CS388: Natural Language Processing

Lecture 9: Language Modeling + Pretraining

Greg Durrett

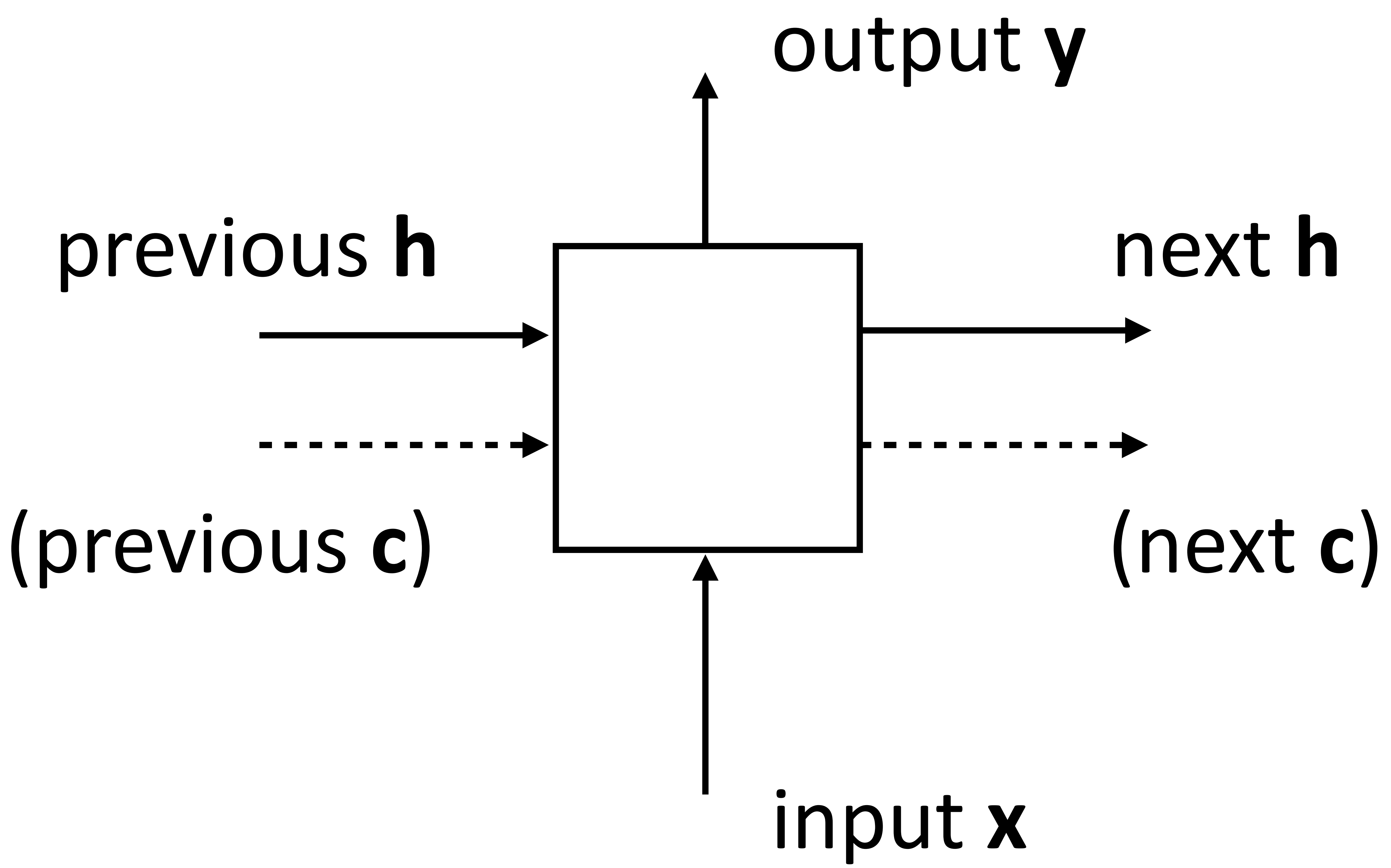
TEXAS
The University of Texas at Austin

Administrivia

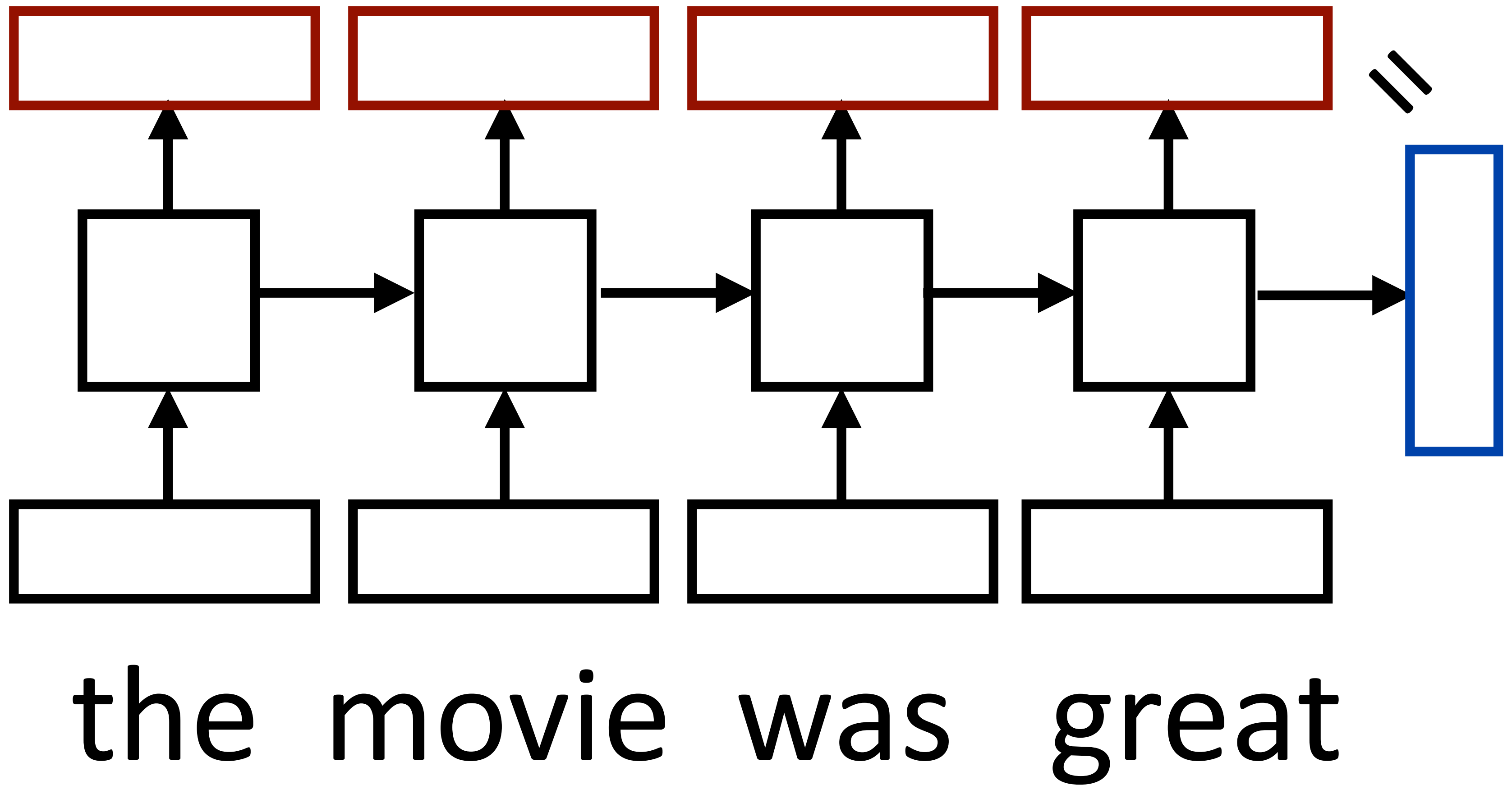
- Mini 2 due March 2
- +3 slip days for everyone
- Rest of the course pushed back
- Final project spec out now

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)

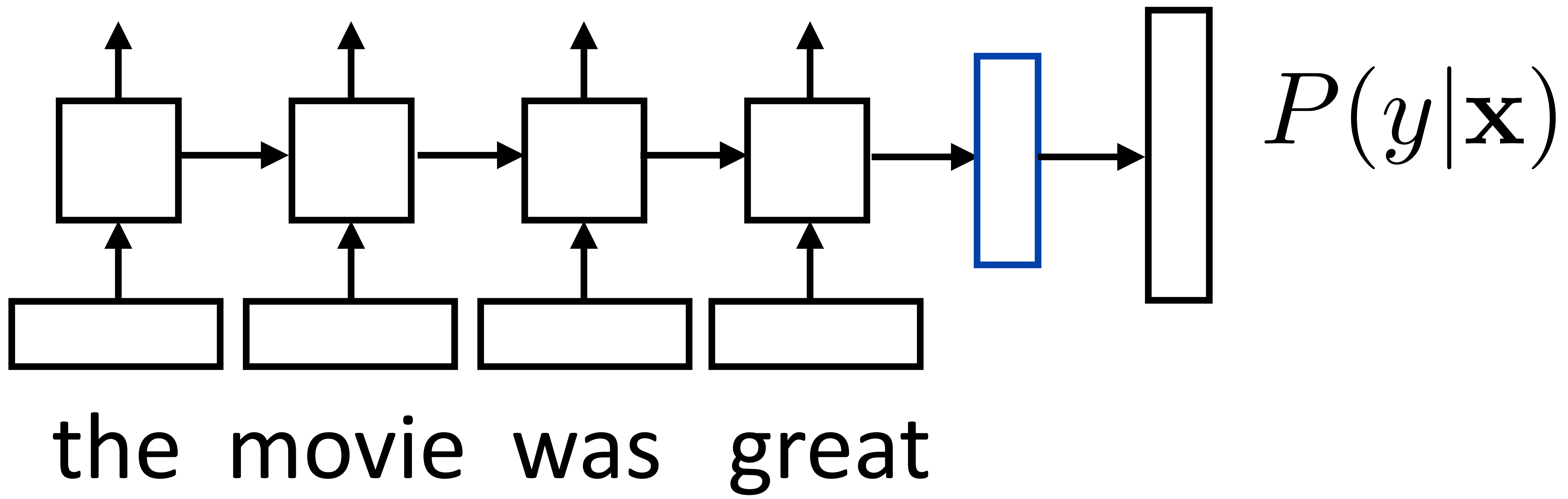


Recall: RNN Abstraction



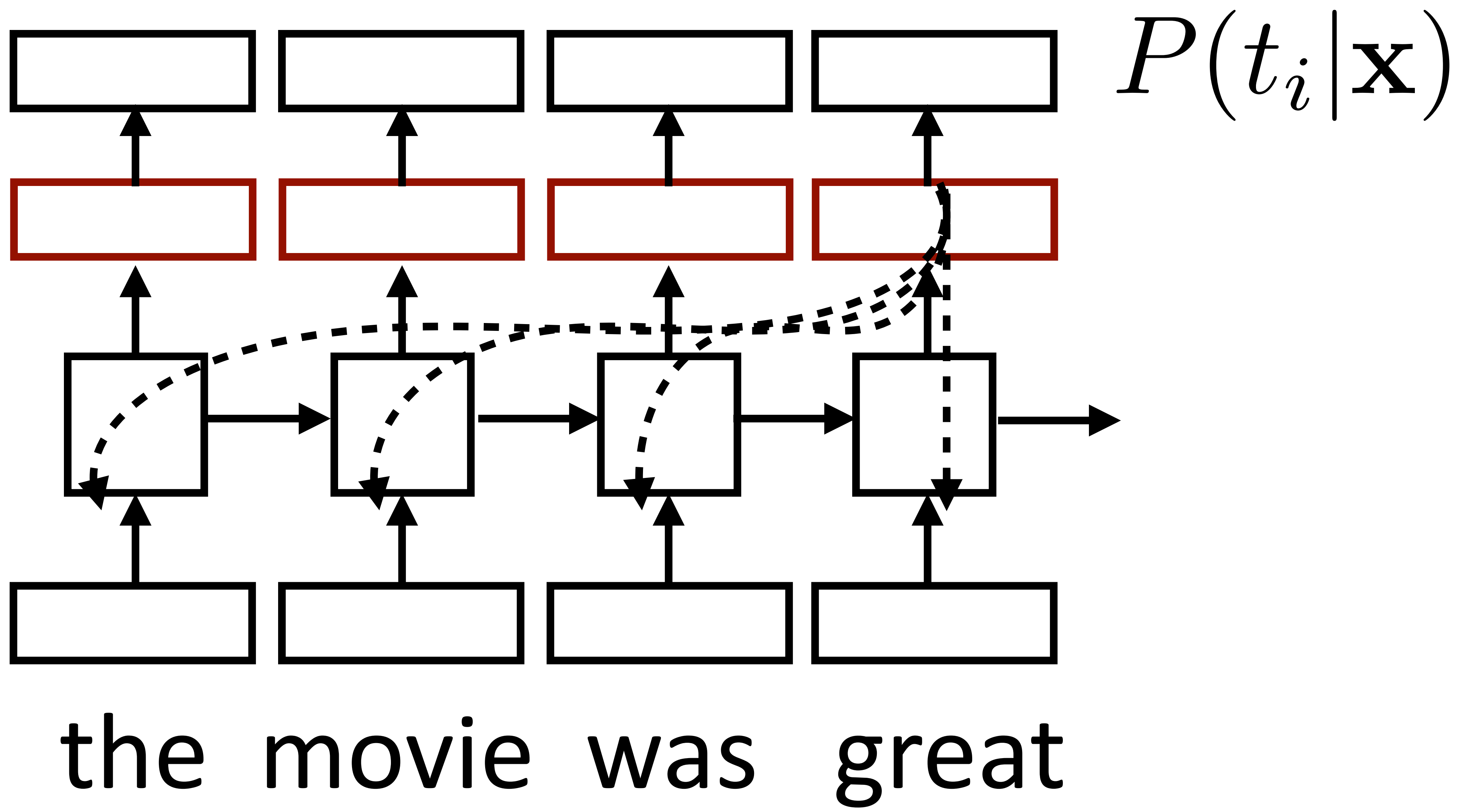
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors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

Recall: Training RNNs



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

Recall: 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

This Lecture

- RNN applications
- Language modeling
  - N-gram models
  - Neural LMs
- LM-based pretraining: ELMo

RNN 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)
- Textual entailment
  - Encode two sentences, predict

Sentiment Analysis

- Semi-supervised method: initialize the language model by training to reproduce the document in a seq2seq fashion (a type of pre-training called a sequential autoencoder)

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

Dai and Le (2015)

Natural Language Inference

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

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

SNLI Dataset

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

100D LSTM: 78% accuracy

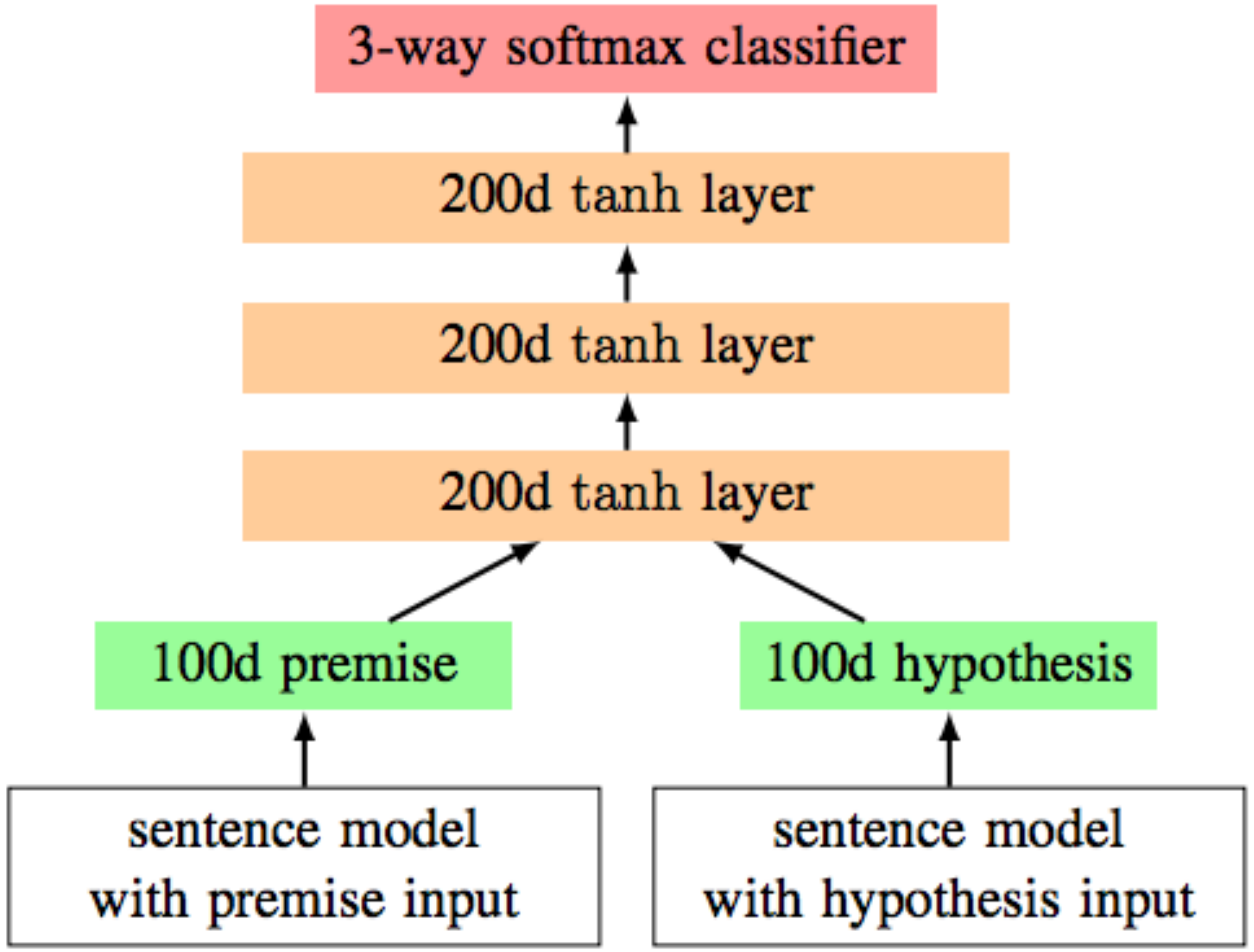
300D LSTM: 80% accuracy

(Bowman et al., 2016)

300D BiLSTM: 83% accuracy

(Liu et al., 2016)

- Later: better models for this

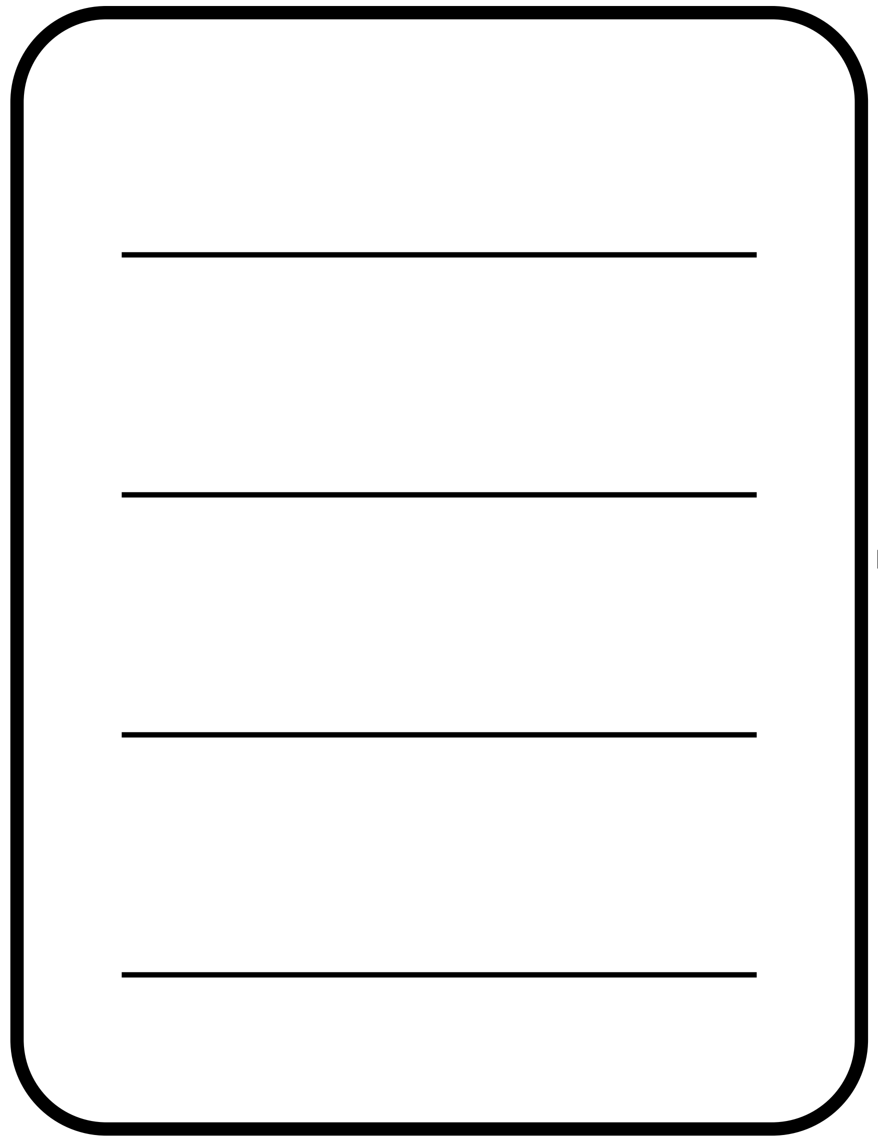


Bowman et al. (2015)

Language Modeling

NLP Analysis Pipeline

Text



Text Analysis

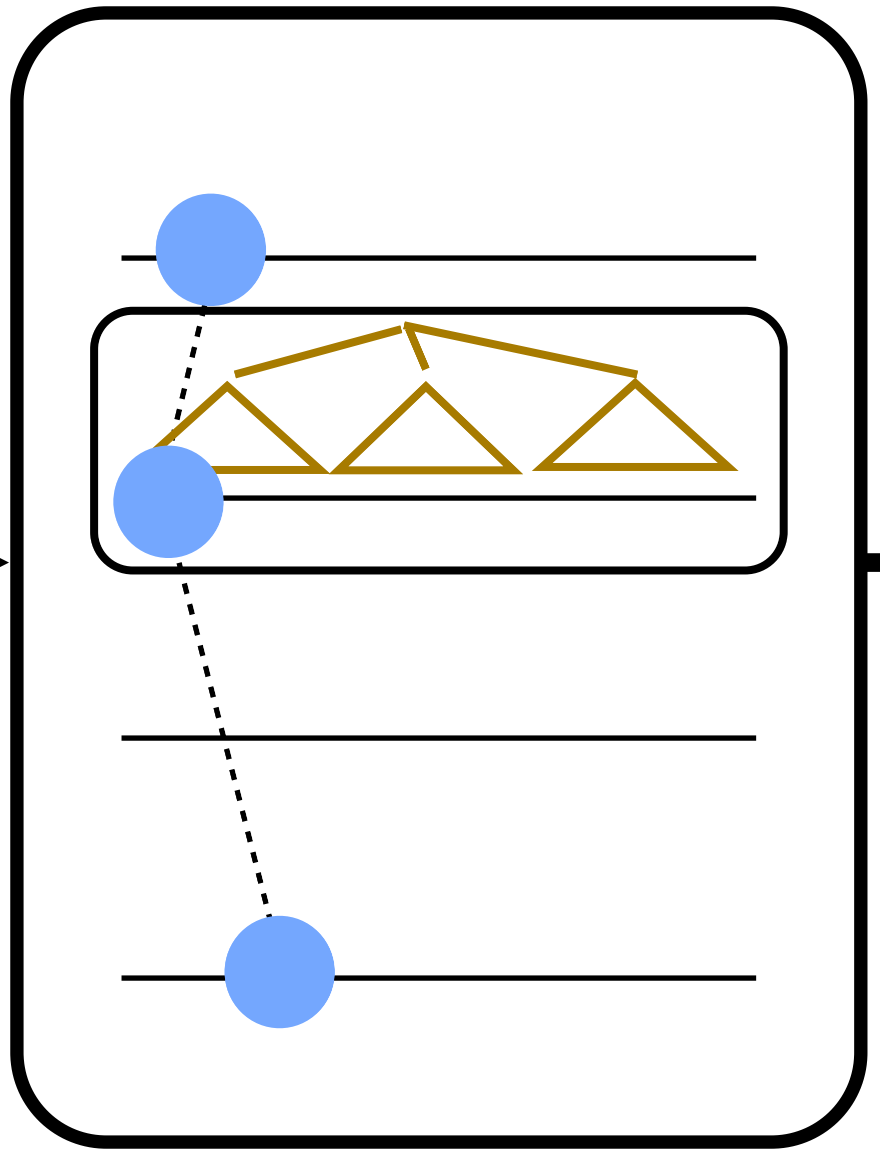
Syntactic parses

Coreference resolution

Entity disambiguation

Discourse analysis

Annotations



Applications

Summarize

Extract information

Answer questions

Identify sentiment

Translate

- So far in this class: mostly text analysis (about tagging, classifying, etc. the structure of text)
- Haven't talked about text *generation* tasks

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 week: 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 Language 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 $P(w_i|w_1, \ldots, w_{i-1}) = P(w_i|w_{i-n+1}, \ldots, w_{i-1})$

I visited San ______ put a distribution over the next word

$$P(w|\text{visited San}) = \frac{\text{count}(\text{visited San}, w)}{\text{count}(\text{visited San})}$$

Maximum likelihood estimate of this 3-gram probability from a corpus

- 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)

Smoothing N-gram Language Models

- What happens when we scale to longer contexts?

$P(w|\text{to})$ — *to* occurs 1M times in corpus

$P(w|\text{go to})$ — *go to* occurs 50,000 times in corpus

$P(w|\text{to go to})$ — *go to* occurs 1500 times in corpus

$P(w|\text{want to go to})$ — *want to go to*: only 100 occurrences

- Probability counts get very sparse, and we often want information from 5+ words away

Smoothing N-gram Language Models

I visited San ______ put a distribution over the next word

- Smoothing is very important, particularly when using 4+ gram models

$$P(w|\text{visited San}) = (1-\lambda)\frac{\text{count}(\text{visited San}, w)}{\text{count}(\text{visited San})} + \lambda\frac{\text{count}(\text{San}, w)}{\text{count}(\text{San})}$$

smooth this too!

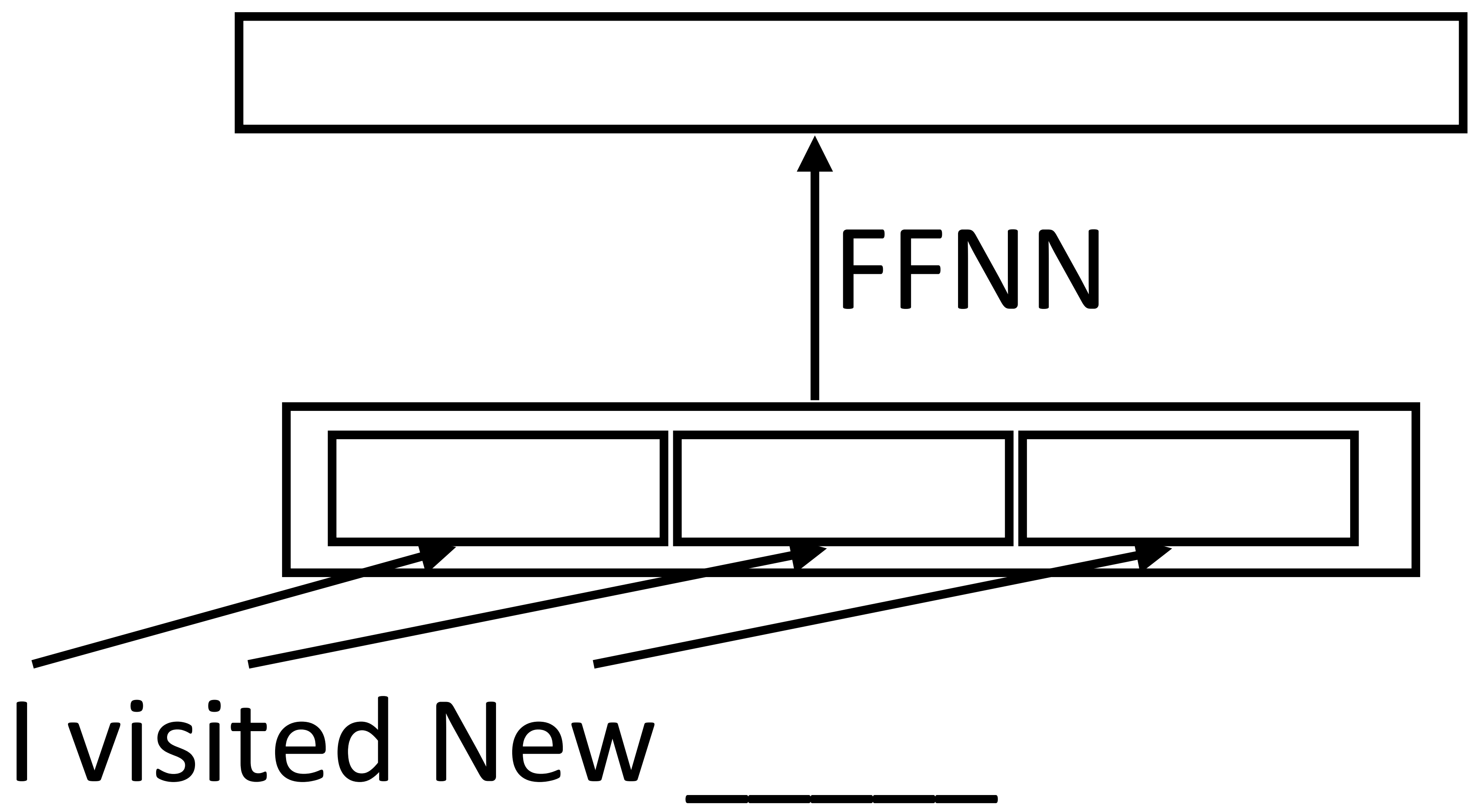
- One technique is "absolute discounting:" subtract off constant *k* from numerator, set lambda to make this normalize (*k*=1 is like leave-one-out)

$$P(w|\text{visited San}) = \frac{\text{count}(\text{visited San}, w) - k}{\text{count}(\text{visited San})} + \lambda\frac{\text{count}(\text{San}, w)}{\text{count}(\text{San})}$$

- Smoothing schemes get very complex!

Neural Language Models

- Early work: feedforward neural networks looking at context



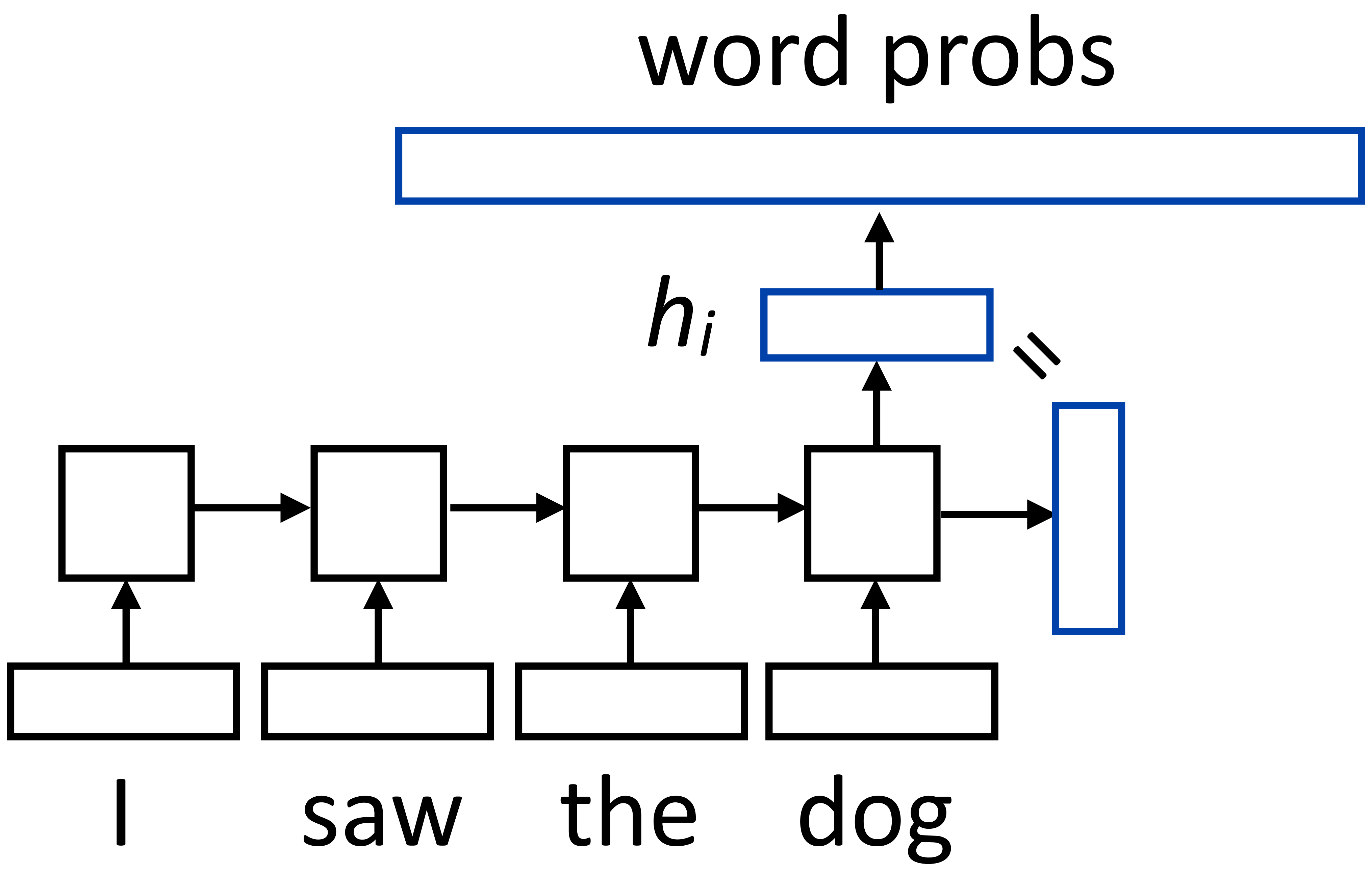
$$P(w_i | w_{i-n}, \ldots, w_{i-1})$$

- Slow to train over lots of data!

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

Mnih and Hinton (2003)

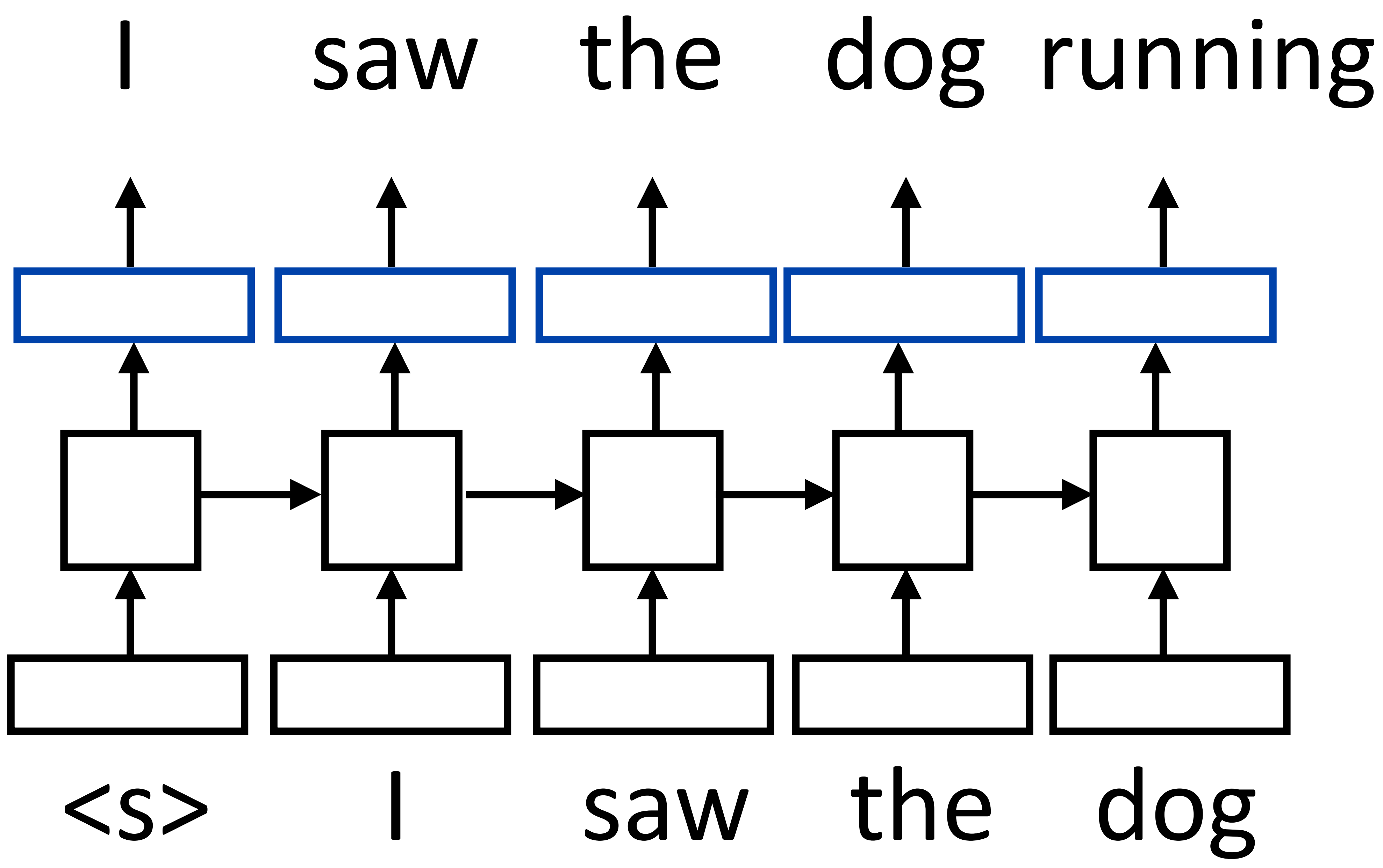
RNN Language Modeling



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

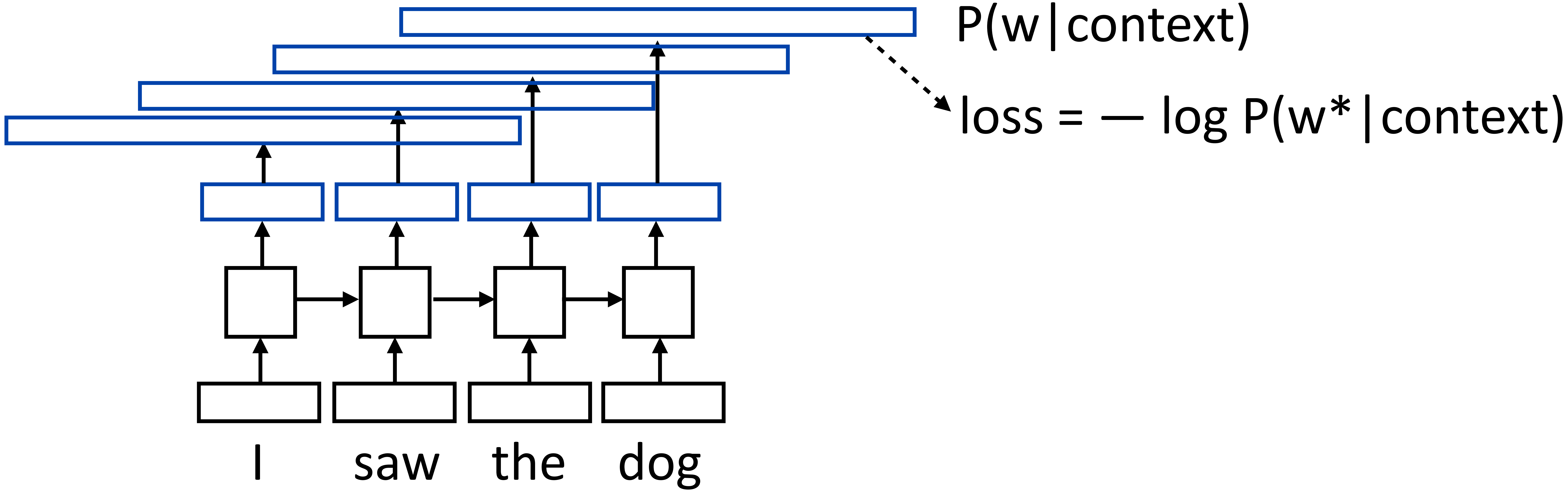
- W is a (vocab size) x (hidden size) matrix

Training RNNLMs



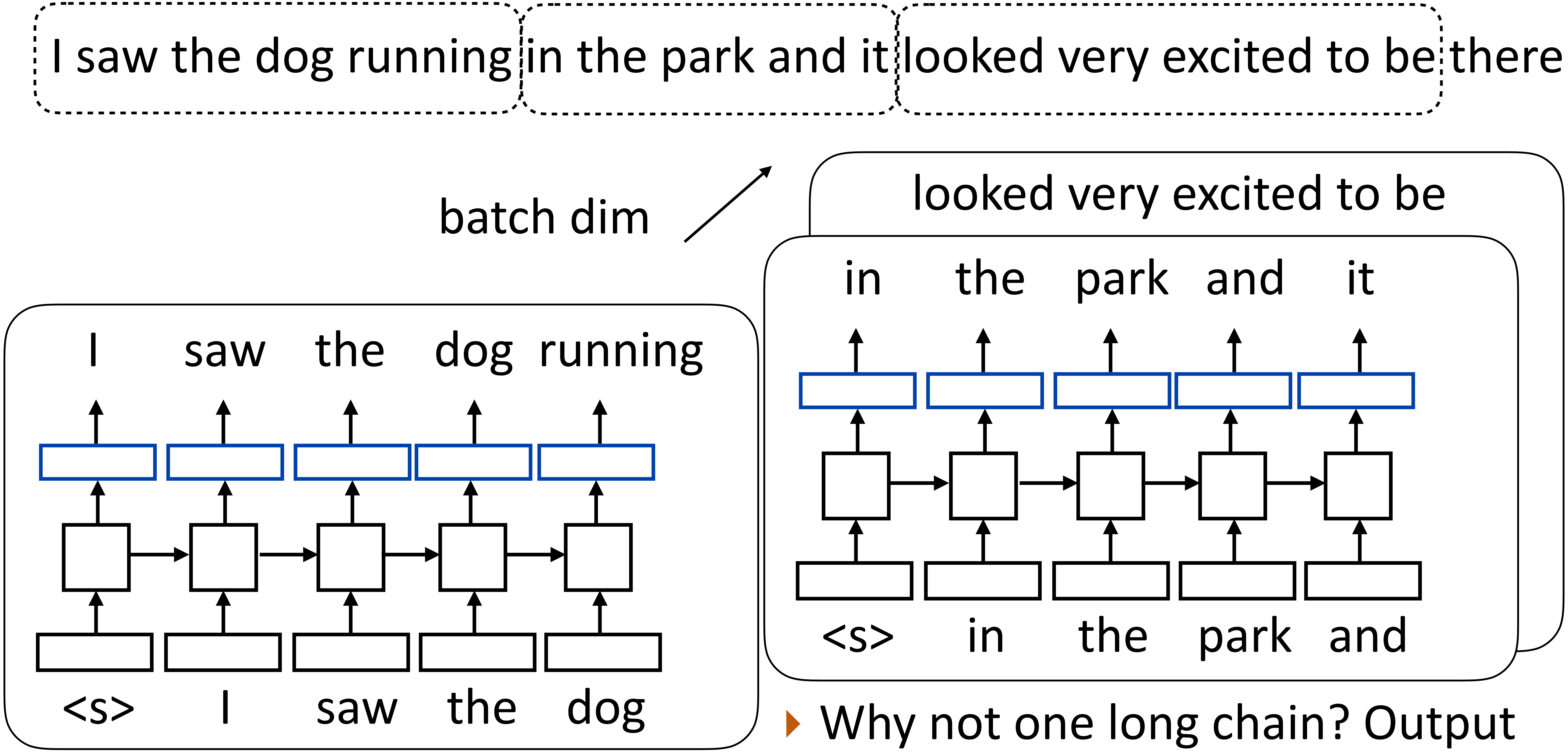
- Input is a sequence of words, output is those words shifted by one,
- Allows us to efficiently batch up training across time (one run of the RNN)

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

Batched LM Training



- Why not one long chain? Output depends on previous timesteps

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
  - Suppose we have probs 1/4, 1/3, 1/4, 1/3 for 4 predictions
  - Avg NLL (base e) = 1.242   Perplexity = 3.464

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

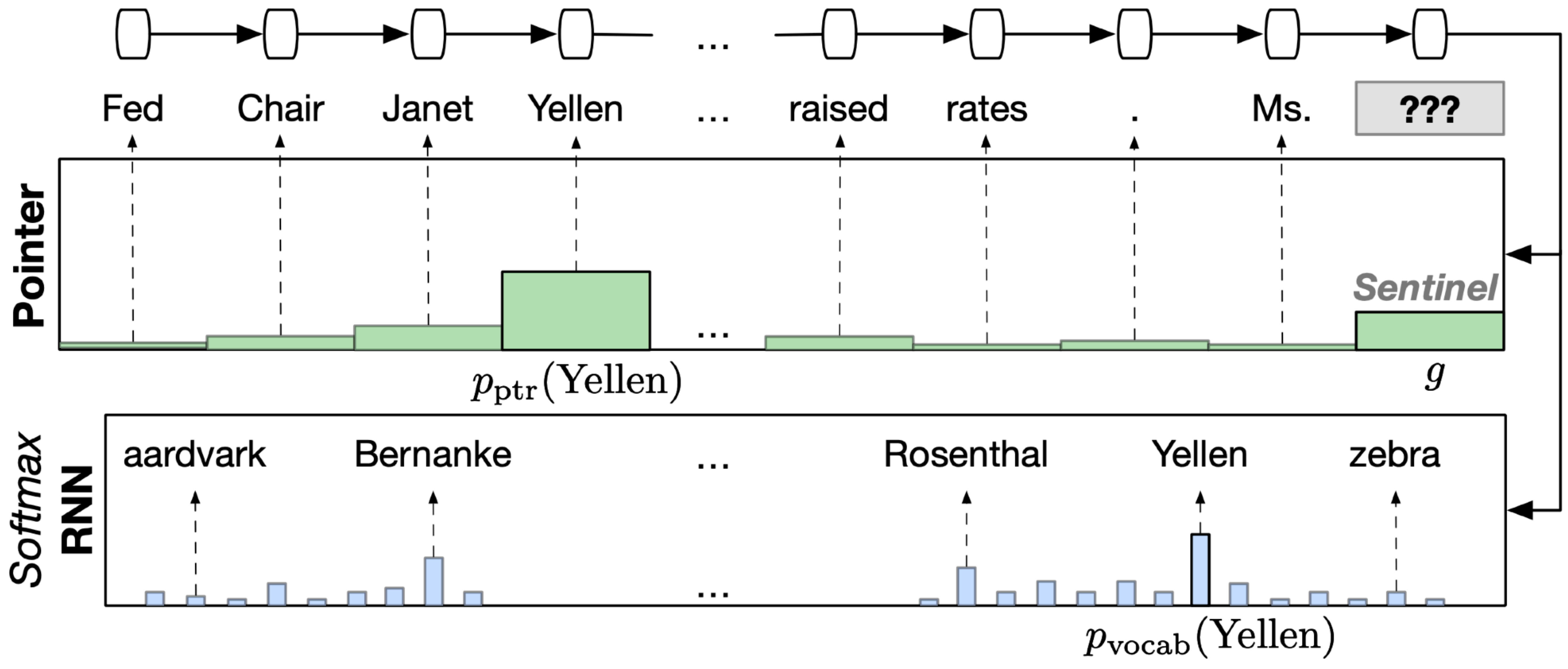
- LSTM: PPL = 60-80 (depending on how much you optimize it)

- 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)

Limitations of LSTM LMs

- Need some kind of pointing mechanism to repeat recent words



Merity et al. (2016)

$$p(\text{Yellen}) = g\, p_{\text{vocab}}(\text{Yellen}) + (1-g)\, p_{\text{ptr}}(\text{Yellen})$$

- Transformers can do this (will discuss later in the course)
- Scaling helps a lot! GPT-2 results:
  PTB perplexity = 65.85 with 117M params => 35.76 w/ 1542M params

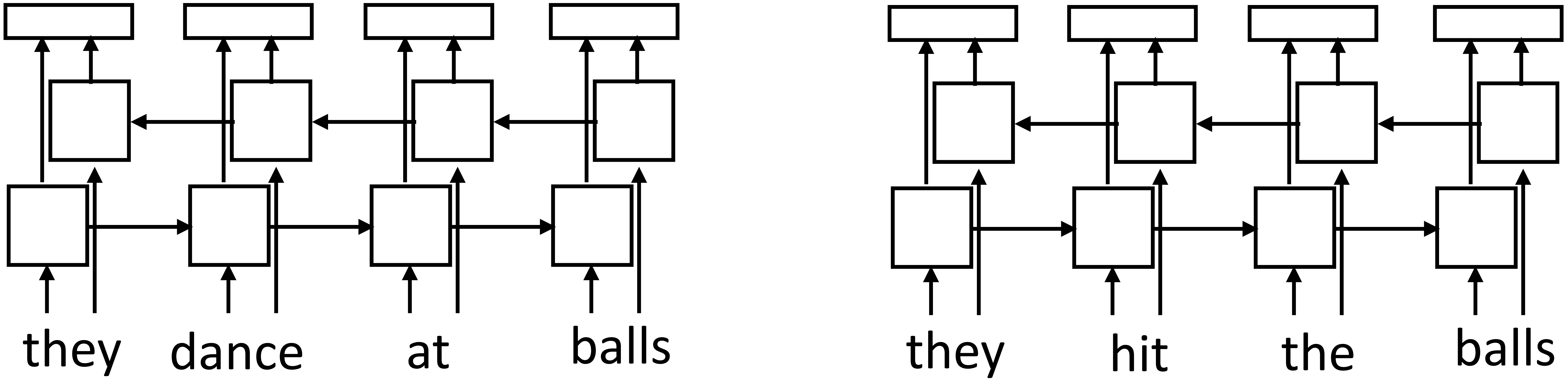
Applications of Language Modeling

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

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)

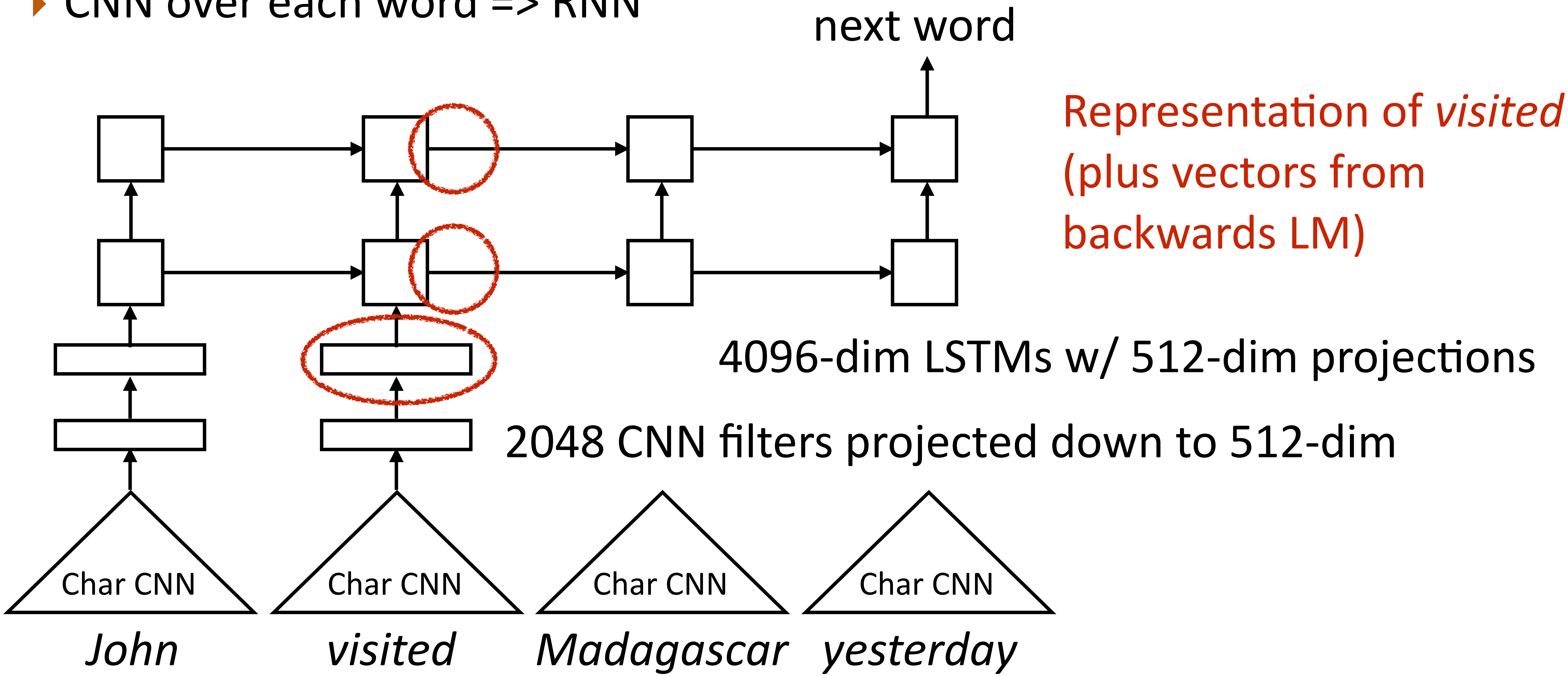
ELMo

- 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, books, crawled stuff from the web, …
- What do we want our LM to look like?

Peters et al. (2018)

ELMo

- CNN over each word => RNN



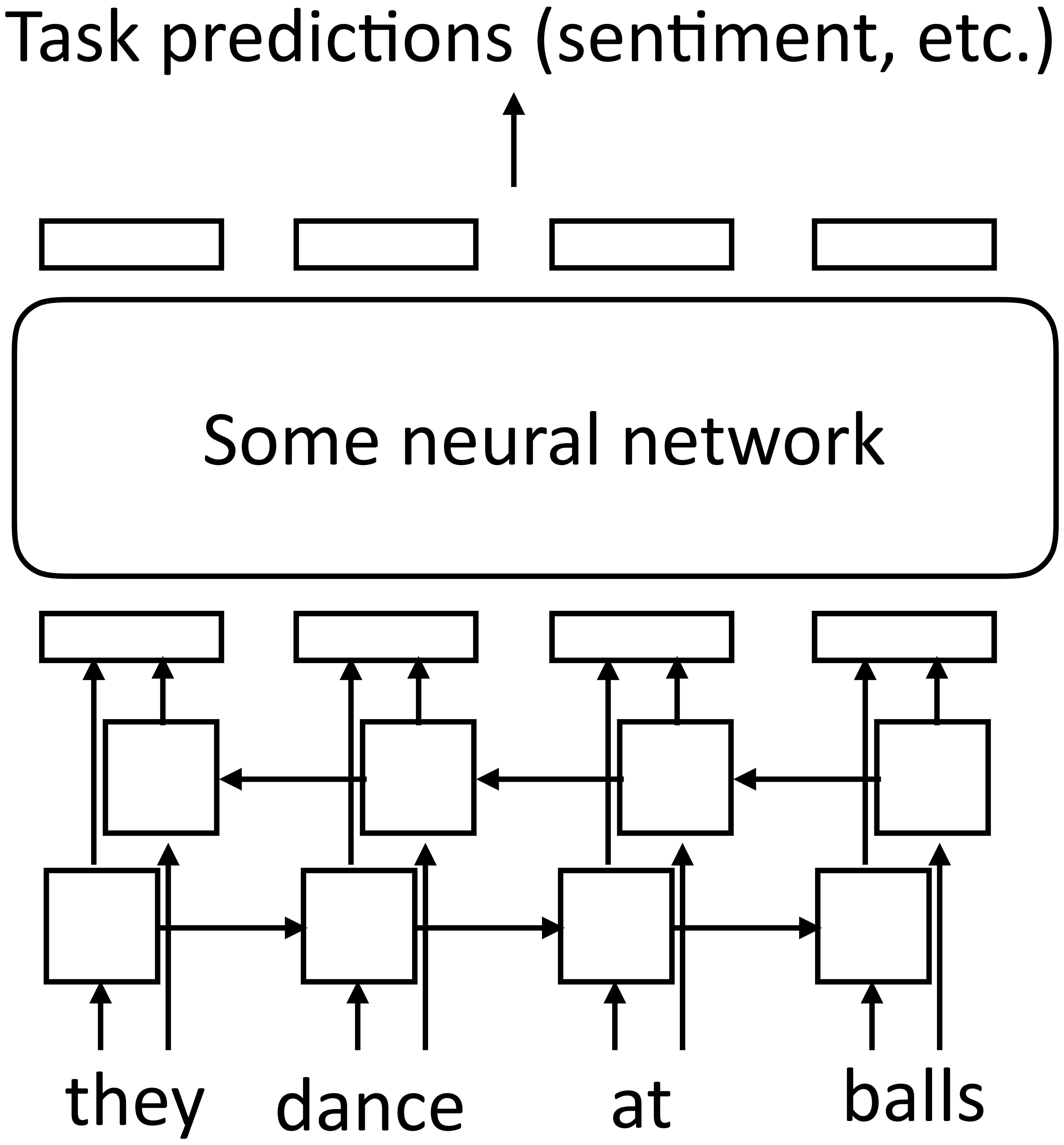
Peters et al. (2018)

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 (discussed later in the course), coreference resolution, named entity recognition, and sentiment analysis

Peters et al. (2018)

How to apply ELMo?

Pretraining	Adaptation	NER	SA	Nat. lang. inference		Semantic textual similarity		
		CoNLL 2003	SST-2	MNLI	SICK-E	SICK-R	MRPC	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: Pretrain	Adapt.	Task	Guidelines
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)

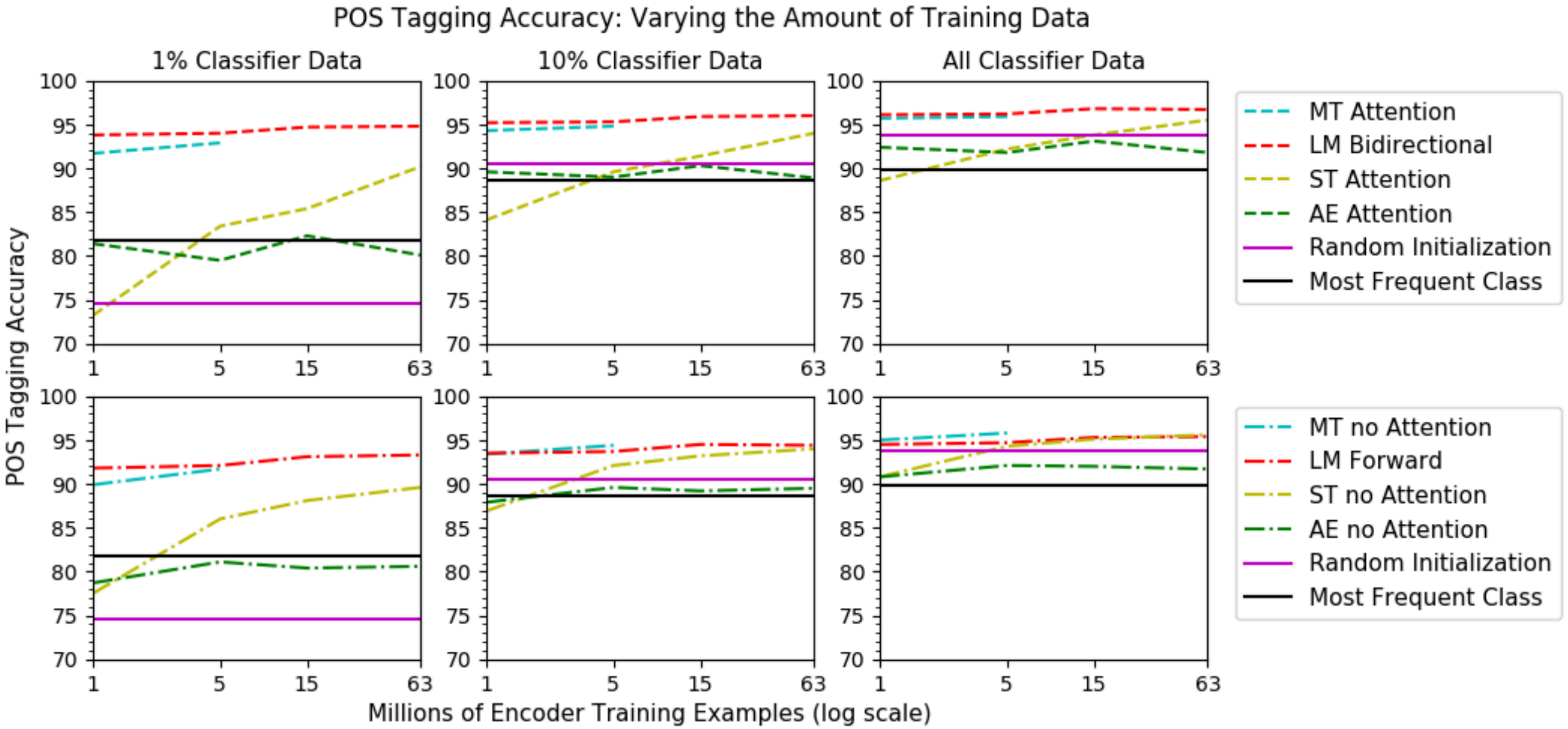
- Successfully predicting next words requires modeling lots of different effects in text

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 is language modeling a good objective?



Zhang and Bowman (2018)

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

- Next time: syntactic parsing