COS 484: Natural Language Processing

Midterm Review

Fall 2019

Announcements

- **Midterm exam**: this Thursday, Oct 24th 1:30-2:45 (75 minutes)
  - Everyone is seated in COS 104 (alternating seats). Please arrive 10 minutes early!
  - One single-sided cheatsheet is allowed
  - No phone/laptop, no calculator or any internet access
  - If you have an exam (e.g. COS 429) at 3pm, you have *an option to start at 1pm*.

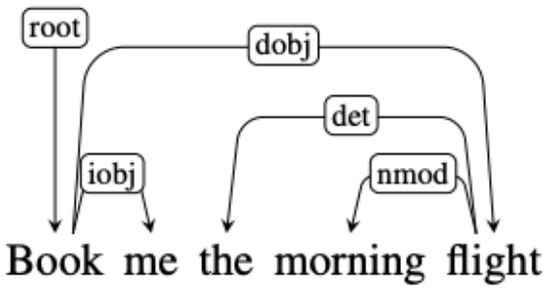
Announcements

- **Assignment 2** grades were out
  - No separate grades for the code submission

- **Assignment 3** due on Friday 11:59pm
- **Assignment 4** will be out on Friday too

Today's Plan

- Dependency parsing (10 mins)
- Midterm review (65 minutes)

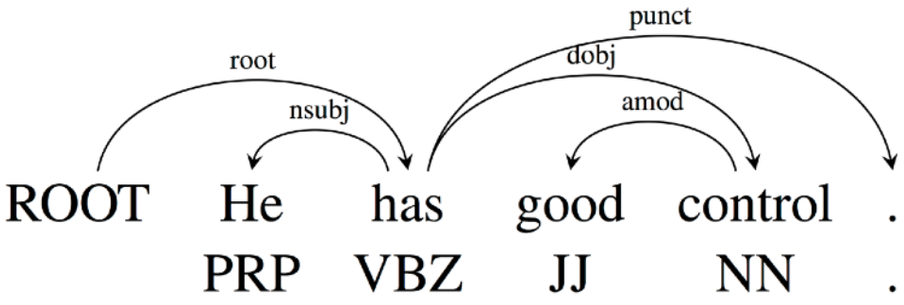


"Book me the morning flight"

Dependency parsing

	stack	buffer	action	added arc
0	[ROOT]	[Book, me, the, morning, flight]	SHIFT	
1	[ROOT, Book]	[me, the, morning, flight]	SHIFT	
2	[ROOT, Book, me]	[the, morning, flight]	RIGHT-ARC(iobj)	(Book, iobj, me)
3	[ROOT, Book]	[the, morning, flight]	SHIFT	
4	[ROOT, Book, the]	[morning, flight]	SHIFT	
5	[ROOT, Book, the, morning]	[flight]	SHIFT	
6	[ROOT, Book, the,morning,flight]	[]	LEFT-ARC(nmod)	(flight,nmod,morning)
7	[ROOT, Book, the, flight]	[]	LEFT-ARC(det)	(flight,det,the)
8	[ROOT, Book, flight]	[]	RIGHT-ARC(dobj)	(Book,dobj,flight)
9	[ROOT, Book]	[]	RIGHT-ARC(root)	(ROOT,root,Book)
10	[ROOT]	[]		

MaltParser



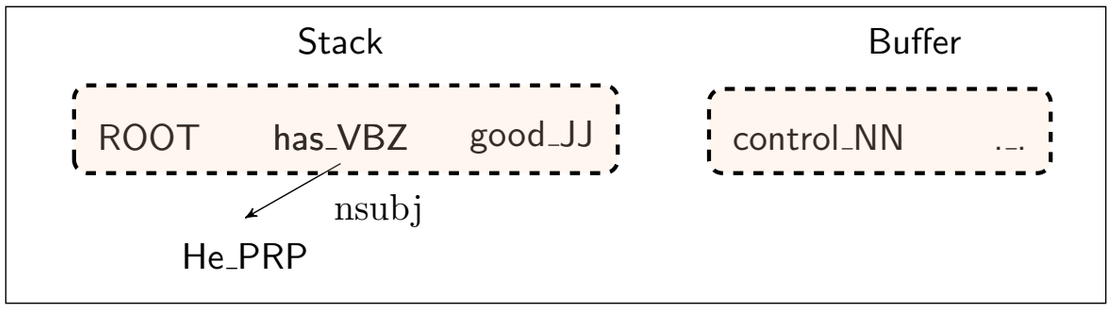
- Extract features from the configuration
- Use your favorite classifier: logistic regression, SVM…

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

w: word, t: part-of-speech tag

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

MaltParser



Feature templates

$s_2 . w \circ s_2 . t$

$s_1 . w \circ s_1 . t \circ b_1 . w$

$lc(s_2) . t \circ s_2 . t \circ s_1 . t$

$lc(s_2) . w \circ lc(s_2) . l \circ s_2 . w$

Features

$s_2 . w = \text{has} \circ s_2 . t = \text{VBZ}$

$s_1 . w = \text{good} \circ s_1 . t = \text{JJ} \circ b_1 . w = \text{control}$

$lc(s_2) . t = \text{PRP} \circ s_2 . t = \text{VBZ} \circ s_1 . t = \text{JJ}$

$lc(s_2) . w = \text{He} \circ lc(s_2) . l = \text{nsubj} \circ s_2 . w = \text{has}$

Usually a combination of 1-3 elements from the configuration

Binary, sparse, millions of features

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing

More feature templates

```
# From Single Words
pair { stack.tag stack.word }
stack { word tag }
pair { input.tag input.word }
input { word tag }
pair { input(1).tag input(1).word }
input(1) { word tag }
pair { input(2).tag input(2).word }
input(2) { word tag }

# From word pairs
quad { stack.tag stack.word input.tag input.word }
triple { stack.tag stack.word input.word }
triple { stack.word input.tag input.word }
triple { stack.tag stack.word input.tag }
triple { stack.tag input.tag input.word }
pair { stack.word input.word }
pair { stack.tag input.tag }
pair { input.tag input(1).tag }

# From word triples
triple { input.tag input(1).tag input(2).tag }
triple { stack.tag input.tag input(1).tag }
triple { stack.head(1).tag stack.tag input.tag }
triple { stack.tag stack.child(-1).tag input.tag }
triple { stack.tag stack.child(1).tag input.tag }
triple { stack.tag input.tag input.child(-1).tag }

# Distance
pair { stack.distance stack.word }
pair { stack.distance stack.tag }
pair { stack.distance input.word }
pair { stack.distance input.tag }
triple { stack.distance stack.word input.word }
triple { stack.distance stack.tag input.tag }
```

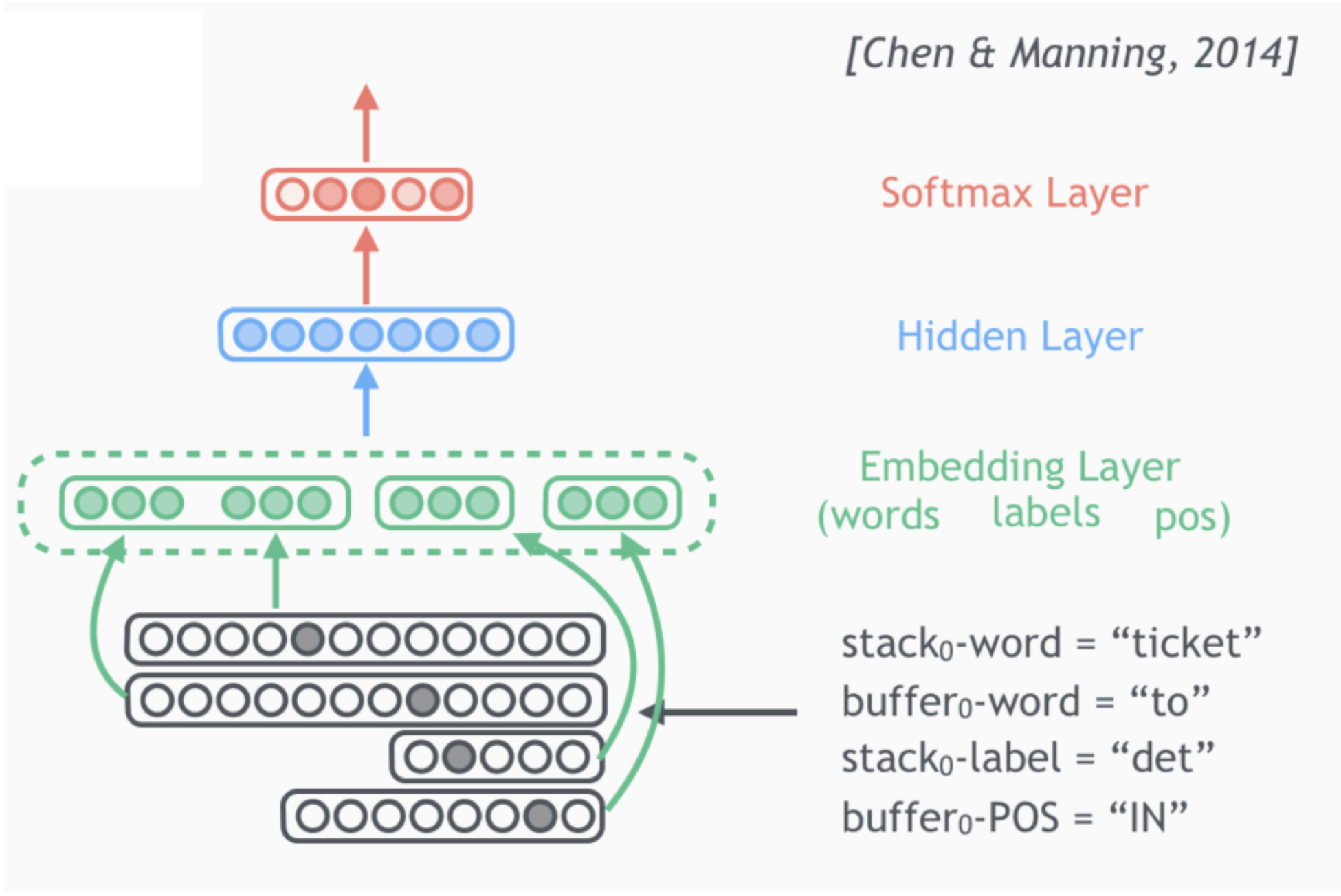
```
# valency
pair { stack.word stack.valence(-1) }
pair { stack.word stack.valence(1) }
pair { stack.tag stack.valence(-1) }
pair { stack.tag stack.valence(1) }
pair { input.word input.valence(-1) }
pair { input.tag input.valence(-1) }

# unigrams
stack.head(1) {word tag}
stack.label
stack.child(-1) {word tag label}
stack.child(1) {word tag label}
input.child(-1) {word tag label}

# third order
stack.head(1).head(1) {word tag}
stack.head(1).label
stack.child(-1).sibling(1) {word tag label}
stack.child(1).sibling(-1) {word tag label}
input.child(-1).sibling(1) {word tag label}
triple { stack.tag stack.child(-1).tag stack.child(-1).sibling(1)
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1).
triple { stack.tag stack.head(1).tag stack.head(1).head(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(1)

# label set
pair { stack.tag stack.child(-1).label }
triple { stack.tag stack.child(-1).label stack.child(-1).sibling(
quad { stack.tag stack.child(-1).label stack.child(-1).sibling(1)
pair { stack.tag stack.child(1).label }
triple { stack.tag stack.child(1).label stack.child(1).sibling(-1
quad { stack.tag stack.child(1).label stack.child(1).sibling(-1).
pair { input.tag input.child(-1).label }
triple { input.tag input.child(-1).label input.child(-1).sibling(
quad { input.tag input.child(-1).label input.child(-1).sibling(1)
```

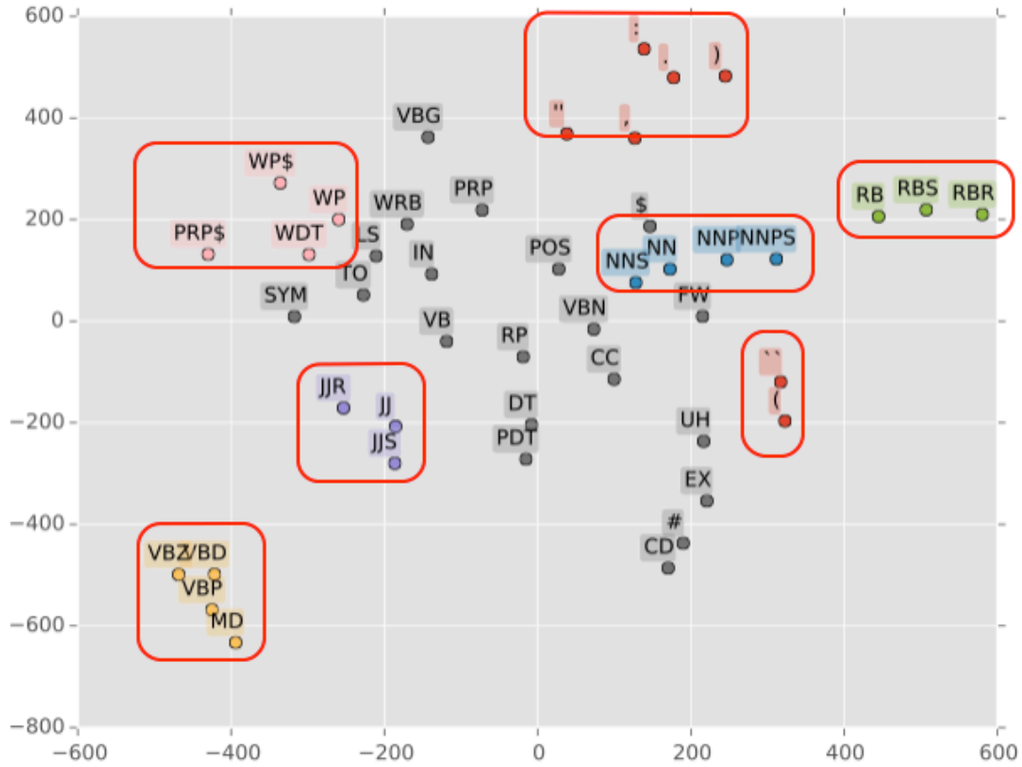
Parsing with neural networks



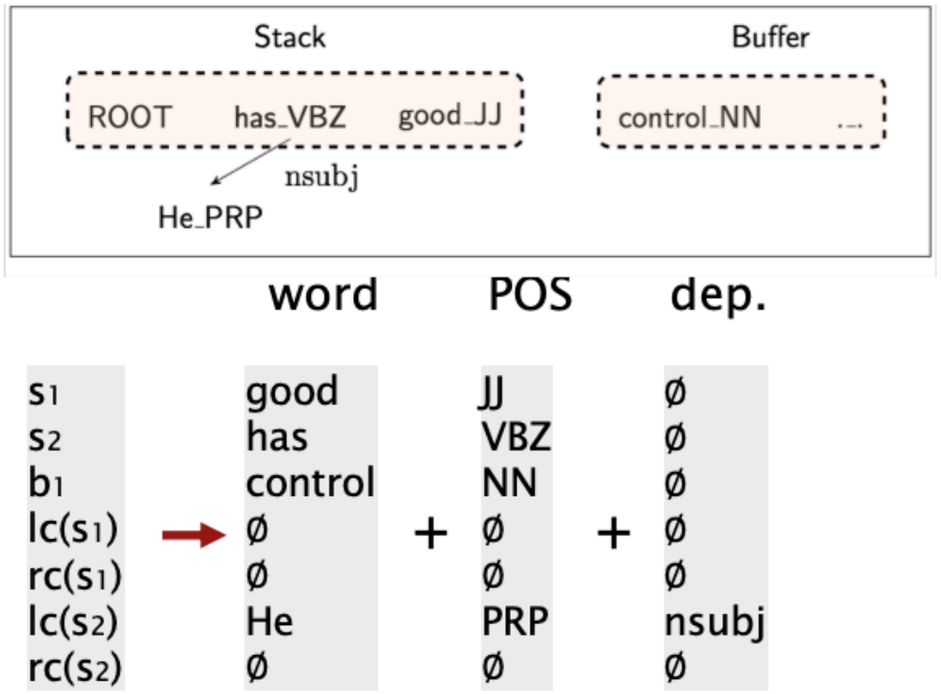
(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks

Parsing with neural networks

- Used pre-trained word embeddings
- Part-of-speech tags and dependency labels are also represented as vectors



- Eliminated feature templates!



- A simple feedforward NN — what is left is backpropagation!

(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks

Further improvements

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global normalization

Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

Google's SyntaxNet and the Parsey McParseFace (English) model

> Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source
>
> Thursday, May 12, 2016

Handling non-projectivity

- The arc-standard algorithm we presented only builds **projective** dependency trees
- Possible directions:
    - Give up!
    - Post-processing
    - Add new transition types (e.g., SWAP)
    - Switch to a different algorithm (e.g., graph-based parsers such as MSTParser)

Language Models

Review

Definition of Language Model (LM)

- For a sequence of words/tokens $w_1, w_2, \ldots, w_N$, a LM outputs the probability of the sequence $P(w_1, w_2, \ldots, w_N)$
- $P(w_1, w_2, \ldots, w_N) = p(w_1)\, p(w_2|w_1)\, p(w_3|w_1, w_2) \times \ldots \times p(w_N|w_1, w_2, \ldots w_{N-1})$
- Each *P(…)* is determined by counting:

$$P(\text{sat}|\text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

$$P(\text{on}|\text{the cat sat}) = \frac{\text{count}(\text{the cat sat on})}{\text{count}(\text{the cat sat})}$$

$$\vdots$$

This process of computing P's is called ***maximum likelihood estimation*** (MLE).

Estimate the parameters of the model such that, according to the model, the ***likelihood*** of the observed data is ***maximized***.

Markhov Assumption

- To reduce the number of parameters, use only the recent past to predict the next token/word.
  - 1st order

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$$

Bigram model

  - 2nd order

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

Trigram model

  - Consider only the last *k* words for context

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

(k+1)gram model

- If we had infinite corpus, larger n → more accurate model

Evaluating a LM

- Once you train a LM on a corpus, you need to test it on a different, unseen corpus.
  - Note: a separate “dev set” is useful for tuning hyperparameters, such as α for Laplace smoothing.
- **Extrinsic evaluation**: Evaluate LM on a downstream task

- **Intrinsic evaluation**: Just use a held-out test corpus

Definition of Perplexity

- Perplexity is how well a probability distribution or model predicts a sample sequence (lower is better).

- Perplexity is 2^{CE}. Fundamentally, cross-entropy (CE) measures how deficient/difficult the model is at predicting the corpus.

- CE $= -\frac{1}{W}\sum_{i=1}^{n}\log_2 P(S_i)$ where W is the num. of words in corpus

- Remember: perplexity is the **inverse probability** of the corpus according to the LM, normalized by the number of words.

- If each n-gram has the same probability $\frac{1}{|V|}$,

$$\text{ppl} = 2^{-\frac{1}{W}W*log(1/|V|)} = |V|$$

Size of vocab

Perplexity in Perspective

- Think of perplexity as a weighted average branching factor: Lower perplexity means it's easier to predict the next word, in the corpus that's being evaluated.

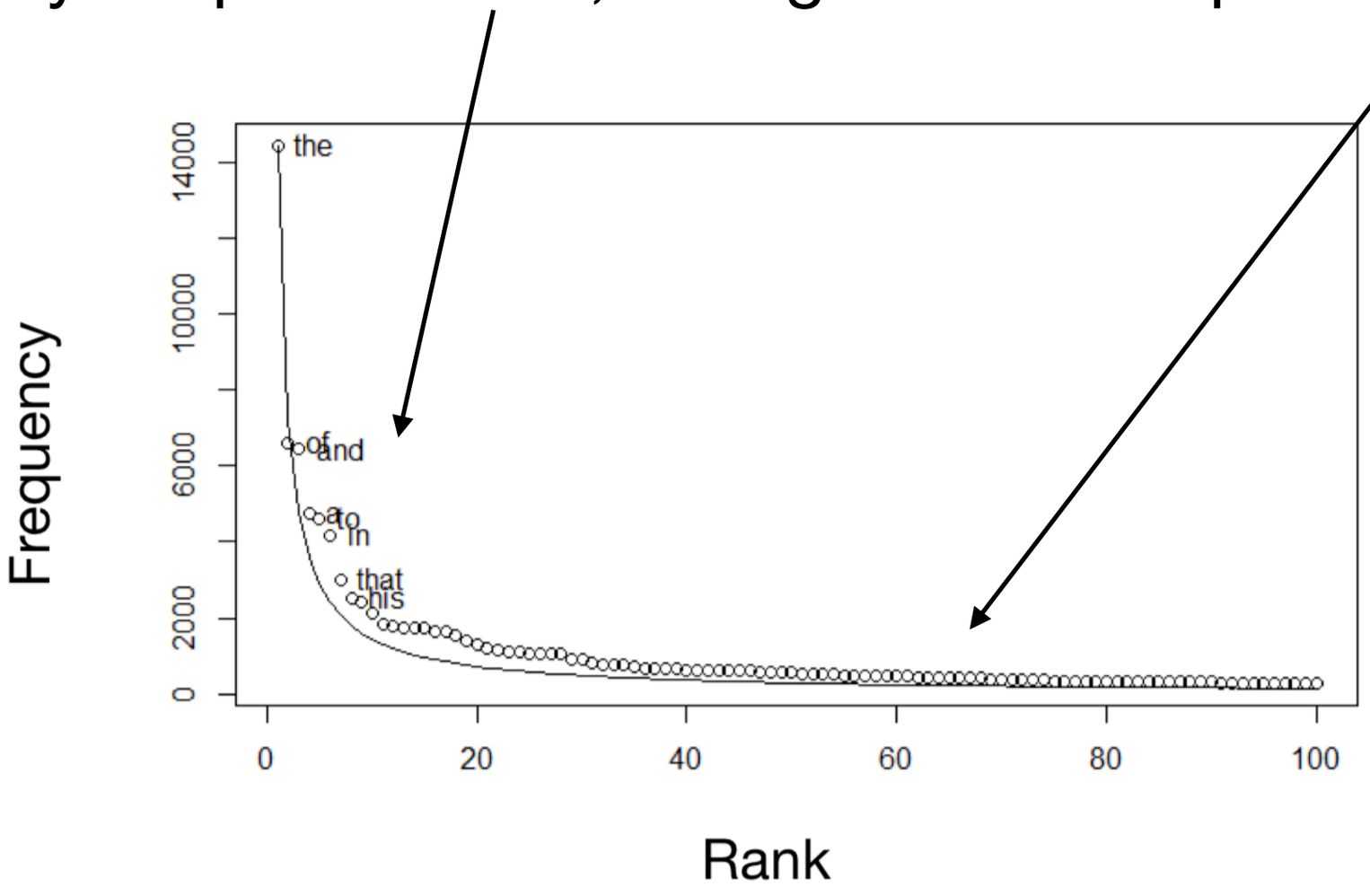
Example: Rolling a 6-sided dice (die)		
Train & test with perfectly-fair die	3 5 1 4 4 2 6…	Testing Perplexity = 6
Train & test with same loaded die	1 1 5 1 1 1 1…	Testing Perplexity < 6
Train & test with differently-loaded die	4 3 4 4 6 4 4…	Testing Perplexity > 6

- Low perplexity does not guarantee good extrinsic results!

Sparsity in Language

- Many n-grams can appear in the test corpus but not training
- A few very frequent words; a long tail of infrequent words

Zipf's law: frequency is inversely proportional to rank



- Need some way to remove zero probabilities

Add-alpha (Laplace) smoothing *Add a small amount to all probabilities*

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha |V|}$$

Linear interpolation *Use a combination of different granularities of n-grams*

$$\begin{aligned}\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) \\ +\lambda_2 P(w_i|w_{i-1}) \\ +\lambda_3 P(w_i)\end{aligned} \quad \text{where} \quad \sum_i \lambda_i = 1$$

Average count *Like simple interpolation, but with more specific lambdas*

$$\begin{aligned}P_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + \\ (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{\text{interp}}(w_i|w_{i-n+2}^{i-1})\end{aligned}$$

where $\lambda_{w_{i-n+1}^{i-1}}$ is based on $\frac{c(w_{i-n+1}^{i-1})}{|w_i : c(w_{i-n+1}^{i}) > 0|}$

average number of counts per non-zero element

"The less sparse the data the larger lambda should be. The more accurate counts we have, the more trustworthy the n-gram is, and the higher we can make lambda."

Absolute discounting *Redistribute probability mass from observed n-grams to unobserved ones*

$$P_{\text{abs\_discount}}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \alpha(w_{i-1})P(w) \quad \text{where} \quad \alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

Back-off *Use lower order n-grams if higher order ones are too sparse*

Katz

$$P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1}) = \begin{cases} d_{w_{i-n+1} \cdots w_i} \dfrac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\ \alpha_{w_{i-n+1} \cdots w_{i-1}} P_{bo}(w_i \mid w_{i-n+2} \cdots w_{i-1}) & \text{otherwise} \end{cases}$$

Focus(1):

- Naive Bayes
- Logistic Regression
  - Training
- Difference between discriminative and generative models
- Evaluation metrics
  - Precision
  - Recall
  - F Score

Naive Bayes

- Example of email classification:
  - Given documents and classes, how to classify another document?
- Formula snippet:

- Bayes Rule:

$$P(c \mid d) = \frac{P(c)\ P(d \mid c)}{P(d)}$$

- Makes strong (naive) independence assumptions

- Option 1: represent the entire sequence of words
  - $P(w_1, w_2, w_3, \ldots, w_k \mid c)$ *(too many sequences!)*
- Option 2: Bag of words
  - Assume position of each word is irrelevant (both absolute and relative)
  - $P(w_1, w_2, w_3, \ldots, w_k \mid c) = P(w_1|c)\ P(w_2|c) \ldots P(w_k|c)$
  - Probability of each word is *conditionally independent* given class **c**

Data Sparsity

Maximum likelihood estimates:

$$\hat{P}(c_j) = \frac{\text{Count}(\text{class} = c_j)}{\sum_c \text{Count}(\text{class} = c)}$$

← Total # of documents

$$\hat{P}(w_i | c_j) = \frac{\text{Count}(w_i, c_j)}{\sum_w \text{Count}(w, c_j)}$$

- Laplace smoothing:

$$\hat{P}(w_i | c) = \frac{\text{Count}(w_i, c) + \alpha}{\left[\sum_w \text{Count}(w, c)\right] + \alpha |V|}$$

↖ Vocabulary size

Summary:

Input: Set of annotated documents $\{(d_i, c_i)\}_{i=1}^{n}$

A. Compute vocabulary **V** of all words

B. Calculate $\hat{P}(c_j) = \frac{\text{Count}(c_j)}{\text{Total \# docs}}$

C. Calculate $\hat{P}(w_i \mid c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} [\text{Count}(w, c_j) + \alpha]}$

D. (Prediction) Given document $d = (w_1, w_2, \ldots, w_k)$

$$c_{MAP} = \arg\max_{c} \hat{P}(c) \cdot \prod_{i=1}^{k} \hat{P}(w_i \mid c)$$

Evaluation Metrics

		Truth	
		Positive	**Negative**
Predicted	**Positive**	100	5
	Negative	45	100

- True positive: Predicted + and actual +
- True negative: Predicted - and actual -
- False positive: Predicted + and actual -
- False negative: Predicted - and actual +

$$\textbf{Precision}(+) = \frac{TP}{TP + FP}$$

$$\textbf{Recall}(+) = \frac{TP}{TP + FN}$$

$$F_\beta = \frac{(1 + \beta^2) \cdot \textbf{Precision} \cdot \textbf{Recall}}{\beta^2 \cdot \textbf{Precision} + \textbf{Recall}}$$

Logistic Regression

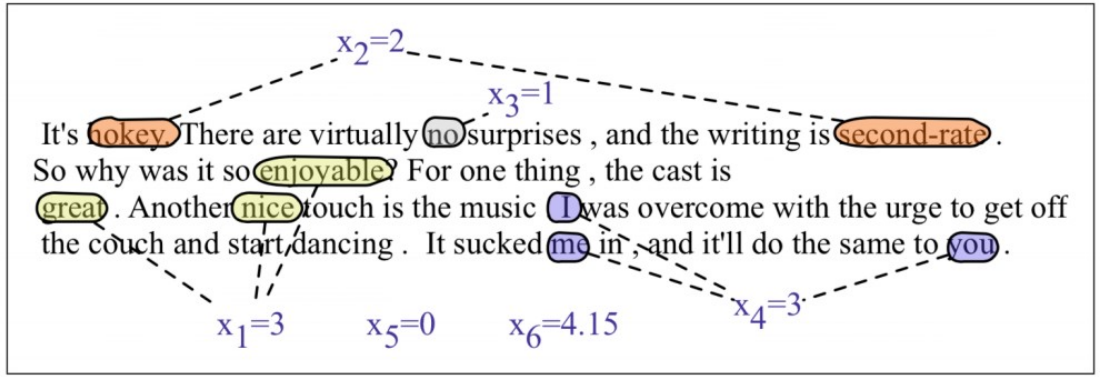
- Logistic Regression: $\hat{c} = \text{argmax}\ P(c|d)$

Using Logistic Regression

- Inputs:
  1. Classification instance in a **feature representation** $[x_1, x_2, \ldots, x_d]$
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y}\,|\,x)$
  3. **Loss function** (for learning)
  4. Optimization **algorithm**
- Train phase: Learn the **parameters** of the model to minimize **loss function**
- Test phase: Apply **parameters** to predict class given a new input x

Feature Representation

Sample feature vector



Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Classification Function

Given x, compute $z = w \cdot x + b$

Compute probabilities: $P(y = 1 \mid x) = \dfrac{1}{1 + e^{-z}}$

$$P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b) = 1 - \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{e^{-(w \cdot x + b)}}{1 + e^{-(w \cdot x + b)}}$$

Decision boundary: $\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Loss Function – Assigning ‘w’ and ‘b’?

Goal: predicted label $\hat{y}$ as close as possible to actual label y

Properties of CE Loss

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

- Ranges from 0 (perfect predictions) to ∞

- Lower the value, better the classifier

- *Cross-entropy* between the true distribution $P(y \mid x)$ and predicted distribution $P(\hat{y} \mid x)$

Optimisation

Gradient for logistic regression

$$\theta = [w; b]$$

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)} + b))]$

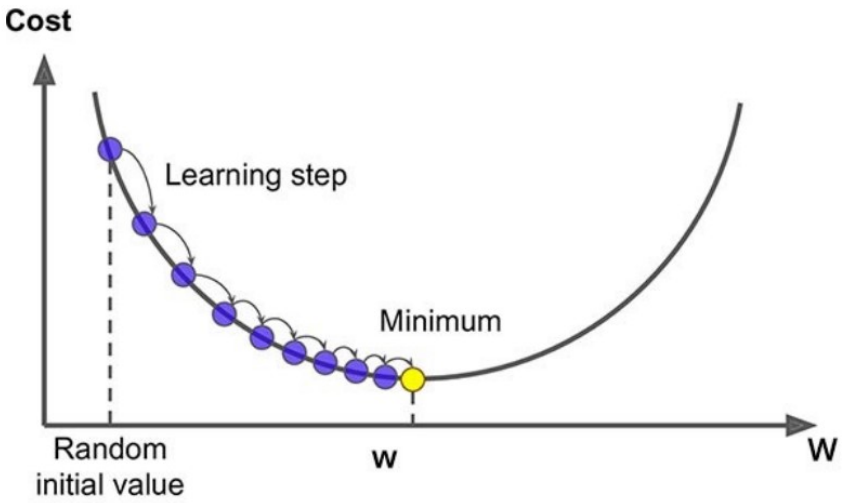
$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- Gradient, $\frac{dL_{CE}(w,b)}{dw_j} = \sum_{i=1}^{n} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)}$

Diff between true y and prediction

input feature value

- $\frac{dL_{CE}(w,b)}{db} = \sum_{i=1}^{n} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}]$



$$\theta^{t+1} = \theta^{t} - \eta \frac{d}{d\theta} f(x; \theta)$$

SGD

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow 0$
repeat til done # see caption
For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
1. Optional (for reporting): # How are we doing on this tuple?
Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

Regularisation

- Prevents Overfitting:

Training objective: $\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} \mid x^{(i)})$

Regularization helps prevent overfitting

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} \mid x^{(i)}) - \alpha R(\theta)$$

Penalize large weights

Word embeddings

COS484, Midterm Review

Word embeddings

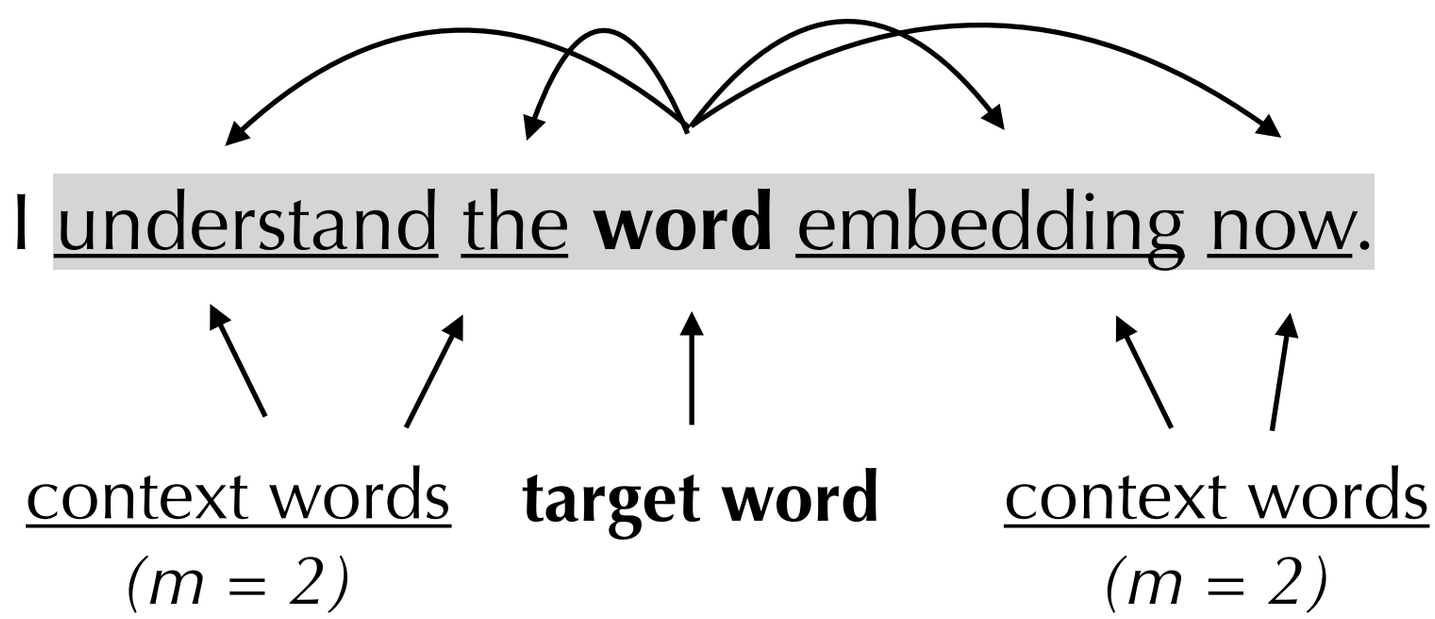
- **Distributional hypothesis:**
  - "words that occur in **similar contexts** tend to have **similar meanings**"
- **Word embedding:**
  - A **vector** that captures the meaning of a word.
  - Can be **sparse** (word-word occurrence) or **dense**.

<u>Goal</u>: represent words as
short (50-300 dimensional)
& **dense** (real-valued) vectors.

$$\text{employees} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 10.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word embeddings

- **Word2Vec**
  - **Skip-gram:** given a **target word**, predict the **context words** in a fixed window of size *m*.

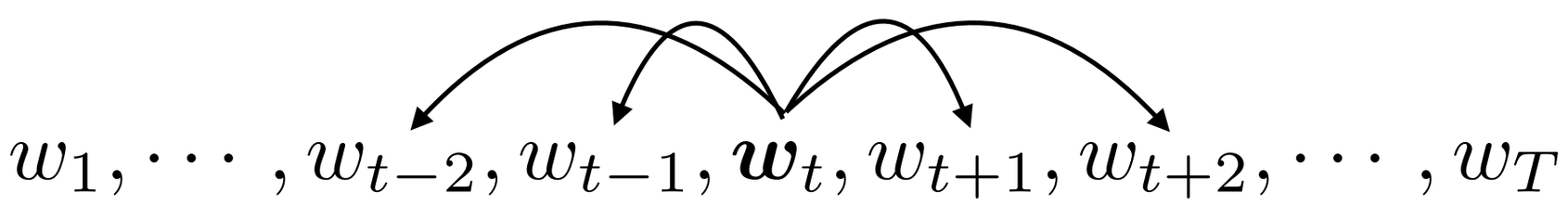


Word embeddings

- **Word2Vec**
  - **Objective function:** Average **N**egative **L**og **L**ikelihood (NLL)

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq t}} \log \mathbb{P}(w_{t+j}|w_t;\theta)$$

given a sentence of length *T*



Word embeddings

- **Word2Vec**

  - **Objective function:** Average **N**egative **L**og **L**ikelihood (NLL)

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq t}} \log \mathbb{P}(w_{t+j}|w_t;\theta)$$

- **Similarity based softmax** (V is vocabulary):

$$\theta = \{\{\boldsymbol{u}_w \in \mathbb{R}^d\}_{w\in V}, \{\boldsymbol{v}_w \in \mathbb{R}^d\}_{w\in V}\}$$

embeddings of target words

embeddings of context words

$$\mathbb{P}(w_{t+j}|w_t;\theta) = \frac{\exp(\boldsymbol{u}_{w_t}\cdot\boldsymbol{v}_{w_{t+j}})}{\sum_{w\in V}\exp(\boldsymbol{u}_{w_t}\cdot\boldsymbol{v}_w)}$$

Word embeddings

$$y = -\log\left(\frac{\exp(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})}{\sum_{w \in V} \exp(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_w)}\right)$$

- **Word2Vec**
  - **Gradient of objective function:**

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m<k<m} \nabla_\theta - \log(\mathbb{P}(w_{t+j}|w_t;\theta))$$

$$\Rightarrow \begin{cases} \frac{\partial y}{\partial \boldsymbol{u}_{w_t}} = -\boldsymbol{v}_{w_c} + \sum_{w \in V} \mathbb{P}(w|w_t)\boldsymbol{v}_w \\ \frac{\partial y}{\partial \boldsymbol{v}_w} = -1(w = w_c)\boldsymbol{u}_{w_t} + \mathbb{P}(w|w_t)\boldsymbol{u}_{w_t} \end{cases}$$

Word embeddings

$$y = -\log\left(\frac{\exp(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})}{\sum_{w \in V} \exp(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_w)}\right)$$

- **Word2Vec**
  - **Gradient of objective function:**

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m<k<m} \nabla_\theta - \log(\mathbb{P}(w_{t+j}|w_t;\theta))$$

$$\Rightarrow \begin{cases} \frac{\partial y}{\partial \boldsymbol{u}_{w_t}} = -\boldsymbol{v}_{w_c} + \sum_{w \in V} \mathbb{P}(w|w_t)\boldsymbol{v}_w \\ \frac{\partial y}{\partial \boldsymbol{v}_w} = -1(w = w_c)\boldsymbol{u}_{w_t} + \mathbb{P}(w|w_t)\boldsymbol{u}_{w_t} \end{cases}$$

- **Full softmax is computationally intractable**

Word embeddings

$$y = -\log\left(\frac{\sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})}{\prod_{k=1}^{K} \sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_k})}\right)$$

- **Word2Vec**
  - **Gradient of objective function:**

Sample $w_k \sim \mathbb{P}_{\text{neg}}(w)$

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m<k<m} \nabla_\theta - \log(\mathbb{P}_{\texttt{NS}}(w_{t+j}|w_t;\theta))$$

$$\begin{cases} \frac{\partial y}{\partial \boldsymbol{u}_{w_t}} = -\sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})\boldsymbol{v}_{w_c} + \sum_{k=1}^{K} \sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_k})\boldsymbol{v}_{w_k} \\ \frac{\partial y}{\partial \boldsymbol{v}_w} = -1_{w=w_c}\sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})\boldsymbol{u}_{w_t} + \sum_{k=1}^{K} \frac{\partial \boldsymbol{v}_{w_k}}{\partial \boldsymbol{v}_w}\sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_w)\boldsymbol{u}_{w_t} \end{cases}$$

- **Negative Sampling:** Randomly sample *K* (5-20) negative examples.

Word embeddings

$$y = -\log\left(\frac{\sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})}{\prod_{k=1}^{K} \sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_k})}\right)$$

- **Word2Vec**
  - **Gradient of objective function:**

Sample $w_k \sim \mathbb{P}_{\text{neg}}(w)$

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m<k<m} \nabla_\theta - \log(\mathbb{P}_{\texttt{NS}}(w_{t+j}|w_t;\theta))$$

$$\begin{cases} \frac{\partial y}{\partial \boldsymbol{u}_{w_t}} = -\sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})\boldsymbol{v}_{w_c} + \sum_{k=1}^{K} \sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_k})\boldsymbol{v}_{w_k} \\ \frac{\partial y}{\partial \boldsymbol{v}_w} = -1_{w=w_c}\sigma(-\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_{w_c})\boldsymbol{u}_{w_t} + \sum_{k=1}^{K} \frac{\partial \boldsymbol{v}_{w_k}}{\partial \boldsymbol{v}_w}\sigma(\boldsymbol{u}_{w_t} \cdot \boldsymbol{v}_w)\boldsymbol{u}_{w_t} \end{cases}$$

- **Update** for **sampled** target-context word pairs (w_t, w_c)**:**

$$\begin{cases} \boldsymbol{u}_{w_t} \leftarrow \boldsymbol{u}_{w_t} - \eta \frac{\partial y}{\partial \boldsymbol{u}_{w_t}} \\ \boldsymbol{v}_{w_c} \leftarrow \boldsymbol{v}_{w_c} - \eta \frac{\partial y}{\partial \boldsymbol{v}_{w_c}} \end{cases}, \quad \boldsymbol{v}_{w_k} \leftarrow \boldsymbol{v}_{w_k} - \eta \frac{\partial y}{\partial \boldsymbol{v}_{w_k}}$$

Word embeddings

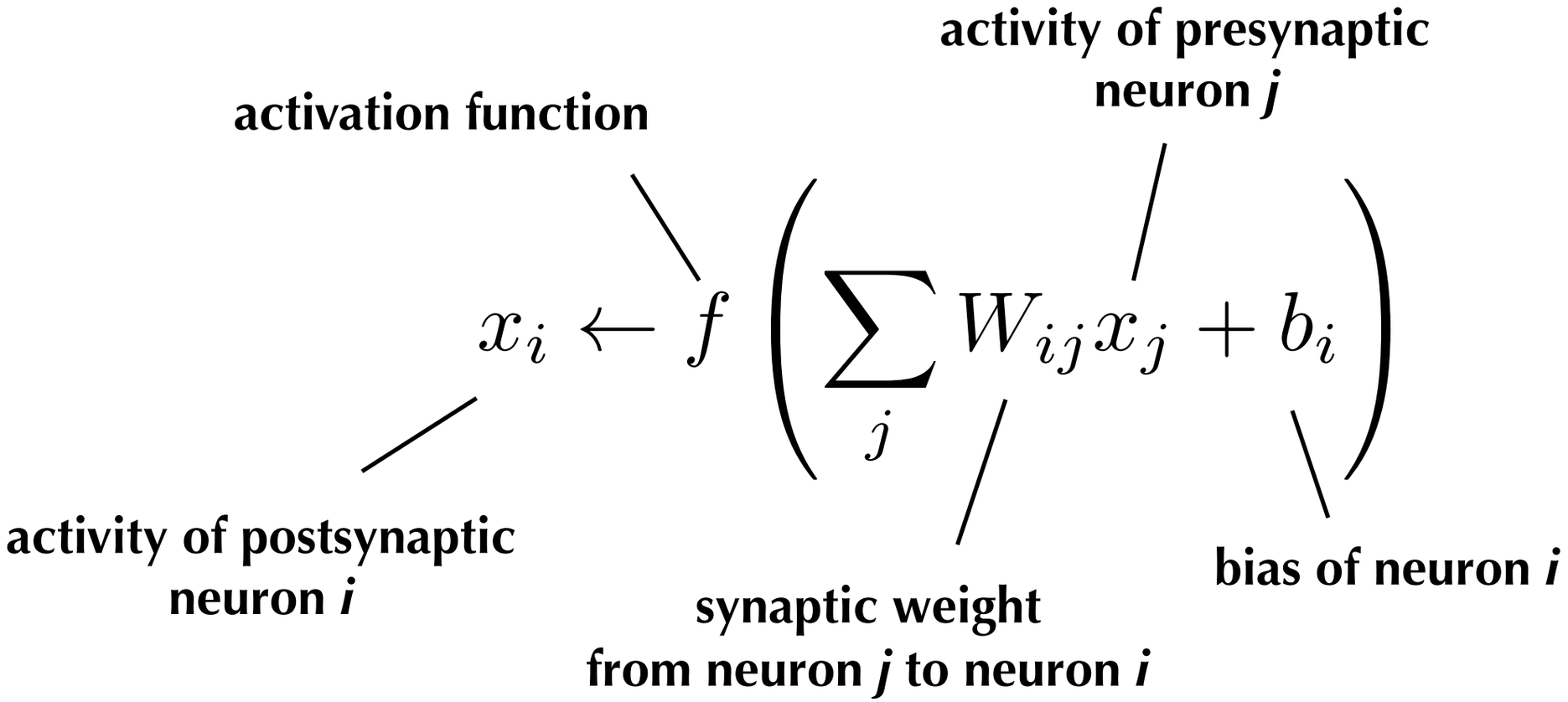
- **Evaluation**
  - **Intrinsic**
    - Evaluate on an **intermediate** subtask (e.g. word similarity)
    - **Fast to compute**
    - Not clear if it really helps the downstream task
  - **Extrinsic**
    - Use word embeddings in **downstream tasks** and measure the performance improvement
    - Time-costly but still the **most important** evaluation metric

Neural Networks

COS484, Midterm Review

Neural Networks

- **The activity of neuron *i*:**



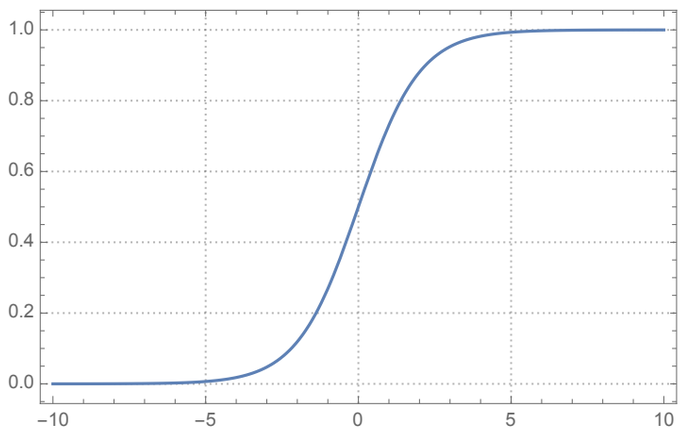
- repeat for other neurons **in some order** to be specified

Neural Networks

- **Useful activation functions and their derivatives**

sigmoid (0,1)

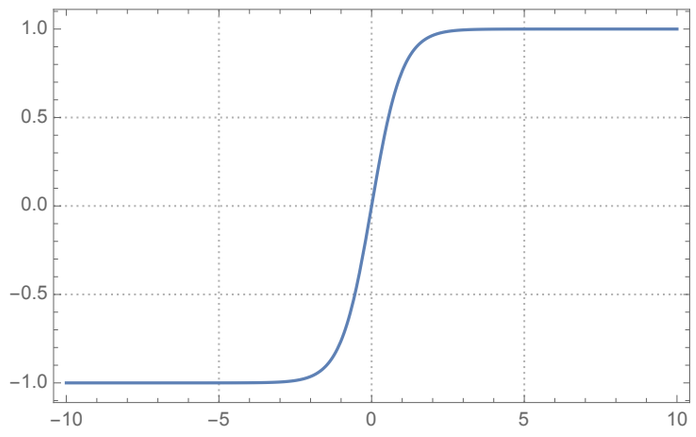
$$f(z) = \frac{1}{1+\exp(-z)}$$



$$f'(z) = f(z)(1-f(z))$$

tanh (-1,1)

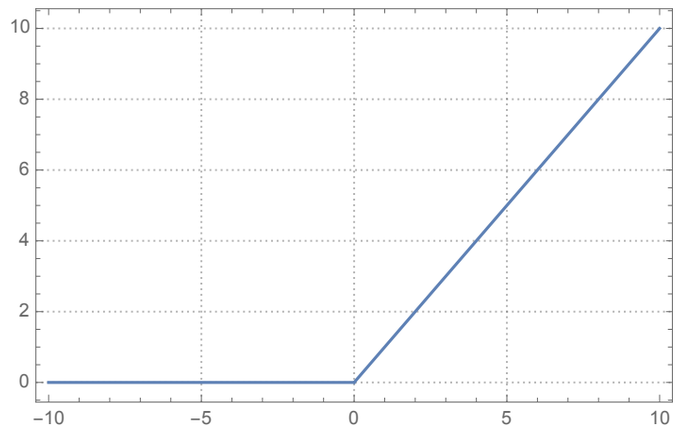
$$f(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$$



$$f'(z) = 1-f^2(z)$$

ReLU [0,+∞)

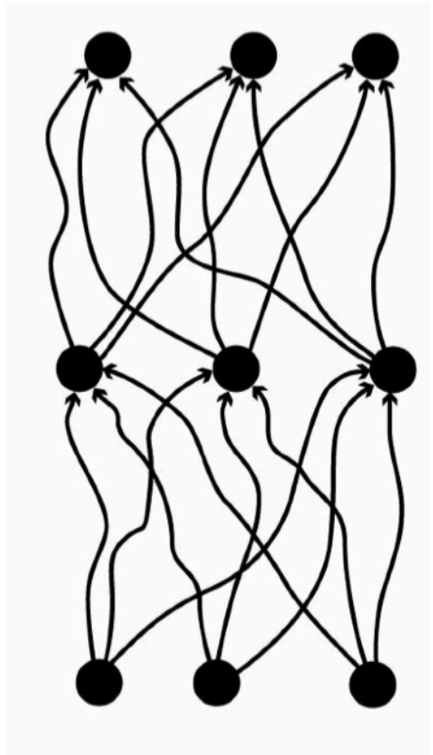
$$f(z) = \max(0,z)$$



$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Neural Networks

- ***L*-Multilayer Formulation** (using matrix-vector notation)



For layer $\ell = 0$ to $L\text{-}1$:

$$\boldsymbol{x}^{\ell} \leftarrow f(W^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell})$$

neural activity of previous layer

weight matrix

bias vector

$$\boldsymbol{x}^0 \xrightarrow{W^1, \boldsymbol{b}^1} \boldsymbol{x}^1 \xrightarrow{W^2, \boldsymbol{b}^2} \cdots \xrightarrow{W^L, \boldsymbol{b}^L} \boldsymbol{x}^L$$

Forward Pass

Neural Networks

- **Loss function** (error between output and ground truth)
  - Logistic Regression / Classification - Cross Entropy (CE)
  - Linear Regression - Mean Squared Error (MSE)
  - ….

ground truth

$$\boldsymbol{x}^0 \xrightarrow{W^1, \boldsymbol{b}^1} \boldsymbol{x}^1 \xrightarrow{W^2, \boldsymbol{b}^2} \cdots \xrightarrow{W^L, \boldsymbol{b}^L} \boldsymbol{x}^L \longrightarrow \mathcal{L}(x^L, y^*)$$

Forward Pass **Loss Function**

- **Update weights and biases using gradient descent (Backprop)**

$$\theta := \{\{W^\ell\}, \{\boldsymbol{b}^\ell\}\} \qquad \theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

Neural Networks

- **Backpropagation** (using chain rule of derivative)

$$\boldsymbol{x}^0 \xrightarrow{W^1, \boldsymbol{b}^1} \boldsymbol{x}^1 \xrightarrow{W^2, \boldsymbol{b}^2} \cdots \xrightarrow{W^L, \boldsymbol{b}^L} \boldsymbol{x}^L \longrightarrow \mathcal{L}(x^L, y^*)$$

Forward Pass $\quad \boldsymbol{x}^\ell \leftarrow f(W^\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}^\ell)$

Define dual variables to maintain negative gradients $\quad \boldsymbol{\rho}^L \leftarrow -\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L} \circ f'(f^{-1}(\boldsymbol{x}^L))$

$$\boldsymbol{\rho}^0 \xleftarrow{W^1, \boldsymbol{b}^1} \boldsymbol{\rho}^1 \xleftarrow{W^2, \boldsymbol{b}^2} \cdots \xleftarrow{W^L, \boldsymbol{b}^L} \boldsymbol{\rho}^L \longleftarrow \mathcal{L}(x^L, y^*)$$

Backward Pass $\quad \boldsymbol{\rho}^{\ell-1} \leftarrow f'(f^{-1}(\boldsymbol{x}^{\ell-1})) \circ (W^\ell)^\intercal \boldsymbol{\rho}^\ell$

Neural Networks

- **Backpropagation** (using chain rule of derivative)

Define dual variables to maintain negative gradients

$$\boldsymbol{\rho}^L \leftarrow -\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L} \circ f'(f^{-1}(\boldsymbol{x}^L))$$

$$\boldsymbol{\rho}^0 \xleftarrow{W^1, \boldsymbol{b}^1} \boldsymbol{\rho}^1 \xleftarrow{W^2, \boldsymbol{b}^2} \cdots \xleftarrow{W^L, \boldsymbol{b}^L} \boldsymbol{\rho}^L \quad \leftarrow \mathcal{L}(x^L, y^*)$$

Backward Pass

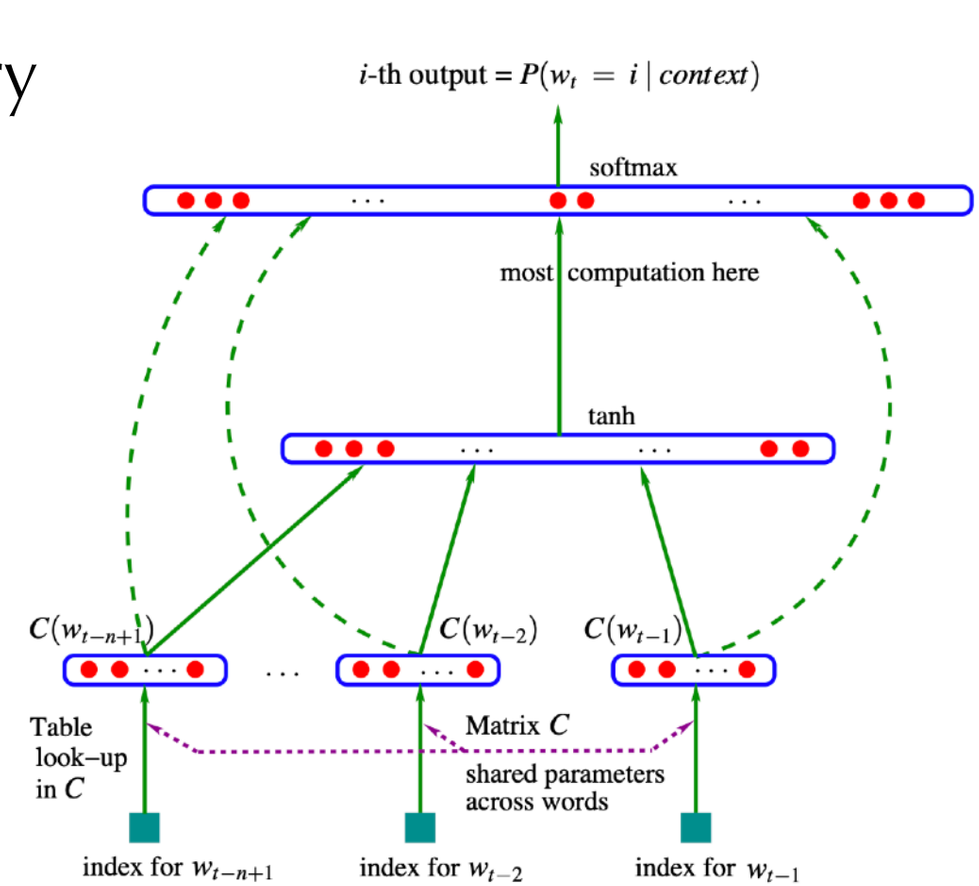
$$\boldsymbol{\rho}^{\ell-1} \leftarrow f'(f^{-1}(\boldsymbol{x}^{\ell-1})) \circ (W^\ell)^\intercal \boldsymbol{\rho}^\ell$$

- **Update weights and biases**

$$\Delta W^\ell \propto \boldsymbol{\rho}^\ell (\boldsymbol{x}^{\ell-1})^\intercal \qquad \Delta \boldsymbol{b}^\ell \propto \boldsymbol{\rho}^\ell$$

Neural Networks

- **Feedforward Language Model**
  - **N-gram models:** *P*(mat | the cat sat on the)
  - **Input**: concatenation of previous words (with fixed context size)
  - **Hidden Layer**: fully connected, use tanh activation
  - **Output**: softmax over vocabulary



Neural Networks

- **Recurrent neural networks**
  - **Simple RNN** (Key: weight sharing)

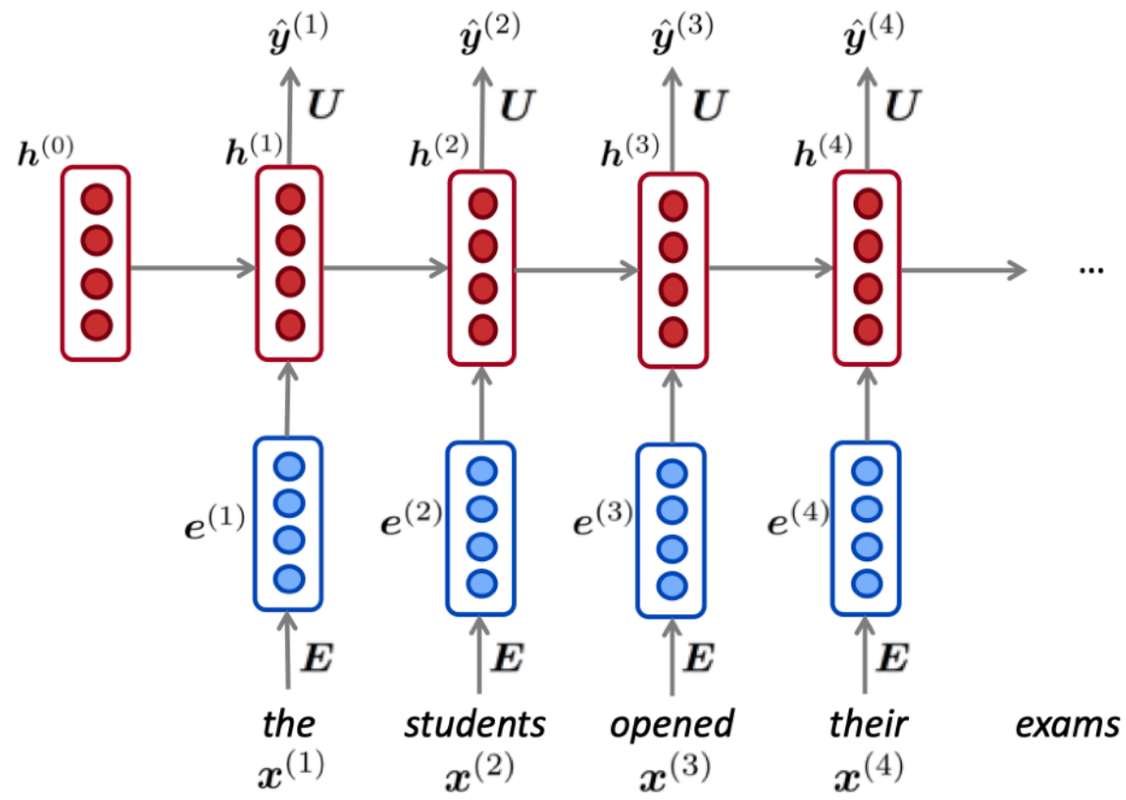
$$\boldsymbol{h}_t = f(W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t + \boldsymbol{b})$$

previous hidden state — **current input**

- **Recurrent Neural Langhage Models**

Neural Networks

- **Recurrent neural networks**
  - **Simple RNN** (Key: weight sharing)



- **Recurrent Neural Langhage Models**

Neural Networks

- **Recurrent neural networks**
  - **Backprop through time** (expensive for long sequences)

$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$
$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$
$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$
$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

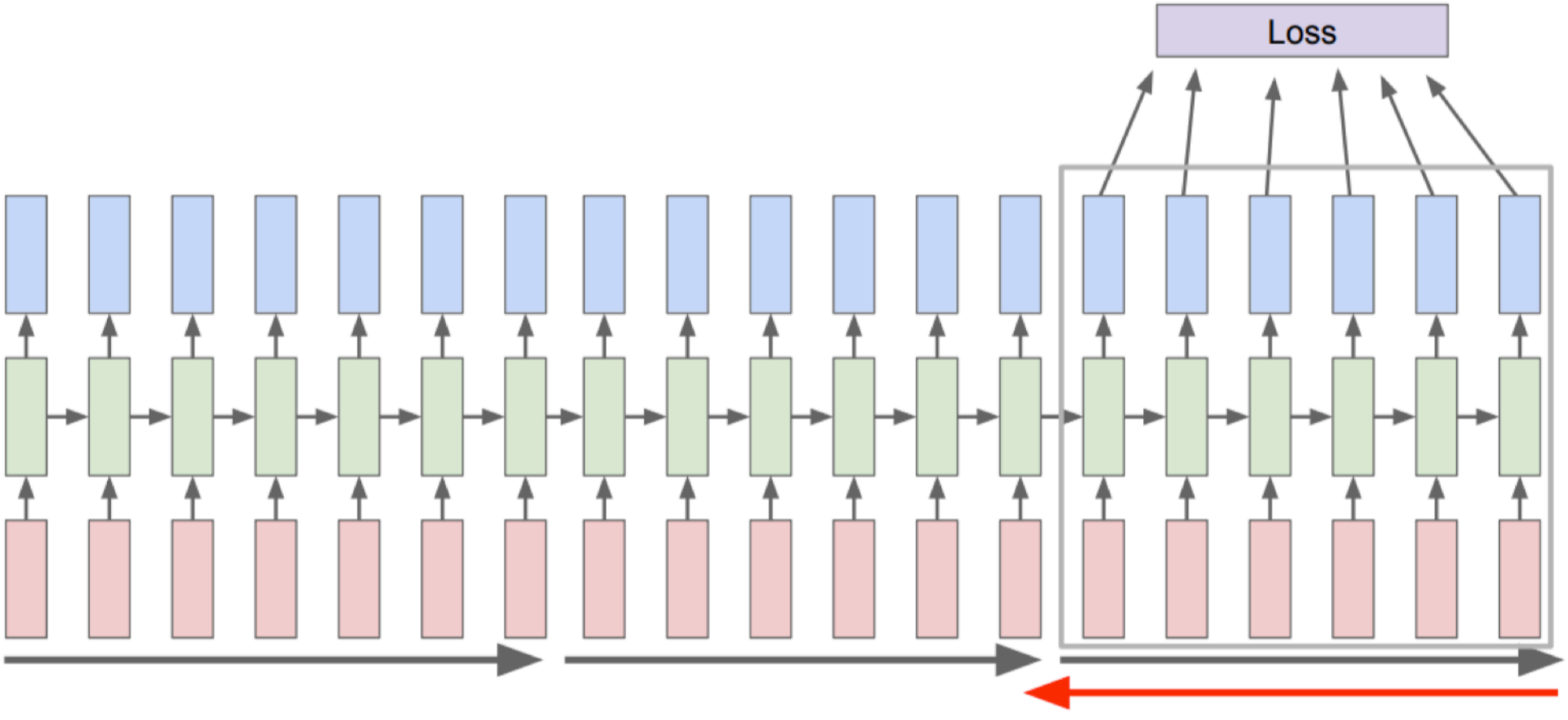
You should know how to compute: $\frac{\partial L_3}{\partial \mathbf{h}_3}$

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

$$\boxed{\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}}$$

Neural Networks

- **Recurrent neural networks**
  - **Backprop through time** (expensive for long sequences)
  - Run forward and backward through **chunks of the sequence**
  - **Only backpropagate for some smaller number of steps**



Neural Networks

- **Long Short-term Memory (LSTM)**

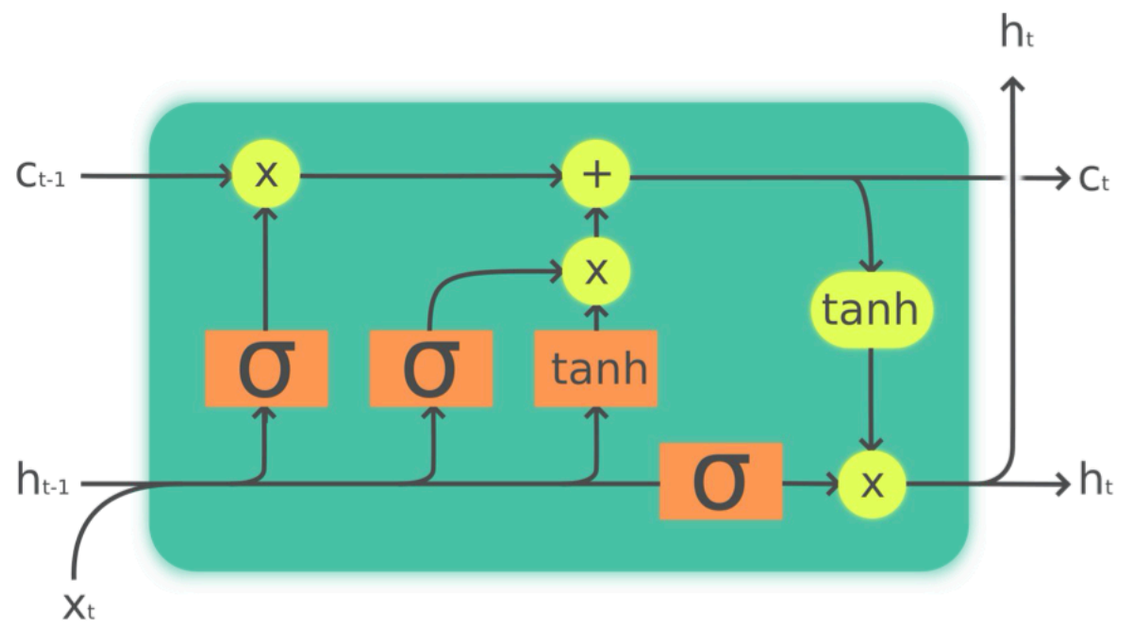
cell state vector $$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c)$$

forget gate $$\mathbf{f}_t = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f\right)$$

input gate $$\mathbf{i}_t = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i\right)$$

output gate $$\mathbf{o}_t = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o\right)$$

hidden state vector/output vector $$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

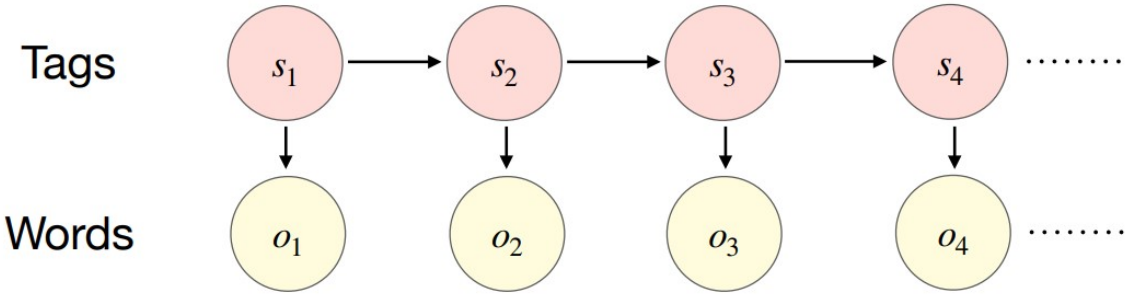


Focus(2):

- HMM
- MEMM
- 3 Problems in HMM
  - Decoding
  - Observation sequence
  - Training

HMM

- A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are hidden hidden: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags hidden because they are not observed.



Problem 1 (Likelihood):	Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O\|\lambda)$.
Problem 2 (Decoding):	Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q.
Problem 3 (Learning):	Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

1. Markov assumption:

$$P(s_{t+1} \mid s_1, \ldots, s_t) = P(s_{t+1} \mid s_t)$$

2. Output independence:

$$P(o_t \mid s_1, \ldots, s_t) = P(o_t \mid s_t)$$

Problem 2 – Decoding

$$\hat{S} = \underset{S}{\operatorname{argmax}}\, P(S|O) = \underset{S}{\operatorname{argmax}}\, \frac{P(S)\,P(O|S)}{P(O)} \quad \text{[Bayes]}$$

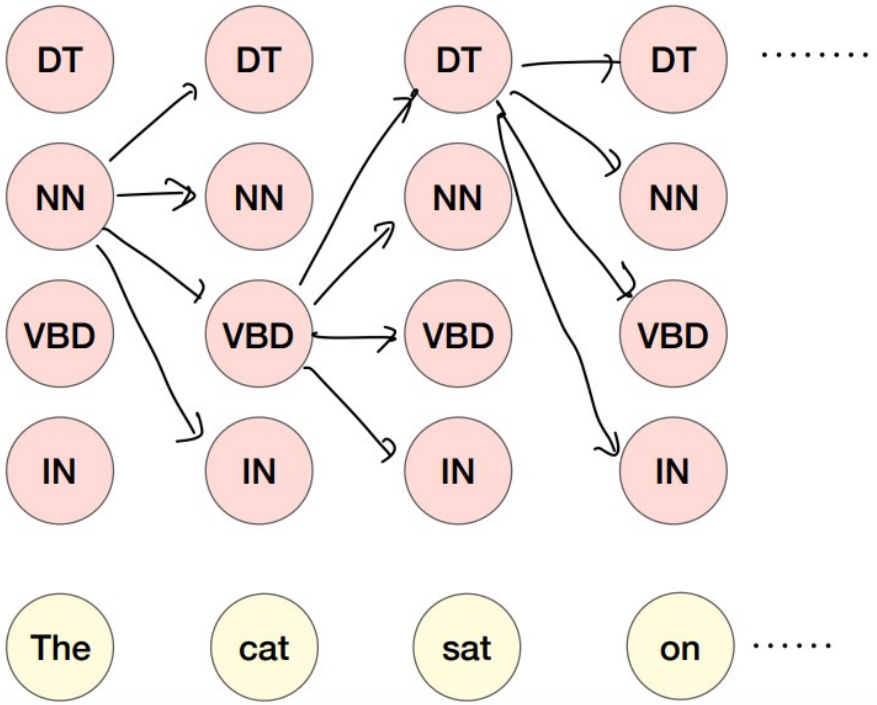
$$= \underset{S}{\operatorname{argmax}}\, P(S)\,P(O|S)$$

Greedy Decoding:

$$\forall t, \quad \hat{S}_{t+1} = \underset{S}{\operatorname{argmax}}\, P(S|\hat{S}_t)\,P(O_{t+1}|S)$$

Viterbi:

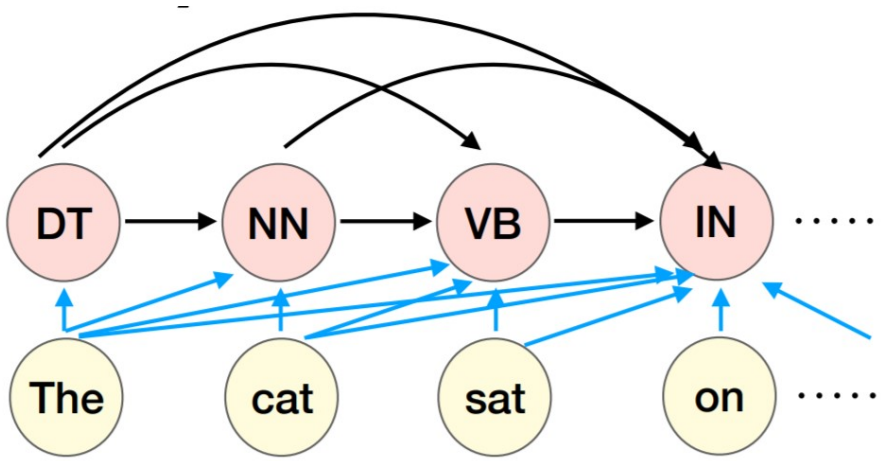
$M[i, j]$: Most probable sequence of states ending with state **j** at time **i**



$$M[i, j] = \max_k M[i-1, k]\ P(s_j \mid s_k)\ P(o_i \mid s_j) \quad 1 \le k \le K \quad 1 \le i \le n$$

Backward: Pick $\max_k M[n, k]$ and backtrack

MEMM



MEMM

In general, we can use all observations and all previous states:

$$\hat{S} = \arg\max_{S} P(S \mid O) = \arg\max_{S} \prod_{i} P(s_i \mid o_n, o_{i-1}, \ldots, o_1, s_{i-1}, \ldots, s_1)$$

Problem 1: Prob. of Observation sequence

- Forward and backward probabilities:

Define:

$\alpha_s(j) = P(x_1, \ldots, x_{j-1}, y_j = s \,|\, \theta, \phi)$ (forward probability)

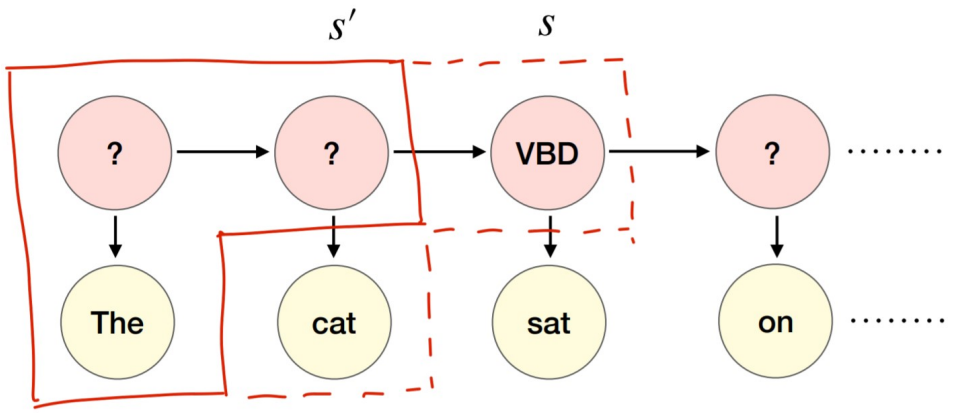
$$= \sum_{s'} \alpha_{s'}(j-1)\, \phi_{s' \to x_{j-1}}\, \theta_{s' \to s}$$

$\beta_s(j) = P(x_j, \ldots, x_m \,|\, y_j = s, \theta, \phi)$ (backward probability)

$$\beta_s(j) = \phi_{s \to x_j} \sum_{s'} \beta_{s'}(j+1) \quad \theta_{s \to s'}$$

Observation likelihood,

$Z = P(x_1, x_2, \ldots, x_m \,|\, \theta, \phi) = \sum_s \alpha_s(j)\beta_s(j)$ for any $j \in 1,\ldots,m$



Problem3: HMM Training

- Intuitive Idea of EM:
  - θ^t is the parameter vector at the t^{th} iteration
  - Choose θ^0 at random (or using smart heuristics)
  - (E step): Compute *expected* counts

    $$\overline{Count}(r) = \sum_{i=1}^{n} \sum_{y} P(y \mid x_i, \theta^{t-1})\ Count(x_i, y, r)$$

    for every parameter θ_r
  - e.g.

    $$\overline{Count}(DT \rightarrow NN) = \sum_{i} \sum_{y} P(S \mid O_i, \theta^{t-1})\ Count(O_i, S, \theta_{DT \rightarrow NN})$$

  - (M step): Re-estimate parameters using expected counts to maximize likelihood

    $$\text{e.g. } \theta_{DT \rightarrow NN} = \frac{\overline{Count}(DT \rightarrow NN)}{\sum_{\beta} \overline{Count}(DT \rightarrow \beta)}$$

Continued: HMM Training

- Forward – backward algorithm:

(E-Step)

$$\overline{Count}(\theta_k) = \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{t-1}, \phi^{t-1})\ Count(X_i, Y, \theta_k)$$

$$= \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{t-1}, \phi^{t-1})\ Count(Y, \theta_k)$$

$$\overline{Count}(\phi_k) = \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{t-1}, \phi^{t-1})\ Count(X_i, Y, \phi_k)$$

(M-Step)

$$\theta_k^t = \frac{\overline{Count}(\theta_k)}{\sum_{\theta' \in M(\theta_k)} \overline{Count}(\theta')}$$ where $M(\theta_k)$ is the set of all transitions $(a \rightarrow b,\ \text{all } b)$ that share the same previous state as the k^{th} transition $(a \rightarrow c \text{ for some } c)$.

$$\phi_k^t = \frac{\overline{Count}(\phi_k)}{\sum_{\phi' \in M'(\phi_k)} \overline{Count}(\phi')}$$ where $M'(\phi_k)$ is the set of all emissions $(a \rightarrow x,\ \text{all } x)$ that share the same hidden state as the k^{th} emission $(a \rightarrow x',\ \text{for some } x')$.

Continued

$$P(y_j = s \mid X, \theta, \phi) = \frac{\alpha_s(j)\beta_s(j)}{Z}$$

$$P(y_j = s, y_{j+1} = s' \mid X, \theta, \phi) = \frac{\alpha_s(j)\ \theta_{s \to s'}\ \phi_{s \to x_j}\ \beta_{s'}\ (j+1)}{Z}$$

Given these, we can now estimate:

$$\overline{Count}(\theta_{s \to s'}) = \sum_i \sum_{j=1}^{m} P(y_j = s, y_{j+1} = s' \mid X_i, \theta, \phi)$$

$$\overline{Count}(\phi_{s \to o}) = \sum_i \sum_{j : X_{ij} = o} P(y_j = s \mid X_i, \theta, \phi)$$