

# TTIC 31190: Natural Language Processing

Kevin Gimpel  
Spring 2018

Lecture 15:  
Review

# Roadmap

- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

# Why is NLP hard?

- ambiguity and variability of linguistic expression:
  - variability: many forms can mean the same thing
  - ambiguity: one form can mean many things
- many different kinds of variability and ambiguity
- each NLP task must address distinct kinds

# Sample Question

Spammers often use modifications of certain words to avoid being flagged. For example:

F . R E E

d ! e t

c o u p 0 n

This poses challenges for building spam filters.

1. Is this challenge caused by ambiguity (A) or variability (V) of natural language?

# Sample Question

Spammers often use modifications of certain words to avoid being flagged. For example:

F . R E E

d ! e t

c o u p 0 n

This poses challenges for building spam filters.

1. Is this challenge caused by ambiguity (A) or variability (V) of natural language? **V**

# Sample Question

Spammers often use modifications of certain words to avoid being flagged. For example:

F . R E E

d ! e t

c o u p 0 n

This poses challenges for building spam filters.

2. If you noticed spammers starting to do this, how might you change your system to deal with it?

# Sample Question

2. If you noticed spammers starting to do this, how might you change your system to deal with it?

you could train word clusters or embeddings on data that contains both spam and non-spam

you could also use features based on subword structure, like character n-grams

# Roadmap

- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks



# Types and Tokens

- once text has been tokenized, let's count the words
- **types**: entries in the vocabulary
- **tokens**: instances of types in a corpus
- example sentence: *If they want to go , they should go .*
  - how many types? 8
  - how many tokens? 10
- **type/token ratio**: useful statistic of a corpus (here, 0.8)
- as we add data, what happens to the type-token ratio?

# Morphology

- **morphemes:**
  - the small meaningful units that make up words
  - **stems:** core meaning-bearing units
  - **affixes:** bits and pieces that adhere to stems
    - often with grammatical functions

# Kinds of Word Formation

- **inflection**: modifying word with affix to change grammatical function (tense, number, etc.)
  - result is “different form of same word”
  - examples: *book* → *books*, *walk* → *walked*
- **derivation**: adding affix to stem to create a new word
  - examples: *great* → *greatly*, *great* → *greatness*
- **compounding**: combining two stems
  - examples: *lawsuit*, *keyboard*, *bookcase*

# Morphology in NLP

- two common tasks:
  - lemmatization
  - stemming

# Sample Question

- Which task would stemming/lemmatization be more helpful for: word sense disambiguation (WSD) or part-of-speech (POS) tagging? Why?

# Sample Question

- Which task would stemming/lemmatization be more helpful for: word sense disambiguation (WSD) or part-of-speech (POS) tagging? Why?
- WSD
  - WSD is more focused on semantics, which is mostly preserved by stemming/lemmatization
  - POS tagging can benefit much more from morphological indicators (to differentiate tenses of verbs, number of nouns, etc.)

# Many ways to get word vectors

some based on counting, some based on prediction/learning

some sparse, some dense

some have interpretable dimensions, some don't

shared ideas:

model meaning of a word by “embedding” it in a vector space

these word vectors are also called “**embeddings**”

contrast: in traditional NLP, word meaning is represented by a vocabulary index (“word #545”)

# Context words of “cooked” with highest PMIs

9.30533	beef	7.66406	chili
8.88418	shrimp	7.56264	rice
8.63397	potatoes	7.56167	soup
8.61946	ate	7.45315	flour
8.56584	dishes	7.43874	steamed
8.50945	eaten	7.43715	crushed
8.4931	beans	7.41193	meals
8.33137	texture	7.39793	digest
8.29489	vegetables	7.39175	rockies
8.25088	soda	7.34773	ramsay
8.20831	meat	7.33211	honey
8.15708	sauce	7.32253	toxicity
8.08345	consuming	7.29057	cared
7.9532	cuisine	7.28626	tomatoes
7.94043	raw	7.27912	boiling
7.78435	curry	7.27769	dal
7.7563	juice	7.27485	citrus
7.74444	vegetable	7.25649	doncaster



# Word Sense Ambiguity

- many words have multiple meanings

- two ways to categorize the patterns of multiple meanings of words:
  - **homonymy**: the multiple meanings are unrelated (coincidental?)
  - **polysemy**: the multiple meanings are related

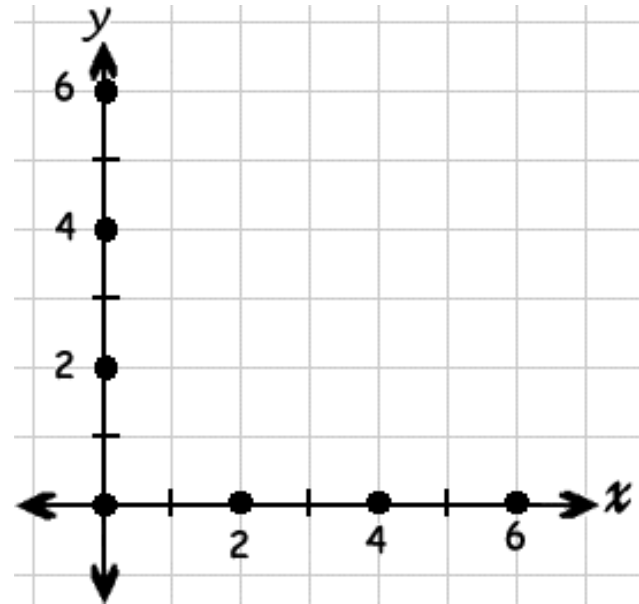
# Homonymy or Polysemy?

axes

an edge tool with a heavy bladed head mounted across a handle



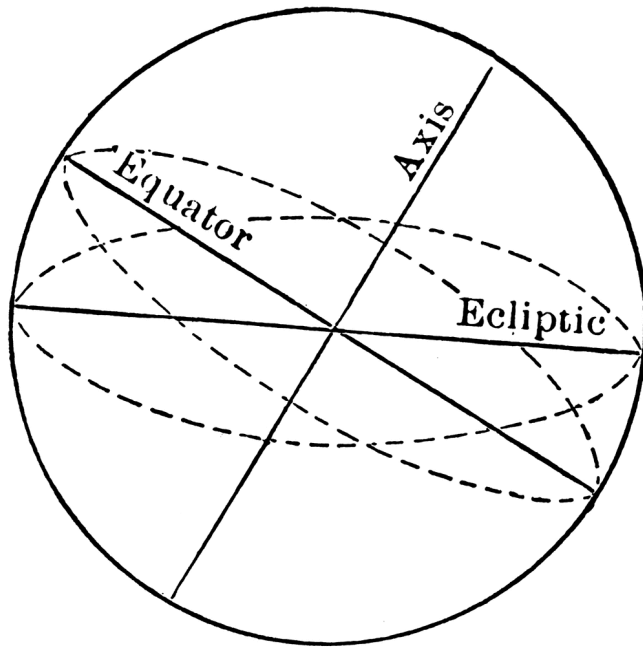
a fixed reference line for the measurement of coordinates



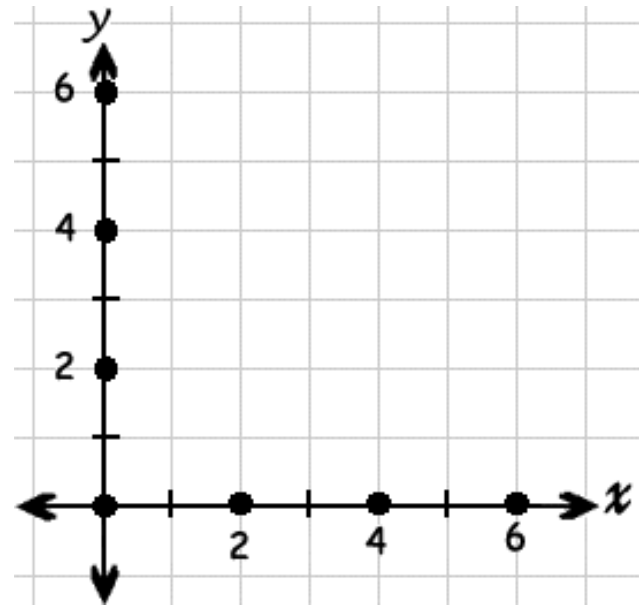
# Homonymy or **Polysemy**?

## axes

an imaginary line about which a body rotates



a fixed reference line for the measurement of coordinates



# Synonyms

- words with same meaning in some or all contexts:
  - *big / large*
  - *water / H<sub>2</sub>O*
- two lexemes are synonyms if they can be substituted for each other in all situations

# Antonyms

- senses that are opposites with respect to one feature of meaning
- otherwise, they are very similar!

*dark/light short/long fast/slow rise/fall*

*hot/cold up/down in/out*

# Sample Question

Word embeddings often have high similarity for antonym pairs, e.g., rise/fall, up/down, increase/decrease.

Suppose you are developing an NLP system for a financial application and you want to avoid this.

How would you learn word embeddings that avoid having high similarity for antonyms?

# Senses of *bass* in WordNet

## Noun

- **S: (n) bass** (the lowest part of the musical range)
- **S: (n) bass, bass part** (the lowest part in polyphonic music)
- **S: (n) bass, basso** (an adult male singer with the lowest voice)
- **S: (n) sea bass, bass** (the lean flesh of a saltwater fish of the family Serranidae)
- **S: (n) freshwater bass, bass** (any of various North American freshwater fish with lean flesh (especially of the genus *Micropterus*))
- **S: (n) bass, bass voice, basso** (the lowest adult male singing voice)
- **S: (n) bass** (the member with the lowest range of a family of musical instruments)
- **S: (n) bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

## Adjective

- **S: (adj) bass, deep** (having or denoting a low vocal or instrumental range) "*a deep voice*"; "*a bass voice is lower than a baritone voice*"; "*a bass clarinet*"



# Word Sense Disambiguation (WSD)

- given:
  - a word in context
  - a fixed inventory of potential word senses
- decide which sense of the word this is
- why? machine translation, question answering, sentiment analysis, text-to-speech



# Supersenses: top level hypernyms in hierarchy

(counts from Schneider & Smith's Streusel corpus)

## Noun

---

GROUP	1469	<i>place</i>
PERSON	1202	<i>people</i>
ARTIFACT	971	<i>car</i>
COGNITION	771	<i>way</i>
FOOD	766	<i>food</i>
ACT	700	<i>service</i>
LOCATION	638	<i>area</i>
TIME	530	<i>day</i>
EVENT	431	<i>experience</i>
COMMUNIC.*	417	<i>review</i>
POSSESSION	339	<i>price</i>
ATTRIBUTE	205	<i>quality</i>
QUANTITY	102	<i>amount</i>
ANIMAL	88	<i>dog</i>

BODY	87	<i>hair</i>
STATE	56	<i>pain</i>
NATURAL OBJ.	54	<i>flower</i>
RELATION	35	<i>portion</i>
SUBSTANCE	34	<i>oil</i>
FEELING	34	<i>discomfort</i>
PROCESS	28	<i>process</i>
MOTIVE	25	<i>reason</i>
PHENOMENON	23	<i>result</i>
SHAPE	6	<i>square</i>
PLANT	5	<i>tree</i>
OTHER	2	<i>stuff</i>

---

## Verb

---

STATIVE	2922	<i>is</i>
COGNITION	1093	<i>know</i>
COMMUNIC.*	974	<i>recommend</i>
SOCIAL	944	<i>use</i>
MOTION	602	<i>go</i>
POSSESSION	309	<i>pay</i>
CHANGE	274	<i>fix</i>
EMOTION	249	<i>love</i>
PERCEPTION	143	<i>see</i>
CONSUMPTION	93	<i>have</i>
BODY	82	<i>get...done</i>
CREATION	64	<i>cook</i>
CONTACT	46	<i>put</i>
COMPETITION	11	<i>win</i>
WEATHER	0	—

---

# Sample Question

Is supersense tagging more similar to WSD or POS tagging? Why?

# Sample Question

Is supersense tagging more similar to WSD or POS tagging? **POS tagging**

Why?

same labels used for multiple word types (in WSD, each word type has its own labels)

typically, there are 20-30 noun supersenses and 10-20 verb supersenses

but in some ways, supersense tagging is similar to WSD because it pertains more to semantics than syntax

# Roadmap

- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

# NLP Datasets

- NLP datasets include inputs (usually text) and outputs (usually some sort of **annotation**)

# Sample Question

Suppose you want to build a system to automatically detect the most important/salient words in a piece of text, say an email, a social media post, a news article, etc.

The idea is that the system can automatically highlight those words for a user to potentially enable faster reading for users in a hurry.

How might you collect naturally-annotated data for this task?



# What is a classifier?

- a function from inputs  $\mathbf{x}$  to classification labels  $y$
- one simple type of classifier:
  - for any input  $\mathbf{x}$ , assign a score to each label  $y$ , parameterized by parameters  $\mathbf{w}$ :

$$\text{score}(\mathbf{x}, y, \mathbf{w})$$

- classify by choosing highest-scoring label:

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{y}{\operatorname{argmax}} \text{score}(\mathbf{x}, y, \mathbf{w})$$

# Modeling, Inference, Learning

**inference:** solve  $\operatorname{argmax}$

**modeling:** define score function

$$\operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname{argmax}_y \operatorname{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

**learning:** choose  $\boldsymbol{w}$

- We will use this formulation throughout
  - even when output space is exponentially large or unbounded (e.g., machine translation)

# Linear Models

- parameters are arranged in a vector  $\mathbf{w}$
- score function is linear in  $\mathbf{w}$ :

$$\text{score}(\mathbf{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}, y) = \sum_i w_i f_i(\mathbf{x}, y)$$

- $\mathbf{f}$  : vector of feature functions

# Unigram Binary Features

- two example features:

$$f_1(\mathbf{x}, y) = \mathbb{I}[y = \text{positive}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

$$f_2(\mathbf{x}, y) = \mathbb{I}[y = \text{negative}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

where  $\mathbb{I}[S] = 1$  if  $S$  is true, 0 otherwise

- we usually think in terms of feature **templates**
- unigram binary feature template:

$$f^{\text{u,b}}(\mathbf{x}, y) = \mathbb{I}[y = \text{label}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{word}]$$

- to create features, this feature template is instantiated for particular labels and words

# Feature Engineering for Text Classification

- Two features:

$$f_1(\mathbf{x}, y) = \mathbb{I}[y = \text{positive}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

$$f_2(\mathbf{x}, y) = \mathbb{I}[y = \text{negative}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

- Let's say we set  $w_1 > w_2$
- On sentences containing “**great**” in the Stanford Sentiment Treebank training data, this would get us an accuracy of 69%
- But “**great**” only appears in 83/6911 examples

# Feature Engineering for Text Classification

- Two features:

$$f_1(\mathbf{x}, y) = \mathbb{I}[y = \text{positive}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

$$f_2(\mathbf{x}, y) = \mathbb{I}[y = \text{negative}] \wedge \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

**ambiguity**: “*great*” may not mean positive sentiment

- On sentences containing “*great*” in the Stanford Sentiment Treebank training data, this would get us an accuracy of 69%
- But “*great*” only appears in 83/6911 examples

**variability**: many other words can indicate positive sentiment

# Sample Question

Typically, bigrams that contain “*good*” are predictive of positive sentiment and bigrams containing “*downside*” are predictive of negative sentiment.

However, Pat trained a linear model text classifier with binary unigram and binary bigram features and found:

- “*only good*” is strongly predictive of **negative** sentiment
- “*only downside*” is strongly predictive of **positive** sentiment

What’s going on here?

- What about a feature like the following?

$$f_3(\mathbf{x}, y) = \mathbb{I}[\mathbf{x} \text{ contains } \textit{great}]$$

- What do you expect its weight to be?
  - Doesn't matter.
  - Why?

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{y \in \{\text{positive}, \text{negative}\}}{\text{argmax}} \quad \mathbf{w}^\top \mathbf{f}(\mathbf{x}, y)$$

- a feature with the same value for all outputs will not affect the argmax



# Sample Question

- Why do our linear model features always have to consider the output label while the features input to a neural network do not?
- features input to a neural network are transformed, then transformed values are paired with the output label in the final layer. Our linear model features are used directly to score output labels.

# Cost Functions

- **cost function**: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task
- usual conventions:  $\text{cost}(y, y) = 0$   
 $\text{cost}(y, y') = \text{cost}(y', y)$

# Cost Functions

- **cost function**: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task
- for classification, what cost should we use?

$$\text{cost}(y, y') = \mathbb{I}[y \neq y']$$

- how about for other NLP tasks?

# Empirical Risk Minimization

(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \mathbb{E}_{P(\boldsymbol{x}, y)} [\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}))]$$



$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

solution: replace “cost loss” (also called “0-1” loss) with a **surrogate** function that is easier to optimize

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

cost loss / 0-1 loss:  $\operatorname{loss}_{\operatorname{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}))$

# Surrogate Loss Functions

cost loss / 0-1 loss:  $\text{loss}_{\text{cost}}(\mathbf{x}, y, \mathbf{w}) = \text{cost}(y, \text{classify}(\mathbf{x}, \mathbf{w}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \mathbf{w})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \mathbf{w}) + \text{cost}(y, y'))$$

# Sample Question

- what is the smallest value this loss can be?

$$\text{loss}_{\text{perc}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \mathbf{w})$$

- 0

# Sample Question

- write down a condition on  $\text{cost}(y, y')$  that will be sufficient to make this loss nonnegative:

$$\text{loss}_{\text{hinge}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \mathbf{w}) + \text{cost}(y, y'))$$

- $\text{cost}(y, y') \geq 0$



# Log Loss

$$\text{loss}_{\log}(\mathbf{x}, y, \mathbf{w}) = -\log p_{\mathbf{w}}(y | \mathbf{x})$$

- minimize negative log of conditional probability of output given input

# Score $\rightarrow$ Probability

- can turn score into probability by exponentiating (to make it positive) and normalizing:

$$p_{\mathbf{w}}(y \mid \mathbf{x}) \propto \exp\{\text{score}(\mathbf{x}, y, \mathbf{w})\}$$

$$p_{\mathbf{w}}(y \mid \mathbf{x}) = \frac{\exp\{\text{score}(\mathbf{x}, y, \mathbf{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\mathbf{x}, y', \mathbf{w})\}}$$

- this is often called a “softmax” function

# Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \underbrace{\lambda R(\boldsymbol{w})}_{\text{regularization term}}$$

Diagram illustrating the components of the regularized empirical risk minimization objective function:


- surrogate loss**: Points to the sum of losses  $\sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$ .
- regularization strength**: Points to the regularization parameter  $\lambda$ .
- regularization term**: Points to the regularization function  $R(\boldsymbol{w})$ .

# Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$



**encourages model  
to fit the training  
data well**



**encourages model to be  
“simpler” in the hope that  
this will help it to  
generalize to new data**

# Probabilistic Language Modeling

- goal: compute the probability of a sequence of words:

$$P(\mathbf{w}) = P(w_1, w_2, \dots, w_n)$$

- related task: probability of next word:

$$P(w_4 \mid w_1, w_2, w_3)$$

- a model that computes either of these:

$$P(\mathbf{w}) \quad \text{or} \quad P(w_k \mid w_1, w_2, \dots, w_{k-1})$$

is called a **language model (LM)**

# Markov Assumption



Andrei Markov

- simplifying assumption:

$P(\text{the } l \text{ its water is so transparent that}) \approx P(\text{the } l \text{ that})$

- or maybe:

$P(\text{the } l \text{ its water is so transparent that}) \approx P(\text{the } l \text{ transparent that})$

# Estimating bigram probabilities

- the maximum likelihood estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

# Probability of Held-out Data

- probability of held-out sentences:

$$\prod_i P(\mathbf{w}^{(i)})$$

- let's work with log-probabilities:

$$\log_2 \prod_i P(\mathbf{w}^{(i)}) = \sum_i \log_2 P(\mathbf{w}^{(i)})$$

- divide by number of words  $M$  in held-out sentences:

$$\frac{1}{M} \sum_i \log_2 P(\mathbf{w}^{(i)})$$



# Probability -> Perplexity

- average log-probability of held-out words:

$$\ell = \frac{1}{M} \sum_i \log_2 P(\mathbf{w}^{(i)})$$

- perplexity:

$$PP = 2^{-\ell}$$

# Intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w \mid \text{denied the})$

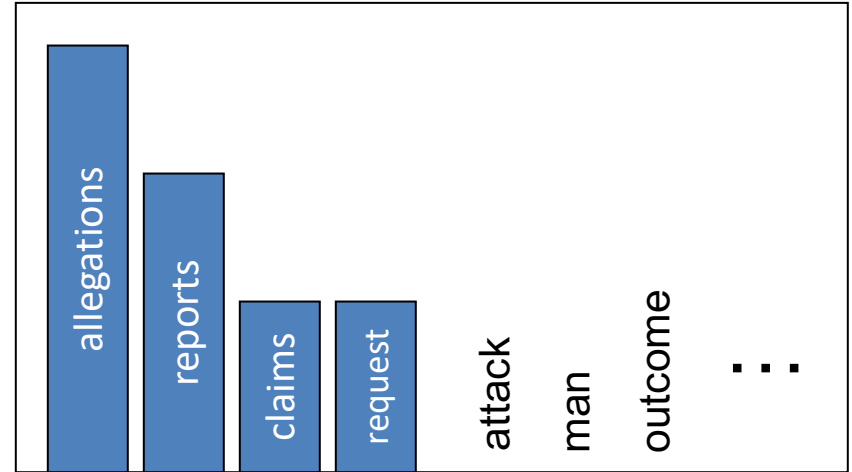
3 *allegations*

2 *reports*

1 *claims*

1 *request*

7 total



- Steal probability mass to generalize better:

$P(w \mid \text{denied the})$

2.5 *allegations*

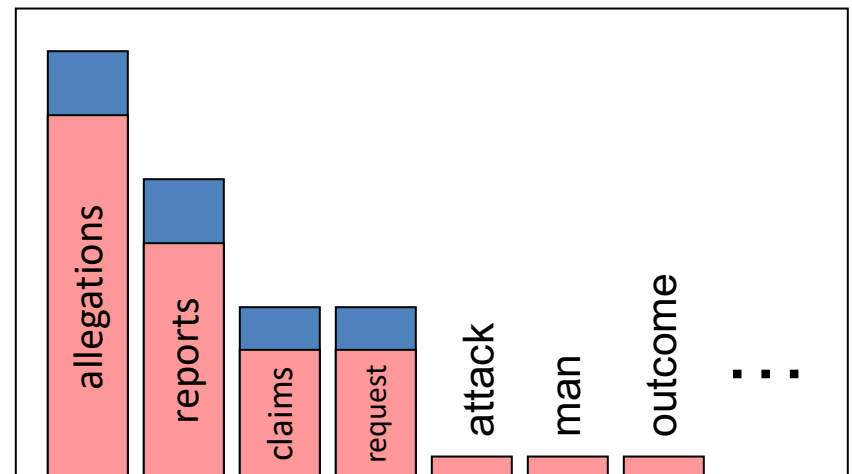
1.5 *reports*

0.5 *claims*

0.5 *request*

2 *other*

7 total



# “Add-1” estimation

- just add 1 to all counts
- MLE estimate:

$$P_{\text{MLE}}(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- Add-1 estimate:

$$P_{\text{add-1}}(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |\mathcal{V}|}$$

# Backoff and Interpolation

- sometimes it helps to use **less** context
  - condition on less context for contexts you haven't learned much about
- **backoff:**
  - use trigram if you have good evidence, otherwise bigram, otherwise unigram
- **interpolation:**
  - mixture of unigram, bigram, trigram (etc.) models
- interpolation works better

# Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

**Figure 4.8** For all bigrams in 22 million words of AP newswire of count 0, 1, 2,...,9, the counts of these bigrams in a held-out corpus also of 22 million words.

# Absolute Discounting

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21

observed bigrams have counts that are **overestimated**  
unobserved bigrams have counts that are **underestimated**

# Kneser-Ney Smoothing

- how many times is  $w$  a novel continuation?

$$P_{\text{continuation}}(w) \propto \underbrace{|\{w' : \text{count}(w', w) > 0\}|}$$

number of unique words that appeared before  $w$

# Kneser-Ney Smoothing

- how many times is  $w$  a novel continuation?

$$P_{\text{continuation}}(w) \propto |\{w' : \text{count}(w', w) > 0\}|$$

- normalize by total number of word bigram types:

$$P_{\text{continuation}}(w) = \frac{|\{w' : \text{count}(w', w) > 0\}|}{|\{\langle w', w \rangle : \text{count}(w', w) > 0\}|}$$



## A Neural Probabilistic Language Model

**Yoshua Bengio**

**Réjean Ducharme**

**Pascal Vincent**

**Christian Jauvin**

*Département d'Informatique et Recherche Opérationnelle*

*Centre de Recherche Mathématiques*

*Université de Montréal, Montréal, Québec, Canada*

BENGIOY@IRO.UMONTREAL.CA

DUCHARME@IRO.UMONTREAL.CA

VINCENTP@IRO.UMONTREAL.CA

JAUVINC@IRO.UMONTREAL.CA

- idea: use a neural network for  $n$ -gram language modeling:

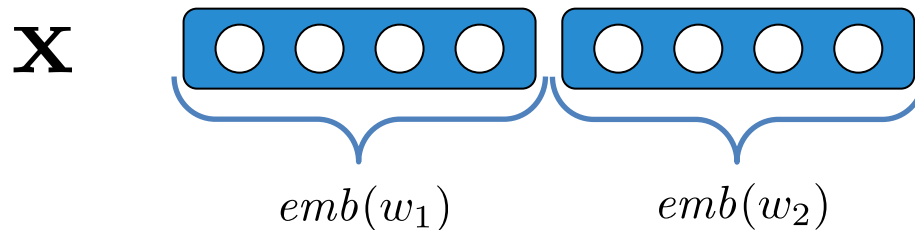
$$P_{\theta}(w_t \mid w_{t-n+1}, \dots, w_{t-2}, w_{t-1})$$

# A Simple Neural Trigram Language Model

- given previous words  $w_1$  and  $w_2$ , predict next word

# A Simple Neural Trigram Language Model

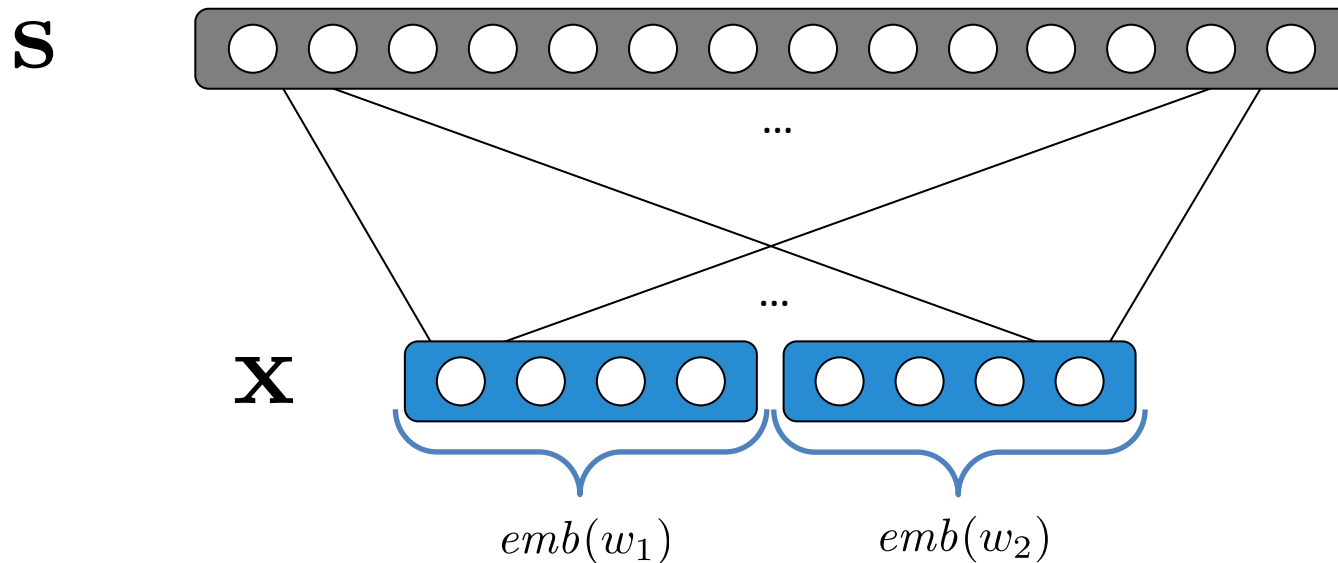
- given previous words  $w_1$  and  $w_2$ , predict next word
- input is concatenation of vectors (embeddings) of previous words:



$$\mathbf{x} = \text{cat}(emb(w_1), emb(w_2))$$

# A Simple Neural Trigram Language Model

- output vector contains scores of possible next words:



$$\mathbf{s} = \mathbf{U}\mathbf{x}$$

$$s_i = \text{score}(\mathbf{x}, w_i, \mathbf{U})$$

$$\text{score}(\mathbf{x}, w_i, \mathbf{U}) = \mathbf{x}^\top \mathbf{U}_{i,1:d}$$

# Sample Question

A feed-forward neural language model typically concatenates the embeddings of the previous words and predicts the next word.

Consider an alternative architecture that **averages** the embeddings of the previous words instead of concatenating them.

When would this “context-averaging” model work better than the “context-concatenation” model?

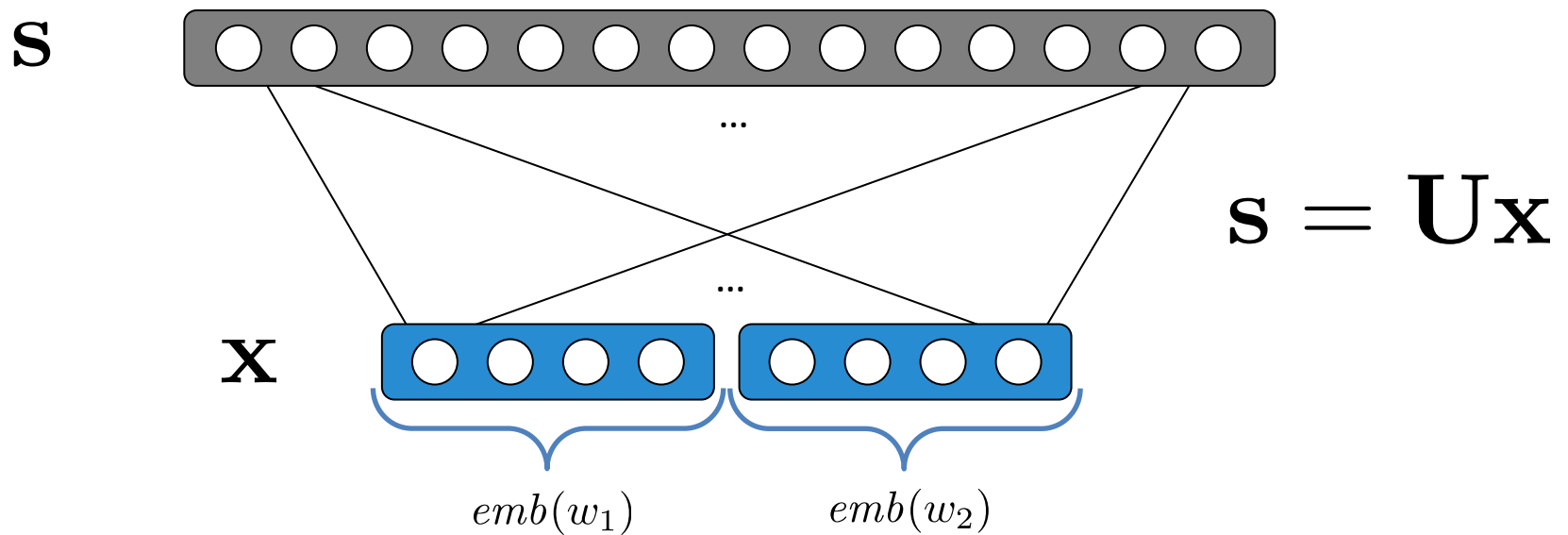
# Sample Question

When would this “context-averaging” model work better than the “context-concatenation” model?

if you are using a very large context or if you have very little data (since it has fewer parameters)

maybe free word order languages?

# A Simple Neural Trigram Language Model

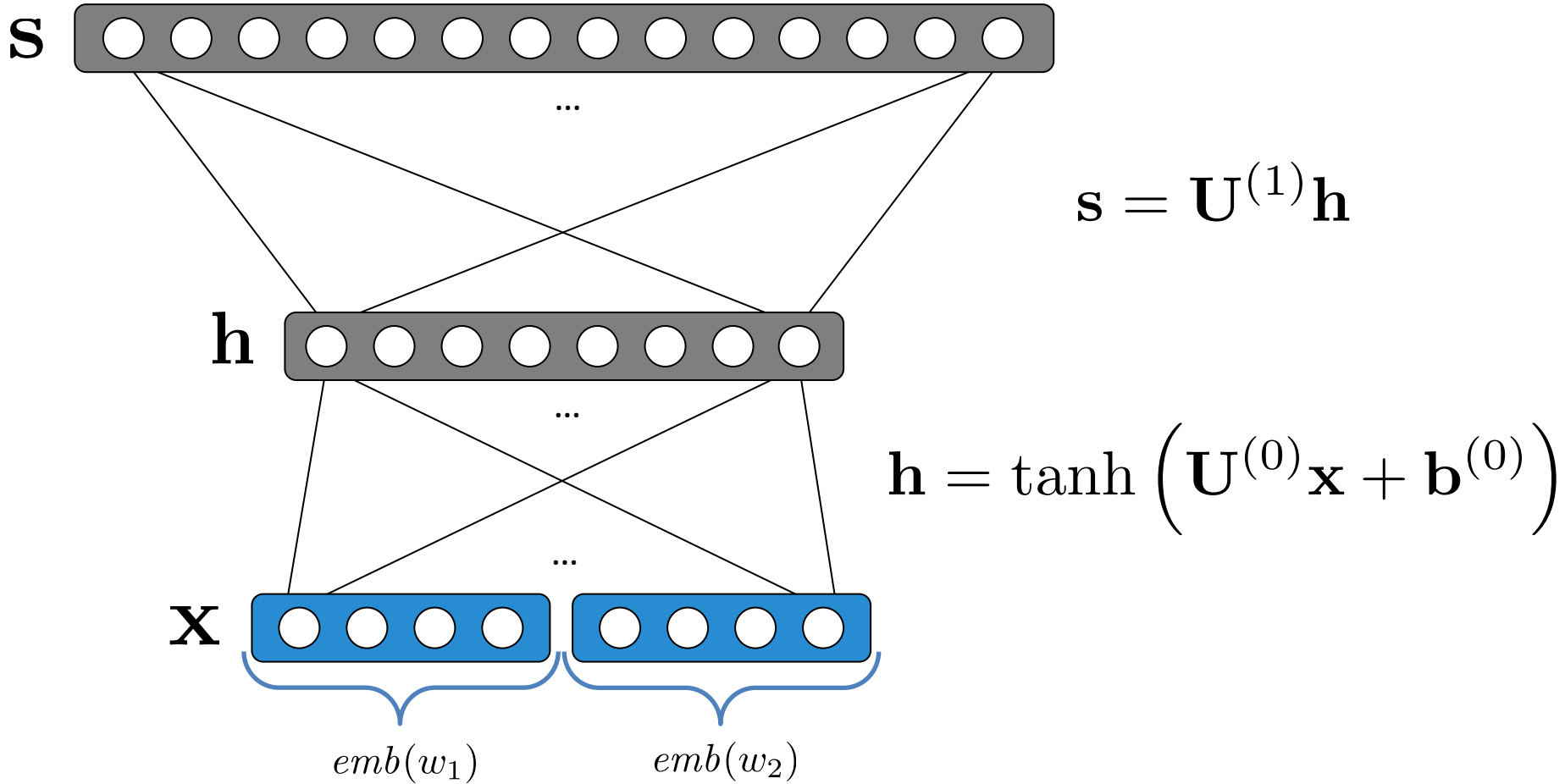


- most common way to train: log loss

$$\text{loss}_{\log}(\langle w_1, w_2 \rangle, w_3, \boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(w_3 \mid \langle w_1, w_2 \rangle)$$

$$p_{\boldsymbol{\theta}}(w_3 \mid \langle w_1, w_2 \rangle) \propto \exp\{\text{score}(\text{cat}(emb(w_1), emb(w_2)), w_3, \mathbf{U})\}$$

# Adding a Hidden Layer





# skip-gram training data (window size = 5)

corpus (English Wikipedia):

*agriculture is the traditional mainstay of the cambodian economy .  
but benares has been destroyed by an earthquake .*

...

inputs (x)	outputs (y)
agriculture	<s>
agriculture	is
agriculture	the
is	<s>
is	agriculture
is	the
is	traditional
the	is
...	...

# CBOW training data (window size = 5)

corpus (English Wikipedia):

*agriculture is the traditional mainstay of the cambodian economy .  
but benares has been destroyed by an earthquake .*

...

inputs (x)	outputs (y)
{<s>, is, the, traditional}	agriculture
{<s>, agriculture, the, traditional}	is
{agriculture, is, traditional, mainstay}	the
{is, the, mainstay, of}	traditional
{the, traditional, of, the}	mainstay
{traditional, mainstay, the, cambodian}	of
{mainstay, of, cambodian, economy}	the
...	...

# word2vec Score Functions

- skip-gram:

$$\text{score}(x, y, \mathbf{w}) = \mathbf{w}^{(\text{in}, x)} \cdot \mathbf{w}^{(\text{out}, y)}$$

inputs (x)	outputs (y)
agriculture	<s>
agriculture	is
agriculture	the

- CBOW:

$$\text{score}(\mathbf{x}, y, \mathbf{w}) = \left( \frac{1}{|\mathbf{x}|} \sum_i \mathbf{w}^{(\text{in}, x_i)} \right) \cdot \mathbf{w}^{(\text{out}, y)}$$

inputs (x)	outputs (y)
{<s>, is, the, traditional}	agriculture
{<s>, agriculture, the, traditional}	is
{agriculture, is, traditional, mainstay}	the

# skip-gram

- skip-gram objective: log loss

$$\min_{\mathbf{w}} \sum_{1 \leq t \leq |\mathcal{T}|} \sum_{-c \leq j \leq c, j \neq 0} -\log P_{\mathbf{w}}(x_{t+j} \mid x_t)$$



sum over  
positions in  
corpus



sum over context  
words in window

$$\min_{\mathbf{w}} \sum_{1 \leq t \leq |\mathcal{T}|} \sum_{-c \leq j \leq c, j \neq 0} -\log P_{\mathbf{w}}(x_{t+j} \mid x_t)$$

from score to probability:

$$P_{\mathbf{w}}(y \mid x) \propto \exp\{\text{score}(x, y, \mathbf{w})\}$$

$$P_{\mathbf{w}}(y \mid x) \propto \exp\{\mathbf{w}^{(\text{in}, x)} \cdot \mathbf{w}^{(\text{out}, y)}\}$$

$$\min_{\mathbf{w}} \sum_{1 \leq t \leq |\mathcal{T}|} \sum_{-c \leq j \leq c, j \neq 0} -\log P_{\mathbf{w}}(x_{t+j} \mid x_t)$$

normalization requires sum over entire vocabulary:

$$P_{\mathbf{w}}(y \mid x) = \frac{\exp\{\mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y)}\}}{\sum_{y'} \exp\{\mathbf{w}^{(\text{in},x)} \cdot \mathbf{w}^{(\text{out},y')}\}}$$

# Negative Sampling

(Mikolov et al., 2013)

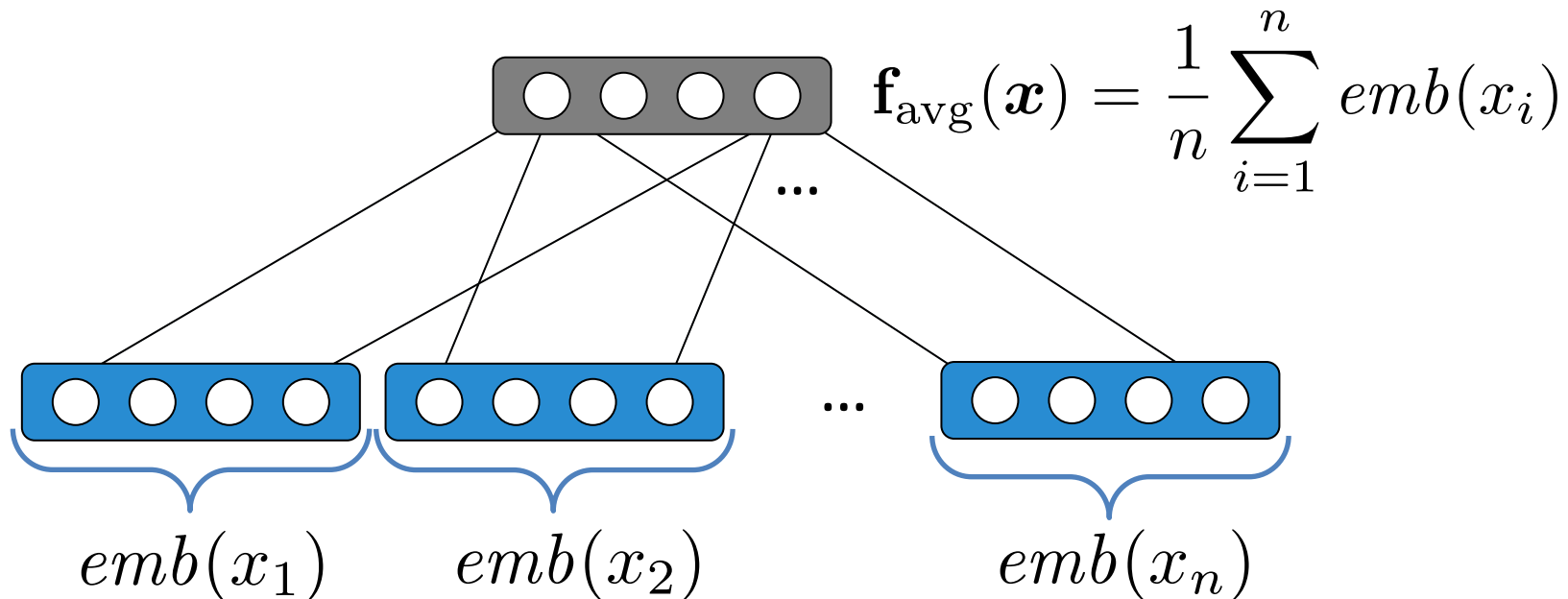
- rather than sum over entire vocabulary, generate samples and sum over them
- instead of a multiclass classifier, use a binary classifier:

$$\min_{\mathbf{w}} \sum_{1 \leq t \leq |\mathcal{T}|} \sum_{-c \leq j \leq c, j \neq 0} -\log \sigma(\text{score}(x_t, x_{t+j}, \mathbf{w})) + \sum_{x \in \text{NEG}} \log \sigma(\text{score}(x_t, x, \mathbf{w}))$$

- where sigma is logistic sigmoid function

# A Simple Neural Text Classification Model

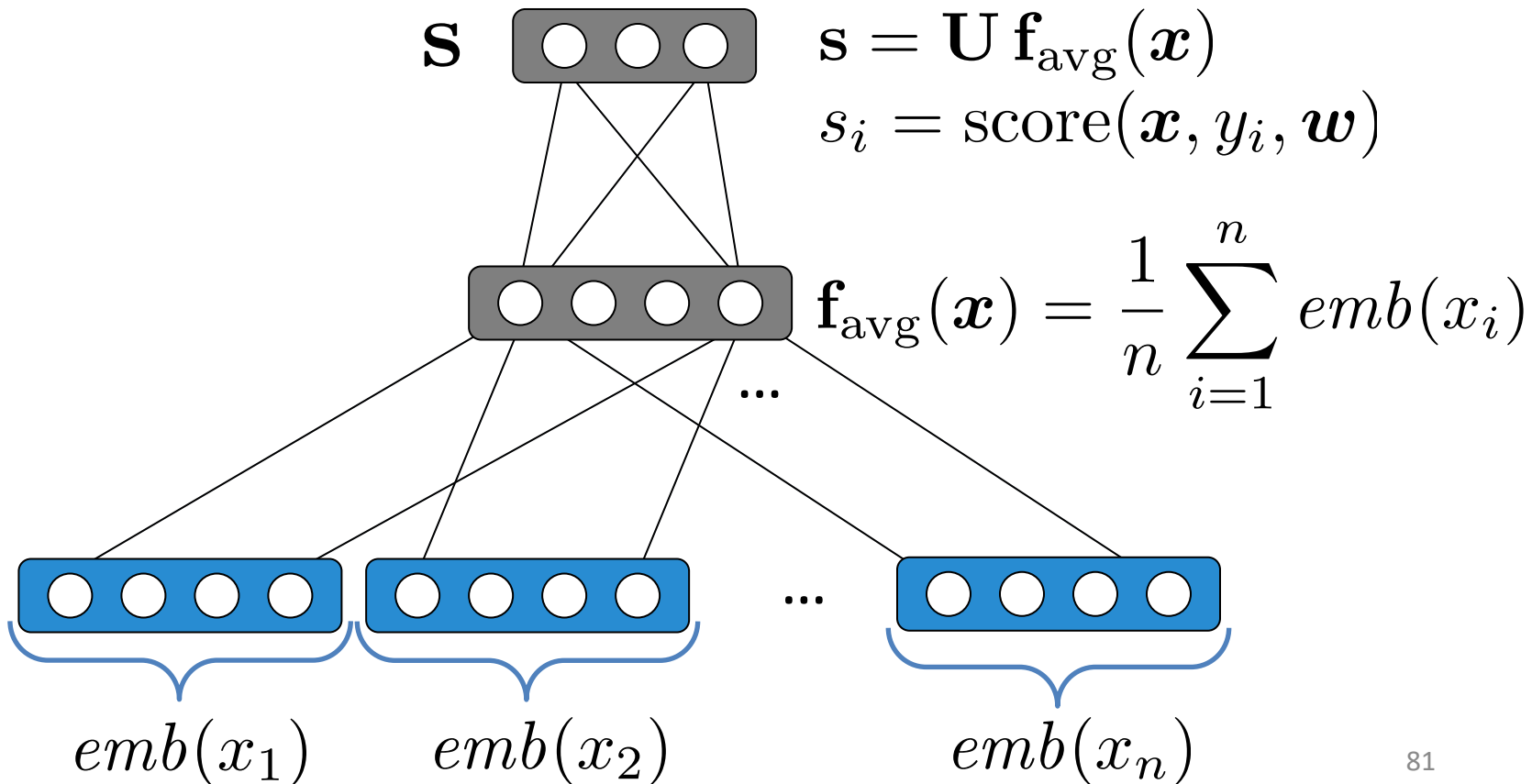
- given a word sequence  $\mathbf{x}$ , predict its label
- represent  $\mathbf{x}$  by averaging its word embeddings:





# A Simple Neural Text Classification Model

- represent  $\mathbf{x}$  by averaging its word embeddings
- output is a score vector over all possible labels:



# Encoders

- encoder: a function to represent a word sequence as a vector
- simplest: average word embeddings:

$$\mathbf{f}_{\text{avg}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \text{emb}(x_i)$$

# Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum
- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{att}}(\mathbf{x}) = \sum_{i=1}^n \underbrace{\text{att}(x_i, i, \mathbf{x})}_{\text{“attention” function, returns a scalar}} \text{emb}(x_i)$$

“attention” function,  
returns a scalar

# Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum
- e.g., for a word averaging encoder:

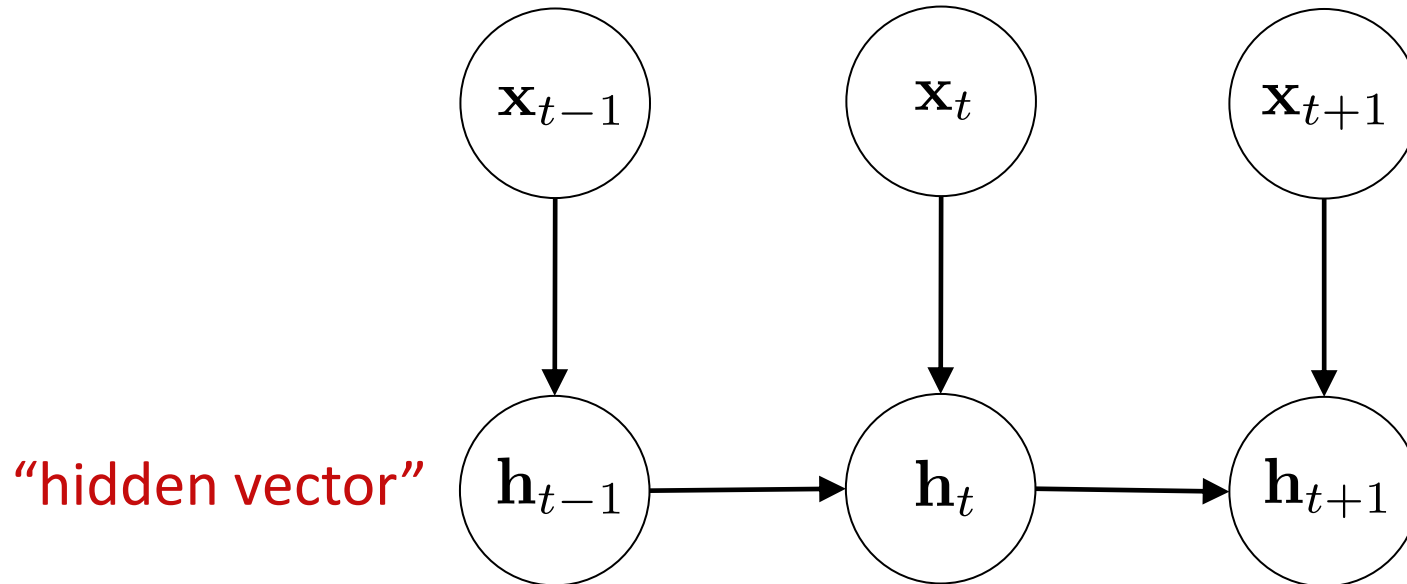
$$\mathbf{f}_{\text{att}}(\mathbf{x}) = \sum_{i=1}^n \text{att}(x_i, i, \mathbf{x}) \text{emb}(x_i)$$

$$\sum_{i=1}^n \text{att}(x_i, i, \mathbf{x}) = 1$$

- many attention functions are possible!

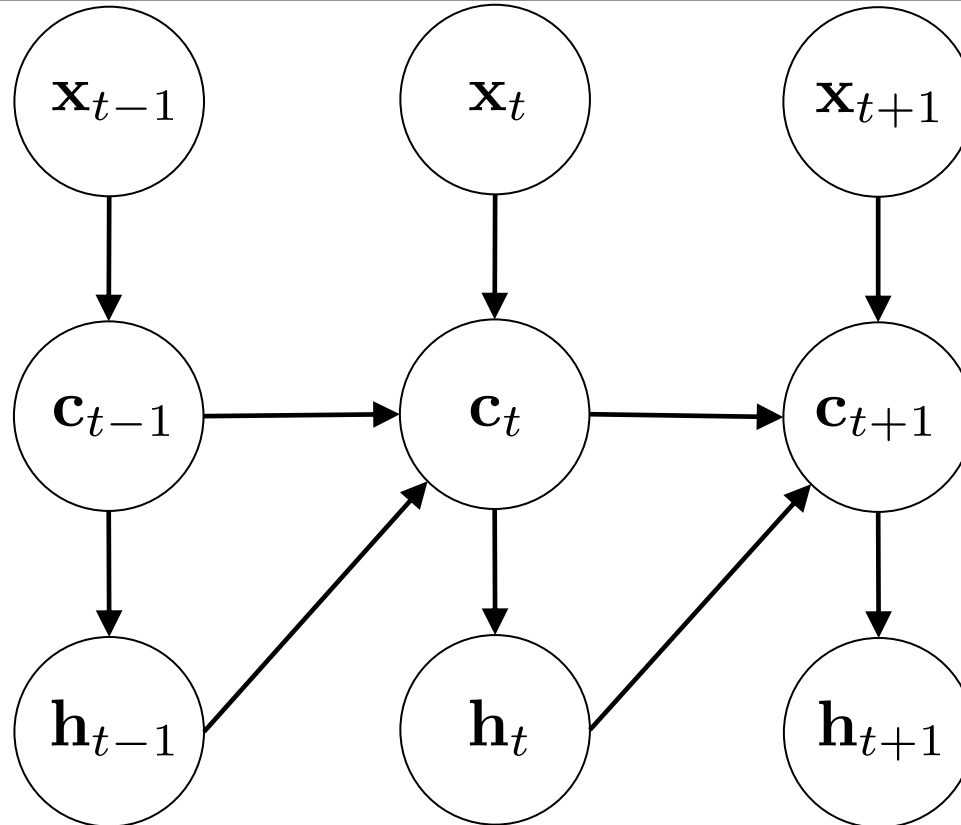
# Recurrent Neural Networks

$$\mathbf{h}_t = \tanh \left( \mathbf{W}^{(x)} \mathbf{x}_t + \mathbf{W}^{(h)} \mathbf{h}_{t-1} + \mathbf{b} \right)$$



# Long Short-Term Memory Networks (gateless)

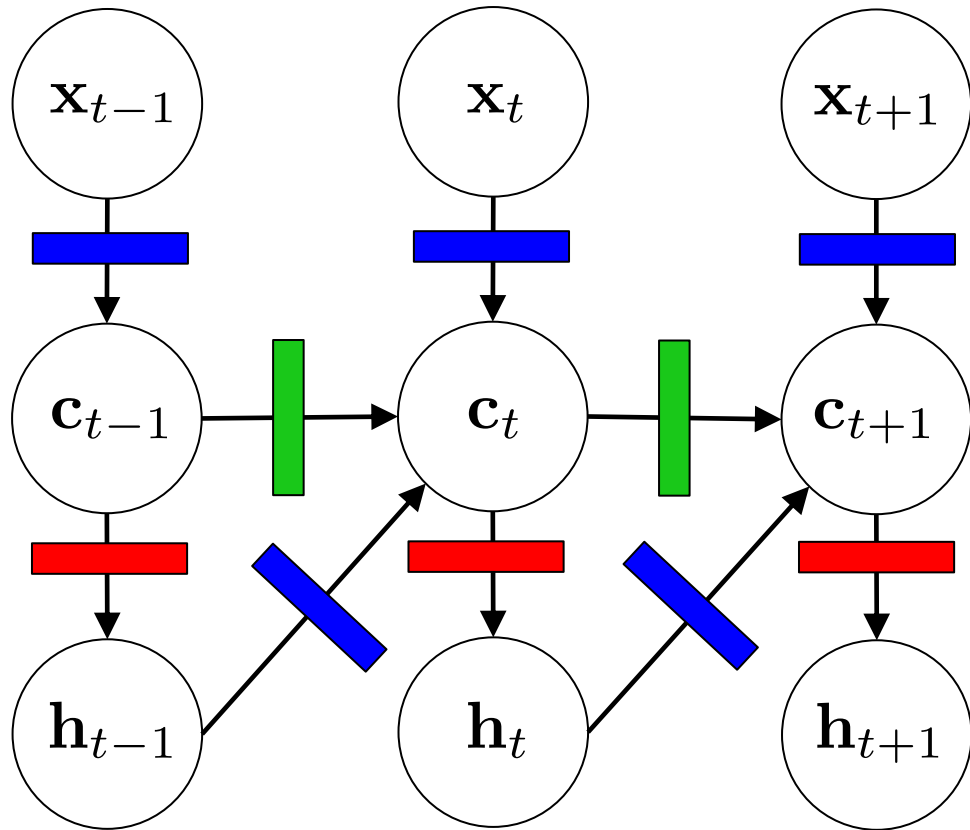
$$\mathbf{c}_t = \mathbf{c}_{t-1} + \tanh \left( \mathbf{W}^{(xc)} \mathbf{x}_t + \mathbf{W}^{(hc)} \mathbf{h}_{t-1} + \mathbf{b}^{(c)} \right)$$



$$\mathbf{h}_t = \tanh(\mathbf{c}_t)$$

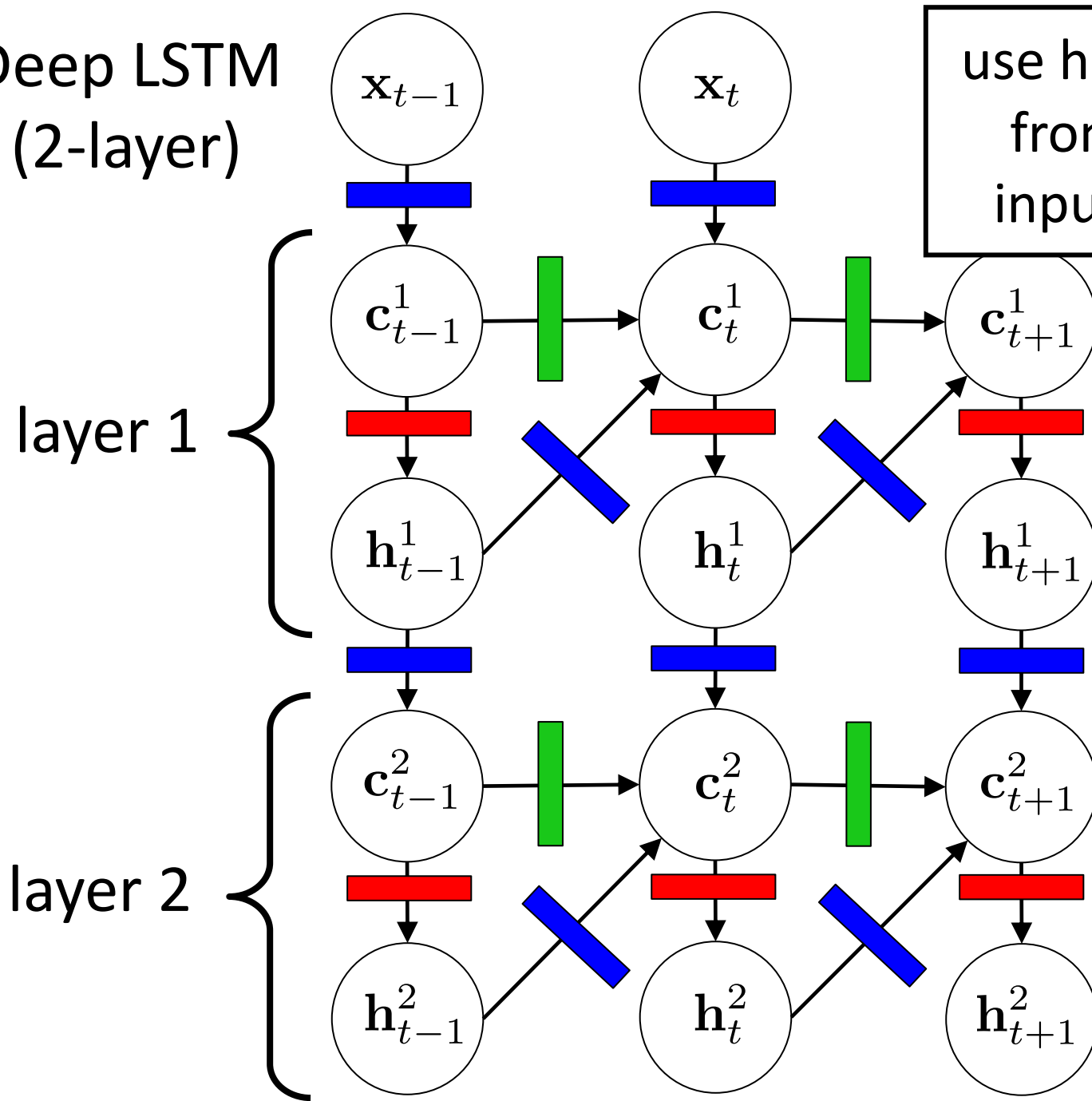
# All Gates

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh \left( \mathbf{W}^{(xc)} \mathbf{x}_t + \mathbf{W}^{(hc)} \mathbf{h}_{t-1} + \mathbf{b}^{(c)} \right)$$



$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

# Deep LSTM (2-layer)

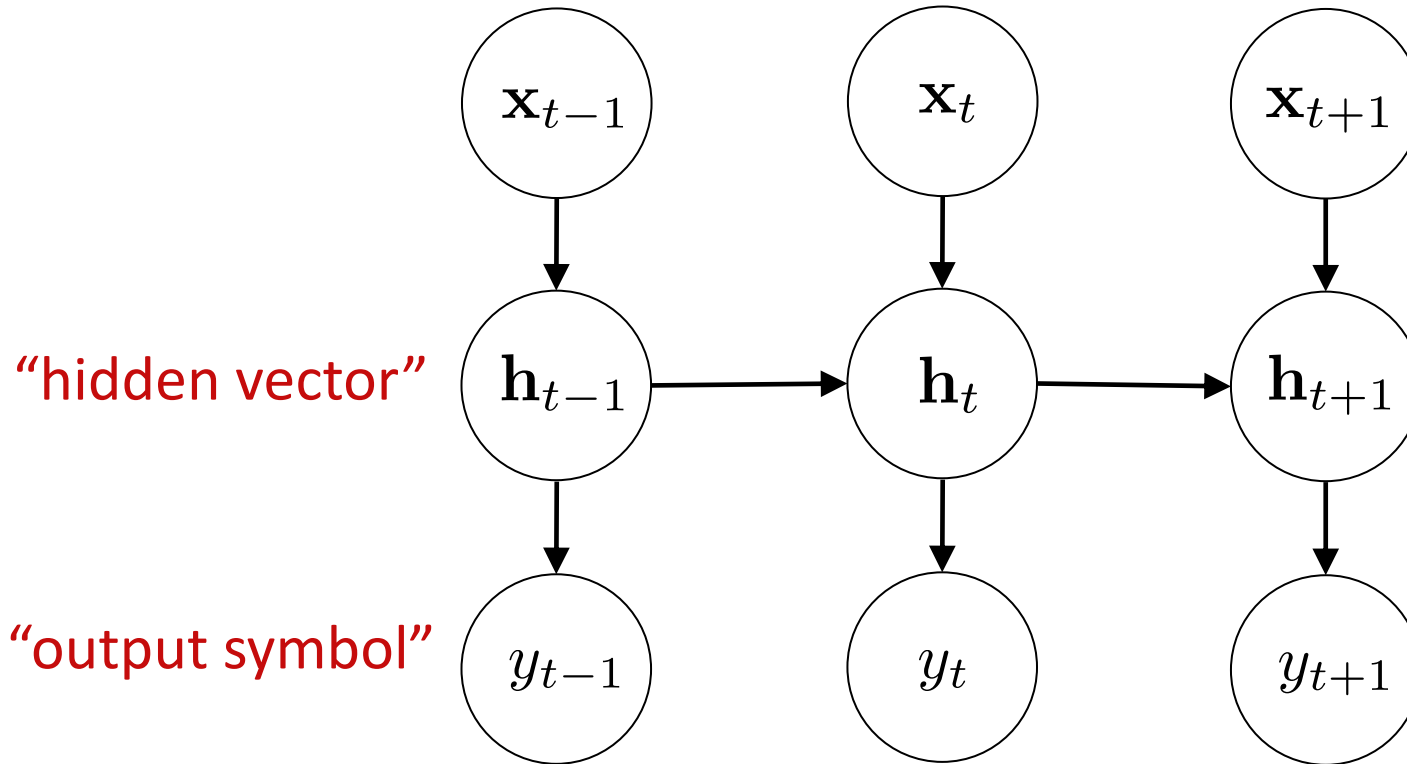


use hidden vectors from layer 1 as inputs to layer 2

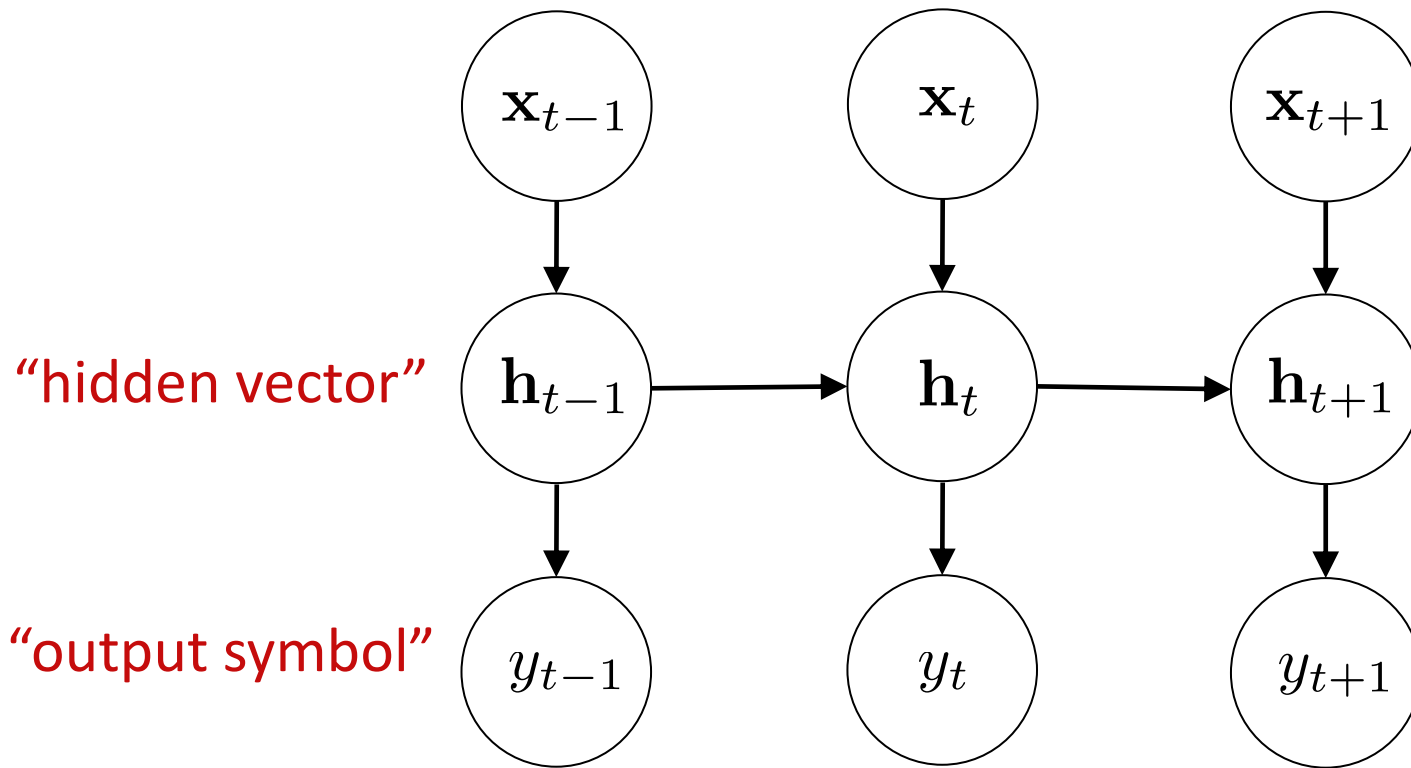


# “Output” Recurrent Neural Networks

$$\mathbf{h}_t = \tanh \left( \mathbf{W}^{(x)} \mathbf{x}_t + \mathbf{W}^{(h)} \mathbf{h}_{t-1} + \mathbf{b} \right)$$



$$y_t = \operatorname{argmax}_{y \in \mathcal{O}} \mathbf{h}_t^\top \operatorname{emb}(y)$$

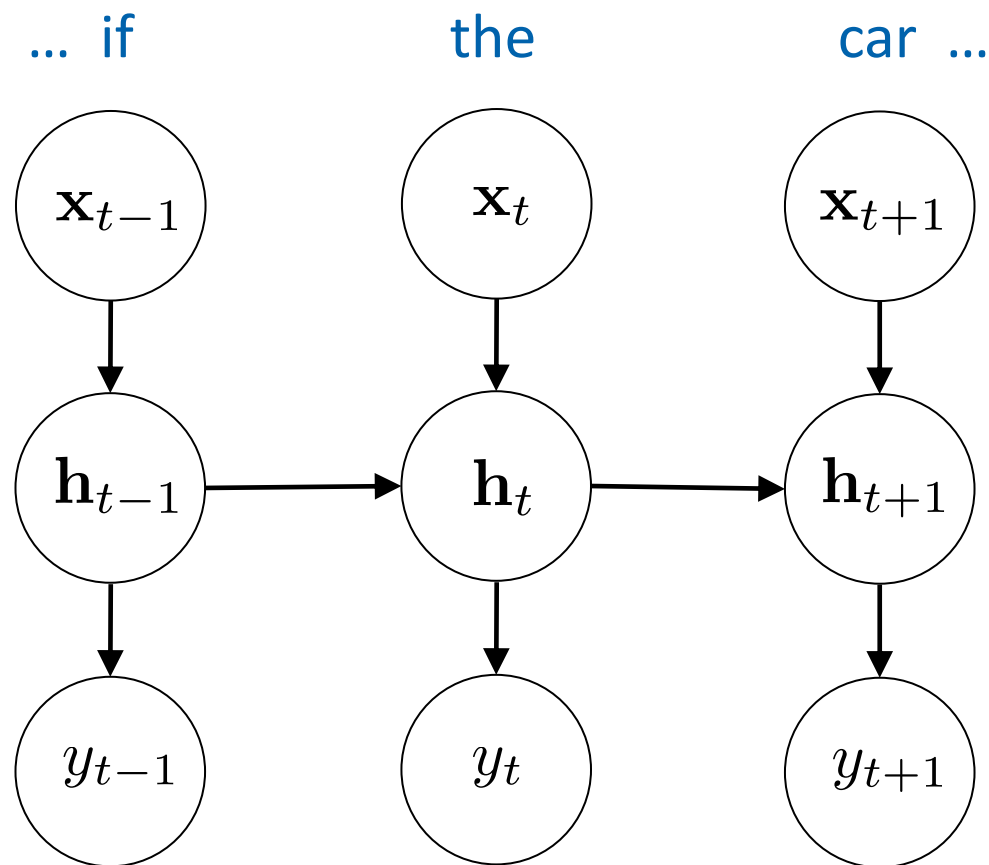


$$y_t = \operatorname{argmax}_{y \in \mathcal{O}} \mathbf{h}_t^\top \operatorname{emb}(y)$$

$$P(Y_t) = \operatorname{softmax}(\mathbf{W}\mathbf{h}_t)$$

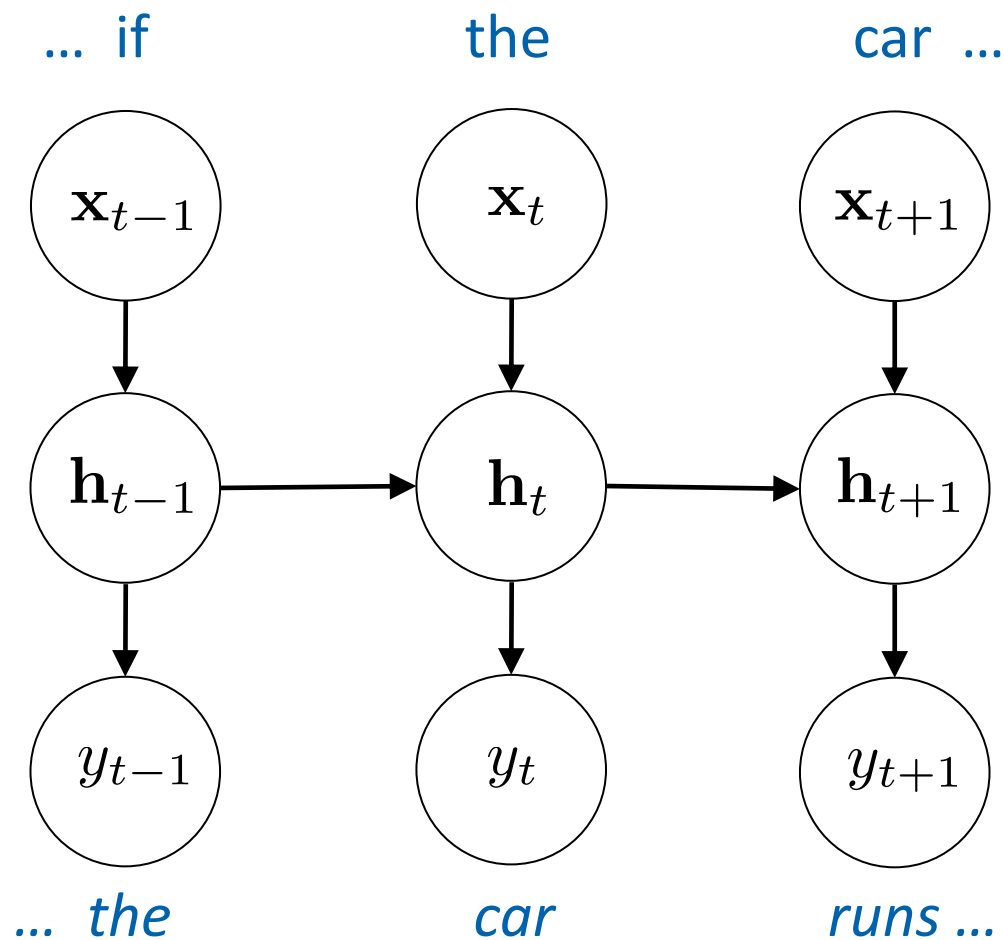
$$\mathbf{W} = [\operatorname{emb}(y_1)^\top; \operatorname{emb}(y_2)^\top; \dots; \operatorname{emb}(y_{|\mathcal{O}|})^\top]$$

# Example: Language Modeling



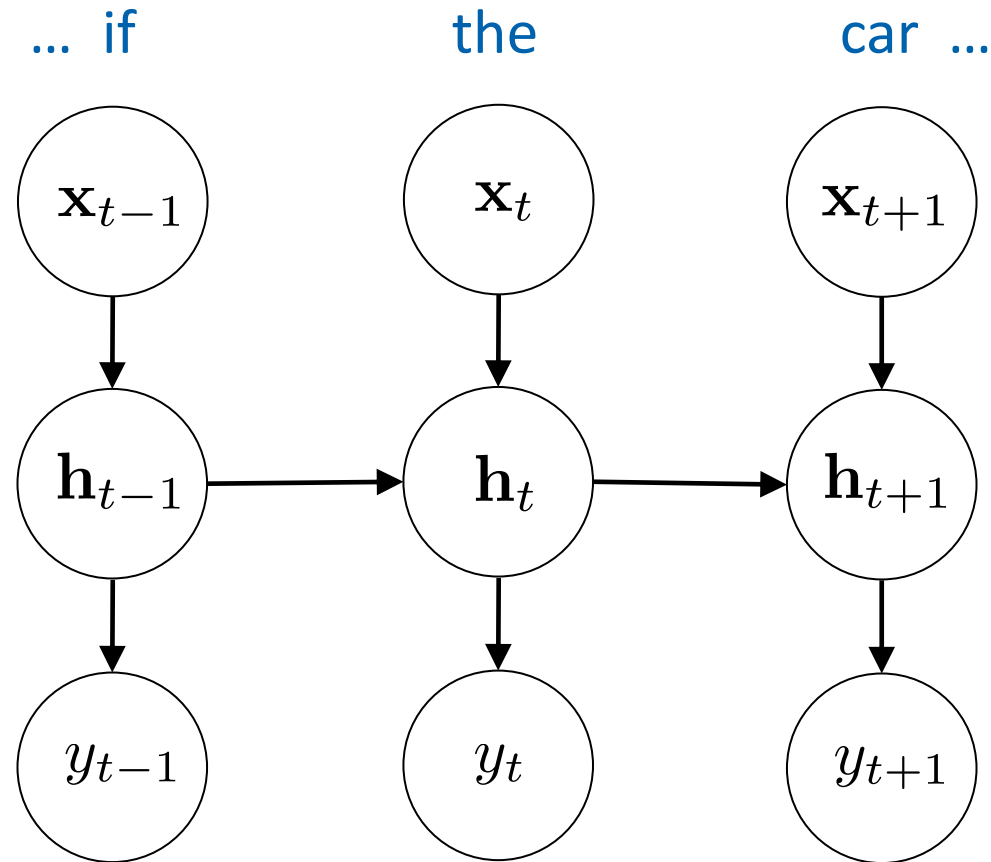
- input: a word sequence
- output?

# Example: Language Modeling



- target output at each position: next word in the sequence!


# Language Modeling: Training



$$-\log P(Y_{t-1} = \text{"the"}) - \log P(Y_t = \text{"car"}) \dots$$

# Convolution

$x = \textit{not that great}$

$$\mathbf{x} = [0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]^\top$$


vector for *not*    vector for *that*    vector for *great*


consider a single convolutional filter  $\mathbf{w} \in \mathbb{R}^d$

# Convolution

compute dot product of filter and each word vector:

$x = \textit{not that great}$

$$\mathbf{x} = \overset{\mathbf{W}}{[0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]}^\top$$


  
vector for *not*    vector for *that*    vector for *great*

$$c_1 = \mathbf{W} \cdot \mathbf{X}_{1:d}$$

# Convolution

compute dot product of filter and each word vector:

$x = \textit{not that great}$

$$\mathbf{x} = [0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]^\top$$


vector for *not*    vector for *that*    vector for *great*

$$c_1 = \mathbf{W} \cdot \mathbf{X}_{1:d}$$

$$c_2 = \mathbf{W} \cdot \mathbf{X}_{d+1:2d}$$




# Convolution

compute dot product of filter and each word vector:

$x =$  *not that great*

$$\mathbf{x} = [0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]^\top$$

  
vector for *not*    vector for *that*    vector for *great*

$$c_1 = \mathbf{W} \cdot \mathbf{X}_{1:d}$$

$$c_2 = \mathbf{W} \cdot \mathbf{X}_{d+1:2d}$$

$$c_3 = \mathbf{W} \cdot \mathbf{X}_{2d+1:3d}$$

# Convolution

$x =$  *not that great*

$$\mathbf{x} = [0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]^\top$$

vector for *not*    vector for *that*    vector for *great*

$$c_1 = \mathbf{W} \cdot \mathbf{X}_{1:d}$$

$$c_2 = \mathbf{W} \cdot \mathbf{X}_{d+1:2d}$$

$$c_3 = \mathbf{W} \cdot \mathbf{X}_{2d+1:3d}$$

Note: it's common to add a bias  $b$  and use a nonlinearity  $g$ :

$$c_1 = g(\mathbf{w} \cdot \mathbf{x}_{1:d} + b)$$

# Convolution

$x =$  *not that great*

$$\mathbf{x} = [0.4 \dots 0.9 \quad 0.2 \dots 0.7 \quad 0.3 \dots 0.6]^\top$$

vector for *not*    vector for *that*    vector for *great*

$$c_1 = \mathbf{W} \cdot \mathbf{X}_{1:d}$$

$$c_2 = \mathbf{W} \cdot \mathbf{X}_{d+1:2d}$$

$$c_3 = \mathbf{W} \cdot \mathbf{X}_{2d+1:3d}$$

$\mathbf{c}$  = “feature map” for this filter,  
has an entry for each position in input (in this case, 3 entries)

# Sequence Labeling Tasks in NLP

## Part-of-Speech Tagging

determiner    verb (past)    prep.    proper noun    proper noun    poss.    adj.    noun  
Some    questioned    if    Tim    Cook    's    first    product

modal    verb    det.    adjective    noun    prep.    proper noun    punc.  
would    be    a    breakaway    hit    for    Apple    .

## Named Entity Recognition

O                    O                    O    B-PERSON    I-PERSON    O                    O                    O  
Some    questioned    if    Tim    Cook    's    first    product

O                    O                    O                    O                    O                    O    B-ORGANIZATION    O  
would    be    a    breakaway    hit    for    Apple    .

# Part-of-Speech

- **open-class:**
  - nouns, verbs, adjectives, adverbs
  - “open” because new words in these categories are often created
- **closed-class:**
  - function words like determiners and prepositions
  - new function words rarely catch on
  - (though new forms/variants of function words do appear, especially in “conversational text”)

# POS Ambiguity

<b>Types:</b>		<b>WSJ</b>	<b>Brown</b>
<b>Unambiguous</b>	(1 tag)	44,432 ( <b>86%</b> )	45,799 ( <b>85%</b> )
<b>Ambiguous</b>	(2+ tags)	7,025 ( <b>14%</b> )	8,050 ( <b>15%</b> )
<b>Tokens:</b>			
<b>Unambiguous</b>	(1 tag)	577,421 ( <b>45%</b> )	384,349 ( <b>33%</b> )
<b>Ambiguous</b>	(2+ tags)	711,780 ( <b>55%</b> )	786,646 ( <b>67%</b> )

**Figure 10.2** The amount of tag ambiguity for word types in the Brown and WSJ corpora, from the Treebank-3 (45-tag) tagging. These statistics include punctuation as words, and assume words are kept in their original case.

- most word types have only one tag
  - frequent word types have more tags
  - rare words are often nouns or verbs

# Universal Tag Set

- contains 12 tags:
  - noun, verb, adjective, adverb, pronoun, determiner, adposition, numeral, conjunction, particle, punctuation, other

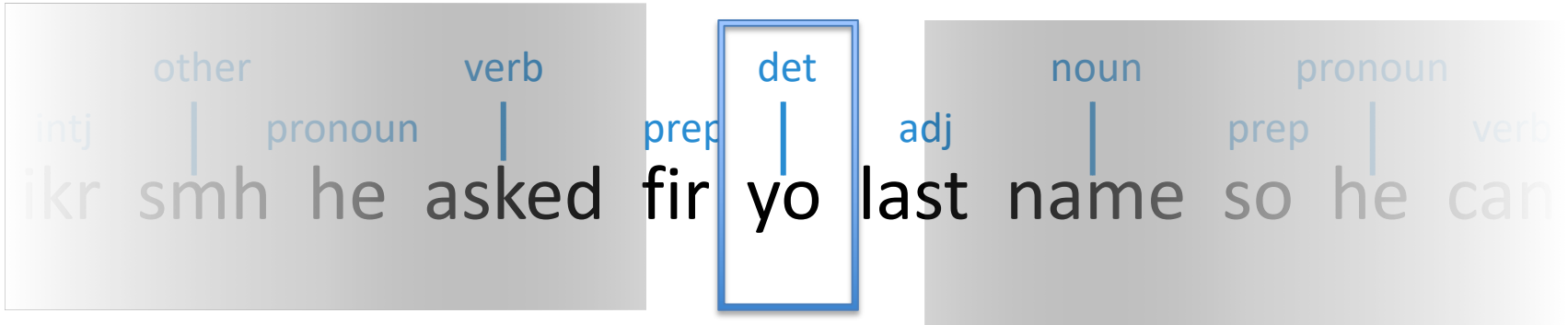
---

sentence:	The	oboist	Heinz	Holliger	has	taken	a	hard	line	about	the	problems	.
original:	DT	NN	NNP	NNP	VBZ	VRB	DT	JJ	NN	IN	DT	NNS	.
universal:	DET	NOUN	NOUN	NOUN	VERB	VERB	DET	ADJ	NOUN	ADP	DET	NOUN	.

Figure 1: Example English sentence with its language specific and corresponding universal POS tags.

*Petrov, Das, McDonald (2011)*

# Feed-Forward Neural Networks for Twitter POS Tagging

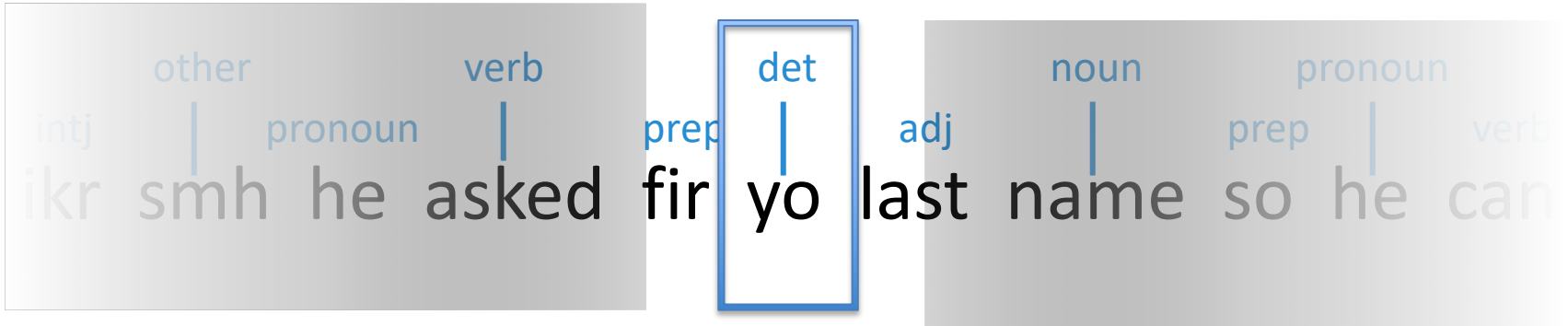


- e.g., predict tag of *yo* given context
- what should the input  $\mathbf{x}$  be?

$$\mathbf{x} = \underbrace{[0.4 \ 0.1 \ \dots \ 0.9]}_{\text{word vector for } yo}^T$$



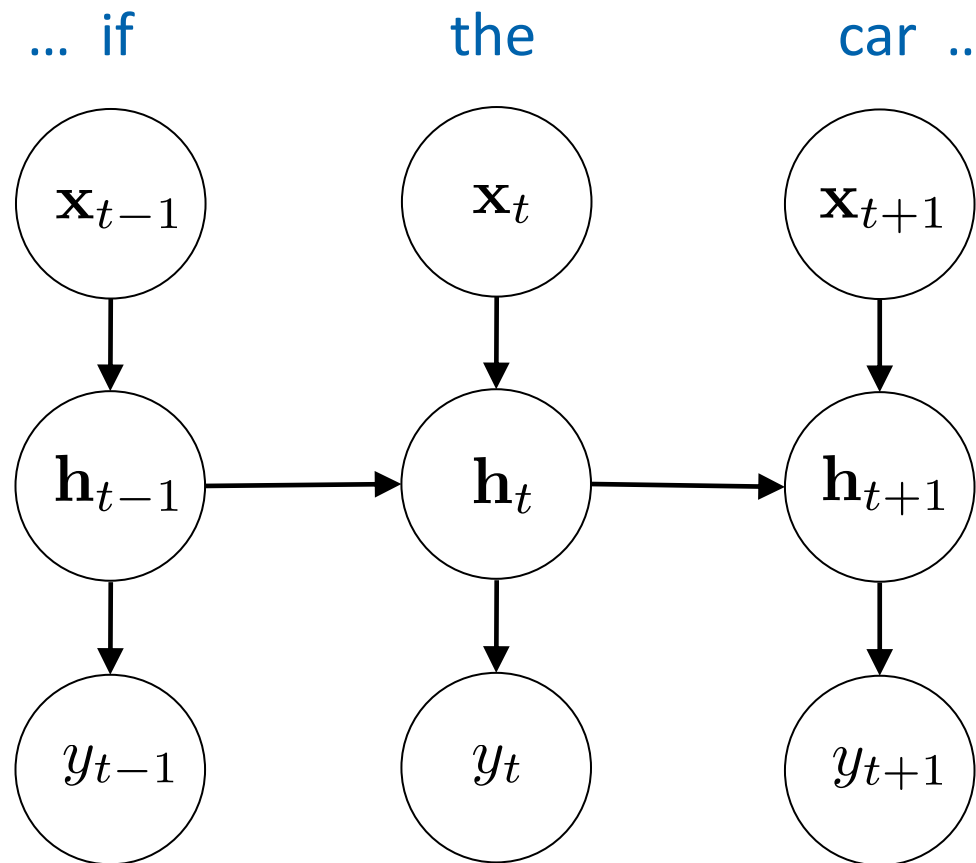
# Feed-Forward Neural Networks for Twitter POS Tagging



- when using word vectors as part of input, we can also treat them as more parameters to be learned!
- this is called “updating” or “fine-tuning” the vectors (since they are initialized using something like `word2vec`)

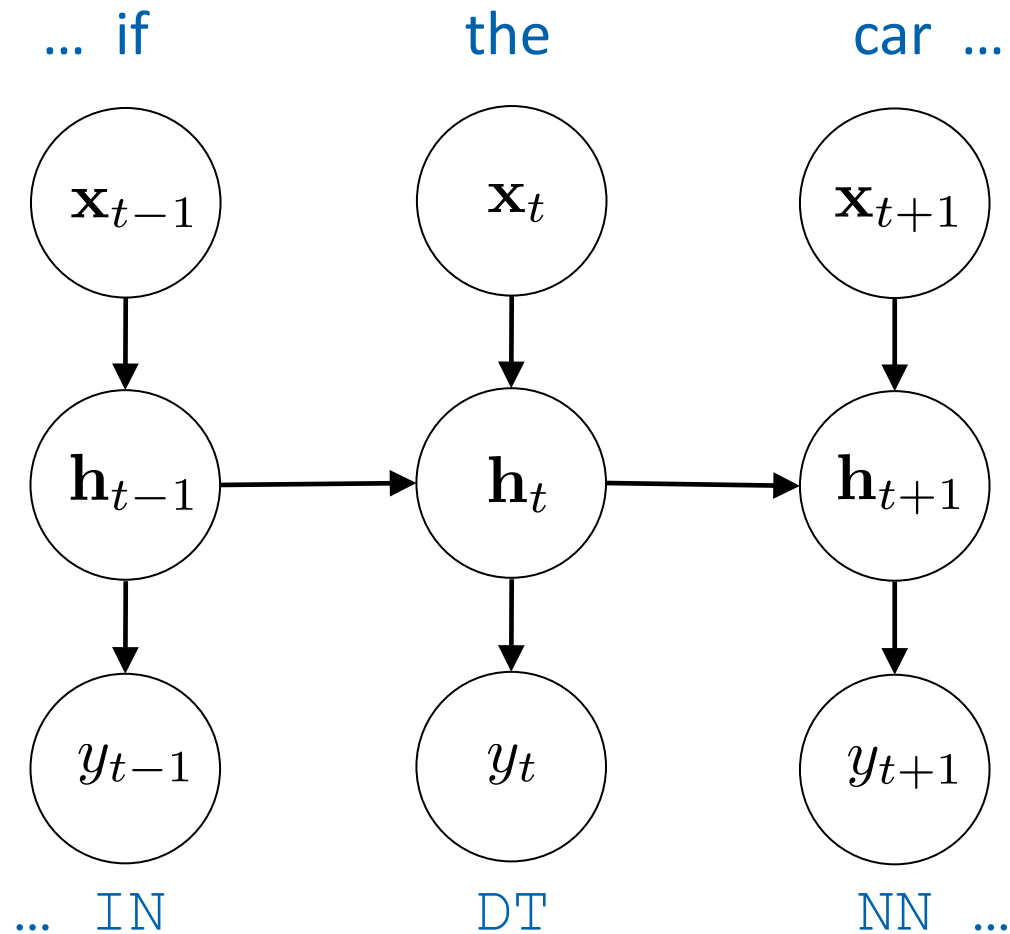
$$\mathbf{x} = \underbrace{[-0.2 \ 0.5 \ \dots \ 0.8]}_{\text{word vector for } \textit{fir}} \underbrace{[0.4 \ 0.1 \ \dots \ 0.9]}_{\text{word vector for } \textit{yo}}^{\top}$$

# RNNs for Part-of-Speech Tagging



- input: a word sequence

# RNNs for Part-of-Speech Tagging

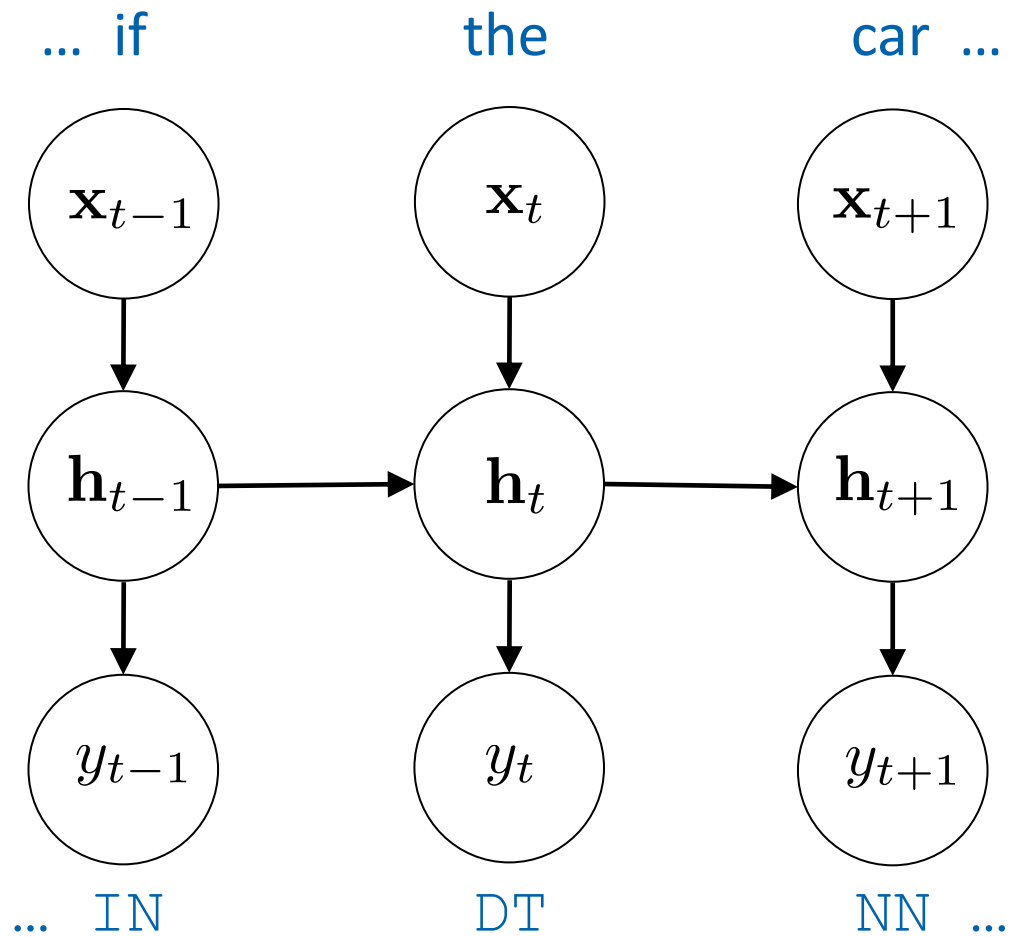


- target output at each position: POS tag for corresponding word

# Feed-Forward Networks for POS Tagging

- feed-forward networks are OK for tagging
- they tend to work best with very small contexts (e.g., 1 word to left & right)
- can also use convolutional networks defined on a window centered on the target word

# RNNs for Part-of-Speech Tagging



# RNN Taggers

- RNN POS taggers are simple and effective
- most common is to use some sort of bidirectional RNN, like a BiLSTM or BiGRU

# Modeling, Inference, Learning in Structured Prediction

**inference: solve**  $\operatorname{argmax}$

**modeling: define** score **function**

$$\operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname{argmax}_{\boldsymbol{y}} \operatorname{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w})$$

**learning: choose**  $\boldsymbol{w}$

# Applications of our Classifier Framework so far

task	input ( $x$ )	output ( $y$ )	output space ( $\mathcal{L}$ )	size of $\mathcal{L}$
text classification	a sentence	gold standard label for $x$	pre-defined, small label set (e.g., {positive, negative})	2-10
word sense disambiguation	instance of a particular word (e.g., <i>bass</i> ) with its context	gold standard word sense of $x$	pre-defined sense inventory from WordNet for <i>bass</i>	2-30
learning skip-gram word embeddings	instance of a word in a corpus	a word in the context of $x$ in a corpus	vocabulary	$ V $
part-of-speech tagging	a sentence	gold standard part-of-speech tags for $x$	all possible part-of-speech tag sequences with same length as $x$	$ P ^{ x }$



# Applications of our Classifier Framework so far

task	input ( $x$ )	output ( $y$ )	output space ( $\mathcal{L}$ )	size of $\mathcal{L}$
text classification	a sentence	gold standard label for $x$	pre-defined, small label set (e.g., {positive, negative})	2-10
word sense disambiguation	instance of a particular word (e.g., <i>bass</i> in its context)	gold standard	pre-defined sense inventory from	2-30
learning skip-gram word embeddings	instance of a word in a context	gold standard	pre-defined word set	2-10,000
part-of-speech tagging	a sentence	gold standard part-of-speech tags for $x$	all possible part-of-speech tag sequences with same length as $x$	$ P ^{ x }$

exponential in size of input!  
 “structured prediction”

$$|P|^{|x|}$$

# Inference for Structured Prediction

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{\mathbf{y}}{\text{argmax}} \text{ score}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

- how do we efficiently search over the space of all structured outputs?
- this space may have size exponential in the size of the input, or be unbounded

# Feature Locality

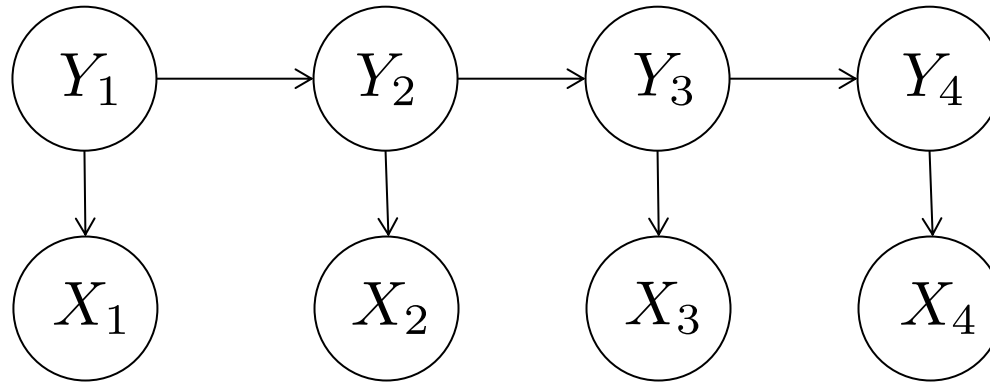
- **feature locality**: how “big” are your features?
- we need to be mindful of this to enable efficient inference
- features can be arbitrarily big in terms of the *input*
- but features **cannot** be arbitrarily big in terms of the *output*!

# Hidden Markov Models

- $n$ -gram language models define a probability distribution over word sequences  $\mathbf{x}$
- HMMs define a joint probability distribution over input sequences  $\mathbf{x}$  and output sequences  $\mathbf{y}$
- conditional independence assumptions (“Markov assumption”) are used to factorize this joint distribution into small terms

# Graphical Model for an HMM

(for a sequence of length 4)



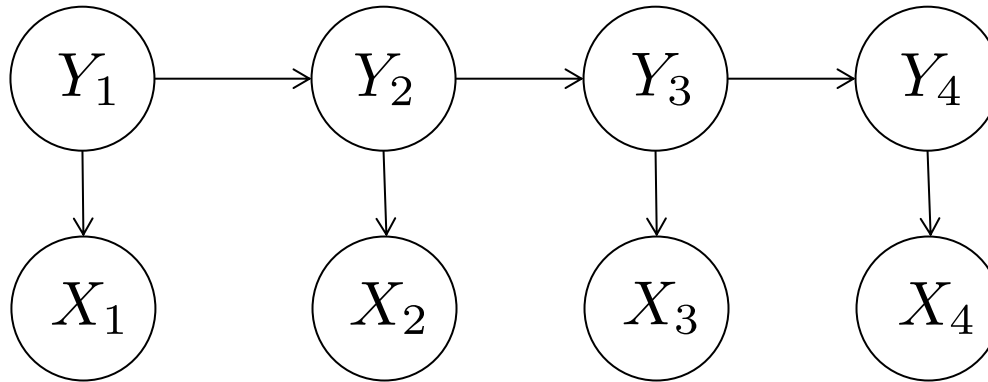
conditional independence statements among random variables are encoded by the edge structure:

$$Y_{t-1} \perp Y_{t+1} \mid Y_t$$

$$X_t \perp Y_{t-1} \mid Y_t$$

# Graphical Model for an HMM

(for a sequence of length 4)



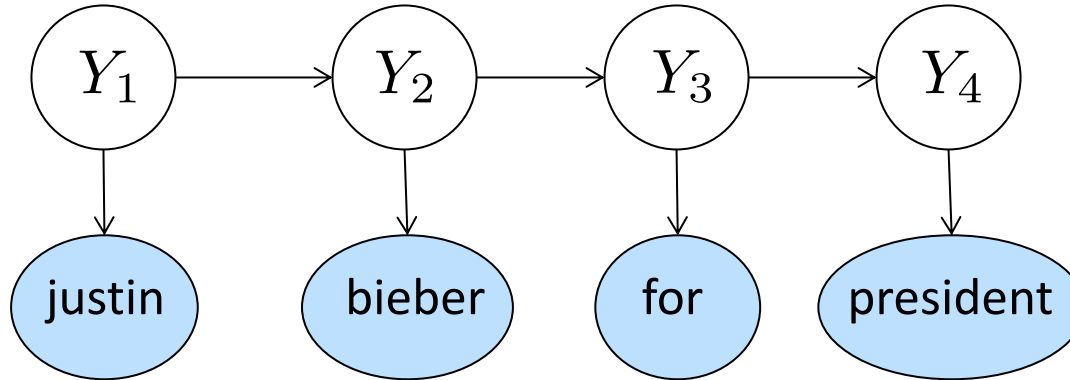
conditional independence statements encoded by edge structure → we only have to worry about **local distributions**:

**transition** parameters:  $p_{\tau}(y_i \mid y_{i-1})$

**emission** parameters:  $p_{\eta}(x_i \mid y_i)$

# HMMs for Word Clustering

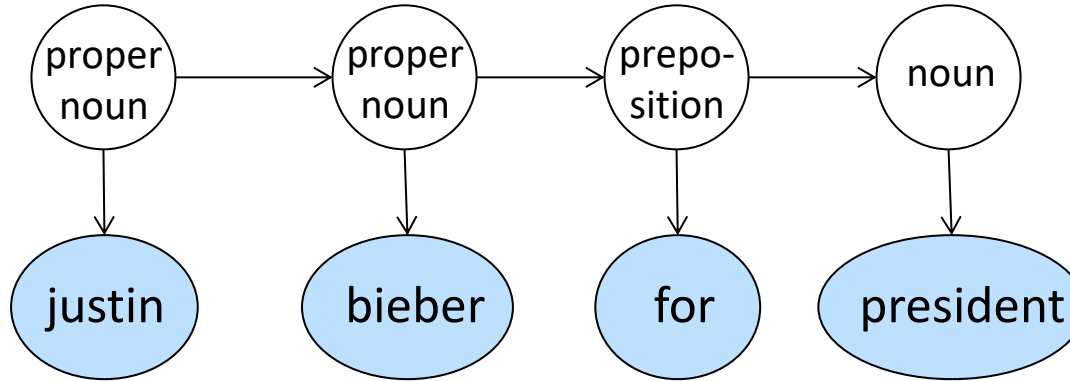
(Brown et al., 1992)



each  $y_i \in \mathcal{L}$  is a cluster ID

so, label space is  $\mathcal{L} = \{1, 2, \dots, 100\}$

# HMMs for Part-of-Speech Tagging



each  $y_i \in \mathcal{L}$  is a part-of-speech tag  
so, label space is  $\mathcal{L} = \{\text{noun, verb, ...}\}$

what parameters need to be learned?

transition parameters:  $p_{\tau}(y_i | y_{i-1})$

emission parameters:  $p_{\eta}(x_i | y_i)$



# Sample Question

Suppose you wanted to use an HMM as a language model, i.e., as a way to compute the probability of a word sequence. How would you do this?

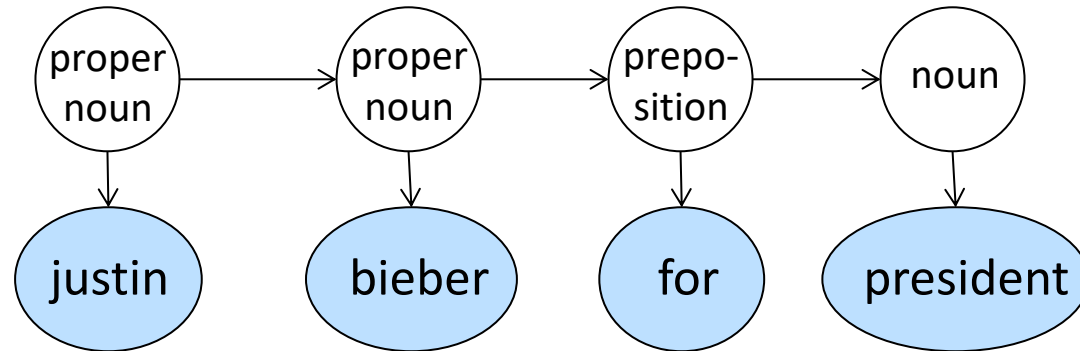
# Sample Question

Suppose you wanted to use an HMM as a language model, i.e., as a way to compute the probability of a word sequence. How would you do this?

marginalize out (sum over) the hidden variables ( $\mathbf{y}$ )

# Supervised HMMs

- given a dataset of input sequences and annotated outputs:



- to estimate transition/emission distributions, use maximum likelihood estimation (count and normalize):

$$p_{\tau}(y | y') \leftarrow \frac{\text{count}(y' y)}{\text{count}(y')} \qquad p_{\eta}(x | y) \leftarrow \frac{\text{count}(y, x)}{\text{count}(y)}$$

$$p_{\tau}(\text{verb} | \text{noun}) \leftarrow \frac{\text{count}(\text{noun verb})}{\text{count}(\text{noun})} \qquad p_{\eta}(\text{walk} | \text{verb}) \leftarrow \frac{\text{count}(\text{verb, walk})}{\text{count}(\text{verb})}$$

# Inference in HMMs

- since the output is a sequence, this argmax requires iterating over an exponentially-large set
- we can use **dynamic programming (DP)** to solve these problems exactly
- for HMMs (and other sequence models), the algorithm for solving this is the **Viterbi algorithm**

# Viterbi Algorithm

- recursive equations + memoization:

## base case:

returns probability of sequence starting with label  $y$  for first word



$$V(1, y) = p_{\eta}(x_1 | y) p_{\tau}(y | \langle s \rangle)$$

$$V(m, y) = \max_{y' \in \mathcal{L}} ( p_{\eta}(x_m | y) p_{\tau}(y | y') V(m - 1, y') )$$



## recursive case:

computes probability of max-probability label sequence that ends with label  $y$  at position  $m$

# Linear Sequence Models

- so, an HMM is:
  - a linear sequence model
  - with particular features on label transitions and label-observation emissions
  - and uses maximum likelihood estimation (count & normalize) for learning
- but we could use any feature functions we like, and use any of our loss functions for learning!

# (Chain) Conditional Random Fields

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{\mathbf{y}}{\text{argmax}} \text{score}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

$$\text{score}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_i w_i f_i(\mathbf{x}, \mathbf{y})$$

- linear sequence model
- arbitrary features of input are permitted
- test-time inference uses Viterbi Algorithm
- learning done by minimizing log loss (DP algorithms used to compute gradients)

# Maximum-Margin Markov Networks

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{\mathbf{y}}{\text{argmax}} \text{score}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

$$\text{score}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_i w_i f_i(\mathbf{x}, \mathbf{y})$$

- linear sequence model
- arbitrary features of input are permitted
- test-time inference uses Viterbi Algorithm
- learning done by minimizing **hinge** loss (DP algorithm used to compute **subgradients**)