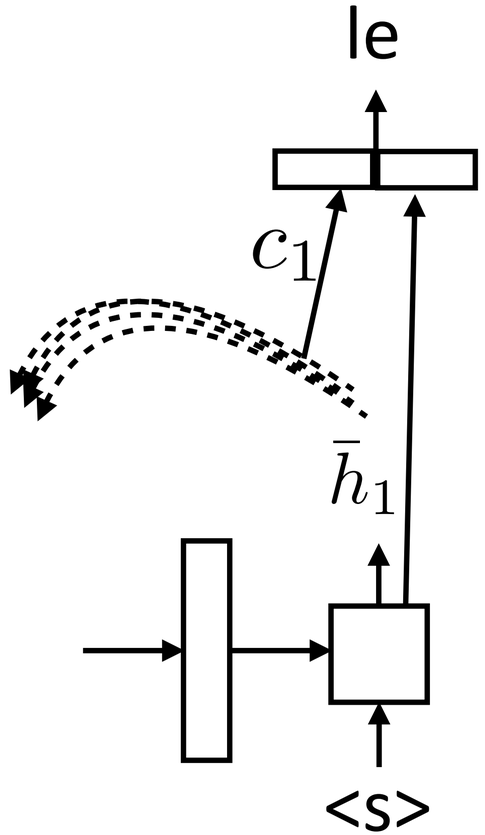
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



$$e_{ij} = f(\bar{h}_i, h_j)$$

- Unnormalized scalar weight

Attention



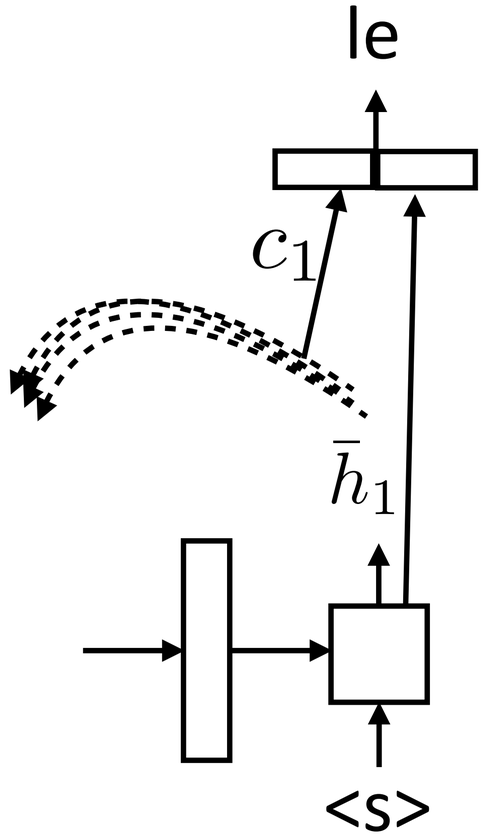
$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

Luong et al. (2015)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

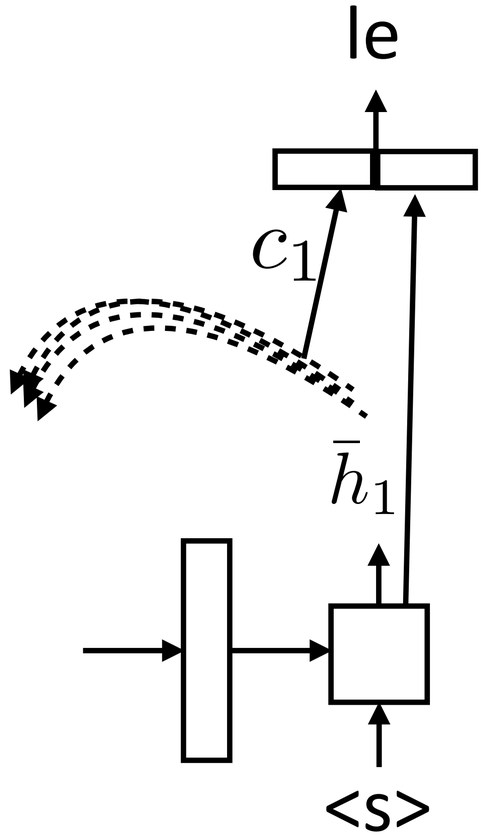
$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

- Bahdanau+ (2014): additive

Luong et al. (2015)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$

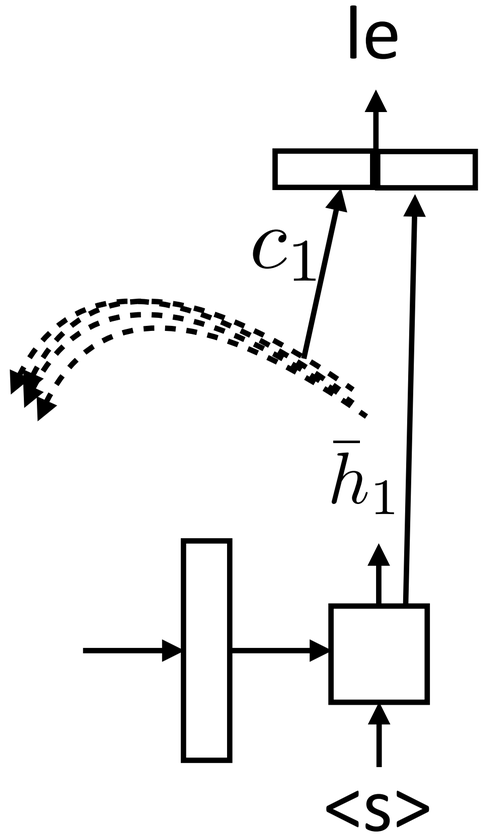
- Bahdanau+ (2014): additive

$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$

- Luong+ (2015): dot product

Luong et al. (2015)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$

- Bahdanau+ (2014): additive

$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$

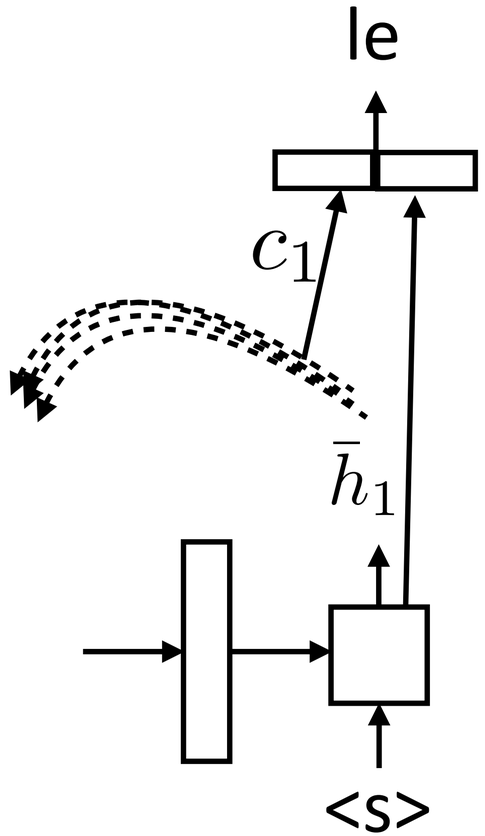
- Luong+ (2015): dot product

$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$

- Luong+ (2015): bilinear

Luong et al. (2015)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$

- Bahdanau+ (2014): additive

$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$

- Luong+ (2015): dot product

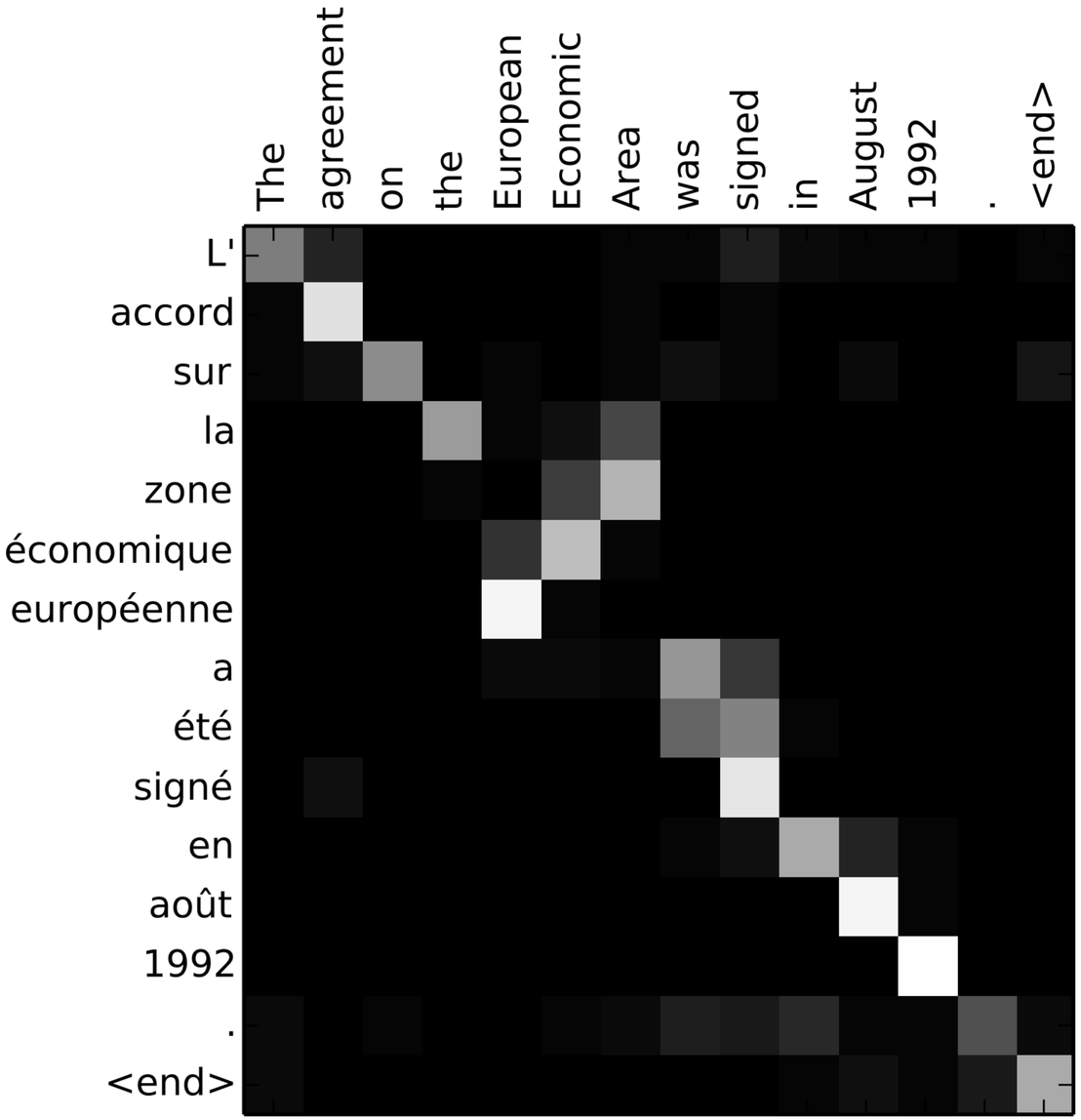
$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$

- Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

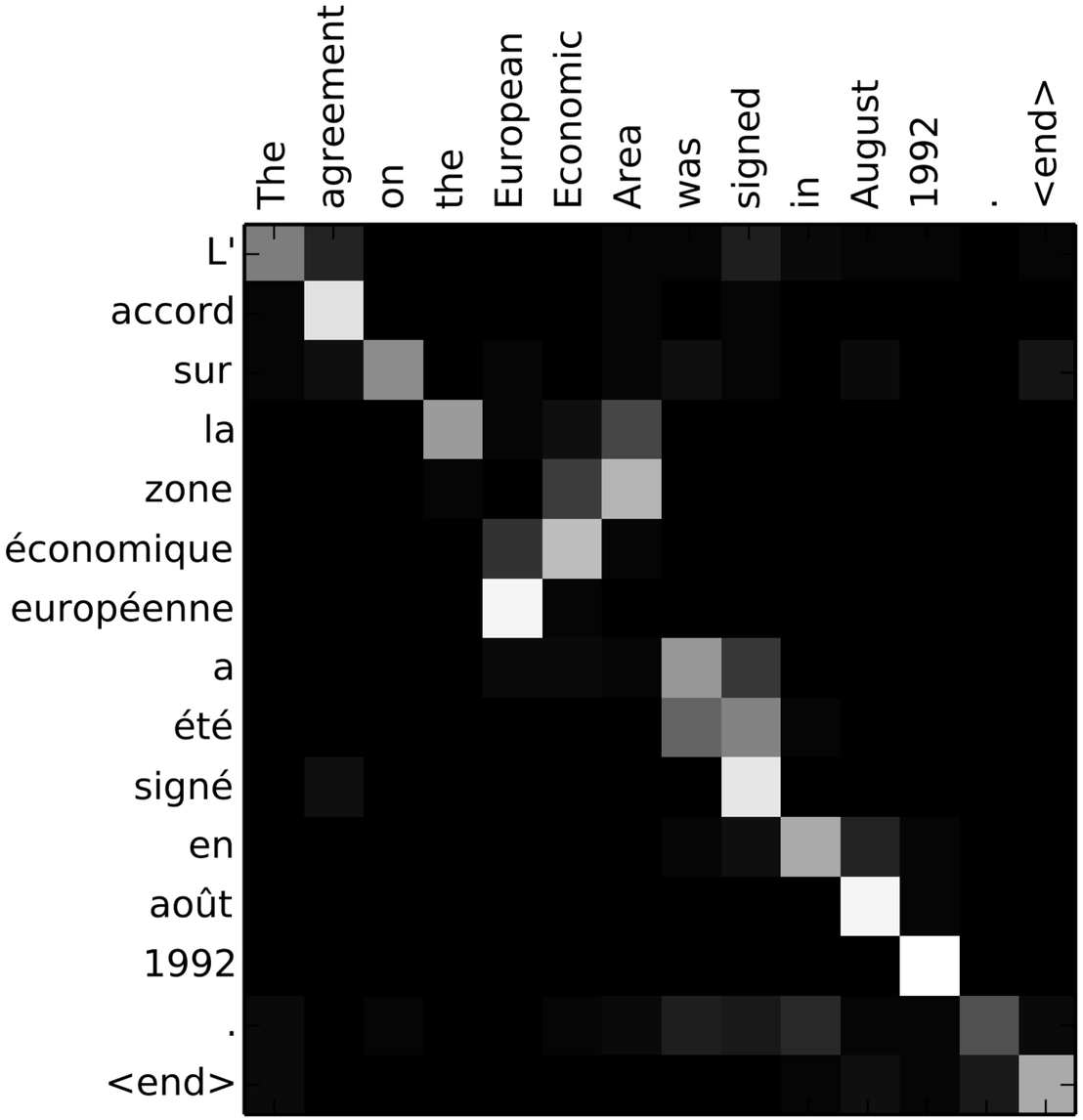
Luong et al. (2015)

Attention


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

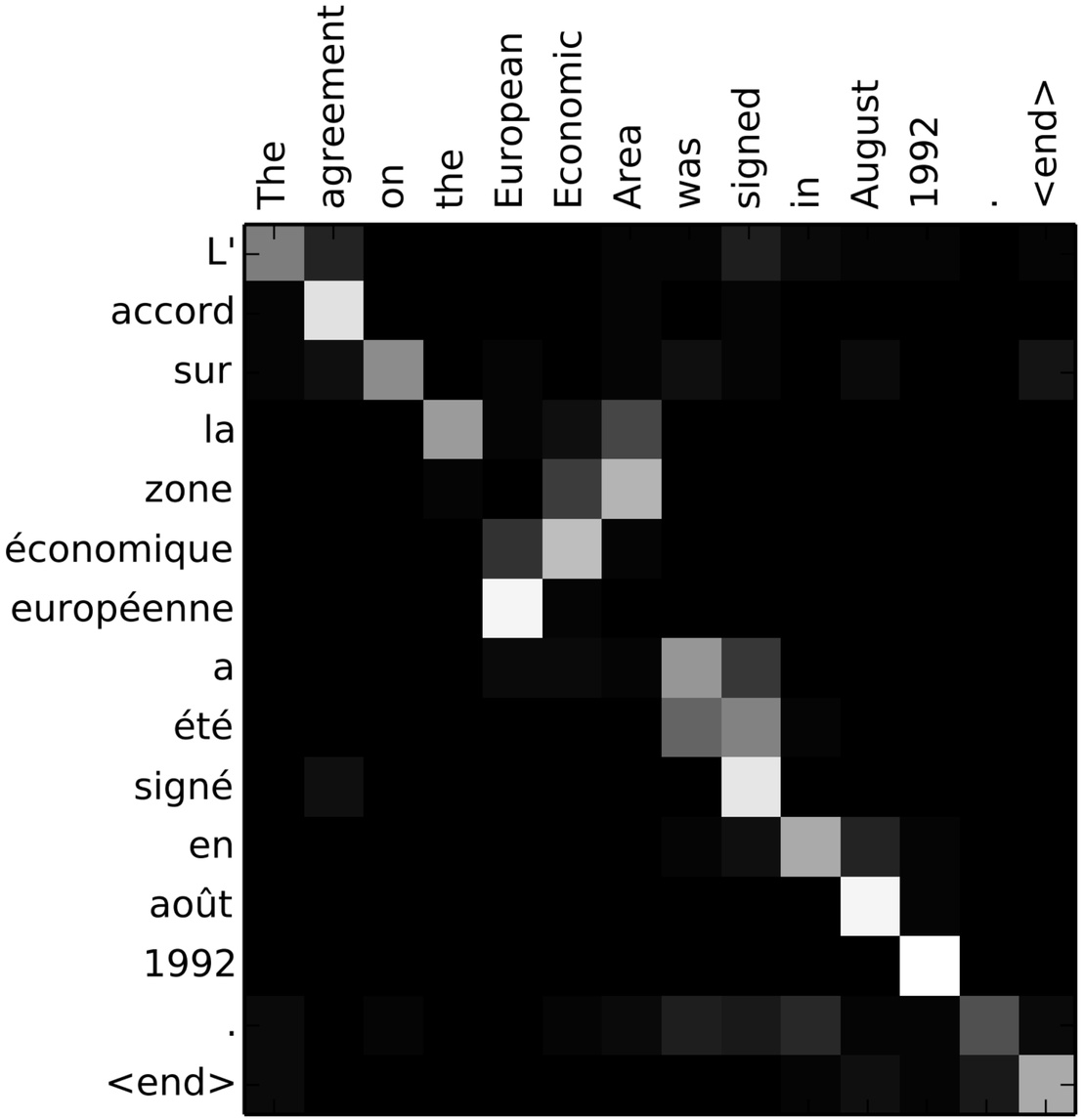
Attention

- Encoder hidden states capture contextual source word identity



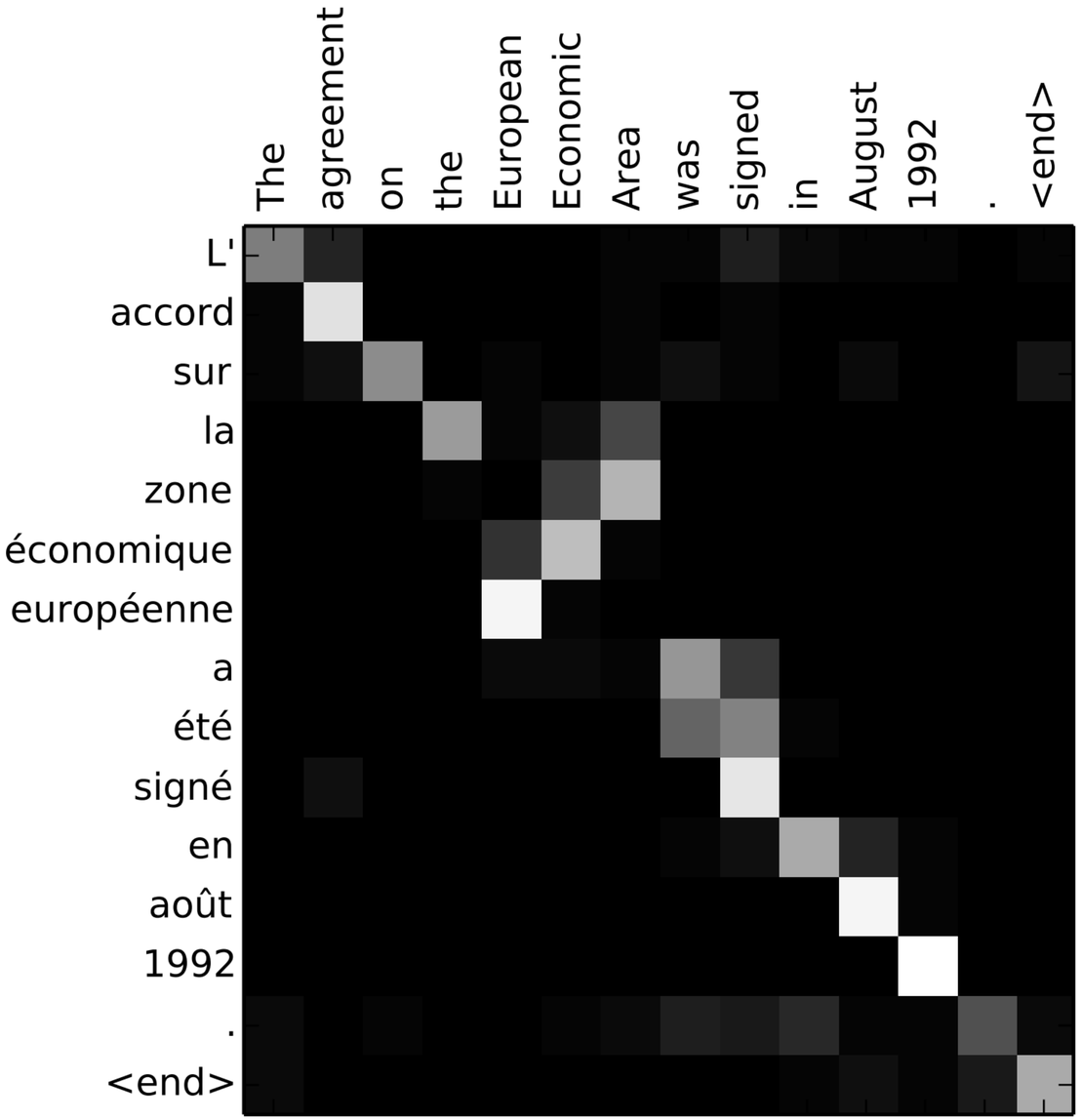
Attention

- Encoder hidden states capture contextual source word identity
- Decoder hidden states are now mostly responsible for selecting what to attend to



Attention

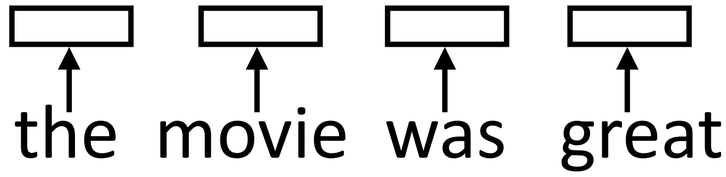
- Encoder hidden states capture contextual source word identity
- Decoder hidden states are now mostly responsible for selecting what to attend to
- Doesn’t take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations



Transformers for MT

Self-Attention

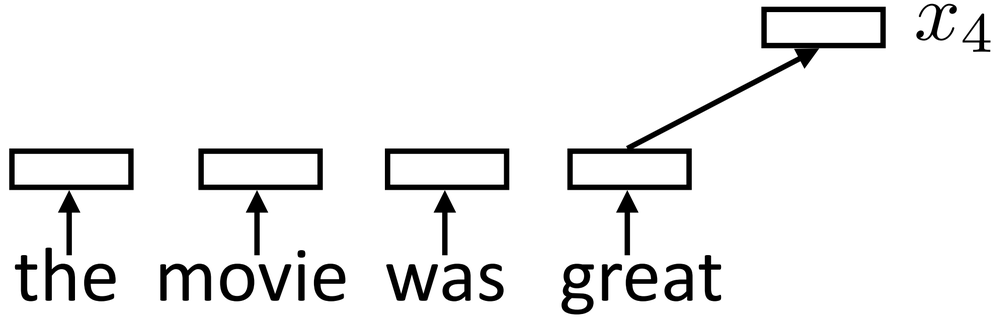
- Each word forms a “query” which then computes attention over each word



Vaswani et al. (2017)

Self-Attention

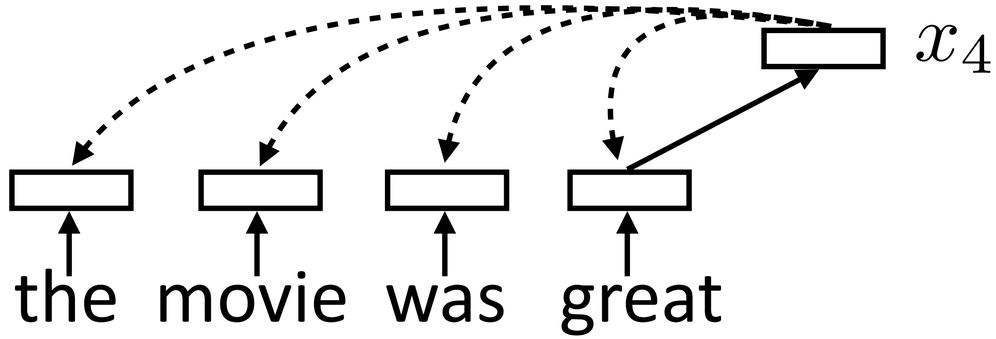
- Each word forms a “query” which then computes attention over each word



Vaswani et al. (2017)

Self-Attention

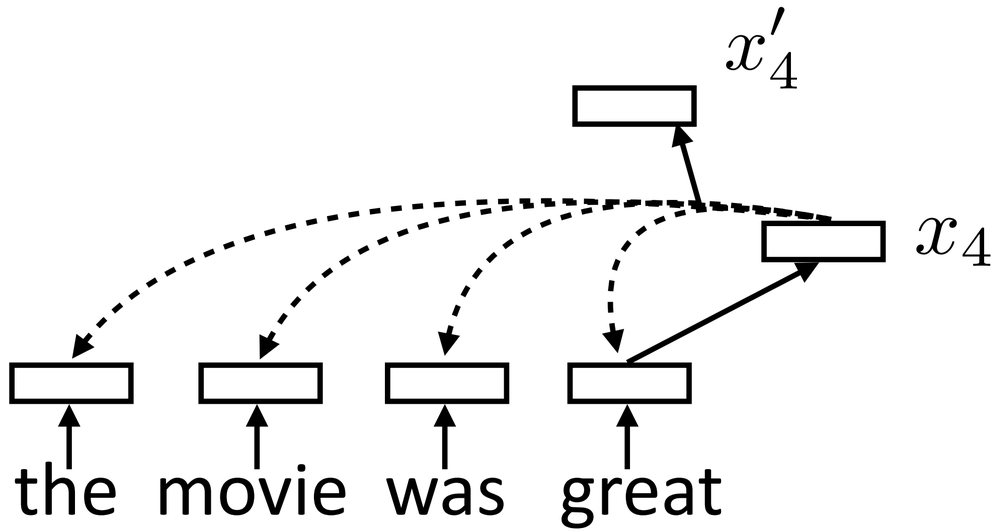
- Each word forms a “query” which then computes attention over each word



Vaswani et al. (2017)

Self-Attention

- Each word forms a “query” which then computes attention over each word

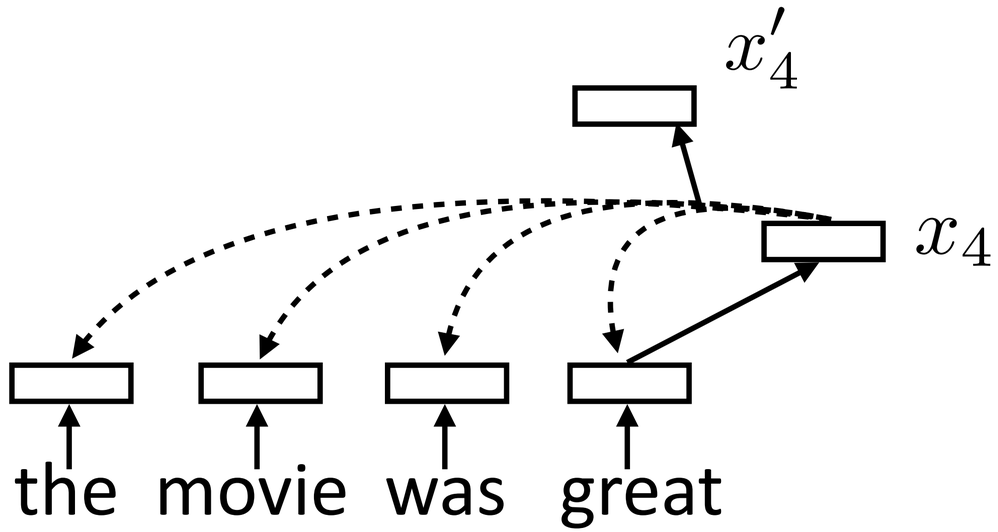


Vaswani et al. (2017)

Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j)$$ scalar



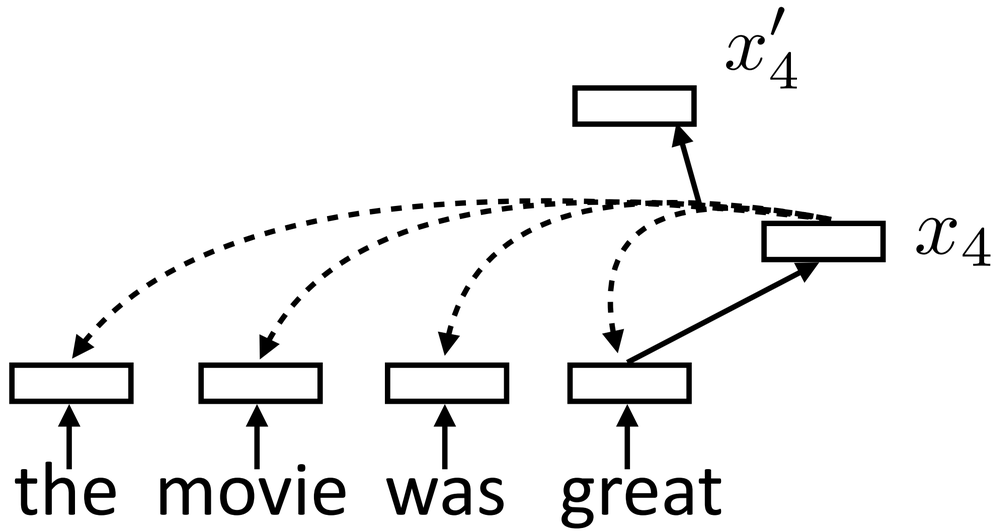
Vaswani et al. (2017)

Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j)$$ scalar

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j$$ vector = sum of scalar * vector



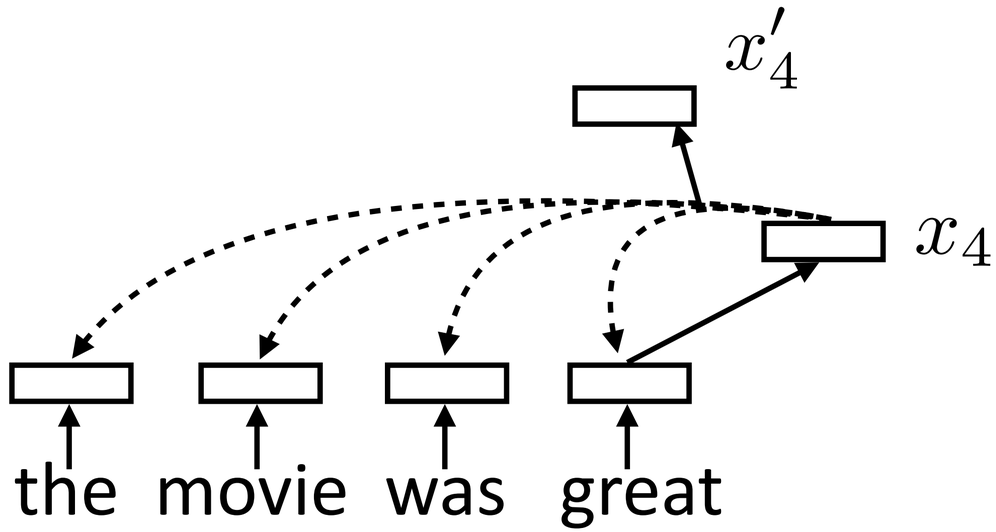
Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$



- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

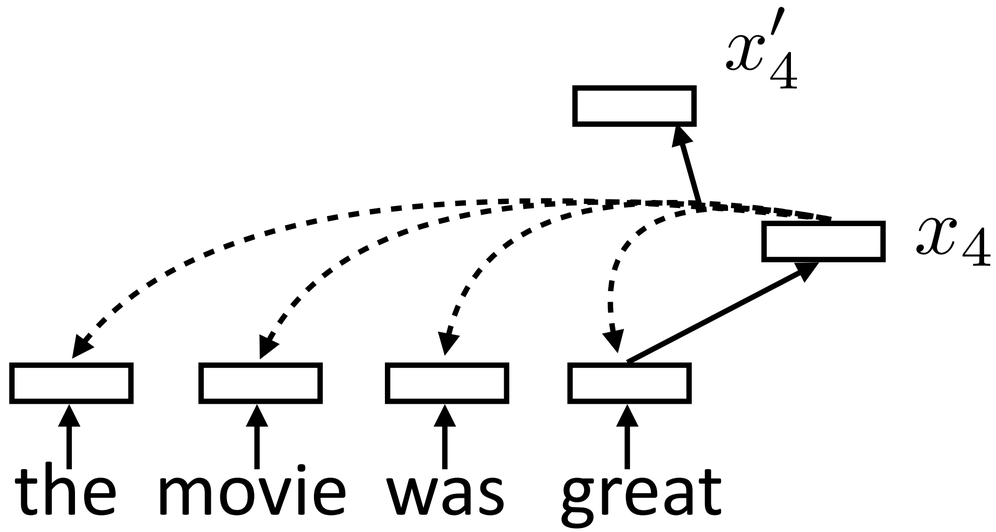
Vaswani et al. (2017)

Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x_i' = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$



- Multiple “heads” analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j)$$

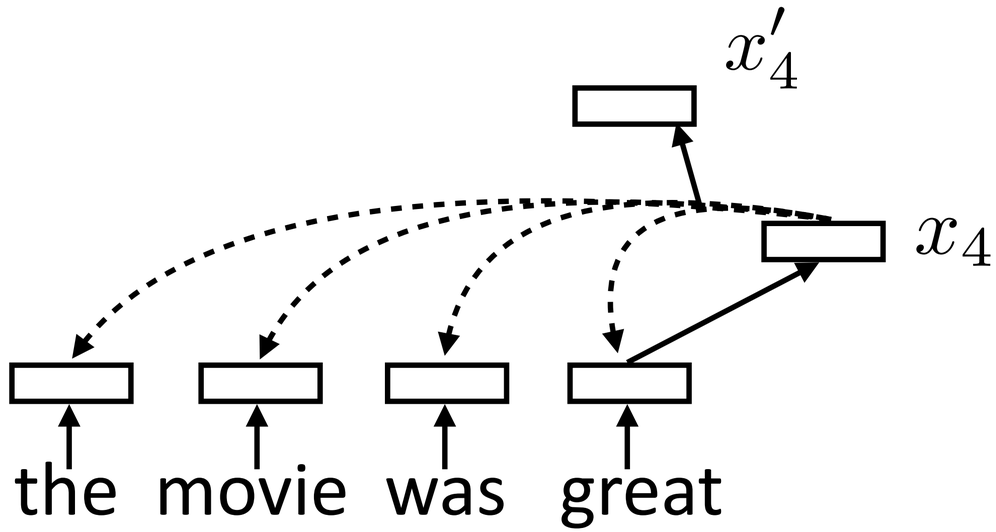
Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$



- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

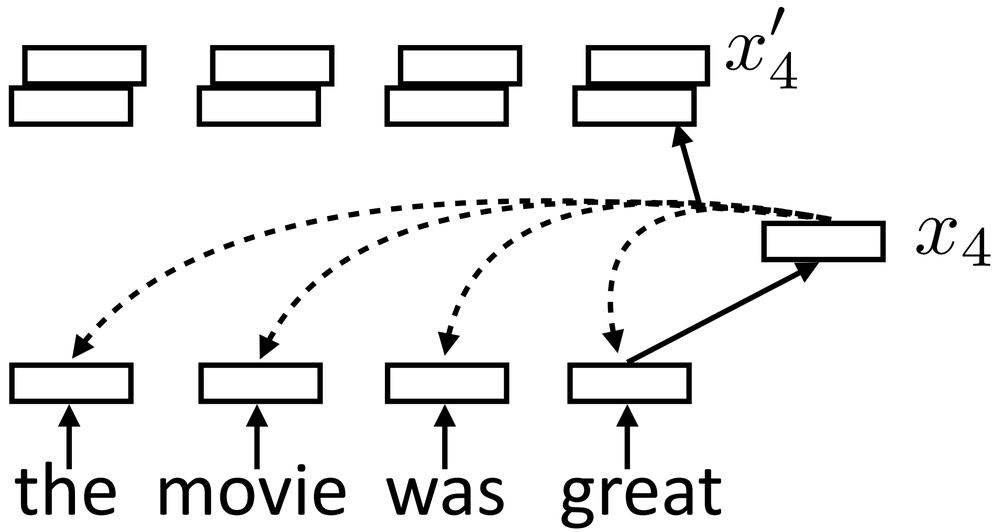
Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$

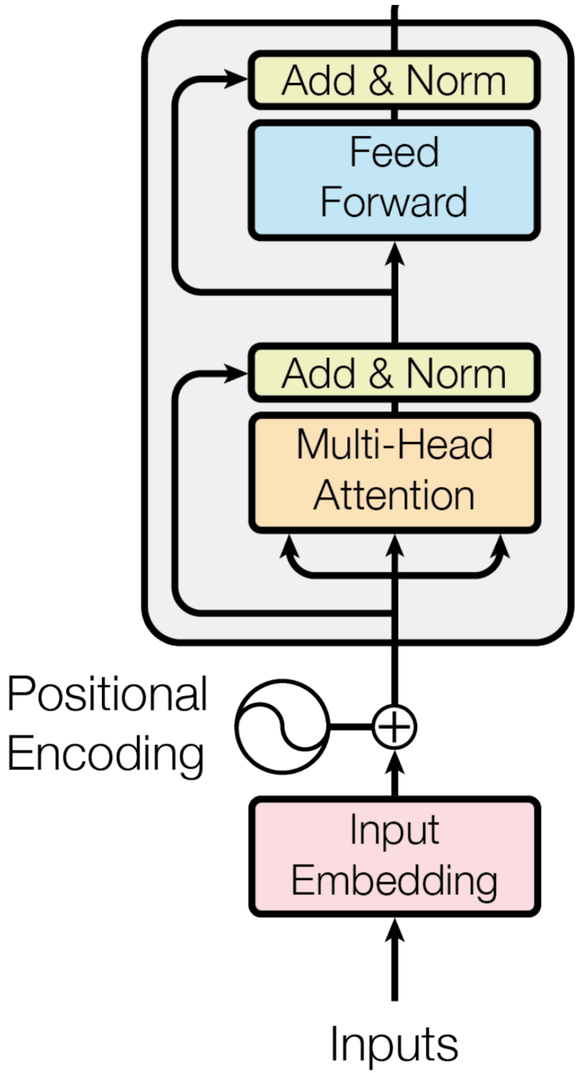


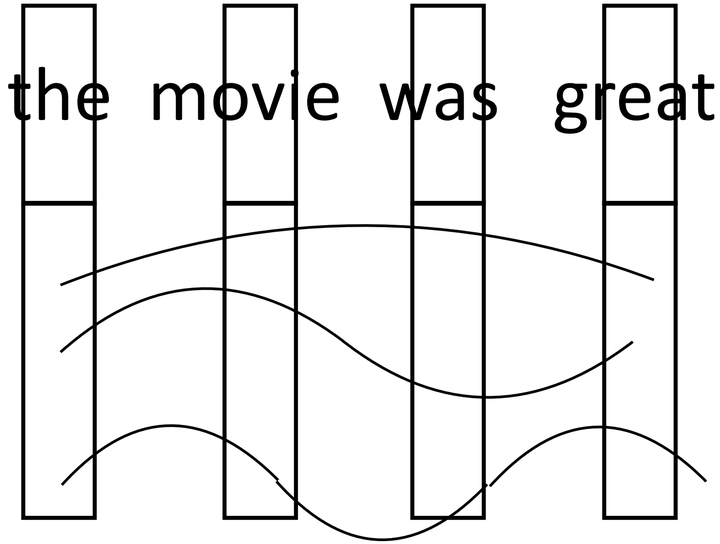
- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

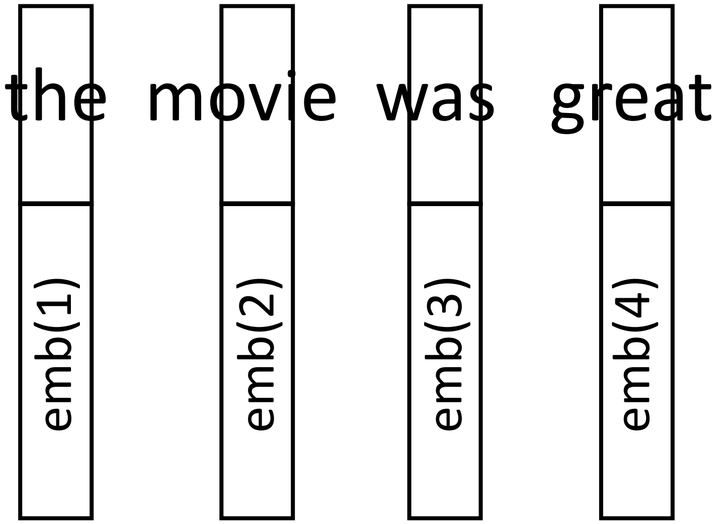
$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

Transformers



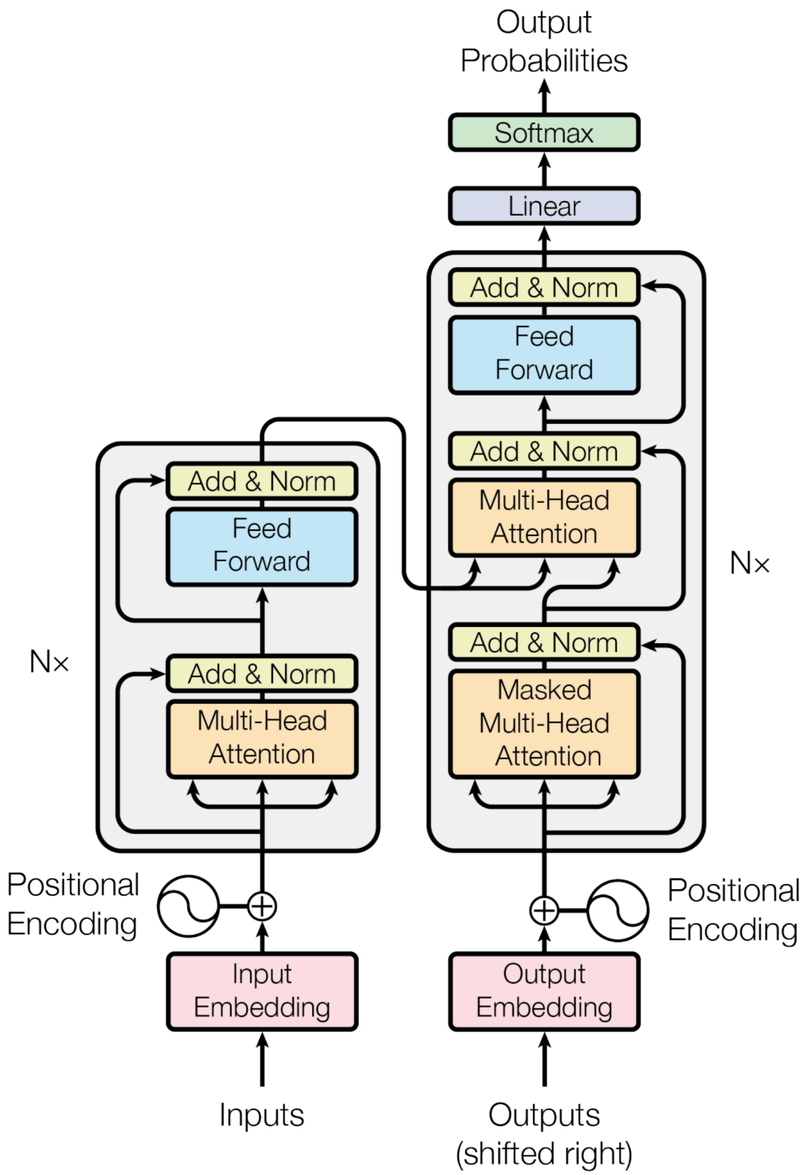




- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

Vaswani et al. (2017)

Transformers



- Encoder and decoder are both transformers
- Decoder consumes the previous generated token (and attends to input), but has *no recurrent state*

Vaswani et al. (2017)

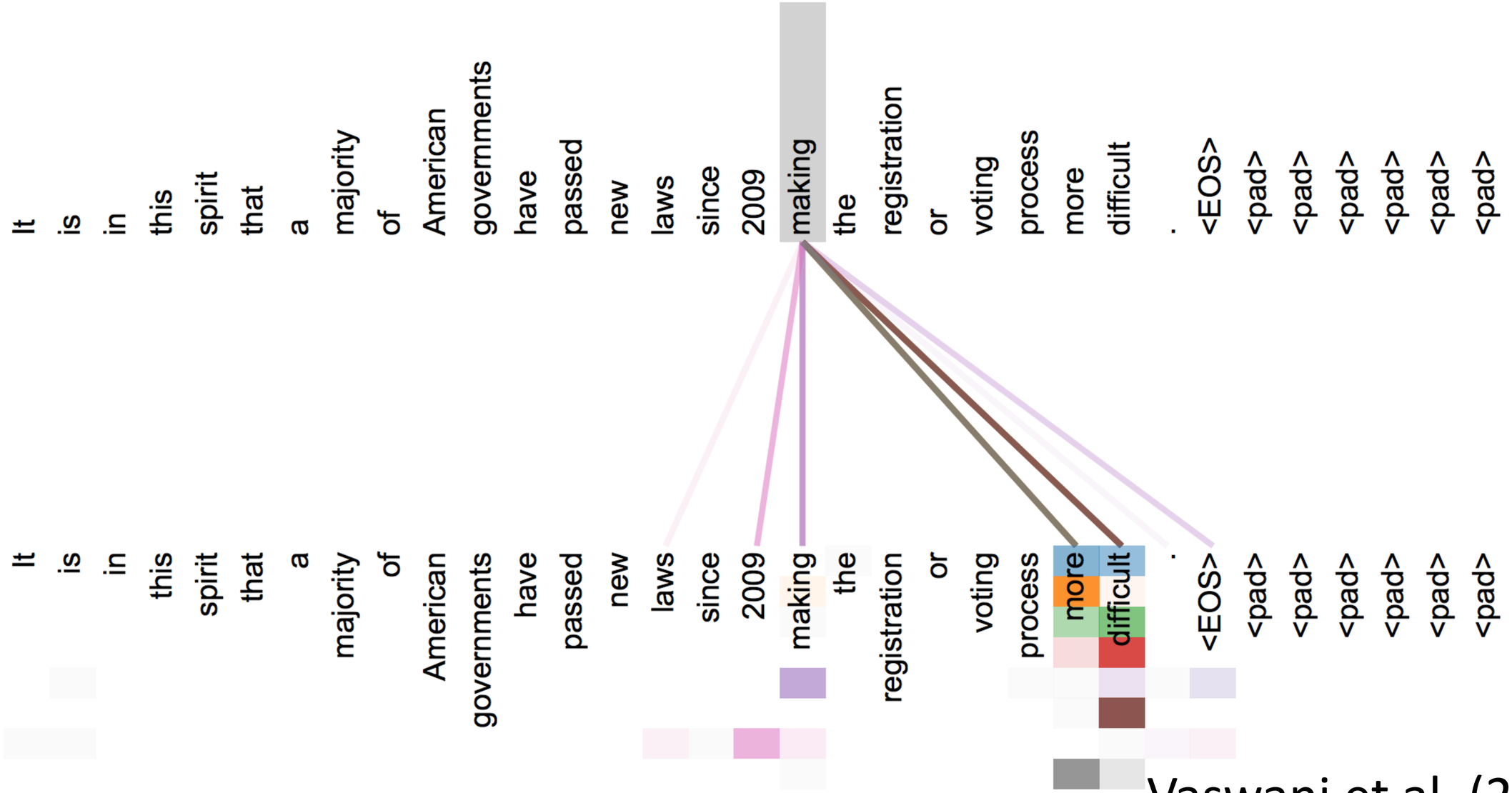
Transformers

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	**41.29**
Transformer (base model)	27.3	38.1
Transformer (big)	**28.4**	**41.8**

- Big = 6 layers, 1000 dim for each token, 16 heads, base = 6 layers + other params halved

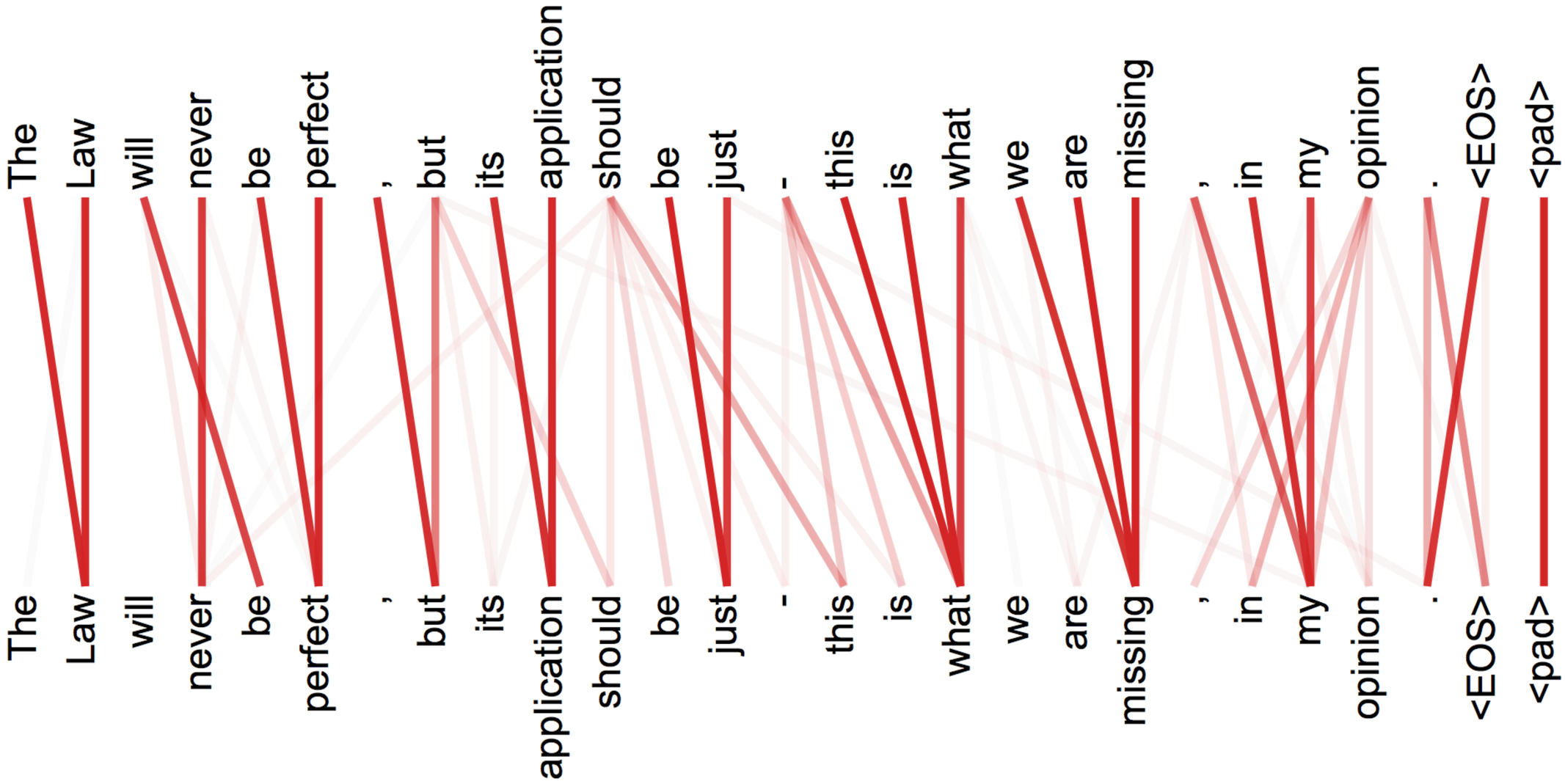
Vaswani et al. (2017)

Visualization



Vaswani et al. (2017)

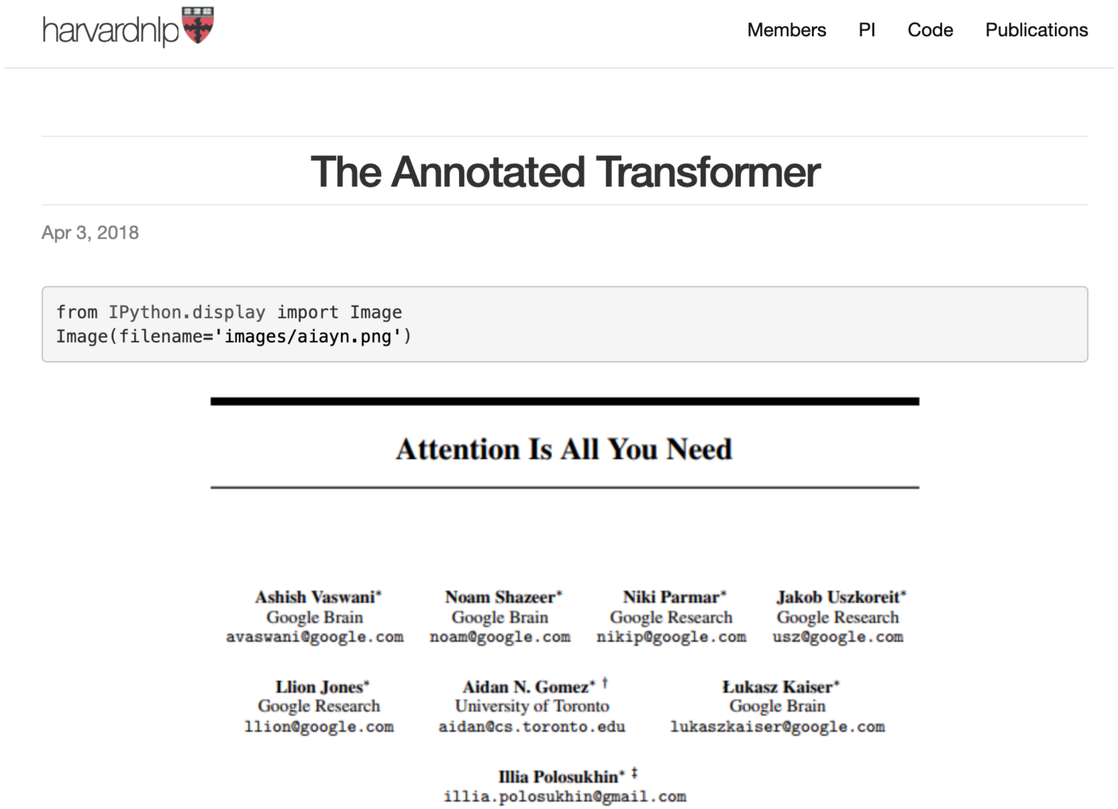
Visualization



Vaswani et al. (2017)

Transformer Implementations

https://nlp.seas.harvard.edu/2018/04/03/attention.html



harvardnlp Members PI Code Publications

The Annotated Transformer

Apr 3, 2018

```
from IPython.display import Image
Image(filename='images/aiayn.png')
```

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer Implementations

https://github.com/pytorch/fairseq

README.md

Fairseq(-py) is a sequence modeling toolkit that allows researchers and developers to train custom models for translation, summarization, language modeling and other text generation tasks.

We provide reference implementations of various sequence modeling papers:

▶ List of implemented papers

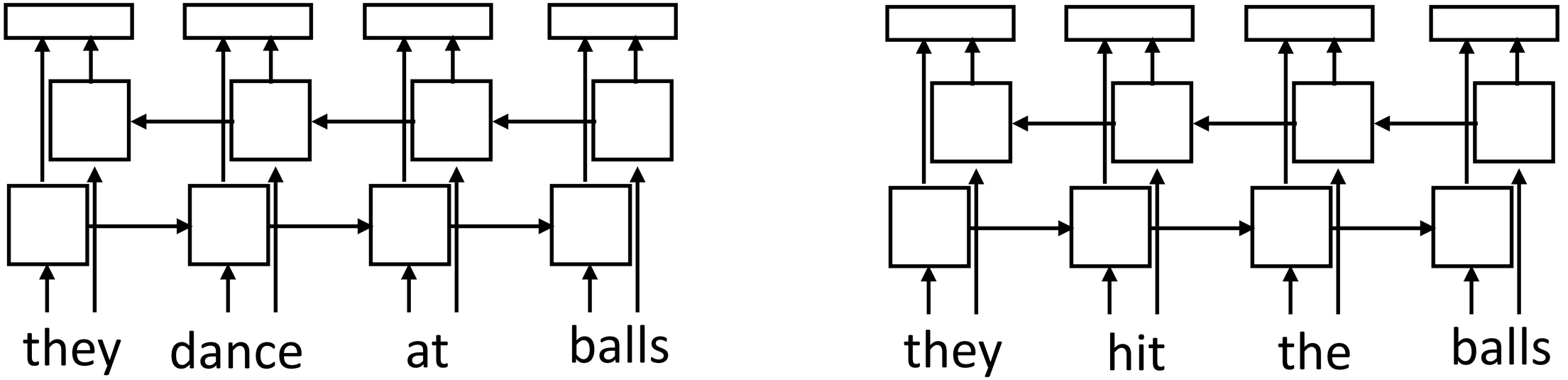
What's New:

- March 2021 Added full parameter and optimizer state sharding + CPU offloading
- February 2021 Added LASER training code
- December 2020: Added Adaptive Attention Span code
- December 2020: GottBERT model and code released
- November 2020: Adopted the Hydra configuration framework

Pretraining / ELMo

Recall: Context-dependent Embeddings

- How to handle different word senses? One vector for *balls*

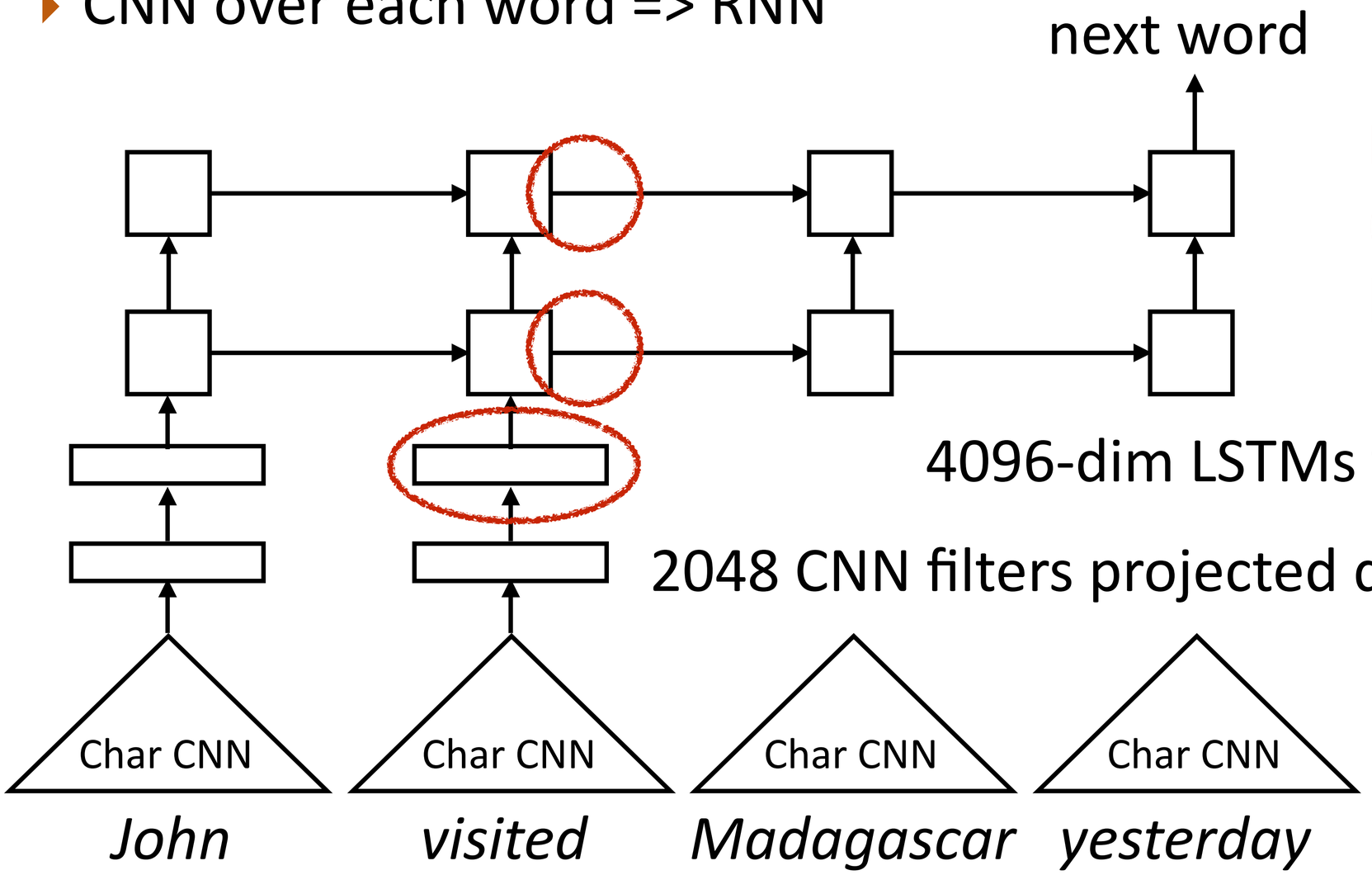


- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors

Peters et al. (2018)

ELMo

- CNN over each word => RNN



Representation of *visited* (plus vectors from backwards LM)

4096-dim LSTMs w/ 512-dim projections

2048 CNN filters projected down to 512-dim

Peters et al. (2018)

Results: Frozen ELMo

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

- Massive improvements across 5 benchmark datasets: question answering, natural language inference, semantic role labeling (discussed later in the course), coreference resolution, named entity recognition, and sentiment analysis

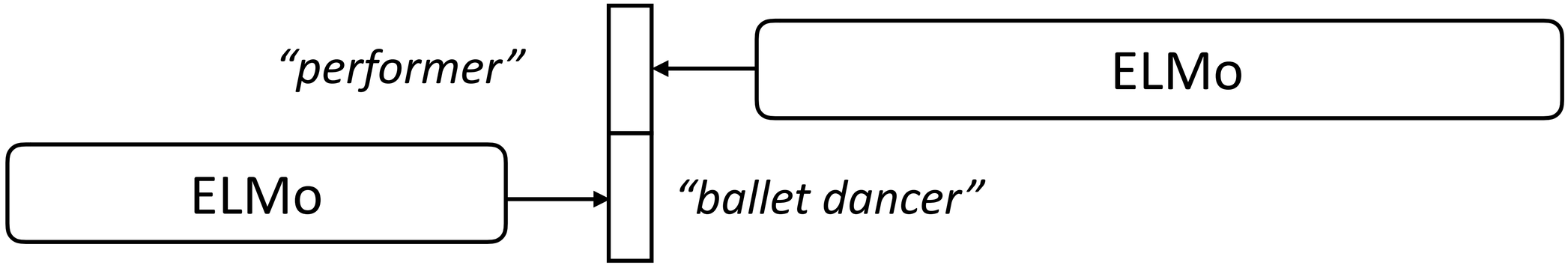
BERT

BERT

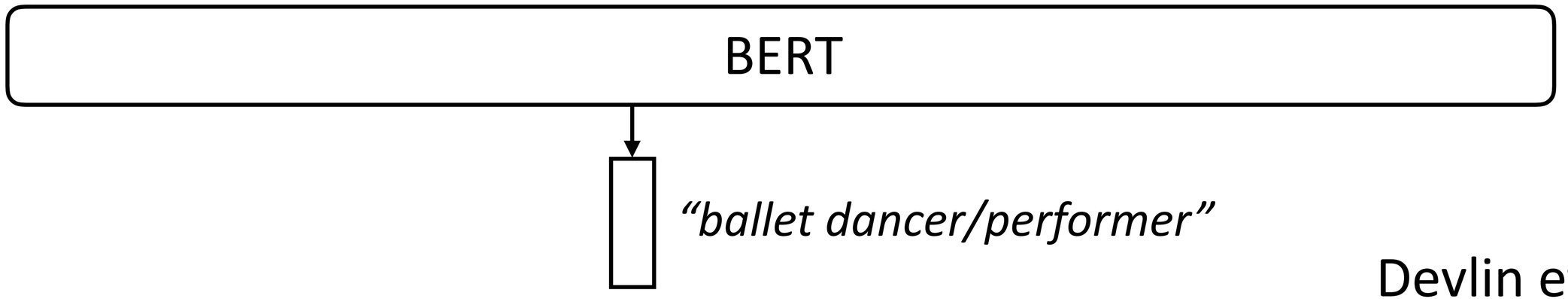
- AI2 made ELMo in spring 2018, GPT was released in summer 2018, BERT came out October 2018
- Three major changes compared to ELMo:
  - Transformers instead of LSTMs (transformers in GPT as well)
  - Bidirectional <=> Masked LM objective instead of standard LM
  - Fine-tune instead of freeze at test time

BERT

- ELMo is a unidirectional model (as is GPT): we can concatenate two unidirectional models, but is this the right thing to do?
- ELMo reprs look at each direction in isolation; BERT looks at them jointly



A stunning ballet dancer, Copeland is one of the best performers to see live.

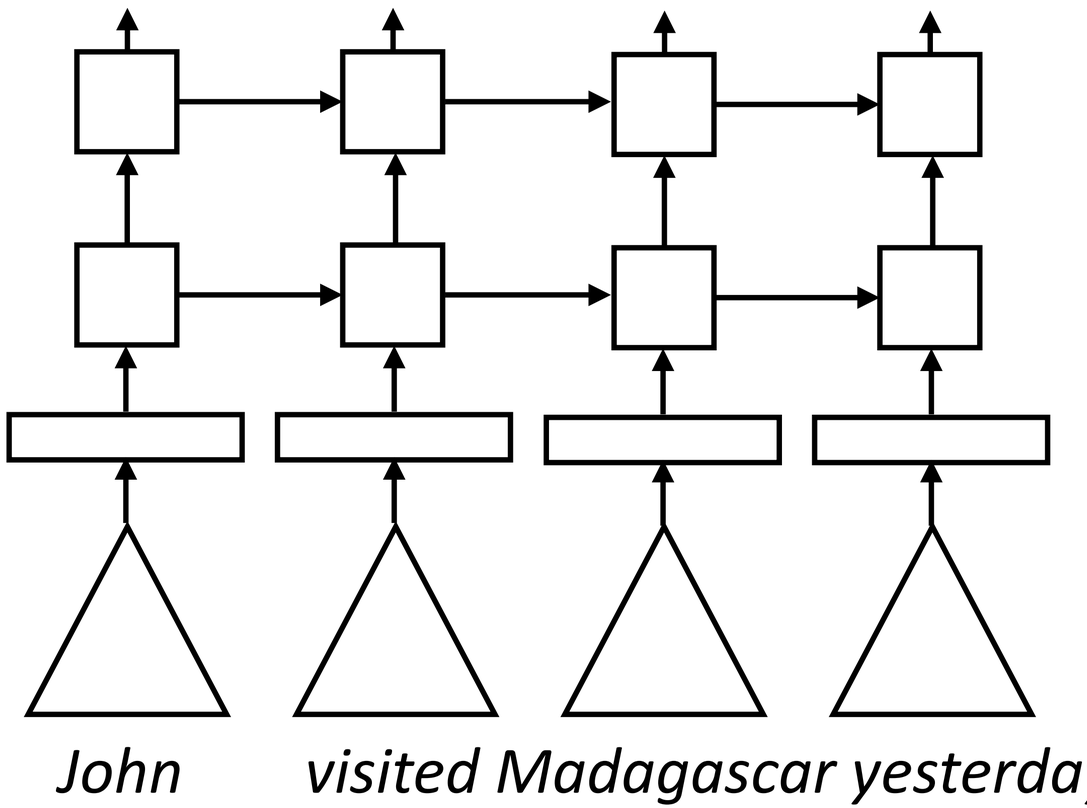


Devlin et al. (2019)

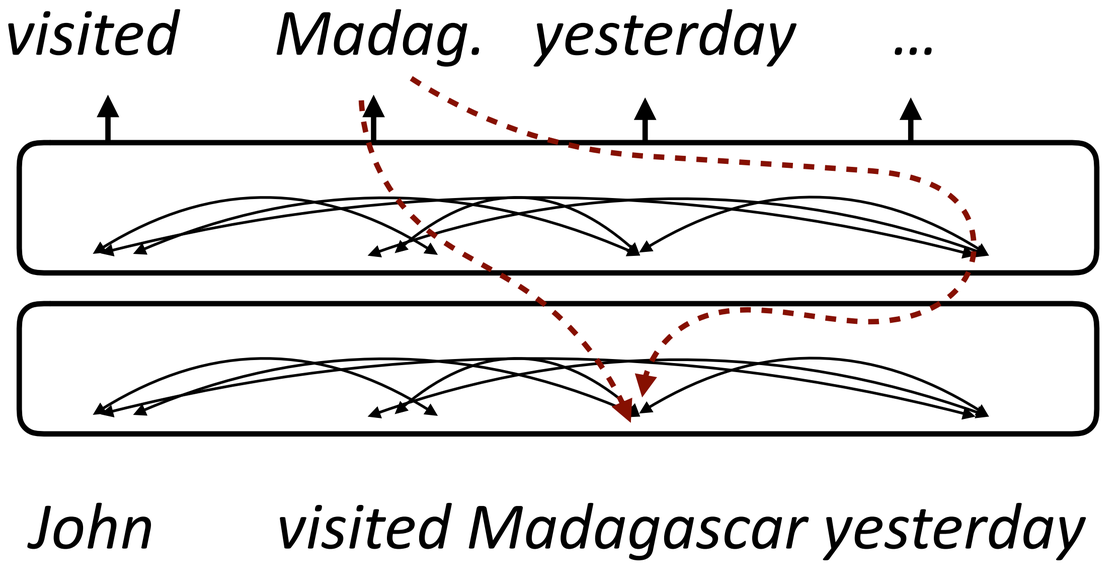
BERT

- How to learn a “deeply bidirectional” model? What happens if we just replace an LSTM with a transformer?

ELMo (Language Modeling)



BERT

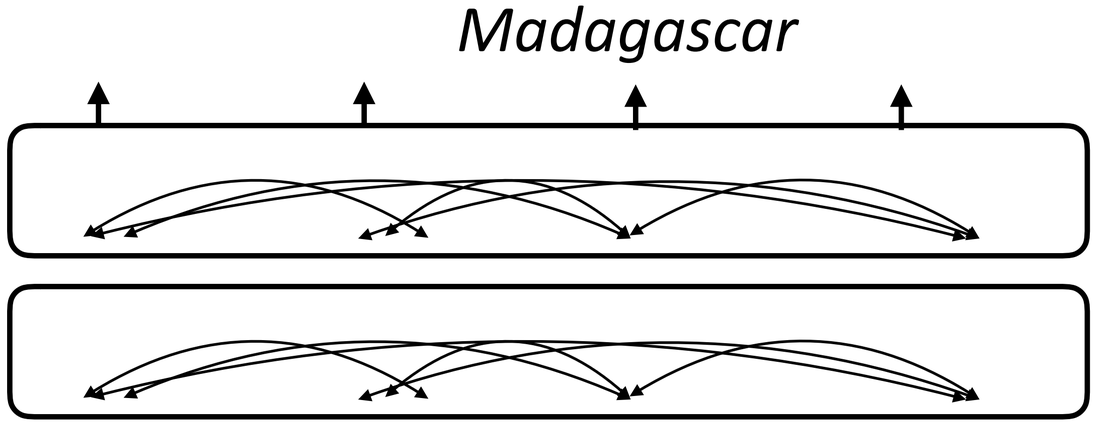


- Transformer LMs have to be “one-sided” (only attend to previous tokens), not what we want

Masked Language Modeling

- How to prevent cheating? Next word prediction fundamentally doesn't work for bidirectional models, instead do *masked language modeling*

- BERT formula: take a chunk of text, predict 15% of the tokens
  - For 80% (of the 15%), replace the input token with [MASK]
  - For 10%, replace w/random
  - For 10%, keep same



John *visited* [MASK] *yesterday*

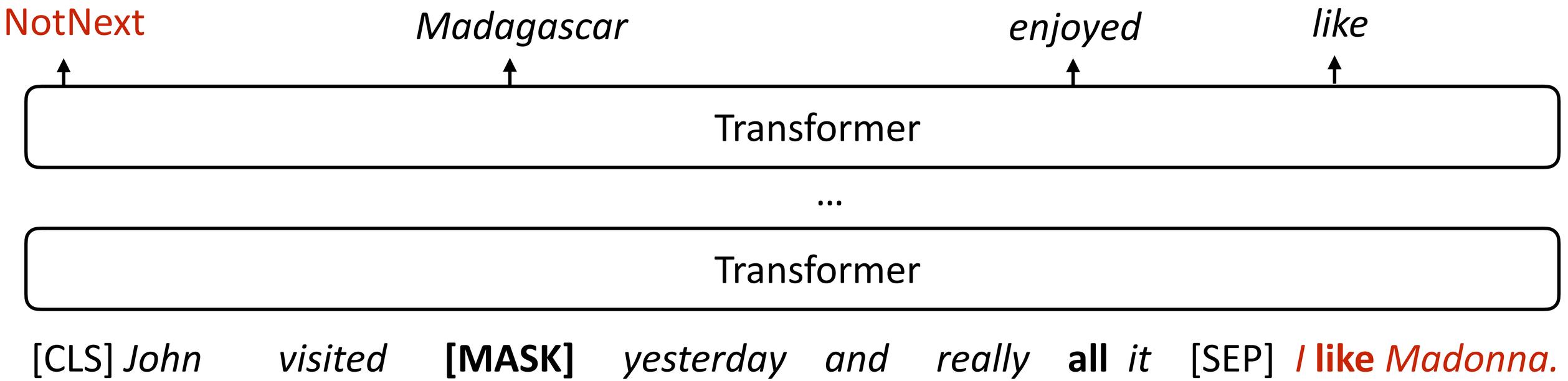
John *visited* **of** *yesterday*

John *visited* Madagascar *yesterday*

Devlin et al. (2019)

Next “Sentence” Prediction

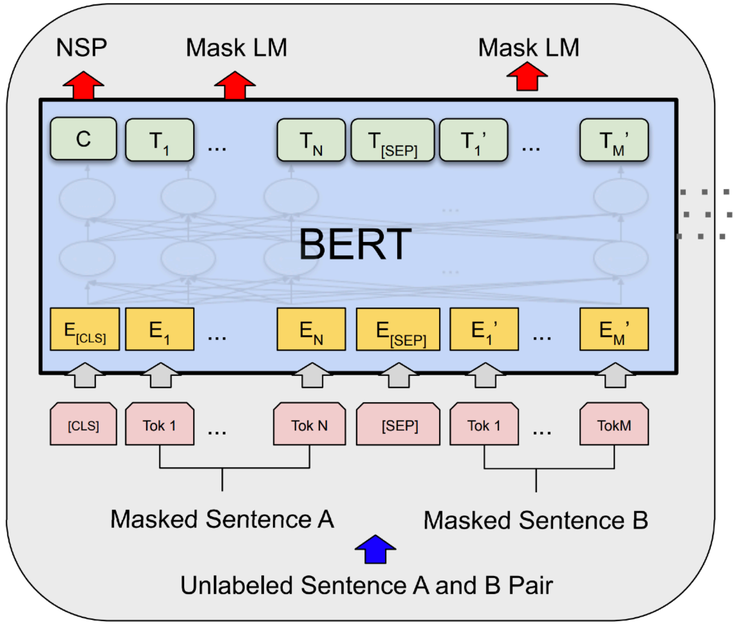
- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk. Predict whether the next chunk is the “true” next
- BERT objective: masked LM + next sentence prediction

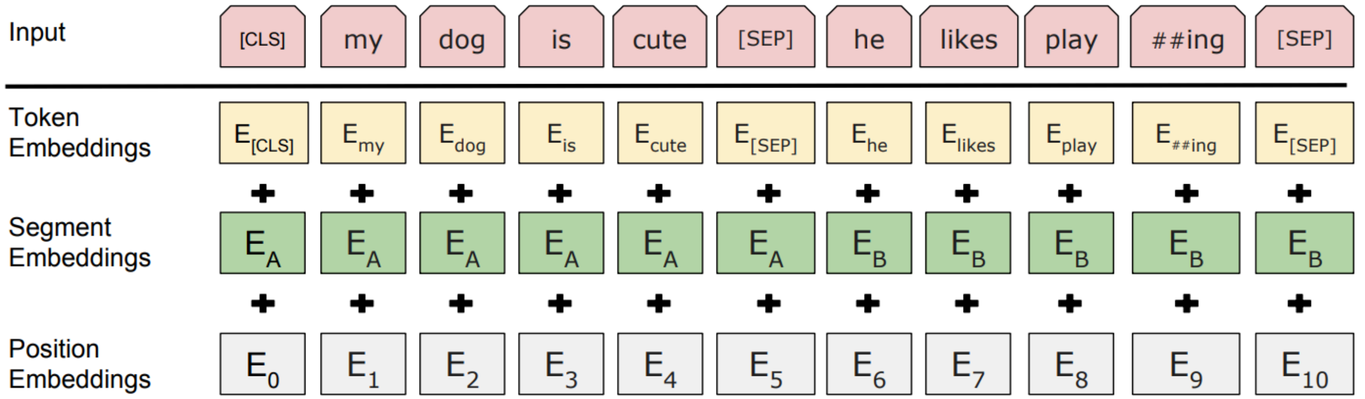


Devlin et al. (2019)

BERT Architecture

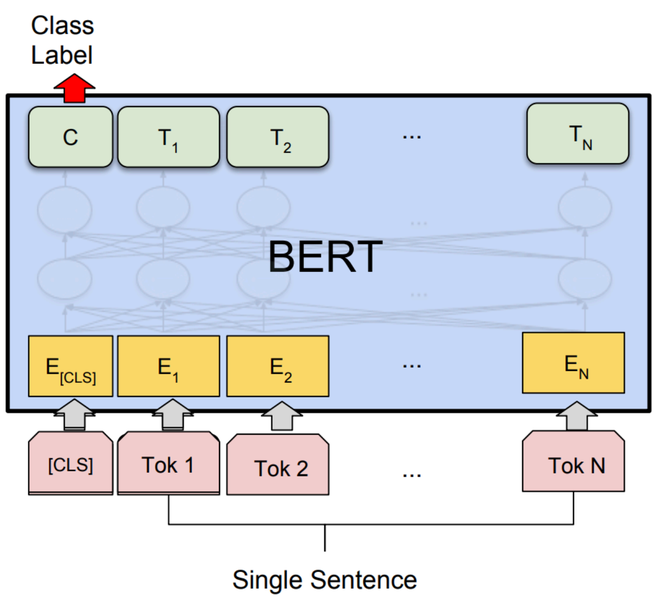
- BERT Base: 12 layers, 768-dim per wordpiece token, 12 heads. Total params = 110M
- BERT Large: 24 layers, 1024-dim per wordpiece token, 16 heads. Total params = 340M
- Positional embeddings and segment embeddings, 30k word pieces
- This is the model that gets **pre-trained** on a large corpus



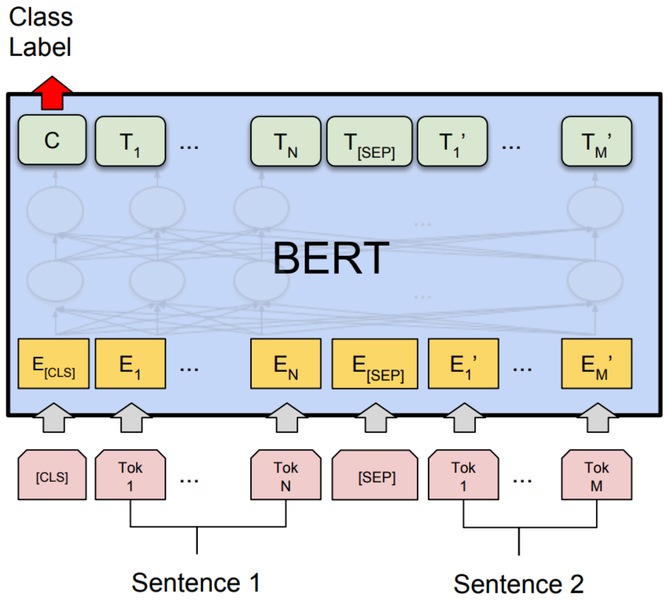


Devlin et al. (2019)

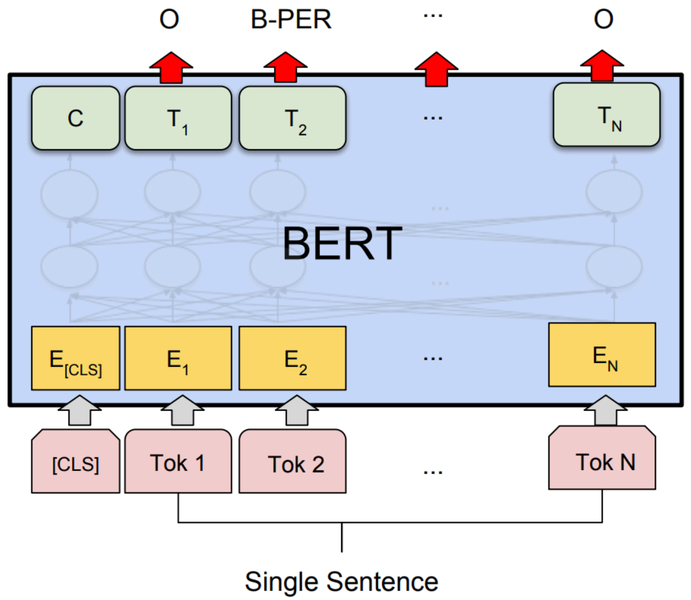
What can BERT do?



(b) Single Sentence Classification Tasks: SST-2, CoLA



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



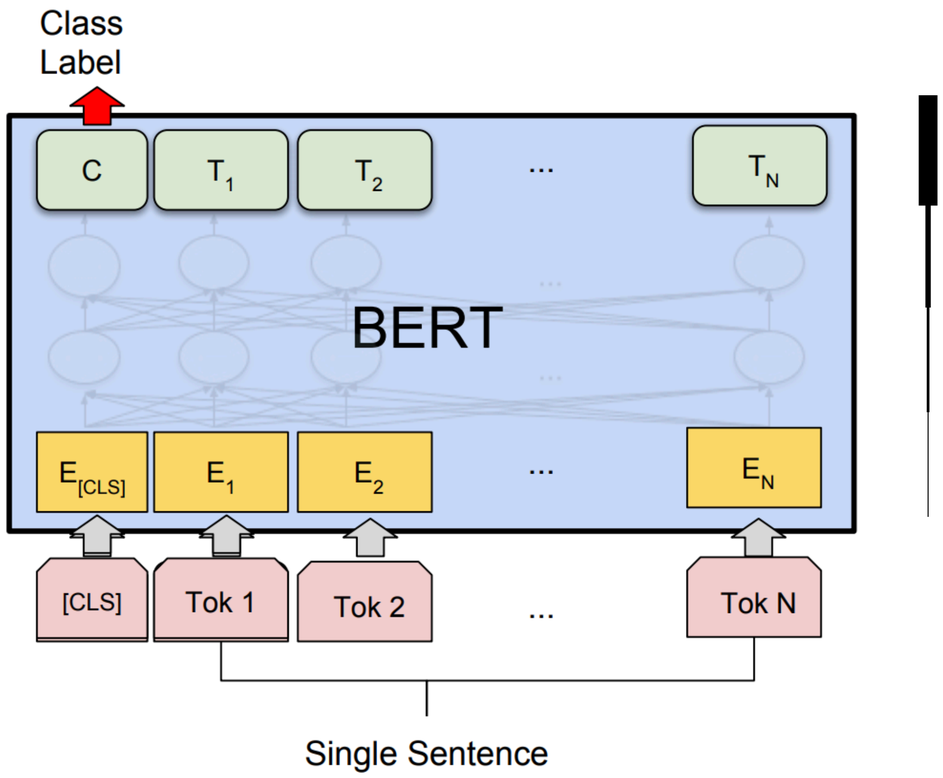
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

- CLS token is used to provide classification decisions
- Sentence pair tasks (entailment): feed both sentences into BERT
- BERT can also do tagging by predicting tags at each word piece

Devlin et al. (2019)

Fine-tuning BERT

- Fine-tune for 1-3 epochs, batch size 2-32, learning rate 2e-5 - 5e-5



(b) Single Sentence Classification Tasks: SST-2, CoLA

- Large changes to weights up here (particularly in last layer to route the right information to [CLS])
- Smaller changes to weights lower down in the transformer
- Small LR and short fine-tuning schedule mean weights don't change much
- More complex "triangular learning rate" schemes exist

Fine-tuning BERT

Pretraining	Adaptation	NER CoNLL 2003	SA SST-2	Nat. lang. inference MNLI	Nat. lang. inference SICK-E	Semantic textual similarity SICK-R	Semantic textual similarity MRPC	Semantic textual similarity STS-B
Skip-thoughts	❄️	-	81.8	62.9	-	86.6	75.8	71.8
ELMo	❄️	91.7	**91.8**	**79.6**	**86.3**	**86.1**	**76.0**	**75.9**
	🔥	**91.9**	91.2	76.4	83.3	83.3	74.7	75.5
	Δ=🔥-❄️	0.2	-0.6	-3.2	-3.3	-2.8	-1.3	-0.4
BERT-base	❄️	92.2	93.0	**84.6**	84.8	86.4	78.1	82.9
	🔥	**92.4**	**93.5**	**84.6**	**85.8**	**88.7**	**84.8**	**87.1**
	Δ=🔥-❄️	0.2	0.5	0.0	1.0	2.3	6.7	4.2

- BERT is typically better if the whole network is fine-tuned, unlike ELMo

Peters, Ruder, Smith (2019)

Evaluation: GLUE

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Wang et al. (2019)

Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

- Huge improvements over prior work (even compared to ELMo)
- Effective at “sentence pair” tasks: textual entailment (does sentence A imply sentence B), paraphrase detection

Devlin et al. (2018)

RoBERTa

- "Robustly optimized BERT"
- 160GB of data instead of 16 GB
- Dynamic masking: standard BERT uses the same MASK scheme for every epoch, RoBERTa recomputes them

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

- New training + more data = better performance

Liu et al. (2019)

v4.5.0 ▼

transformers

Star 43,515

Search docs

GET STARTED

Quick tour

Installation

Philosophy

🔥 SIGN IN 🚀 MODELS 💬 FORUM

Docs » Transformers View page source

Transformers

State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0.

🤗 Transformers (formerly known as *pytorch-transformers* and *pytorch-pretrained-bert*) provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between TensorFlow 2.0 and PyTorch.

This is the documentation of our repository transformers.

GPT/GPT2

OpenAI GPT/GPT2

- "ELMo with transformers" (works better than ELMo)
- Train a single unidirectional transformer LM on long contexts
- GPT2: trained on 40GB of text collected from upvoted links from reddit
- 1.5B parameters — by far the largest of these models trained as of March 2019

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

- Because it's a language model, we can **generate** from it

Radford et al. (2019)

OpenAI GPT2

SYSTEM PROMPT (HUMAN-WRITTEN)	*Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.*
MODEL COMPLETION (MACHINE-WRITTEN, SECOND TRY)	The 19-year-old singer was caught on camera being escorted out of the store by security guards. The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back. Scroll down for video Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured) The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals. She was carrying a pair of black and white striped gloves and a small black bag.

slide credit: OpenAI

Open Questions

1) How novel is the stuff being generated? (Is it just doing nearest neighbors on a large corpus?)

2) How do we understand and distill what is learned in this model?

3) How do we harness these priors for conditional generation tasks (summarization, generate a report of a basketball game, etc.)

4) Is this technology dangerous? (OpenAI has only released 774M param model, not 1.5B yet)

Grover

- Sample from a large language model conditioned on a domain, date, authors, and headline
- Humans rank Grover-generated propaganda as more realistic than real “fake news”
- Fine-tuned Grover can detect Grover propaganda easily — authors argue for releasing it for this reason
- NOTE: Not a GAN, discriminator trained separately from the generator

Discriminator size		Unpaired Accuracy, Generator size: 1.5B	355M	124M	Paired Accuracy, Generator size: 1.5B	355M	124M
	Chance		50.0			50.0	
1.5B	GROVER-Mega	**92.0**	**98.5**	**99.8**	**97.4**	**100.0**	**100.0**
355M	GROVER-Large	**80.8**	**91.2**	**98.4**	**89.0**	**96.9**	**100.0**
	BERT-Large	73.1	75.9	97.5	84.1	91.5	99.9
	GPT2	70.1	78.0	90.3	78.8	87.0	96.8
124M	GROVER-Base	**70.1**	**80.0**	**89.2**	77.5	88.2	95.7
	BERT-Base	67.2	76.6	84.1	**80.0**	**89.5**	**96.2**
	GPT2	66.2	71.9	83.5	72.5	79.6	89.6
11M	FastText	63.8	65.6	69.7	65.9	69.0	74.4

Zellers et al. (2019)

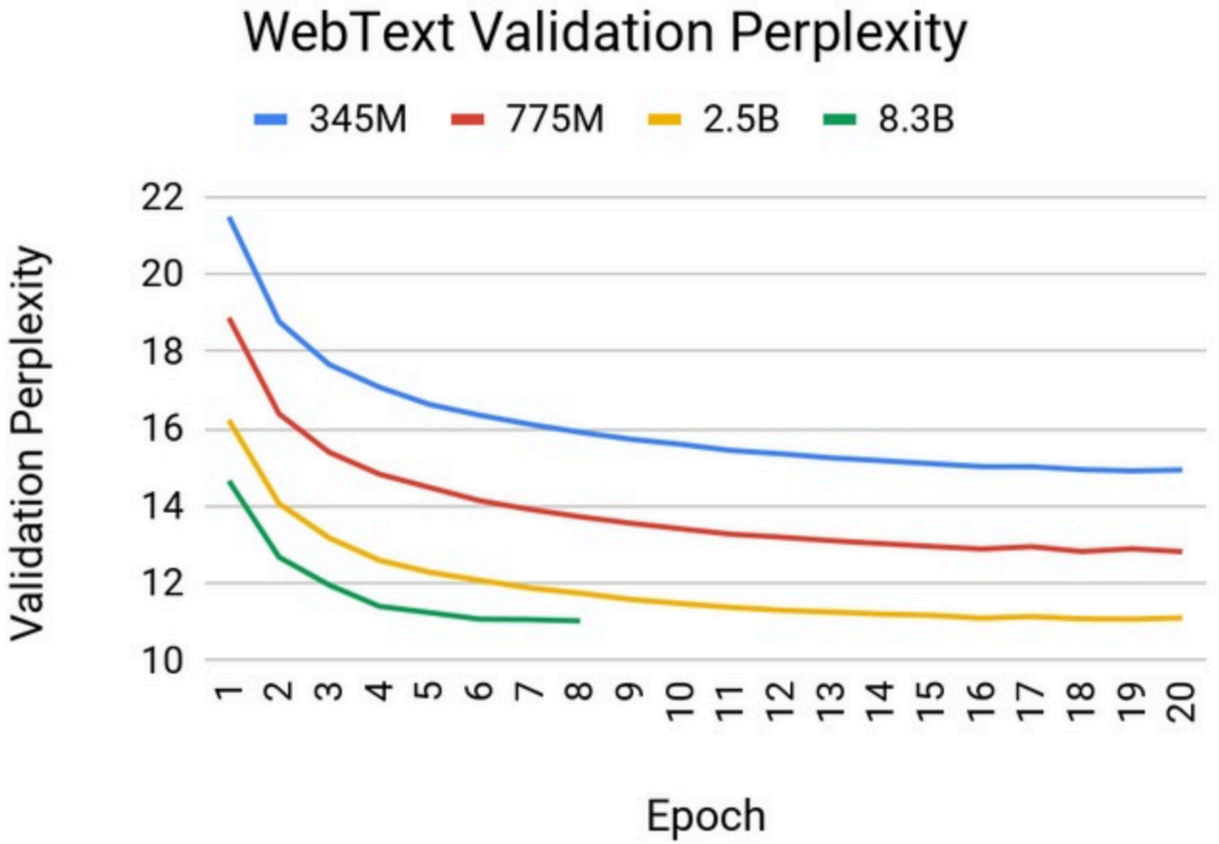
Pre-Training Cost (with Google/AWS)

- BERT: Base $500, Large $7000
- Grover-MEGA: $25,000
- XLNet (BERT variant): $30,000 — $60,000 (unclear)
- This is for a single pre-training run…developing new pre-training techniques may require many runs
- *Fine-tuning* these models can typically be done with a single GPU (but may take 1-3 days for medium-sized datasets)

https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/

Pushing the Limits

- NVIDIA: trained 8.3B parameter GPT model (5.6x the size of GPT-2)
- Arguable these models are still underfit: larger models still get better held-out perplexities



NVIDIA blog (Narasimhan, August 2019)

Google T5

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full dataset	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

- Colossal Cleaned Common Crawl: 750 GB of text
- We still haven't hit the limit of bigger data being useful

Raffel et al. (October 23, 2019)

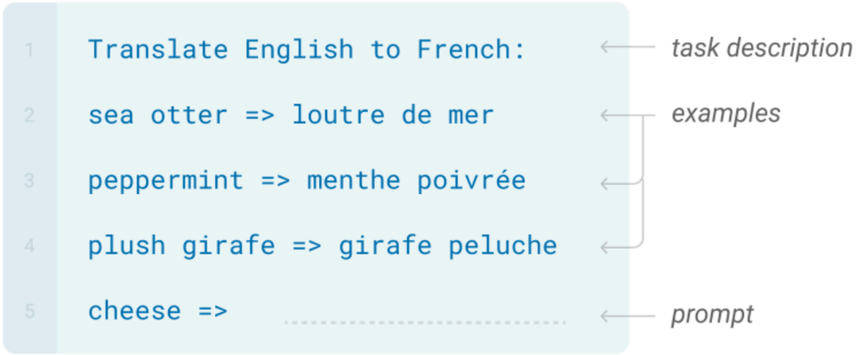
GPT-3

- GPT-2, but more parameters and trained on more data
- 175B parameters, large, filtered web-crawl
- Supports “in-context learning”
  - No gradients! (or parameter updates)
  - Competitive results on few-shot learning

Dataset	Quantity (tokens)
Common Crawl (filtered)	410 billion
WebText2	19 billion
Books1	12 billion
Books2	55 billion
Wikipedia	3 billion

Few-shot

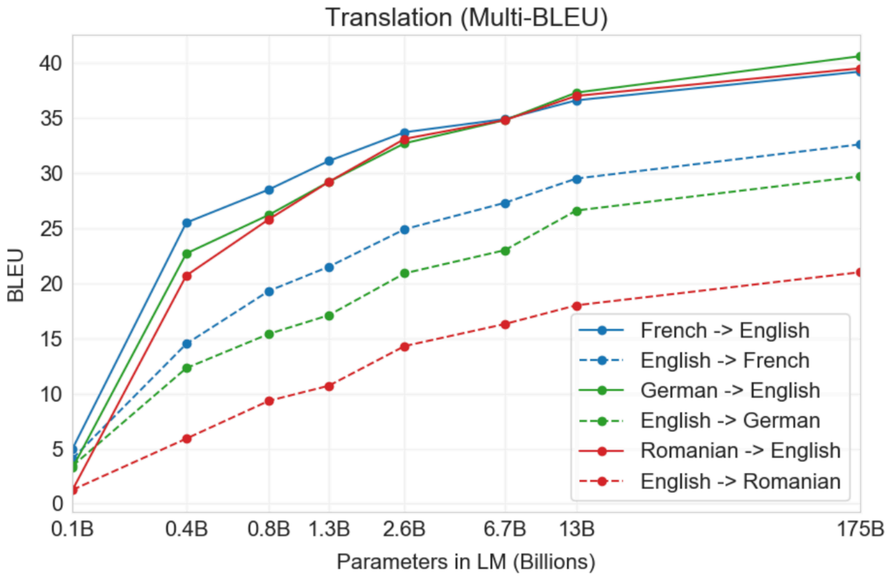
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

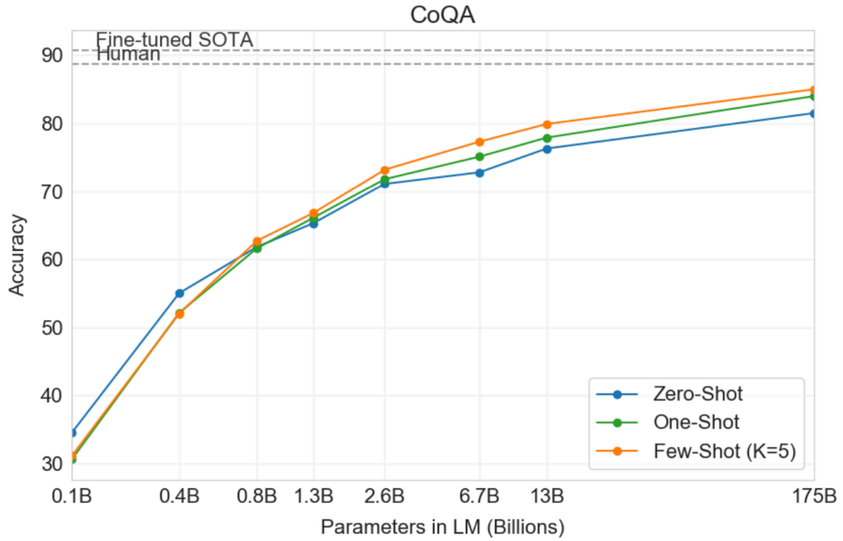


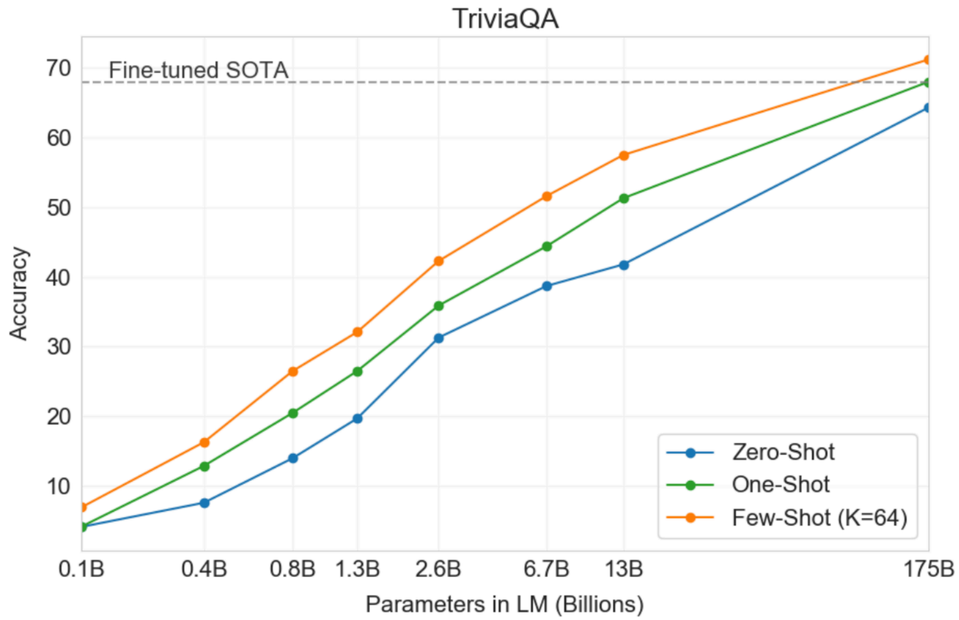
(Brown et. al. 2020)

GPT-3

- Surprisingly good performance with only a few training examples! (For some tasks, at least…)
- Amazing this can work without any gradient updates!
- But, fine-tuning on a large supervised dataset generally works better with much less inference cost… Can only be accessed via API.







Neural Nets for Text Processing (Summary)

- How to encode text as input to a neural network?
  - Word embeddings
  - CBOW / Deep Averaging Networks
- Contextualized embeddings / Pre-training
  - ELmO, BERT
- More pre-training / larger models
  - T5 - 11B, GPT-3 - 175B parameters