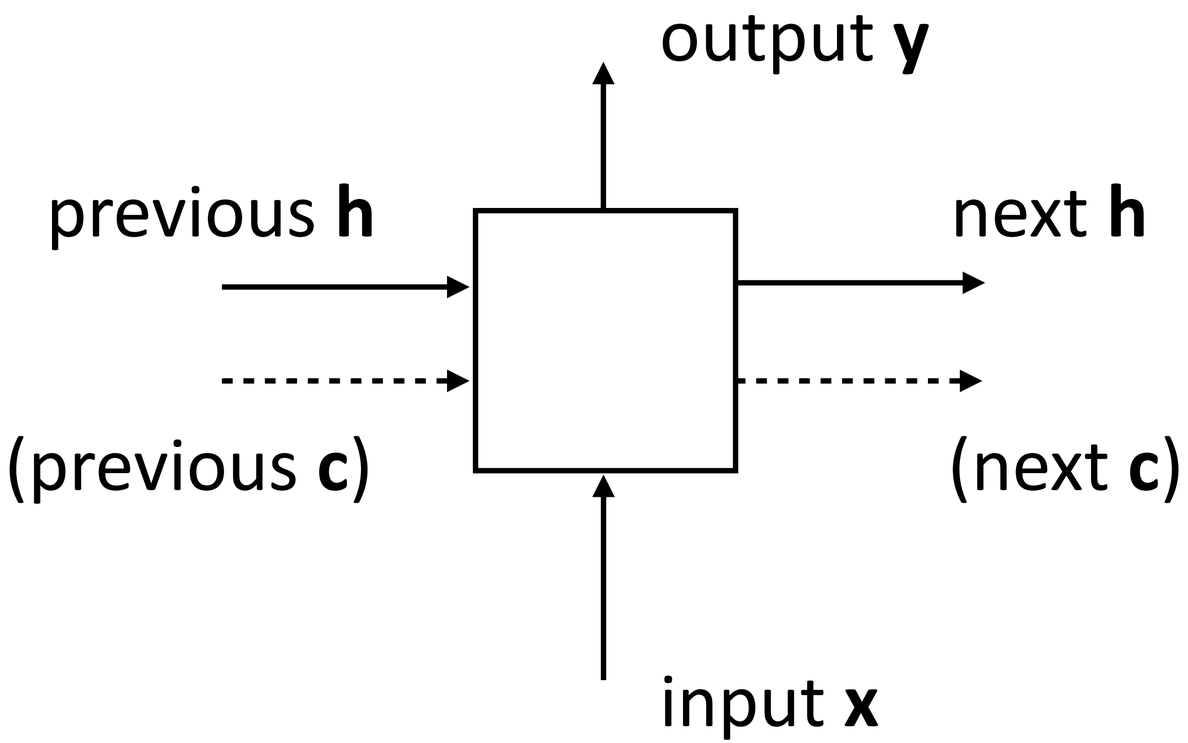
ANLP

13 - Pretraining (NNs, part III)

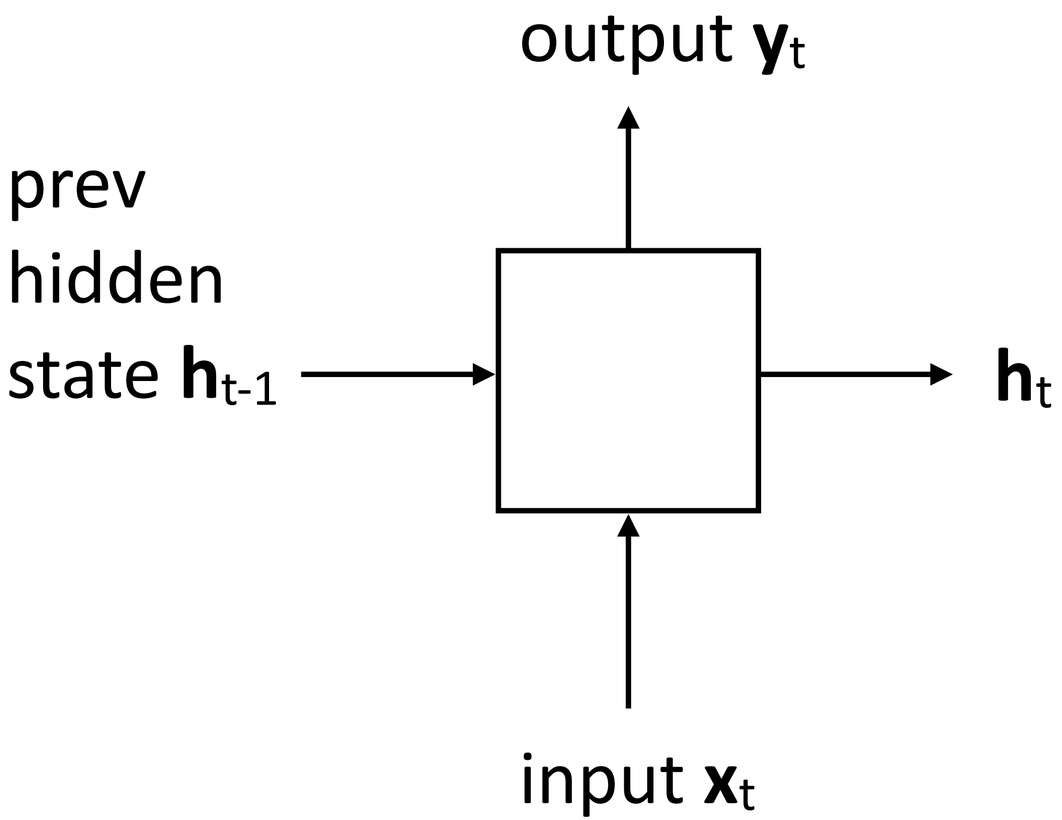
David Schlangen
University of Potsdam, MSc Cognitive Systems
Winter 2019 / 2020

Recall: RNNs

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



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

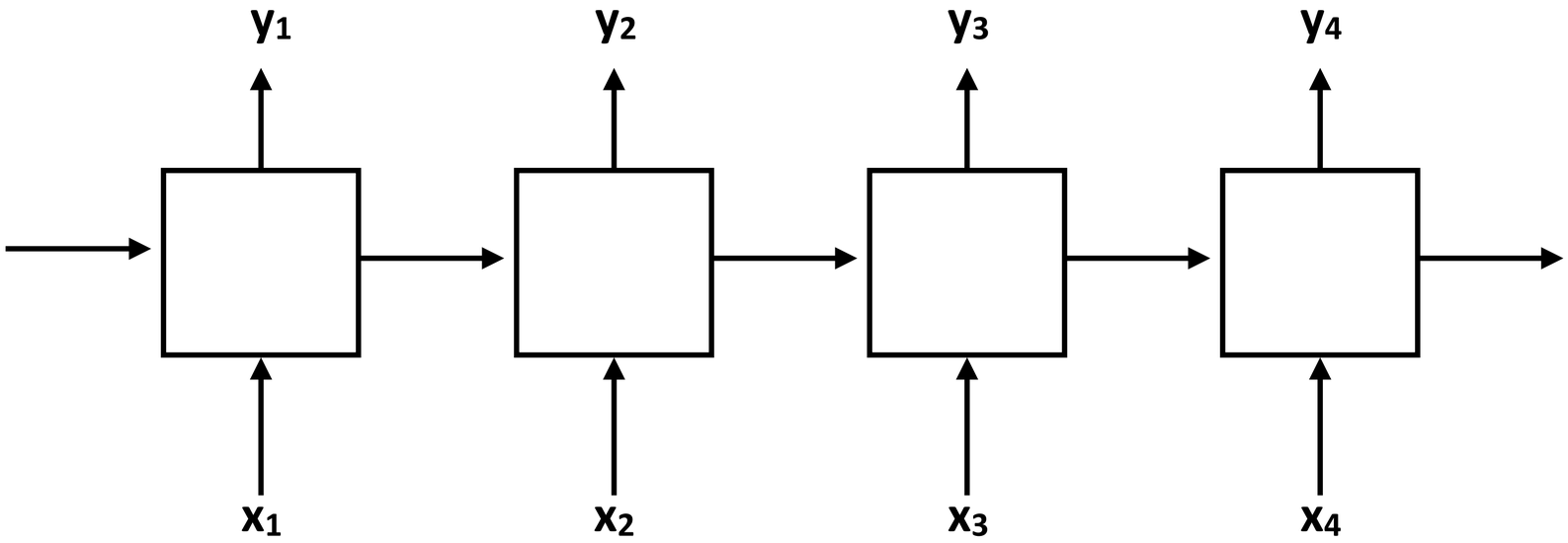
$$\mathbf{y}_t = \tanh(U\mathbf{h_t} + \mathbf{b}_y)$$

- Computes output from hidden state

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

Elman (1990)

unrolled



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

$$\frac{\partial E_t}{\partial W_R} = \sum_{i=0}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_i} \frac{\partial h_i}{\partial W_R}$$

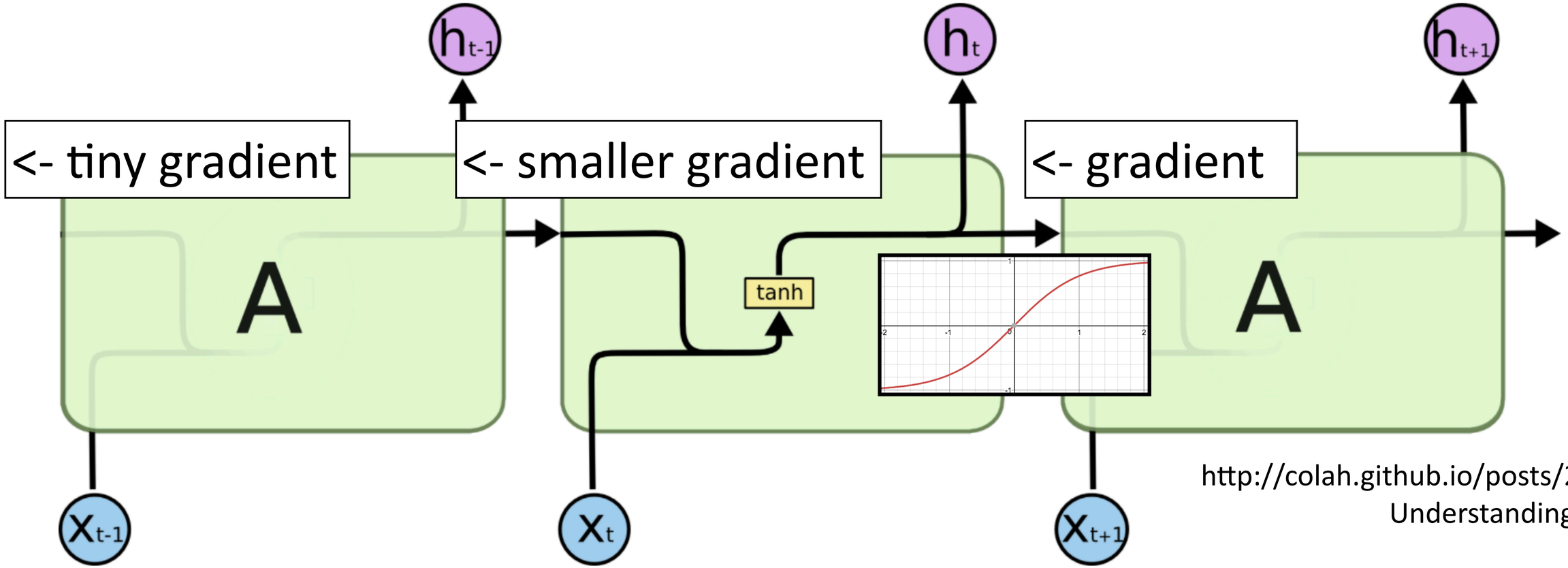
$$\frac{\partial h_t}{\partial h_i} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{i+1}}{\partial h_i} = \prod_{k=i}^{t-1} \frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial h_k}{\partial h_1} = \prod_{i}^{k} diag(f'(W_I x_i + W_R h_{i-1}))W_R$$

the gory details: [1], and original papers

[1] https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html

Vanishing Gradient



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

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

LSTMs/GRUs

Gated Connections

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

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

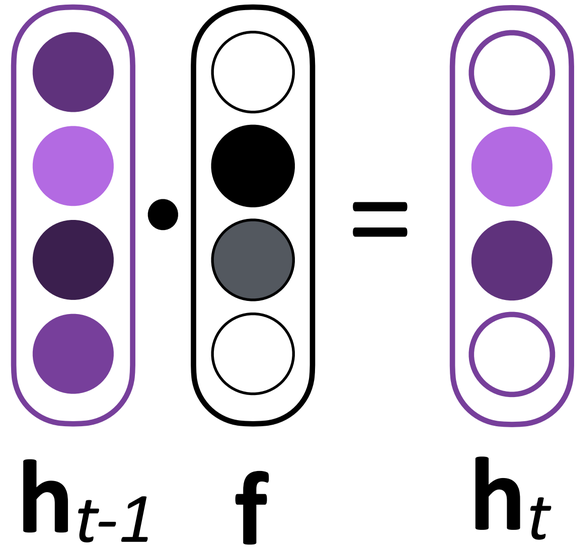
gated

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

Elman

- Vector-valued "forget gate" **f** computed based on input and previous hidden state

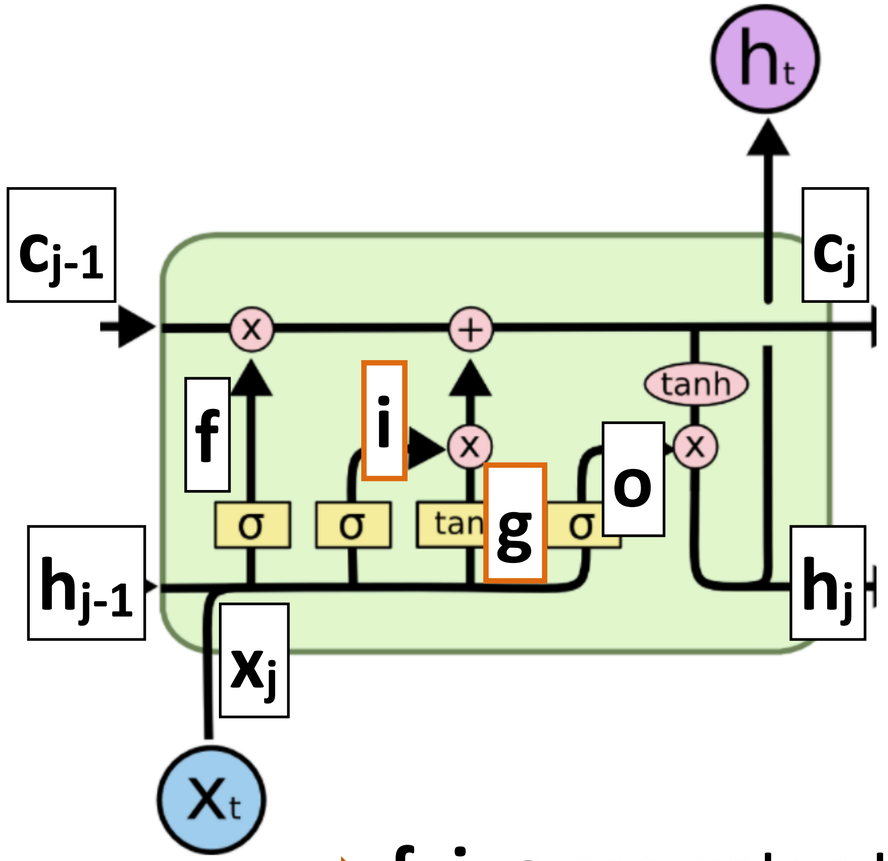
$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$



- Sigmoid: elements of **f** are in (0, 1)

- If **f** ≈ **1**, we simply sum up a function of all inputs — gradient doesn't vanish! More stable without matrix multiply (*V*) as well

LSTMs



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

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

$$\mathbf{g} = \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}})$$

$$\mathbf{i} = \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hi}})$$

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

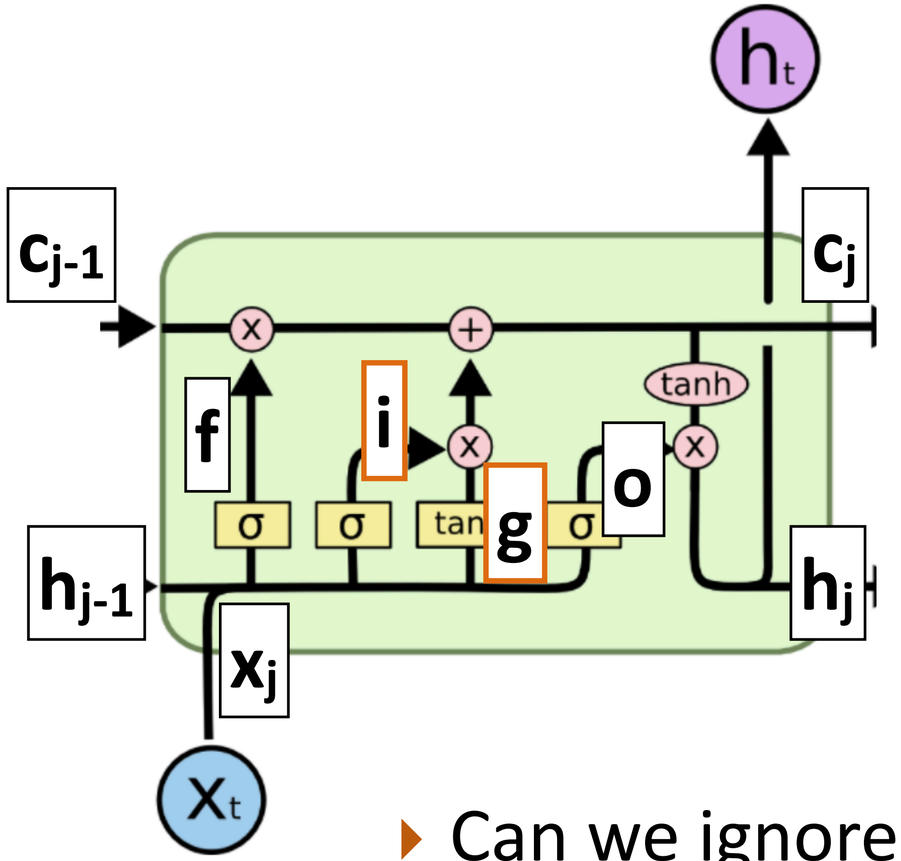
$$\mathbf{o} = \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}})$$

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

Goldberg lecture notes

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

LSTMs



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

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

$$\mathbf{g} = \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}})$$

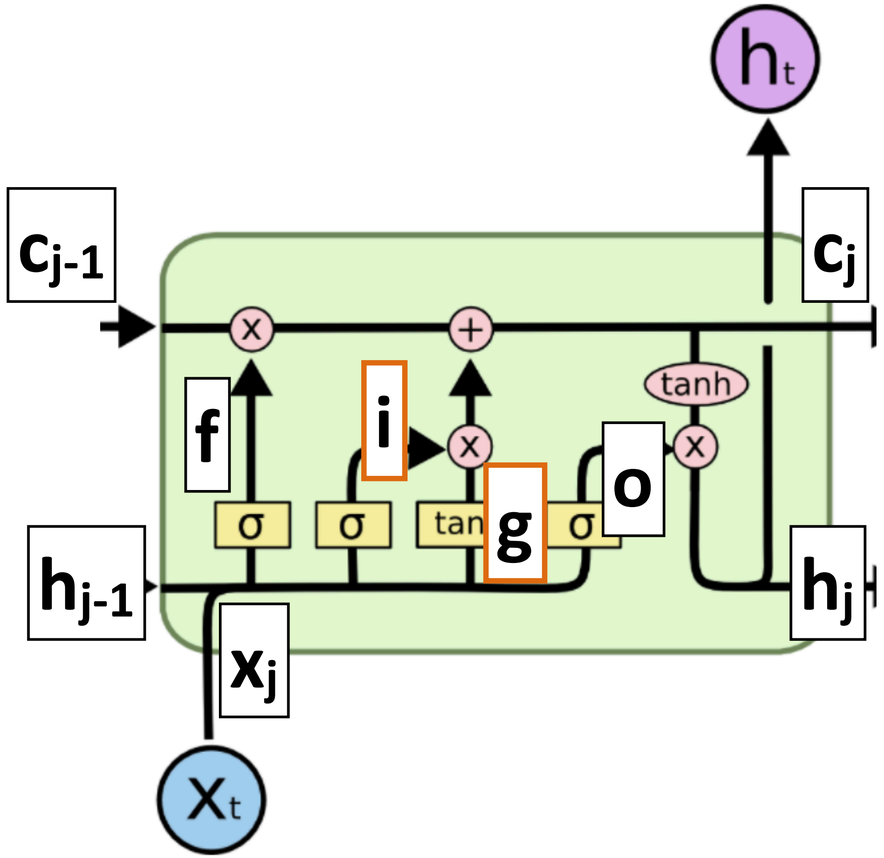
$$\mathbf{i} = \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}})$$

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

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

- Can we ignore the old value of **c** for this timestep?
- Can an LSTM sum up its inputs **x**?
- Can we ignore a particular input **x**?
- Can we output something without changing **c**?

LSTMs

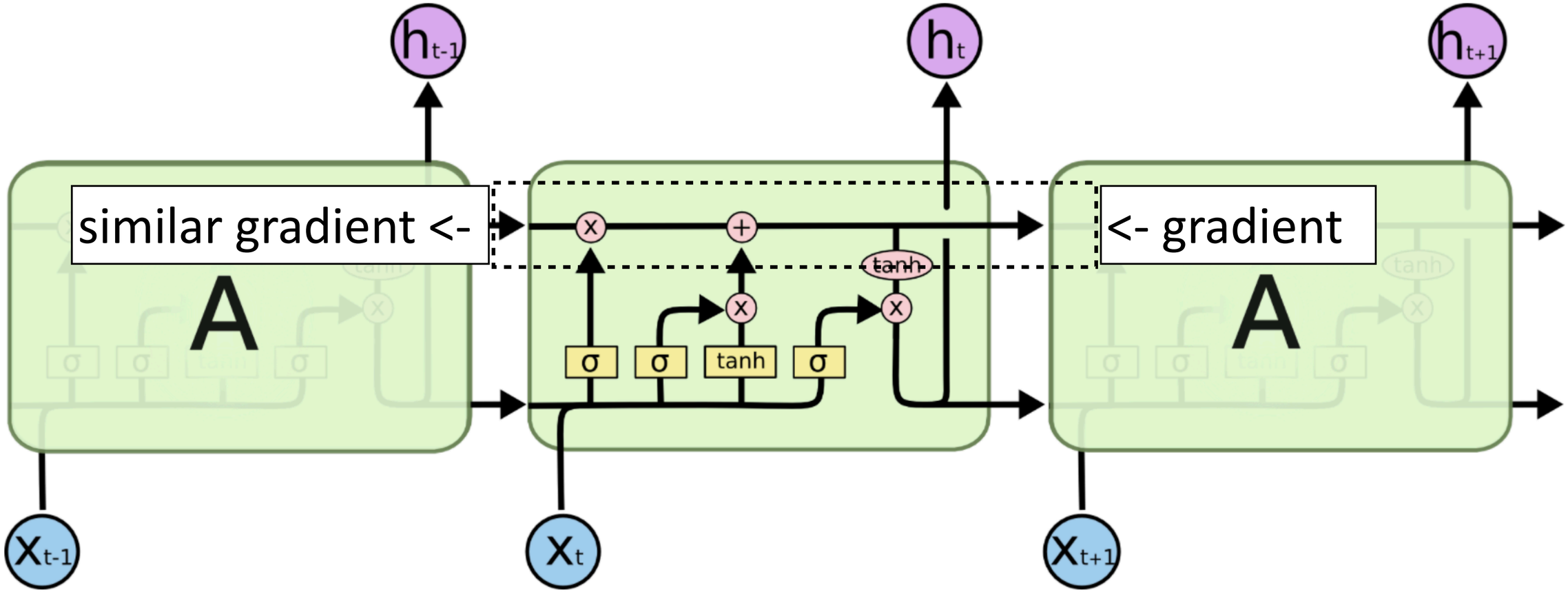


- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Goldberg lecture notes

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

LSTMs

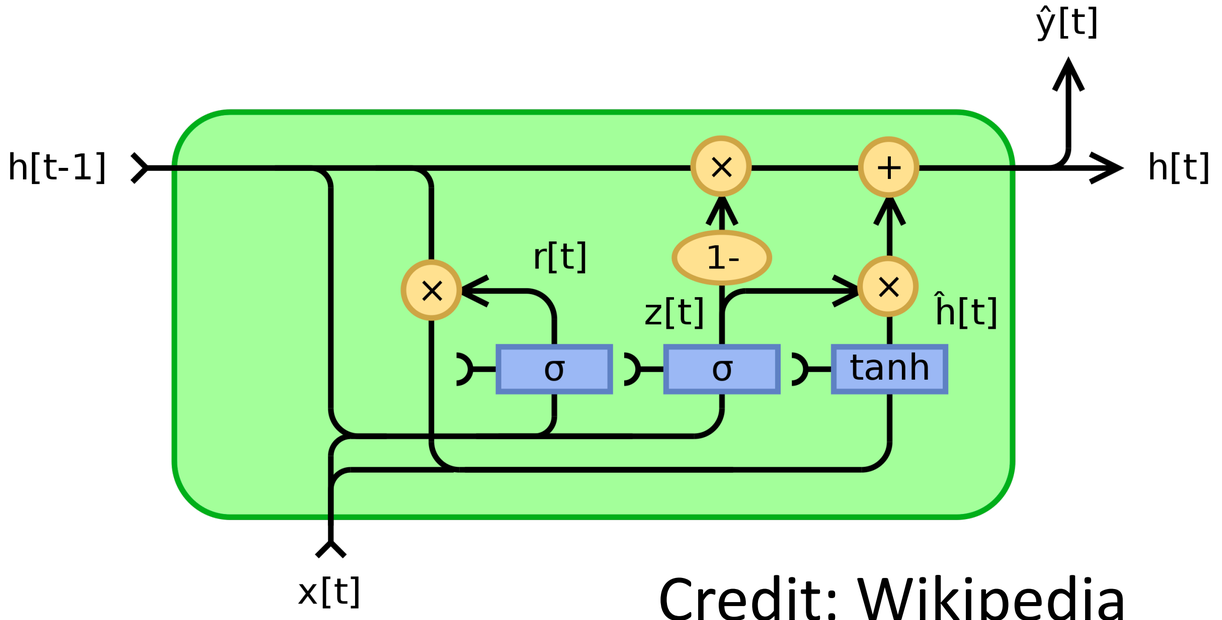


- Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

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

Gated Recurrent Unit (GRU)

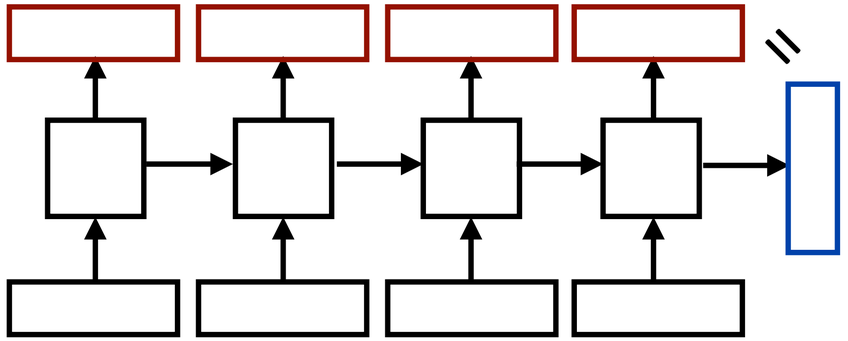
- **z** is update, **r** is reset
- The single hidden state and simpler update gate gives simpler mixing semantics than in LSTMs
- Faster to train and sometimes works better than LSTMs, often a tossup



Credit: Wikipedia

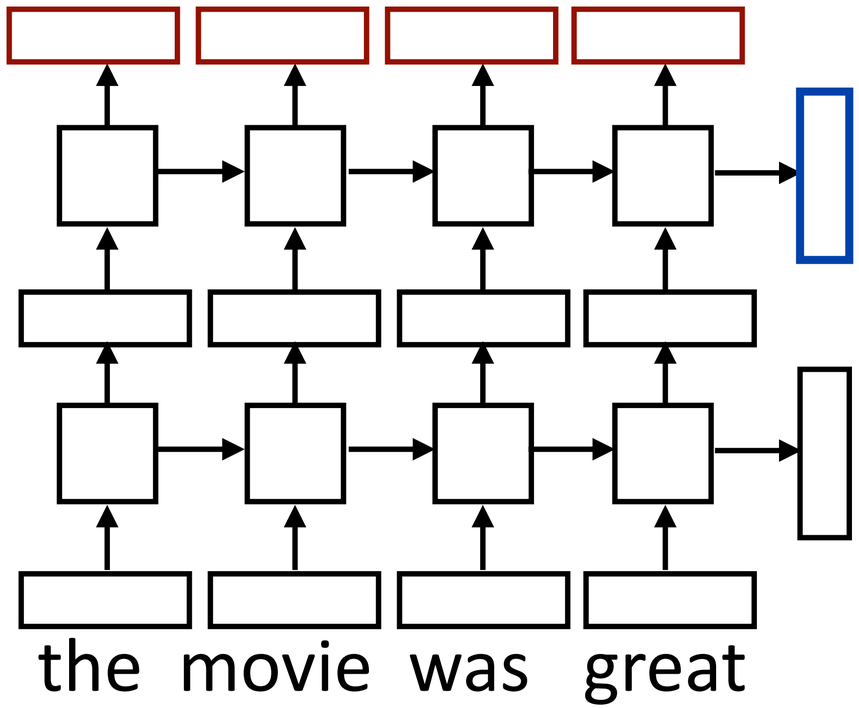
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

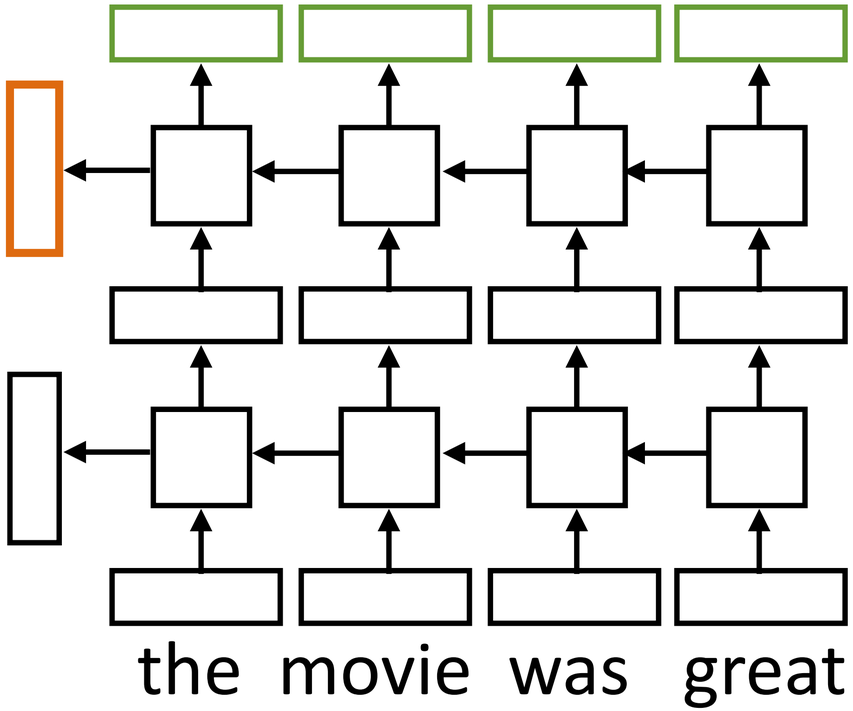
What do RNNs produce?



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

Multilayer Bidirectional RNN

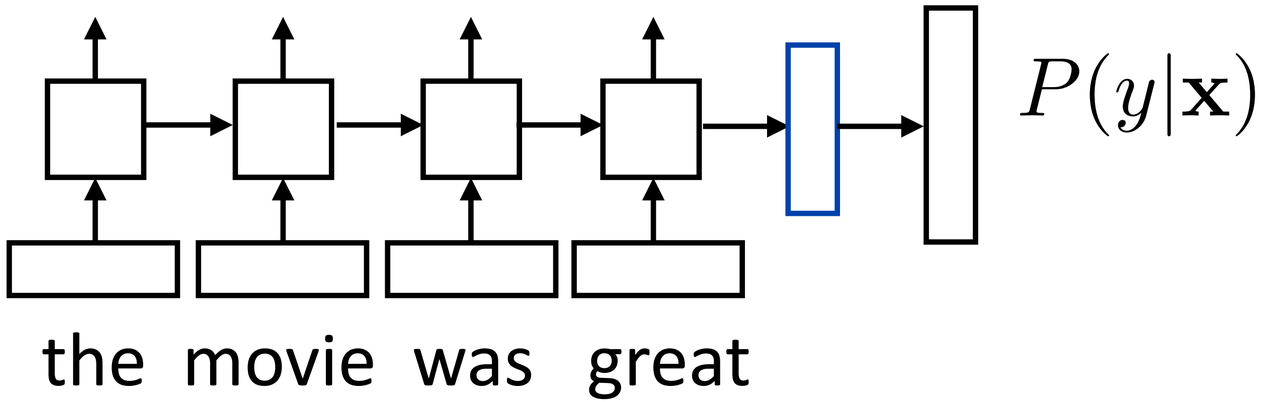




- Sentence classification based on concatenation of both final outputs

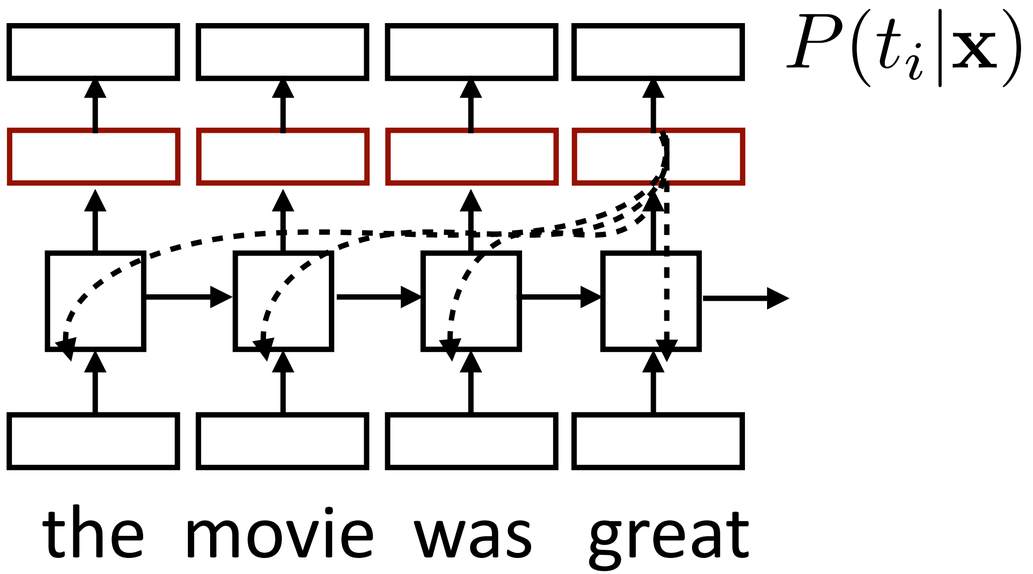
- Token classification based on concatenation of both directions' token representations

Training RNNs



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

Training RNNs



- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network
- Example: language modeling (predict next word given context) or POS tagging

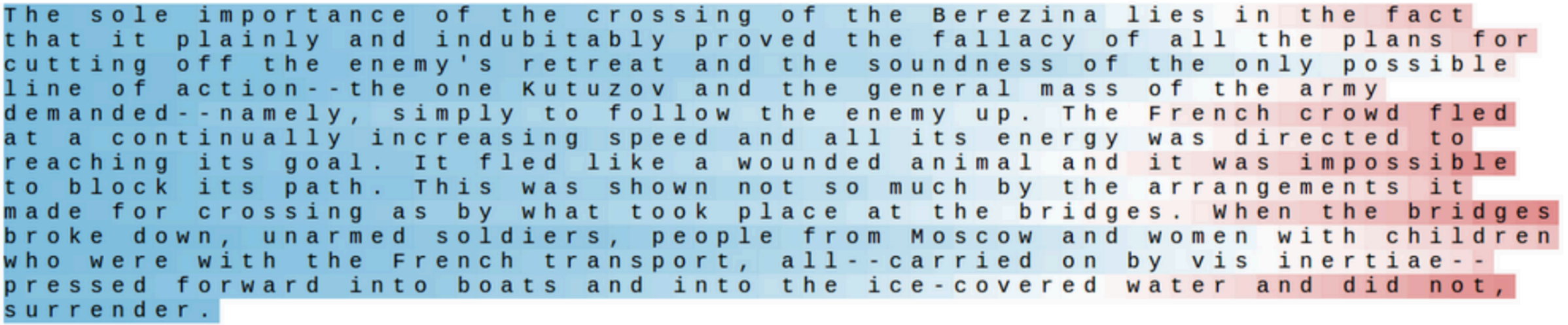
Applications

What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)

Visualizing LSTMs

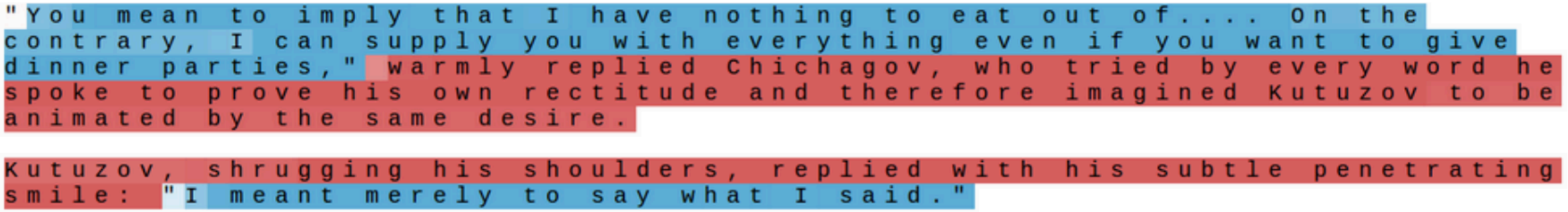
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of **c**) to understand them
- Counter: know when to generate \n



Karpathy et al. (2015)

Visualizing LSTMs

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



Karpathy et al. (2015)

Visualizing LSTMs

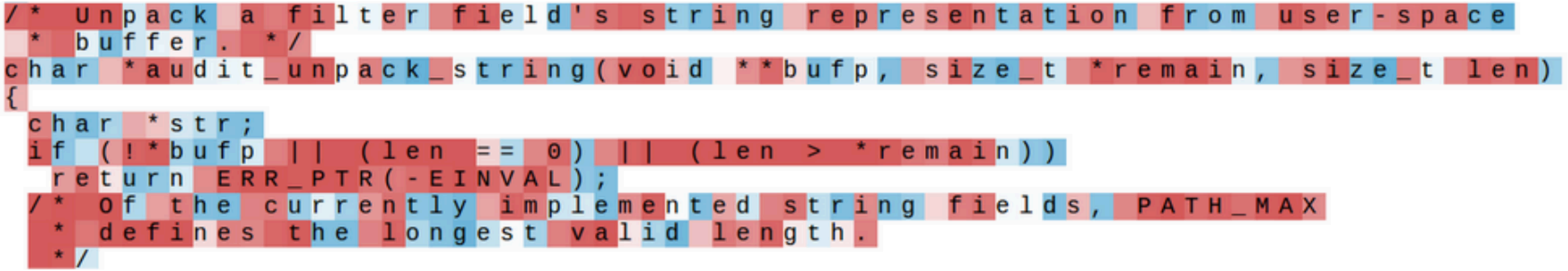
- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Karpathy et al. (2015)

Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Uninterpretable: probably doing double-duty, or only makes sense in the context of another activation



Karpathy et al. (2015)

What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)
- Textual entailment
  - Encode two sentences, predict

Sentiment Analysis

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

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

better than tuned Naive Bayes when using the SA trick

Dai and Le (2015)

Natural Language Inference

Premise		Hypothesis
A boy plays in the snow	*entails*	A boy is outside
A man inspects the uniform of a figure	*contradicts*	The man is sleeping
An older and younger man smiling	*neutral*	Two men are smiling and laughing at cats playing

- Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)
- Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)

SNLI Dataset

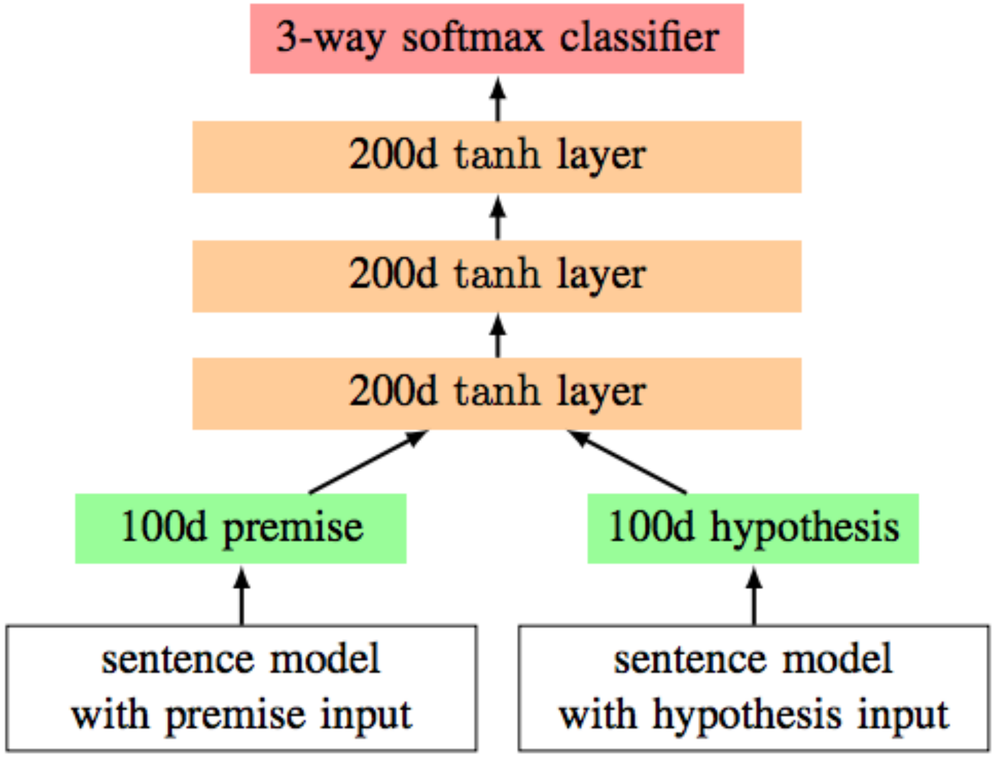
- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process

100D LSTM: 78% accuracy

300D LSTM: 80% accuracy
(Bowman et al., 2016)

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

- Later: better models for this



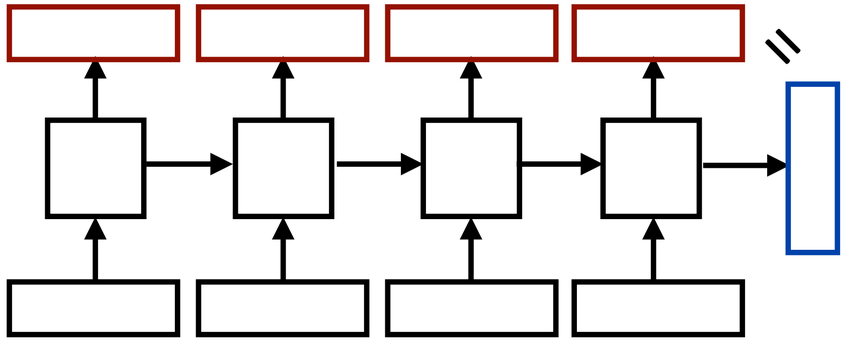
Bowman et al. (2015)

Takeaways

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector

- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation

Recall: RNN Abstraction



- Encoding of the sentence — can pass this to 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

This Lecture

- Language modeling
  - N-gram models (recap)
  - Neural LMs
- LM-based pretraining: ELMo

Language Modeling

Challenges in Text Generation

- Dialogue, machine translation, summarization, etc.
  - What to say (content selection + content planning) and how to say it
- Template-based generation systems always generate fluent output
- For learned systems, how do we make sure language is plausible?
- Language models: place a distribution P(***w***) over strings ***w*** in a language
  - Next weeks: P(T,***w***) modeled by probabilistic context-free grammars
  - Today: *autoregressive* models $P(\mathbf{w}) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots$

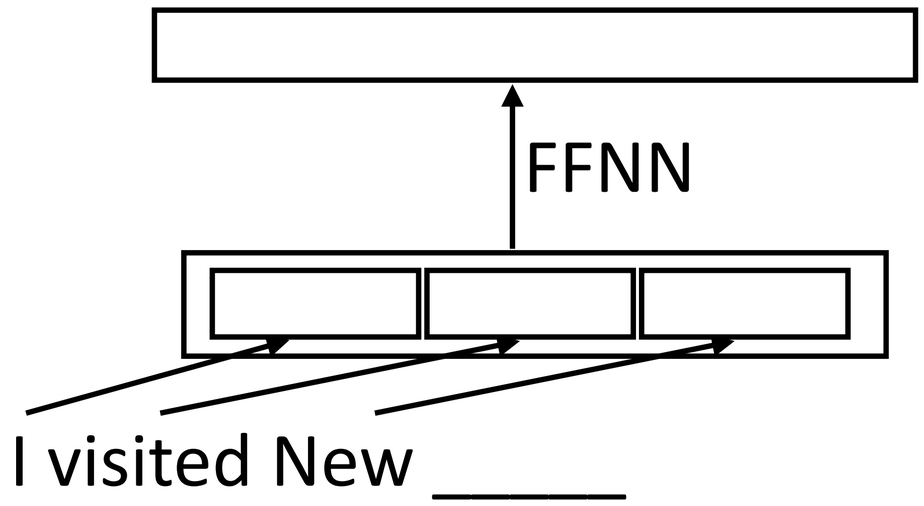
n-gram models

$$P(\mathbf{w}) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots$$

- n-gram models: distribution of next word is a multinomial conditioned on previous n-1 words $P(w_i|w_1, \ldots, w_{i-1}) = P(w_i|w_{i-n+1}, \ldots, w_{i-1})$
- Just relies on counts, even in 2008 could scale up to 1.3M word types, 4B n-grams (all 5-grams occurring >40 times on the Web)
- but in principle limited to fixed n

Neural Language Models

- Early work: feedforward neural networks looking at context



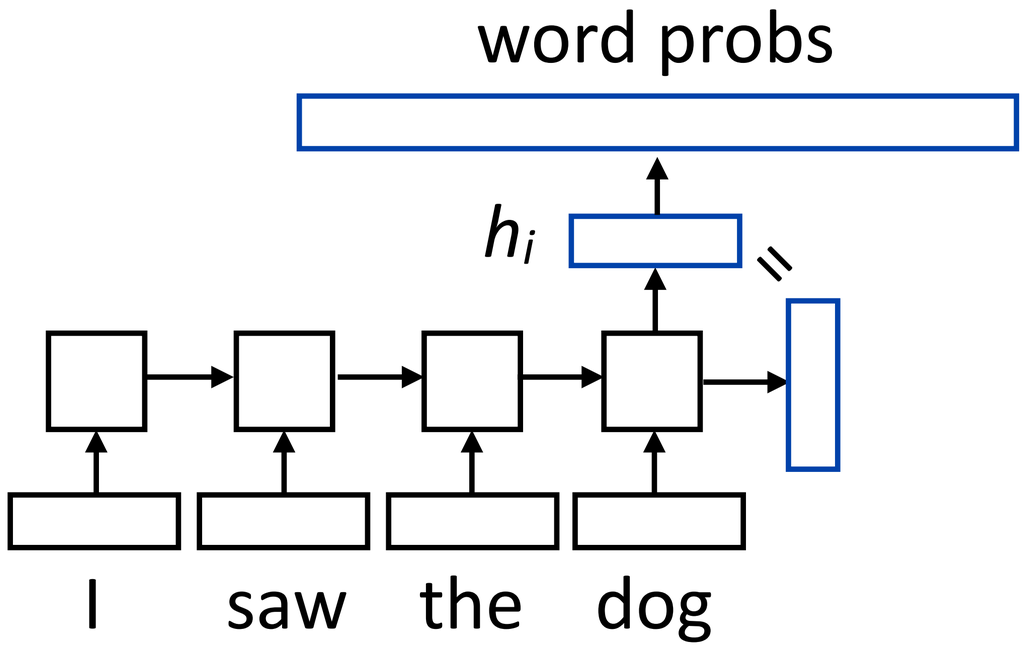
$$P(w_i|w_{i-n},\ldots,w_{i-1})$$

- Slow to train over lots of data!

- Still only look at a fixed window of information…can we use more?

Mnih and Hinton (2003)

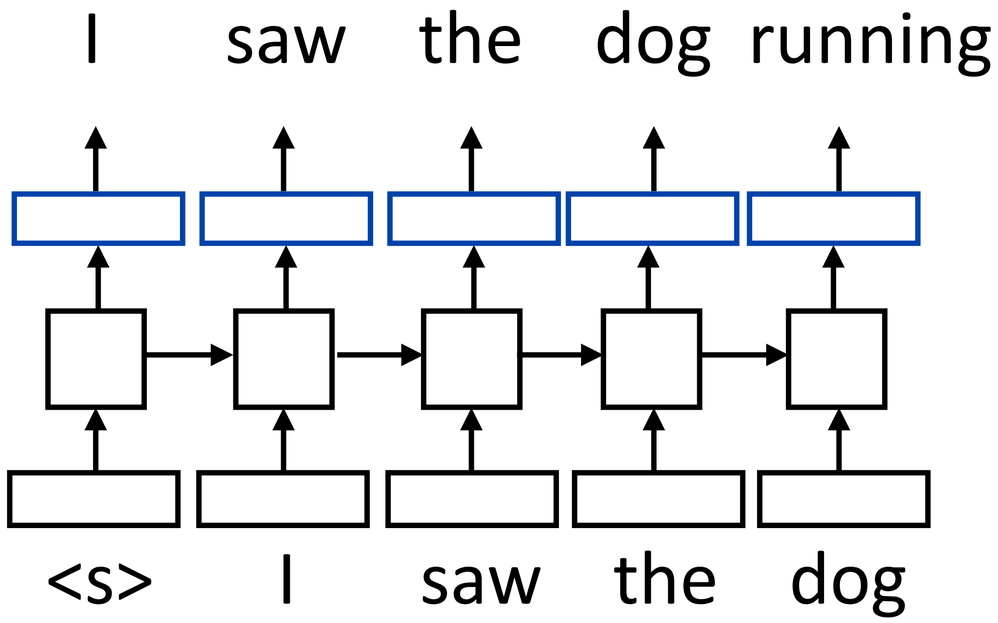
RNN Language Modeling



$$P(w|\text{context}) = \text{softmax}(W\mathbf{h}_i)$$

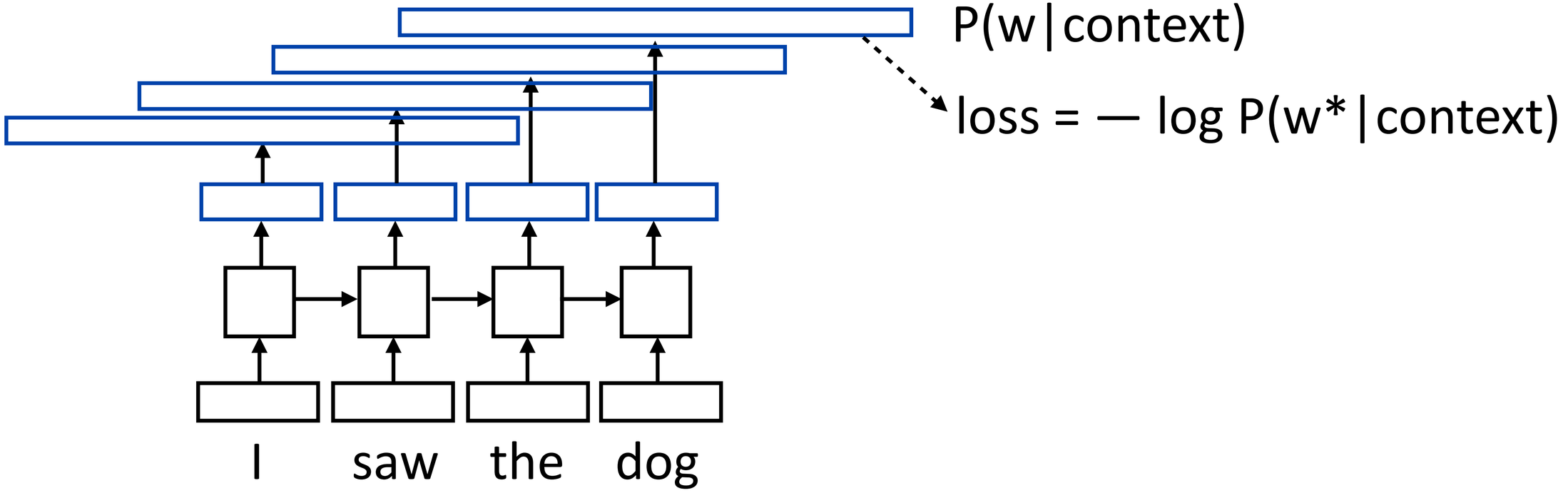
- W is a (vocab size) x (hidden size) matrix

Training RNNLMs



- Input is a sequence of words, output is those words shifted by one,
- Allows us to efficiently batch up training across time (one run of the RNN)

Training RNNLMs



- Total loss = sum of negative log likelihoods at each position
- Backpropagate through the network to simultaneously learn to predict next word given previous words at all positions

LM Evaluation

- Accuracy doesn't make sense — predicting the next word is generally impossible so accuracy values would be very low
- Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

$$\frac{1}{n}\sum_{i=1}^{n}\log P(w_i|w_1,\ldots,w_{i-1})$$

- Perplexity: exp(average negative log likelihood). Lower is better

Results

- Evaluate on Penn Treebank: small dataset (1M words) compared to what's used in MT, but common benchmark
- Kneser-Ney 5-gram model with cache: PPL = 125.7
- LSTM: PPL ~ 60-80 (depending on how much you optimize it)
- Melis et al.: many neural LM improvements from 2014-2017 are subsumed by just using the right regularization (right dropout settings). So LSTMs are pretty good
  - Main tricks: changing some tricky dropout settings + how params are structured (sizes, etc.)

Merity et al. (2017), Melis et al. (2017)

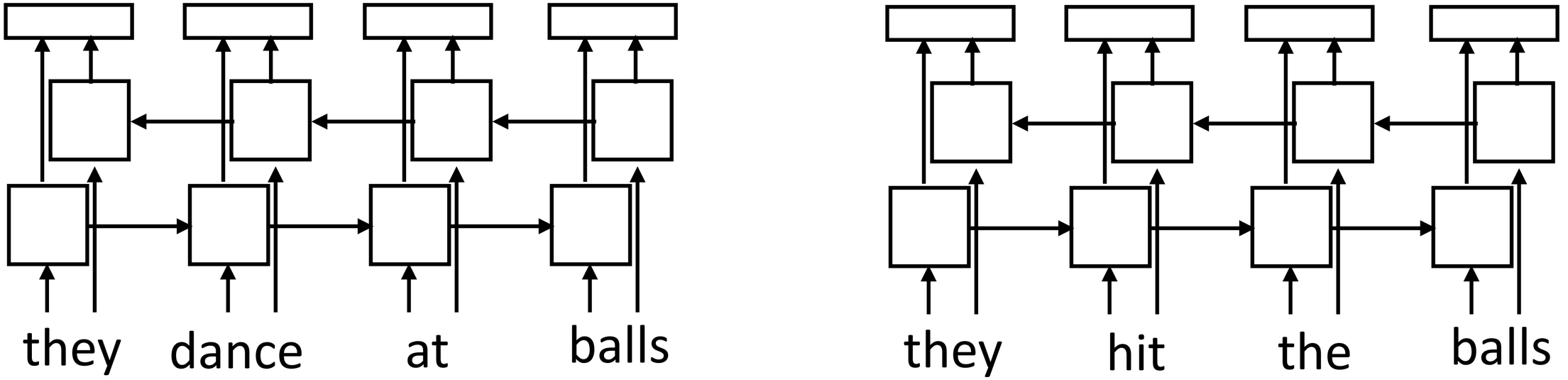
Applications of Language Modeling

- All generation tasks: translation, dialogue, text simplification, paraphrasing, etc.
- Grammatical error correction
- Predictive text
- Pretraining!

Pretraining / ELMo

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)

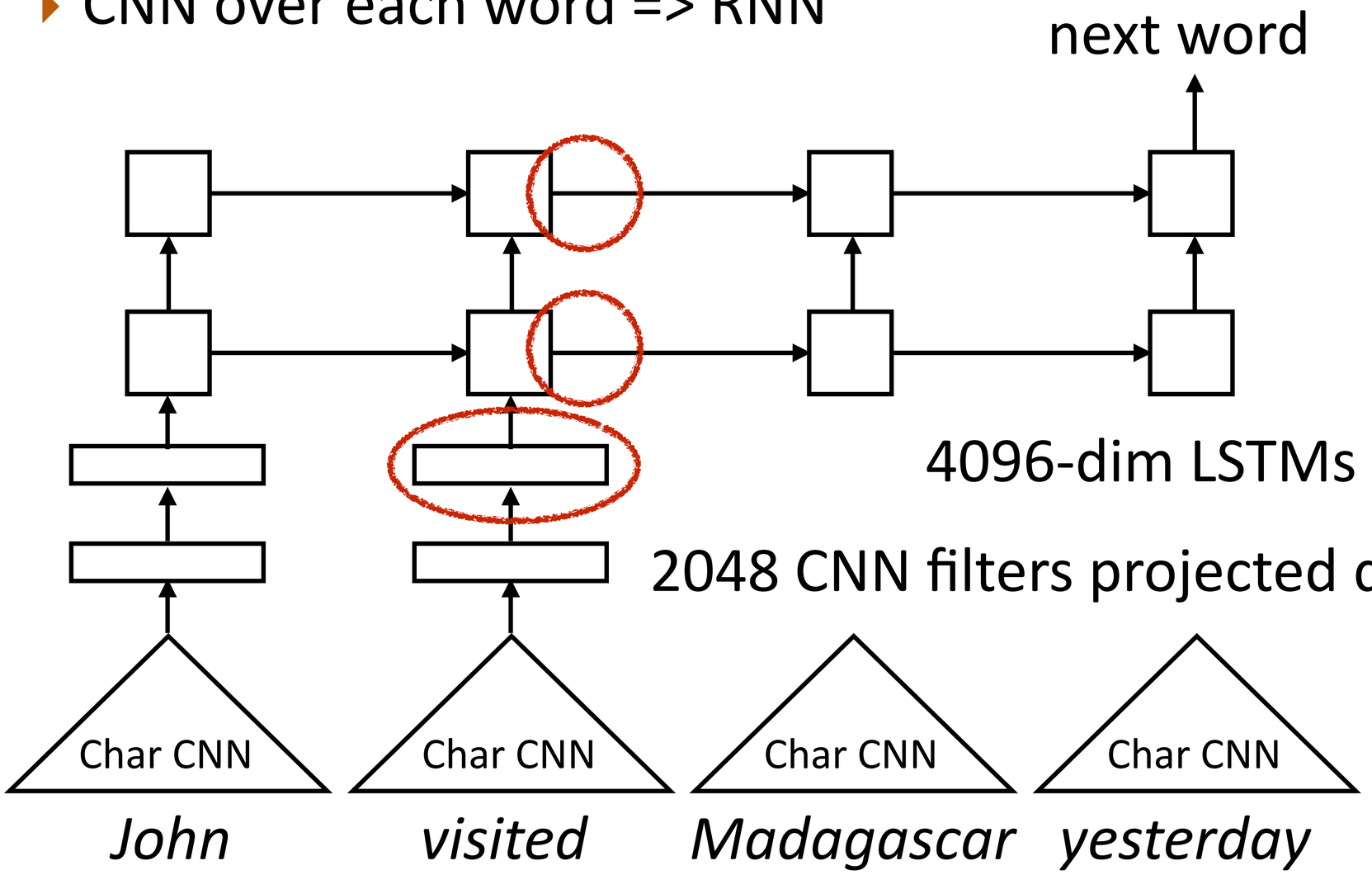
ELMo

- Key idea: language models can allow us to form useful word representations in the same way word2vec did
- Take a powerful language model, train it on large amounts of data, then use those representations in downstream tasks
  - Data: Wikipedia and Toronto Books Corpus
- What do we want our LM to look like?

Peters et al. (2018)

ELMo

- CNN over each word => RNN



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

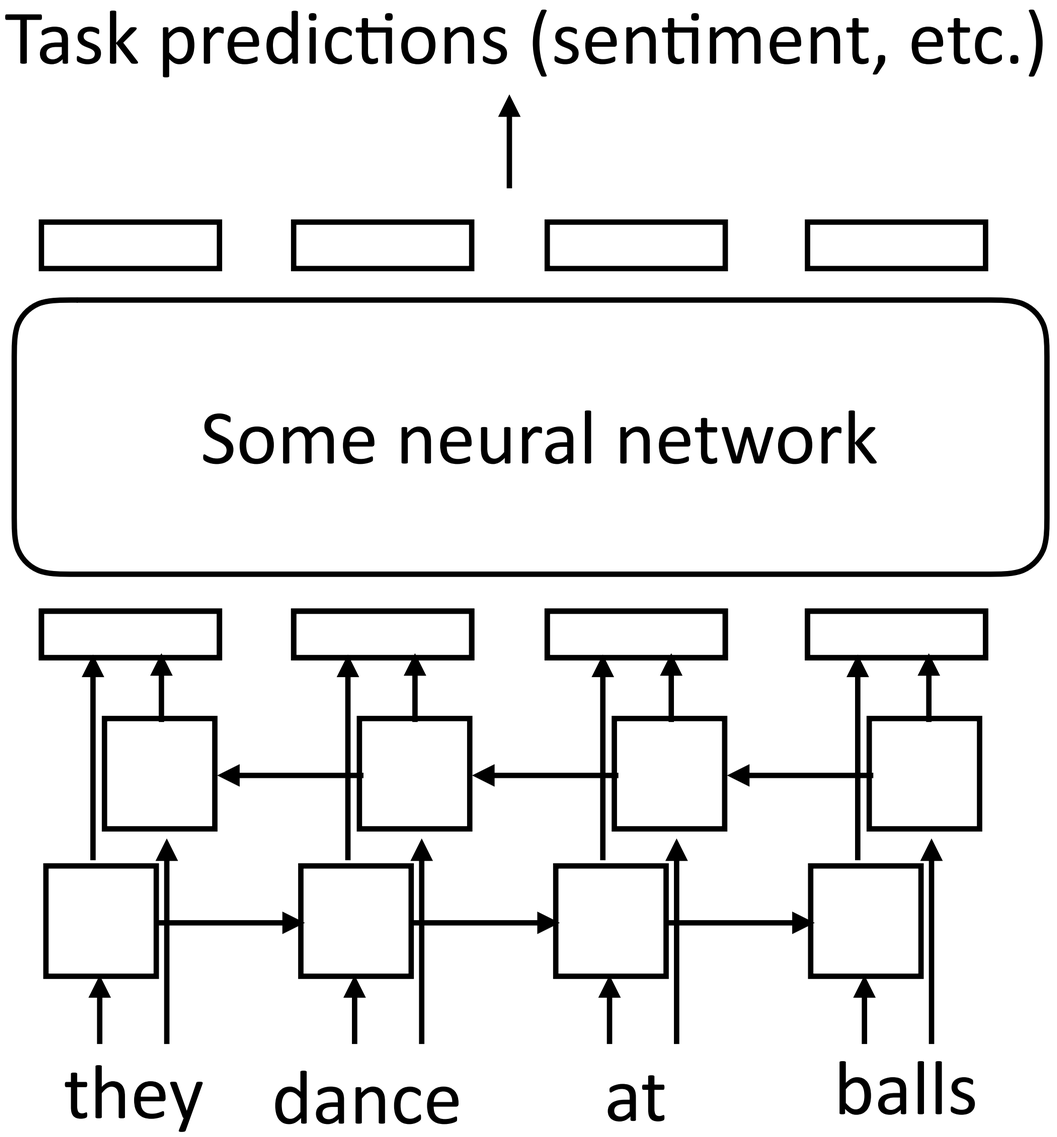
4096-dim LSTMs w/ 512-dim projections

2048 CNN filters projected down to 512-dim

Peters et al. (2018)

How to apply ELMo?

- Take those embeddings and feed them into whatever architecture you want to use for your task
- *Frozen* embeddings: update the weights of your network but keep ELMo's parameters frozen
- *Fine-tuning*: backpropagate all the way into ELMo when training your model



Peters, Ruder, Smith (2019)

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, coreference resolution, named entity recognition, and sentiment analysis

Peters et al. (2018)

How to apply ELMo?

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

- How does frozen (❄️) vs. fine-tuned (🔥) compare?
- Recommendations:

Conditions			Guidelines
Pretrain	Adapt.	Task	
Any	❄️	Any	Add many task parameters
Any	🔥	Any	Add minimal task parameters ⚠️ Hyper-parameters
Any	Any	Seq. / clas.	❄️ and 🔥 have similar performance
ELMo	Any	Sent. pair	use ❄️
BERT	Any	Sent. pair	use 🔥

Peters, Ruder, Smith (2019)

Why is language modeling a good objective?

- "Impossible" problem but bigger models seem to do better and better at distributional modeling (no upper limit yet)
- Successfully predicting next words requires modeling lots of different effects in text

Context: My wife refused to allow me to come to Hong Kong when the plague was at its height and –" "Your wife, Johanne? You are married at last ?" Johanne grinned. "Well, when a man gets to my age, he starts to need a few home comforts.
Target sentence: After my dear mother passed away ten years ago now, I became _____.
Target word: lonely

- LAMBADA dataset (Papernot et al., 2016): explicitly targets world knowledge and very challenging LM examples
- Coreference, Winograd schema, and much more

Why did this take time to catch on?

- Earlier version of ELMo by the same authors in 2017, but it was only evaluated on tagging tasks, gains were 1% or less
- Required: training on lots of data, having the right architecture, significant hyperparameter tuning

Probing ELMo

- From each layer of the ELMo model, attempt to predict something: POS tags, word senses, etc.
- Higher accuracy => ELMo is capturing that thing more strongly

Model	F_1
WordNet 1st Sense Baseline	65.9
Raganato et al. (2017a)	69.9
Iacobacci et al. (2016)	**70.1**
CoVe, First Layer	59.4
CoVe, Second Layer	*64.7*
biLM, First layer	67.4
biLM, Second layer	*69.0*

Table 5: All-words fine grained WSD F_1. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

Model	Acc.
Collobert et al. (2011)	97.3
Ma and Hovy (2016)	97.6
Ling et al. (2015)	**97.8**
CoVe, First Layer	*93.3*
CoVe, Second Layer	92.8
biLM, First Layer	*97.3*
biLM, Second Layer	96.8

Table 6: Test set POS tagging accuracies for PTB. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

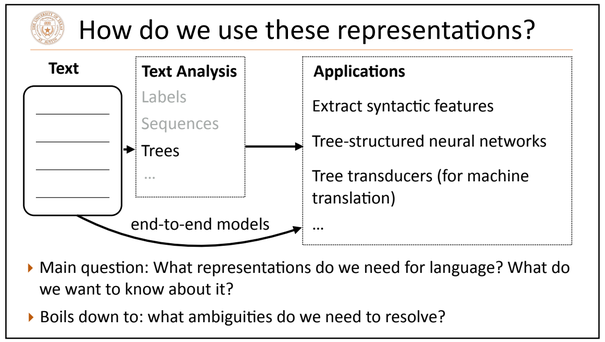
Peters et al. (2018)

Takeaways

- Language modeling involves predicting the next word given context. Several techniques to do this, more later in the course
- Learning a neural network to do this induces useful representations for other tasks, similar to word2vec/GloVe

slide credits

slides that look like this



come from

CS388 given by Greg Durrett at U Texas, Austin

and their use is gratefully acknowledged. I try to make any modifications obvious, but if there are errors on a slide, assume that I added them.