

***NATURAL
LANGUAGE
PROCESSING***

WEEK-2

THIRUMURUGAN.R

Language modeling

This is the ...

house

rat

did

malt

What's the probability of the next word?

$$p(\text{house} \mid \text{this is the}) = ?$$

Toy corpus

This is the house that Jack built.

This is the malt

That lay in the house that Jack built.

This is the rat,

That ate the malt

That lay in the house that Jack built.

This is the cat,

That killed the rat,

That ate the malt

That lay in the house that Jack built.

$$p(\text{house} \mid \text{this is the}) = \frac{c(\text{this is the house})}{c(\text{this is the ...})} = \frac{1}{4}$$

- **Language Model** - A language model is a probability distribution over word sequences that describes how often the sequence occurs as a sentence in some domain of interest.

$$P(w_1, w_2, \dots, w_n) = \prod_{k=1}^n P(w_k | w_1^{k-1})$$

Language Models

- Language models are useful for NLP applications such as
 - Next word prediction
 - Machine translation
 - Spelling correction
 - Authorship Identification
 - Natural language generation
- For intrinsic evaluation of language models, *Perplexity* metric is used.

Toy corpus

*This is the house **that Jack** built.*

This is the malt

***That lay** in the house **that Jack** built.*

This is the rat,

***That ate** the malt*

***That lay** in the house **that Jack** built.*

This is the cat,

***That killed** the rat,*

***That ate** the malt*

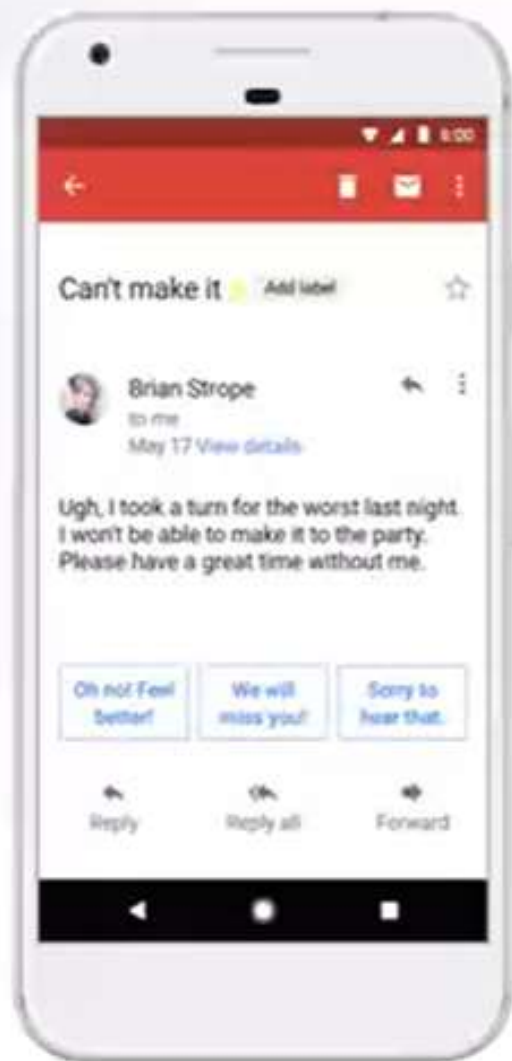
***That lay** in the house **that Jack** built.*

bigrams



$$p(\text{Jack} \mid \text{that}) = \frac{c(\text{that Jack})}{c(\text{that...})} = \frac{4}{10}$$

Where do we need LM?



- Suggestions in messengers
- Spelling correction
- Machine translation
- Speech recognition
- Handwriting recognition
- ...

Bigram language model

So that's what we get for $n = 2$:

$$p(\mathbf{w}) = p(w_1)p(w_2|w_1) \dots p(w_k|w_{k-1})$$

Toy corpus:

This is the malt

That lay in the house that Jack built.

$$p(\text{this is the house}) = \overset{1/12}{p(\text{this})} \overset{1}{p(\text{is}|\text{this})} \overset{1}{p(\text{the}|\text{is})} \overset{1/2}{p(\text{house}|\text{the})}$$

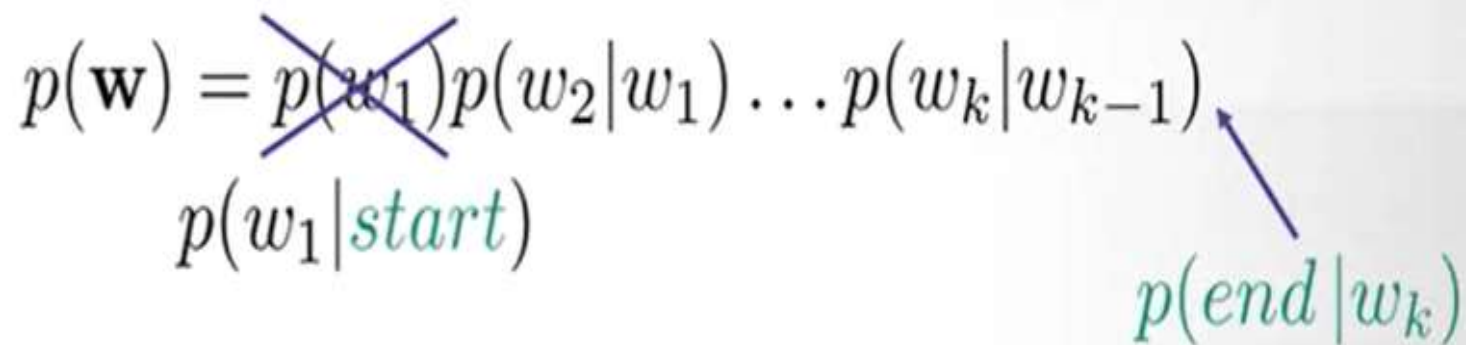
Bigram language model

So that's what we get for $n = 2$:

$$p(\mathbf{w}) = \cancel{p(w_1)} p(w_2|w_1) \dots p(w_k|w_{k-1})$$

$p(w_1|start)$

$p(end|w_k)$



It's normalized separately for each sequence length!

$$p(this) + p(that) = 1.0$$

$$p(this\ this) + p(this\ is) + \dots + p(built\ built) = 1.0$$

Let's check the model

_ dog _

_ dog cat tiger _

_ cat dog cat _

$$p(\text{cat dog cat}) = p(\text{cat} \mid _) p(\text{dog} \mid \text{cat}) p(\text{cat} \mid \text{dog}) p(_ \mid \text{cat})$$

dog	cat tiger	
	cat dog cat tiger	cat dog_
	cat dog cat dog	
	cat dog cat _	
	cat_	



Resume: bigram language model

Define the model:

$$p(\mathbf{w}) = \prod_{i=1}^{k+1} p(w_i | w_{i-1})$$

Estimate the probabilities:

$$p(w_i | w_{i-1}) = \frac{c(w_{i-1} w_i)}{\sum_{w_i} c(w_{i-1} w_i)} = \frac{c(w_{i-1} w_i)}{c(w_{i-1})}$$

It's all about counting!

How to train n-gram models

$$\mathbf{w} = (w_1 w_2 w_3 \dots w_k)$$

Bigram language model:


The first and last, serve as fake tokens!

$$p(\mathbf{w}) = \prod_{i=1}^{k+1} p(w_i | w_{i-1})$$

N-gram language model:

$$p(\mathbf{w}) = \prod_{i=1}^{k+1} p(w_i | w_{i-n+1}^{i-1})$$

$(w_{i-n+1}, \dots, w_{i-1})$



How to train n-gram models

Log-likelihood maximization:

$$\log p(\mathbf{w}_{\text{train}}) = \sum_{i=1}^{N+1} \log p(w_i | w_{i-n+1}^{i-1}) \rightarrow \max$$

Estimates for parameters:

$$p(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i)}{\sum_{w_i} c(w_{i-n+1}^i)} = \frac{c(w_{i-n+1}^i)}{c(w_{i-n+1}^{i-1})}$$

N is the length of the **train corpus** (all words concatenated).

Generated Shakespeare

Unigrams:

*To him swallowed confess hear both. Which. Of save on trail
for are ay device and rote life have. Every enter now severally
so, let. Hill he late speaks; or! a more to leg less first you enter.*

Bigrams:

*What means, sir. I confess she? then all sorts, he is trim,
captain. Why dost stand forth thy canopy, forsooth; he is this
palpable hit the King Henry. Live king. Follow. What we, hath
got so she that I rest and sent to scold and nature bankrupt,
nor the first gentleman?*

3-grams:

Sweet prince, Falstaff shall die. Harry of Monmouth's grave. This shall forbid it should be branded, if renown made it empty. What is't that cried? Indeed the duke; and had a very good friend. Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

4-grams:

King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; Will you not tell me who I am? It cannot be but so. Indeed the short and the long. Marry, 'tis a noble Lepidus. They say all lovers swear more performance than they are wont to keep obliged faith.

LOOKS MORE LIKE A SHAKESPEARE TEXT, THAN THE PREVIOUS TWO!

Which model is better?

The best n might depend on how much data you have:

- bigrams might be not enough
- 7-grams might never occur

5 GRAMS, generally performs well!

Extrinsic evaluation:

- Quality of a downstream task: machine translation, speech recognition, spelling correction...

Intrinsic evaluation:

- Hold-out (text) perplexity!

Evaluate the model on the test set

How to evaluate

Likelihood:

$$\mathcal{L} = p(\mathbf{w}_{\text{test}}) = \prod_{i=1}^{N+1} p(w_i | w_{i-n+1}^{i-1})$$

Perplexity:

$$\mathcal{P} = p(\mathbf{w}_{\text{test}})^{-\frac{1}{N}} = \frac{1}{\sqrt[N]{p(\mathbf{w}_{\text{test}})}}$$

N is the length of the **test corpus** (all words concatenated).

Lower perplexity is better! Greater likelihood!

Perplexity

- It is an evaluation metric for N-gram models.
- It is the weighted average number of choices a random variable can make, i.e. the number of possible next words that can follow a given word.

$$\text{Perplexity} = \sqrt[n]{\frac{1}{\prod_{i=1}^n \Pr(w_i | w_{i-1})}}$$

Motivation for Smoothing

- Even if one n-gram is unseen in the sentence, probability of the whole sentence becomes zero.
- To avoid this, some probability mass has to be reserved for the unseen words.
- **Solution - Smoothing techniques**
- This zero probability problem also occurs in **text categorization** using *Multinomial Naïve Bayes*
 - Probability of a test document given some class can be zero even if a single word in that document is unseen

Out-of-vocabulary words

Toy train corpus:

This is the house that Jack built.

Toy test corpus:

This is the *malt*.

What's the perplexity of the Bigram LM?

$$p(\textit{malt}|\textit{the}) = \frac{c(\textit{the malt})}{c(\textit{the})} = 0$$

$$p(\mathbf{w}_{\text{test}}) = 0$$

$$\mathcal{P} = \inf$$



How can we fix that?

Simple idea:

- Build a vocabulary (e.g. by word frequencies)
- Substitute OOV words by <UNK> (both in train and test!)
- Compute counts as usual for all tokens
- Profit!

OK, no OOV words

Toy train corpus:

This is the house that Jack built.

Toy test corpus:

This is *Jack*.

What's the perplexity of the Bigram LM?

$$p(Jack | is) = \frac{c(is \ Jack)}{c(is)} = 0$$

$$p(\mathbf{w}_{\text{test}}) = 0$$

$$\mathcal{P} = \inf$$



Zero probabilities for test data

Toy train corpus:

This is the house that Jack built.

Toy test corpus:

This is *Jack*.

What's the perplexity of the Bigram LM?

$$p(\textit{Jack} \mid \textit{is}) = \frac{c(\textit{is Jack})}{c(\textit{is})} = 0$$

$$p(\mathbf{w}_{\text{test}}) = 0$$

$$\mathcal{P} = \inf$$



Smoothing

- Smoothing is the task of adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities.
- The name comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward.
- Smoothing not only prevents zero probabilities, attempts to improve the accuracy of the model as a whole.

Laplacian smoothing

Idea:

- Pull some probability from frequent bigrams to infrequent ones
- Just add 1 to the counts (add-one smoothing):

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i) + 1}{c(w_{i-n+1}^{i-1}) + V}$$

- Or tune a parameter (add-k smoothing):

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i) + k}{c(w_{i-n+1}^{i-1}) + Vk}$$

Katz backoff

Problem:

- Longer n-grams are better, but data is not always enough

Idea:

- Try a longer n-gram and back off to shorter if needed

$$\hat{p}(w_i | w_{i-n+1}^{i-1}) = \begin{cases} \tilde{p}(w_i | w_{i-n+1}^{i-1}), & \text{if } c(w_{i-n+1}^i) > 0 \\ \alpha(w_{i-n+1}^{i-1}) \hat{p}(w_i | w_{i-n+2}^{i-1}), & \text{otherwise} \end{cases}$$

Interpolation smoothing

Idea:

- Let us have a mixture of several n-gram models
- Example for a trigram model:

$$\hat{p}(w_i | w_{i-2}w_{i-1}) = \lambda_1 p(w_i | w_{i-2}w_{i-1}) + \lambda_2 p(w_i | w_{i-1}) + \lambda_3 p(w_i)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

- The weights are optimized on a test (dev) set
- Optionally they can also depend on the context

Absolute discounting

Idea:

- Let's compare the counts for bigrams in train and test sets

Experiment (Church and Gale, 1991):

- Subtract 0.75 and get a good estimate for the test count!

Train bigram count	Test bigram count
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21

$$\hat{p}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i) - d}{\sum_x c(w_{i-1}x)} + \lambda(w_{i-1})p(w_i)$$

Kneser-Ney smoothing

Idea:

- The unigram distribution captures the word frequency
- We need to capture the diversity of contexts for the word

$$\hat{p}(w) \propto |\{x : c(x w) > 0\}|$$

This is the ... **malt**
Kong

- Probably, the most popular smoothing technique

Kneser Ney Smoothing

- Augments Absolute Discounting by a more intuitive way to handle backoff distribution.
- Shannon Game : Predict the next word....
 - I can't see without my reading _____.
 - E.g. suppose the required bigram “reading glasses” is absent in the training corpus.
 - Backing off to unigram model, it is observed that “Fransisco” is more common than “glasses”.
 - But, information that “Fransisco” always follows “San” is not at all used, as backed off model is simple unigram model $P(w)$.

Kneser Ney Smoothing (contd.)

- Kneser and Ney, 1995 proposed-
 - Instead of $P(w)$ i.e. “how likely is w ”.
 - Use $P_{\text{continuation}}(w)$ i.e. “how likely w can occur as a novel continuation”.
- This continuation probability is proportional to number of distinct bigrams $(*,w)$ that w completes

$$P_{\text{continuation}}(w_n) = \frac{|\{w_{n-1} : C(w_{n-1}, w_n) > 0\}|}{\sum_{w_n} |\{w_{n-1} : C(w_{n-1}, w_n) > 0\}|}$$

- Final expression:

$$\begin{aligned} P(w_n | w_{n-1}) &= \frac{C(w_n, w_{n-1}) - D}{C(w_{n-1})}, \text{ if } C(w_n, w_{n-1}) > 0 \\ &= \alpha(w_{n-1}) P_{\text{continuation}}(w_n), \text{ otherwise} \end{aligned}$$

Smoothing techniques:

- Add-one (add-k) smoothing
- Katz backoff
- Interpolation smoothing
- Absolute discounting
- Kneser-Ney smoothing

N-gram models + Kneser-Ney smoothing is a strong baseline in Language Modeling!

SEQUENCE TAGGING WITH PROBABILISTIC MODEL:

Sequence labeling

Problem

- Given a sequence of *tokens*, infer the most probable sequence of *labels* for these tokens.

Examples

- part of speech tagging
- named entity recognition

Example: part-of-speech tagging

Tokens are words, and labels are PoS tags.

PRON

VERB

DET

PROPN

NOUN

I

saw

a

Heffalump

today.

PoS tags from Universal Dependencies project

Open class words	
ADJ	adjective
ADV	adverb
INTJ	interjection
NOUN	noun
PROPN	proper noun
VERB	verb

Other	
PUNCT	punctuation
SYM	symbol
X	other

Closed class words	
ADP	adposition
AUX	auxiliary verb
CCONJ	coordinating conjunction
DET	determiner
NUM	numeral
PART	particle
PRON	pronoun
SCONJ	subordinating conjunction

Example: named entity recognition

Once upon a time, a very long time ago now, about **[last Friday]**, **[Winnie-the-Pooh]** lived in a forest all by himself under the name of **[Sanders]**.

Types of named entities

Any real-world object which may have a proper name:

- persons
- organizations
- locations
- ...

Also named entities usually include:

- dates and times
- units
- amounts

Approaches to sequence labeling

1. Rule-based models (example: EngCG tagger)
2. Separate label classifiers for each token
3. Sequence models (HMM, MEMM, CRF)
4. Neural networks

PoS tagging with HMMs

Notation:

$\mathbf{x} = x_1, \dots, x_T$ is a sequence of words (input)

$\mathbf{y} = y_1, \dots, y_T$ is a sequence of their tags (labels)

We need to find the most probable sequence of tags given the sentence:

$$\mathbf{y} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$$

But first, let us define the model...

Hidden Markov Model

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y}) p(\mathbf{y}) \approx \prod_{t=1}^T p(x_t|y_t)p(y_t|y_{t-1})$$

observable hidden

Markov assumption:

$$p(\mathbf{y}) \approx \prod_{t=1}^T p(y_t|y_{t-1})$$

Output independence:

$$p(\mathbf{x}|\mathbf{y}) \approx \prod_{t=1}^T p(x_t|y_t)$$

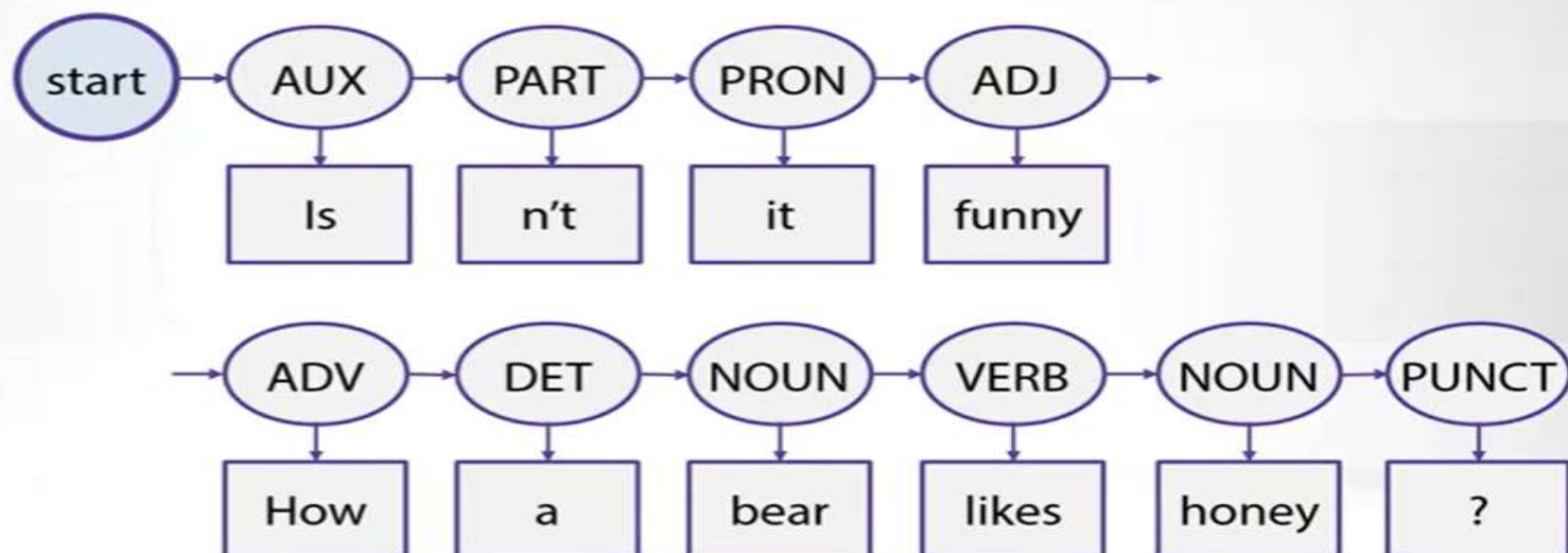
Text generation in HMM

Assume that the text is generated in the following manner:

- One chooses the next PoS tag given the previous tag
- Given the current tag, one generates another word

Thus, the neighboring words do not depend on each other, but they depend on the underlying tags.

Text generation in HMM: an example



Formal definition of HMM

A Hidden Markov Model is specified by:

1. The set $S = s_1, s_2, \dots, s_N$ of hidden states
2. The start state s_0
3. The matrix A of transition probabilities: $a_{ij} = p(s_j | s_i)$
4. The set O of possible visible outcomes
5. The matrix B of output probabilities: $b_{ik} = p(o_k | s_i)$

How to train the model?



If we could see the tags in train set, we would count:

$$a_{ij} = p(s_j | s_i) = \frac{c(s_i \rightarrow s_j)}{c(s_i)}$$

$$b_{ik} = p(o_k | s_i) = \frac{c(s_i \rightarrow o_k)}{c(s_i)}$$

Supervised case: MLE

The same in more formal terms (this is MLE!):

$$a_{ij} = p(s_j | s_i) = \frac{\sum_{t=1}^T [y_{t-1} = s_i, y_t = s_j]}{\sum_{t=1}^T [y_t = s_i]}$$

Note that the corpus is considered as a single sequence of length T with special states between the sentences.

But we do not see the labels! How to train the model?

Baum-Welch algorithm (a sketch)

E-step: posterior probabilities for hidden variables:

$$p(y_{t-1} = s_i, y_t = s_j)$$

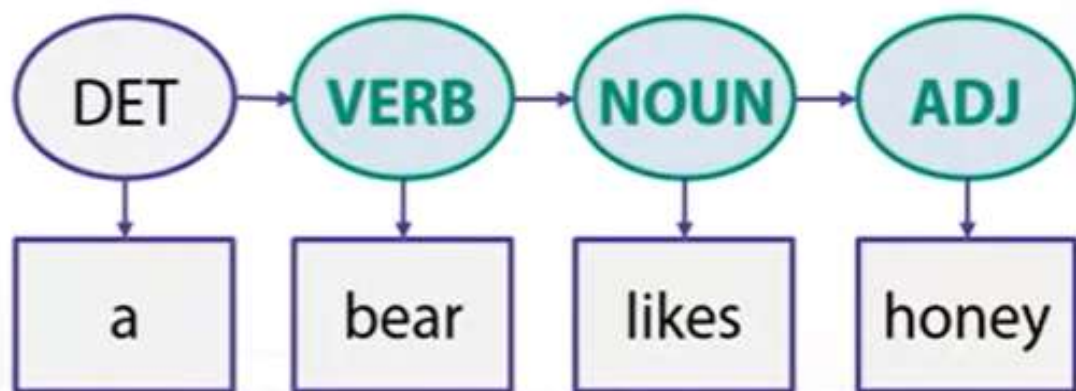
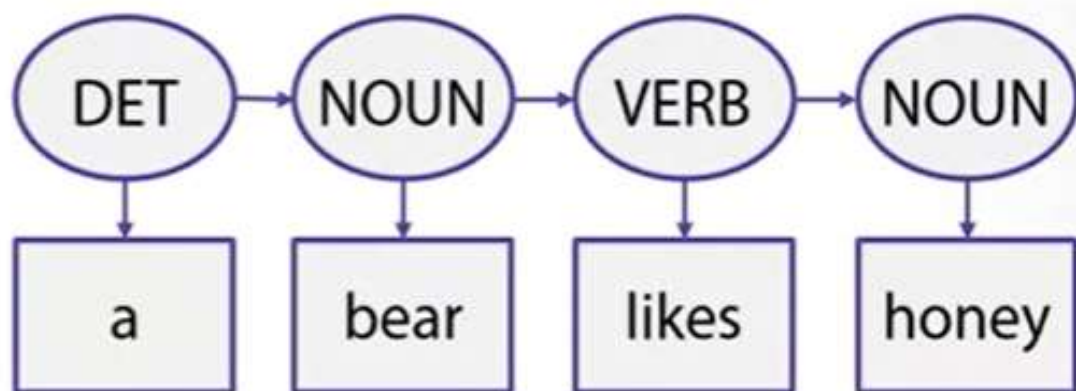
Can be effectively done with dynamic programming
(forward-backward algorithm)

M-step: maximum likelihood updates for the parameters:

$$a_{ij} = p(s_j | s_i) = \frac{\sum_{t=1}^T p(y_{t-1} = s_i, y_t = s_j)}{\sum_{t=1}^T p(y_t = s_i)}$$

Motivation


The same output sentence can be generated by different sequences of hidden states:



Decoding in HMM

$$p(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^T p(y_t | y_{t-1}) p(x_t | y_t)$$

Transition probabilities Output probabilities



Decoding problem:

What is the most probable sequence of hidden states?

$$\mathbf{y} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y} | \mathbf{x}) = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$$

Solve this problem efficiently using dynamic programming!

Viterbi decoding

Let $Q_{t,s}$ be the most probable sequence of hidden states of length t that finishes in the state s and generates o_1, \dots, o_t . Let $q_{t,s}$ be the probability of this sequence.

Then $q_{t,s}$ can be computed dynamically:

$$q_{t,s} = \max_{s'} q_{t-1,s'} \cdot p(s|s') \cdot p(o_t|s)$$


Transition probabilities Output probabilities

$Q_{t,s}$ can be determined by remembering the argmax.

An example of HMM: transition probabilities

Suppose that we have the following PoS tags:
ADJ, NOUN, VERB.

Consider the transition probabilities between tags:

from \ to	ADJ	NOUN	VERB
ADJ	0.4	0.4	0.2
NOUN	0.2	0.4	0.4
VERB	0.1	0.6	0.3

Note that the sum of probabilities in each row is equal to 1.

Let all initial state probabilities be equal (to 1/3).

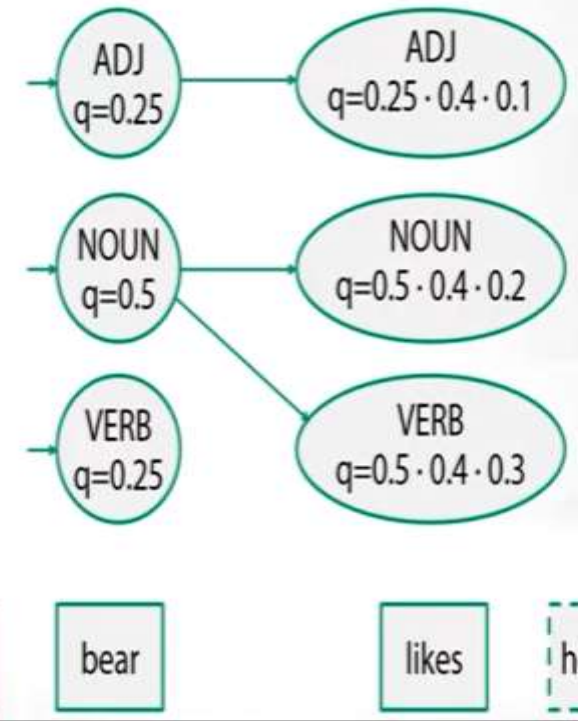
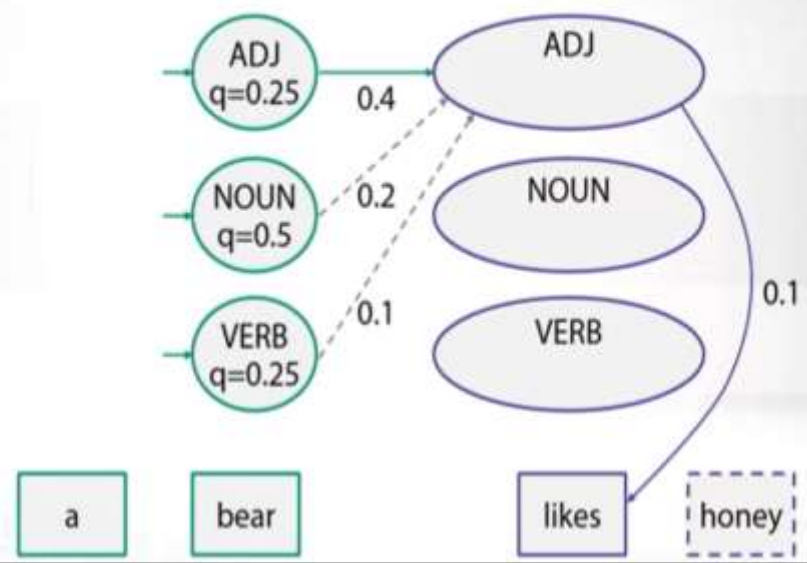
An example of HMM: output probabilities

Consider the following output probabilities for the vocabulary:

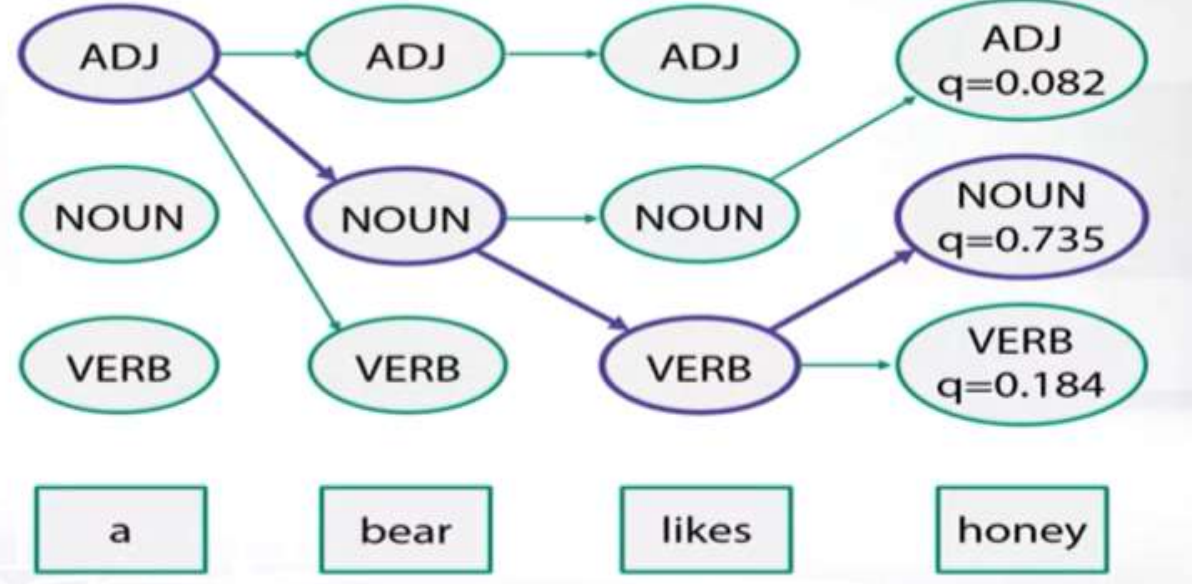
tag\word	a	bear	fly	honey	likes	sweet
ADJ	0.2	0.1	0.1	0.1	0.1	0.4
NOUN	0.1	0.2	0.2	0.2	0.2	0.1
VERB	0.1	0.2	0.2	0.1	0.3	0.1

The probabilities in each row also sum to 1.

The best transition to ADJ



Remember the best transitions!



Backtrace

Viterbi algorithm

Input: observations of length T , state-graph of length N

Output: best-path

for each state s from 1 to N do

$$q[1, s] \leftarrow p(s|s_0) \cdot p(o_1|s)$$

$$\text{backpointers}[1, s] \leftarrow 0$$

for each time step t from 2 to T do

for each state s from 1 to N do

$$q[t, s] \leftarrow \max_{s'=1}^N q[t-1, s'] \cdot p(s|s') \cdot p(o_t|s)$$

$$\text{backpointers}[t, s] \leftarrow \operatorname{argmax}_{s'=1}^N q[t-1, s'] \cdot p(s|s')$$

$$s \leftarrow \operatorname{argmax}_{s'=1}^N q[T, s']$$

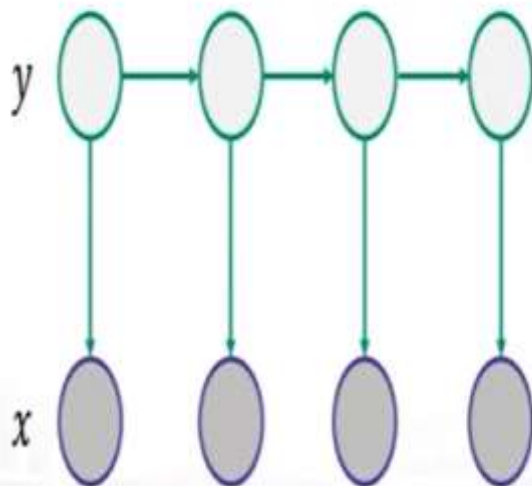
return the backtrace path from backpointers $[T, s]$



Hidden Markov Model

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^T p(y_t | y_{t-1}) p(x_t | y_t)$$

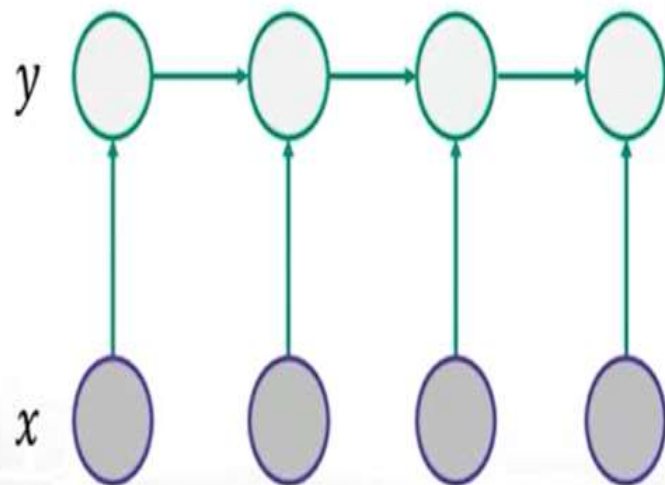
↑
Generative
model



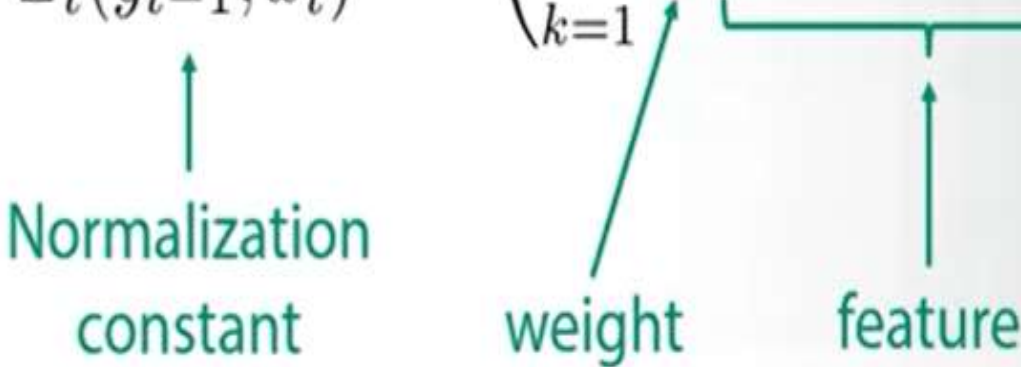
Maximum Entropy Markov Model

$$p(\mathbf{y} | \mathbf{x}) = \prod_{t=1}^T p(y_t | y_{t-1}, x_t)$$

↑
Discriminative
model



Maximum Entropy Markov Model

$$p(y_t | y_{t-1}, x_t) = \frac{1}{Z_t(y_{t-1}, x_t)} \exp \left(\sum_{k=1}^K \theta_k f_k(y_t, y_{t-1}, x_t) \right)$$


Normalization
constant

weight

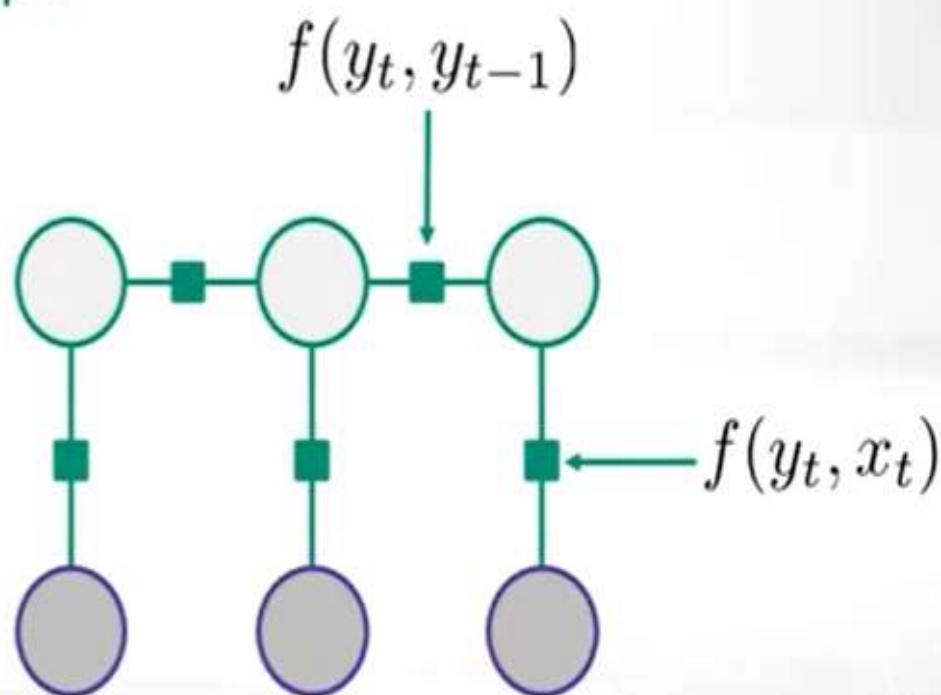
feature

Similar to Soft max applied to logistic regression

Conditional Random Field (linear chain)

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^T \exp \left(\sum_{k=1}^K \theta_k f_k(y_t, y_{t-1}, x_t) \right)$$

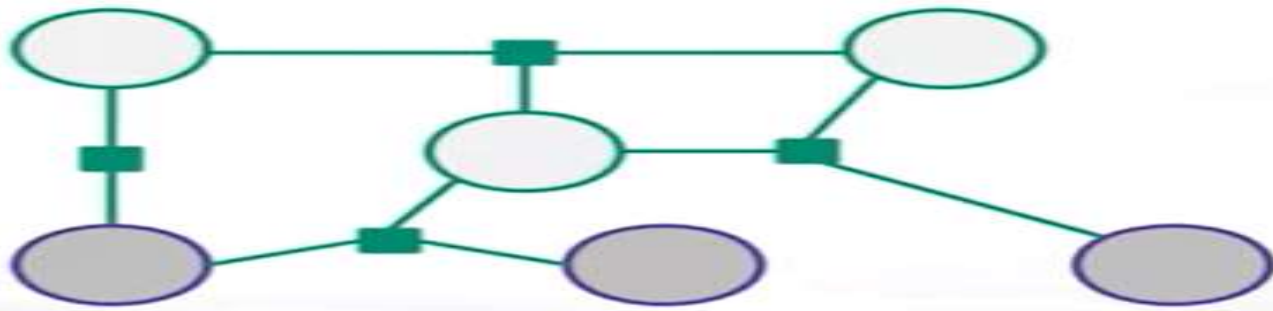
Undirected graph:



Conditional Random Field (general form)

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{a=1}^A \Psi_a(y_a, x_a)$$

↑
Arbitrary factors



CRF++	https://sourceforge.net/projects/crfpp/
MALLET	http://mallet.cs.umass.edu/
GRMM	http://mallet.cs.umass.edu/grmm/
CRFSuite	http://www.chokkan.org/software/crfsuite/
FACTORIE	http://www.factorie.cc

**BLACK BOX
IMPLEMENTATIONS
FOR
CRF!**

Features engineering

Label-observation features:

- $f(y_t, y_{t-1}, x_t) = [y_t = y] g_m(x_t)$
- $f(y_t, y_{t-1}, x_t) = [y_t = y][y_{t-1} = y']$
- $f(y_t, y_{t-1}, x_t) = [y_t = y][y_{t-1} = y'] g_m(x_t)$

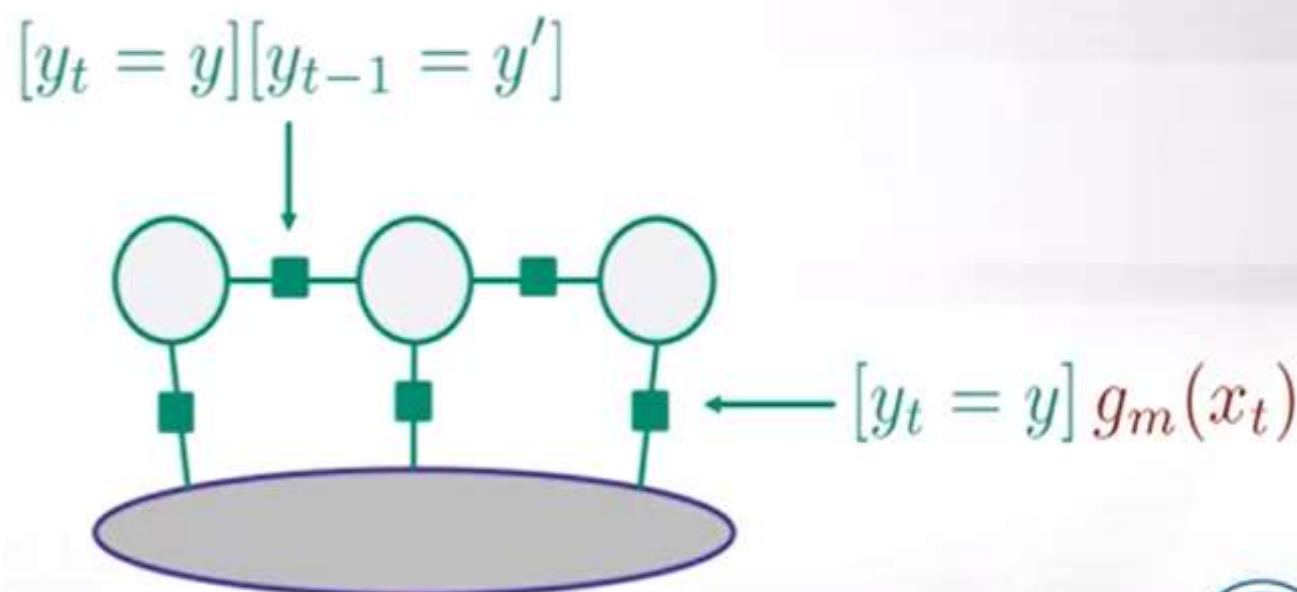
Observation function examples

	$w_t = v$	$\forall v \in V$
	part-of-speech tag for w_t is j	$\forall \text{ tags } j$
	w_t is in a phrase of syntactic type j	$\forall \text{ tags } j$
Capitalized	w_t matches $[A-Z][a-z]^+$	
AllCaps	w_t matches $[A-Z]^+$	
EndsInDot	w_t matches $[\wedge.]^+\backslash\cdot$	
	w_t matches a dash	
	w_t appears in a list of stop words	
	w_t appears in list of capitals	

Dependencies on input

Trick: Pretend the current input x_t contains not only the current word w_t , but also w_{t-1} and w_{t+1} and build observation functions for them as well.

The model is discriminative, so we can use the whole input:



Resume for the lesson

Probabilistic graphical models:

- Hidden Markov Models (generative, directed)
- Maximum Entropy Markov Models (discriminative, directed)
- Conditional Random Field (discriminative, undirected)

Tasks:

- Training – fit parameters (Baum-Welch for HMM)
- Decoding – find the most probable tags (Viterbi for HMM)

Practice:

- Features engineering + black-box implementation