Lecture 5: Linear Sequential Models

André Martins

Deep Structured Learning Course, Fall 2018

Announcements

- The project proposal deadline is today.
- Homework 1 has been graded and the solutions have been sent out through Piazza.
- Need to reschedule October 31 class: can we do October 29 (Monday) same time?

Today's Roadmap

Today we're starting to talk about **structure**, more specifically sequences:

- Generative sequence models: Markov models and hidden Markov models
- Dynamic programming: the Viterbi and Forward-Backward algorithms
- Viterbi decoding and minimum risk decoding
- Unsupervised learning with the Baum-Welch (EM) algorithm
- Brown clustering
- Discriminative sequence models: structured perceptron, conditional random fields, structured SVMs

Outline

Structured Prediction

Structured prediction: a machine learning framework for predicting structured, constrained, and interdependent outputs

NLP deals with *structured* and *ambiguous* textual data:

- machine translation, speech recognition, parsing, ...

Computer vision, computational biology, etc. deal with structured outputs too:

- image segmentation, image parsing, ...
- protein folding, protein design, ...
- time series prediction, ...

Sequence Labeling

Some of these tasks are sequential in nature...

Example: POS Tagging

Map **sentences** to sequences of **part-of-speech tags.**

Time	flies	like	an	arrow	.
noun	verb	prep	det	noun	.

- Need to predict a morphological tag for each word of the sentence
- High correlation between adjacent words!

(Ratnaparkhi, 1999; Brants, 2000; Toutanova et al., 2003)

Example: Named Entity Recognition

From **sentences** extract **named entities.**

Louis	Elsevier	was	born	in	Leuven	.
B-PER	I-PER	O	O	O	B-LOC	.

- Identify word segments that refer to entities (person, organization, location)
- Typically done with sequence models and B-I-O tagging

(Zhang and Johnson, 2003; Ratinov and Roth, 2009)

Notation

- Input set $\mathcal{X}$ (e.g., sentences)

Notation

- Input set $\mathcal{X}$ (e.g., sentences)
- Output set $\mathcal{Y}$ (e.g. POS sequences)—**large and structured!**
  - How many possible outputs for a sentence with length L, assuming K possible tags?

Notation

- Input set $\mathcal{X}$ (e.g., sentences)
- Output set $\mathcal{Y}$ (e.g. POS sequences)—**large and structured!**
  - How many possible outputs for a sentence with length L, assuming K possible tags? $O(K^L)$

Notation

- Input set $\mathcal{X}$ (e.g., sentences)
- Output set $\mathcal{Y}$ (e.g. POS sequences)—**large and structured!**
  - How many possible outputs for a sentence with length L, assuming K possible tags? $O(K^L)$
- Weight vector $\boldsymbol{w} \in \mathbb{R}^m$, feature function $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Linear compatibility function $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$

Notation

- Input set $\mathcal{X}$ (e.g., sentences)
- Output set $\mathcal{Y}$ (e.g. POS sequences)—**large and structured!**
  - How many possible outputs for a sentence with length L, assuming K possible tags? $O(K^L)$
- Weight vector $\boldsymbol{w} \in \mathbb{R}^m$, feature function $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Linear compatibility function $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$
- **Training problem:** learn $\boldsymbol{w}$ from data $\mathcal{T} = \{\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle\}_{t=1}^{|\mathcal{T}|}$

Notation

- Input set $\mathcal{X}$ (e.g., sentences)
- Output set $\mathcal{Y}$ (e.g. POS sequences)—**large and structured!**
  - How many possible outputs for a sentence with length L, assuming K possible tags? $O(K^L)$
- Weight vector $\boldsymbol{w} \in \mathbb{R}^m$, feature function $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- Linear compatibility function $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$
- **Training problem:** learn $\boldsymbol{w}$ from data $\mathcal{T} = \{\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle\}_{t=1}^{|\mathcal{T}|}$
- **Decoding problem:**

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$$

This is what makes structured prediction special!

The Decoding Problem

Also called "the inference problem":

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$$

- In multi-class classification, we just enumerate the scores and pick the argmax
- **But in structure prediction, $\mathcal{Y}$ is too large—it is intractable to enumerate!!**

Key assumption: $F_{\boldsymbol{w}}$ decomposes into (overlapping) *parts*

- For example: for sequences, scores may factor as a sum over label n-grams

Roadmap: Models for Structured Prediction

Binary/Multi-class	Structured Prediction
Naive Bayes	?
Logistic Regression	?
Perceptron	?
SVMs	?

Outline

Outline

Key Problem

How to define a probability distribution over a string (a sequence of discrete symbols)?

- If the length of the string is unbounded, there are infinitely many strings
- How to make sure the distribution normalizes to 1?
- So far, we only talked about distributions on finite random variables...

Probability Distributions Over Strings

- Alphabet Σ (e.g. words in the vocabulary)
- Set of strings: $\Sigma^\star := \{\epsilon\} \cup \Sigma \cup \Sigma^2 \ldots$
- $\Sigma^\star$ is a countably infinite set
- We want to define a probability distribution over $\Sigma^\star$
- This distribution must normalize properly, $\sum_{\boldsymbol{x} \in \Sigma^\star} \mathbb{P}(\boldsymbol{x}) = 1$

Trivial Choices

- Probability 0 to sequences greater than L, uniform otherwise
- Given a sample of N sequences, assign a uniform distribution of $1/N$ to each observed sequence, 0 if the sentence is not in the sample

What if we want *every* sequence to have some probability?

Lower Extreme: Bag-of-Words Model

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L, \text{STOP}) = \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{x}_i)$$

- Also called "unigram" model
- Assumes every word is generated independently of other words
- Probability of a string is insensitive to word order
- How many parameters?
- **Not structured prediction!**

Upper Extreme: Full History Model

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L, \text{STOP}) = \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{i-1})$$

- Assumes the generation of each word depends on the entire history (*all* the previous words)
- Huge expressive power!
- But: too many parameters to estimate! (How many?)
- Cannot generalize well

In-Between: First Order Markov Models

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L, \text{STOP}) = \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{x}_{i-1})$$

- Each word only depends on the previous word
- Which parameters need to be estimated?
- Transition probabilities $\mathbb{P}(\boldsymbol{x}_i | \boldsymbol{x}_{i-1})$...
- ... including initial and final probabilities $\mathbb{P}(\boldsymbol{x}_1 | \text{START})$ and $\mathbb{P}(\text{STOP} | \boldsymbol{x}_L)$
- Total number of parameters: $O(|\Sigma|^2)$

In-Between: Kth Order Markov Models

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L, \text{STOP}) = \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_{i-K})$$

- Each word depends on the Kth previous words
- Transition probabilities $\mathbb{P}(\boldsymbol{x}_i | \boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_{i-K})$
- Total number of parameters: $O(|\Sigma|^{K+1})$
- **Widely used in language modeling**
- **Applications: machine translation, speech recognition, OCR...**

Markov Models: How to Estimate the Parameters

How to estimate these transition probabilities $\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_{i-K})$ from data?

- Maximum likelihood estimation
- Boils down to **counting and normalizing**
- What if we never observe a particular K-gram at training time? What happens at test time?

Problem: Data Sparsity

- As K grows, we suffer more from insufficient data: there are many K-grams with zero counts, therefore the maximum likelihood estimate will be $\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{x}_{i-1},\ldots,\boldsymbol{x}_{i-K}) = 0$
- This is undesirable!
- Example: a language model which gives probability 0 to unseen words
- Just because an event has never been observed in training data does not mean it cannot occur in test data
- So, if the count is zero, what should $\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{x}_{i-1},\ldots,\boldsymbol{x}_{i-K}) = 0$ be?
- We need **smoothing**!

Smoothing

Many smoothing techniques have been proposed:

- Additive smoothing (e.g. add-one, a.k.a. Laplace smoothing)
- Good-Turing estimate
- Jelinek-Mercer smoothing (interpolation)
- Katz smoothing (backoff)
- Witten-Bell smoothing
- Absolute discounting
- Kneser-Ney smoothing
- Modified Kneser-Ney smoothing (considered the best)

This is a black art...

Markov models are extremely simple.

How well do they perform?

Example

- Unigram model estimated on 2.8M words of political blog text.

  this trying our putting and funny and among it herring it obama but certainly foreign my c on byron again but from i i so and i chuck yeah the as but but republicans if this stay oh so or it mccain bush npr this with what and they right i while because obama

(Adapted from Noah Smith's slides.)

Example

- Bigram model estimated on 2.8M words of political blog text.

the lack of the senator mccain hadn t keep this story backwards while showering praise of the kind of gop weakness it was mistaken for american economist anywhere in the white house press hounded the absence of those he s as a wide variety of this election day after the candidate b richardson was polled ri in hempstead moderated by the convention that he had zero wall street journal argues sounds like you may be the primary but even close the bill told c e to take the obama on the public schools and romney fred flinstone s see how a lick skillet road it s little sexist remarks

(Adapted from Noah Smith's slides.)

Example

- Trigram model estimated on 2.8M words of political blog text.

as i can pin them all none of them want to bet that any of the might be conservatism unleashed into the privacy rule book and when told about what paul fans organized another massive fundraising initiative yesterday and i don t know what the rams supposedly want ooh but she did but still victory dinner alone among republicans there are probably best not all of the fundamentalist community asked for an independent maverick now for crystallizing in one especially embarrassing

(Adapted from Noah Smith's slides.)

Example

- 5-gram model estimated on 2.8M words of political blog text.

he realizes fully how shallow and insincere conservative behavior has been he realizes that there is little way to change the situation this recent arianna huffington item about mccain issuing heartfelt denials of things that were actually true or for that matter about the shia sunni split and which side iran was on would get confused about this any more than someone with any knowledge of us politics would get confused about whether neo confederates were likely to be supporting the socialist workers party at the end of the world and i m not especially discouraged now that newsweek shows obama leading by three now

(Adapted from Noah Smith's slides.)

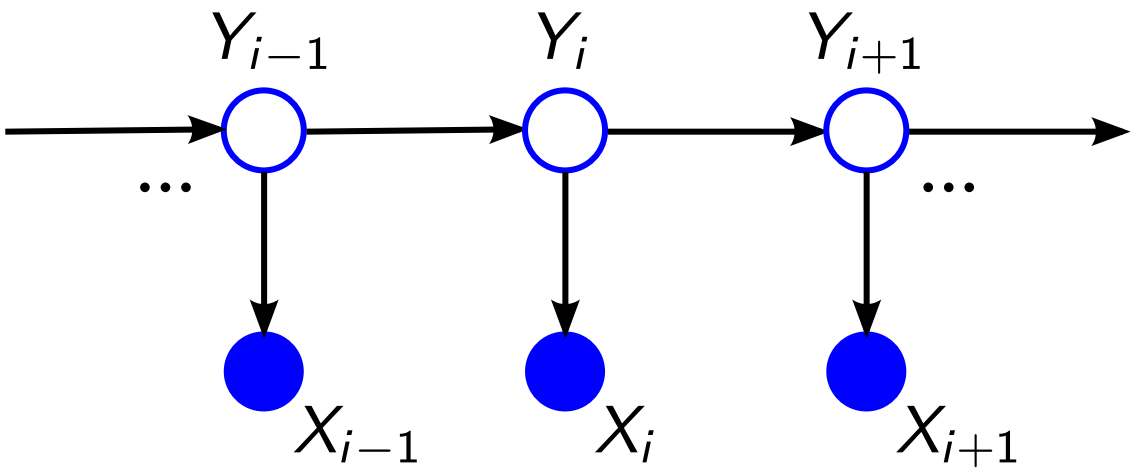
Outline

Hidden Markov Models

- Assume each word x_i is associated with a **hidden state** y_i
- Two alphabets: word alphabet Σ **and state alphabet** Λ
- States are generated according to a (first-order) Markov model
- Word emissions are conditioned on each state
- **Applications: POS tagging, named entity recognition, shallow parsing, ...**

Hidden Markov Models

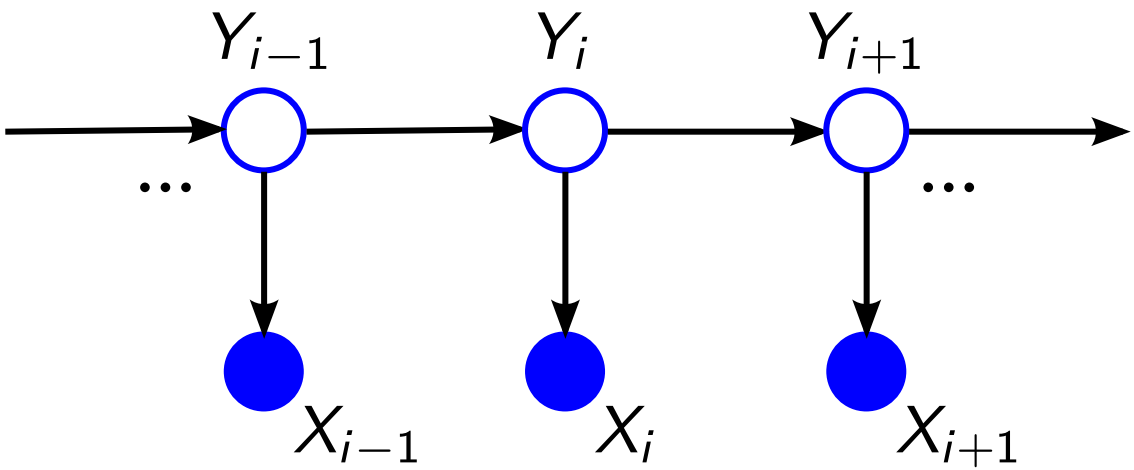
$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \ldots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP}) = \prod_{i=1}^{L+1} \underbrace{\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})}_{\text{transitions}} \times \prod_{i=1}^{L} \underbrace{\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)}_{\text{emissions}}$$



What are the parameters?

Hidden Markov Models

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \ldots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP}) = \prod_{i=1}^{L+1} \underbrace{\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})}_{\text{transitions}} \times \prod_{i=1}^{L} \underbrace{\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)}_{\text{emissions}}$$

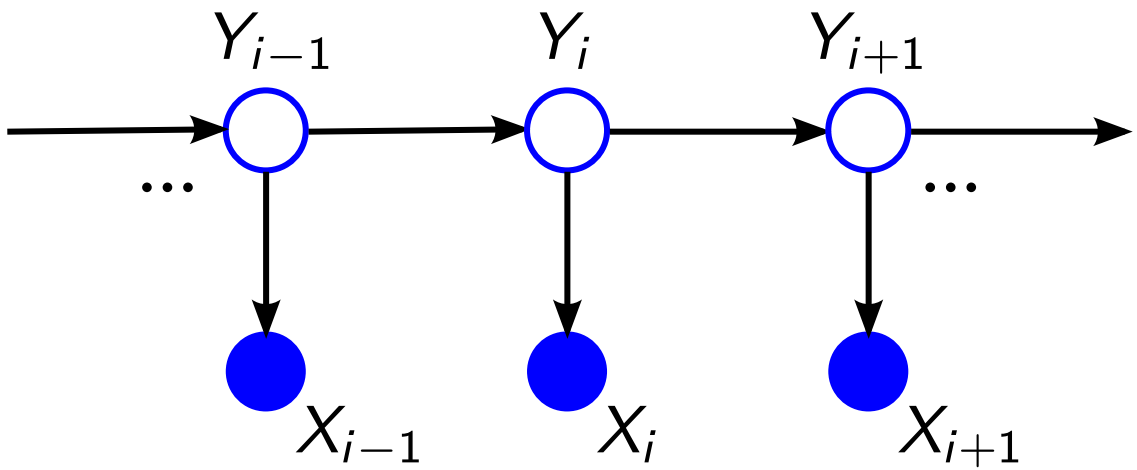


What are the parameters?

- **Transition probabilities** $\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})$
- **Emission probabilities** $\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)$

Hidden Markov Models

$$\mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{x}_2, \boldsymbol{y}_2, \ldots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP}) = \prod_{i=1}^{L+1} \underbrace{\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})}_{\text{transitions}} \times \prod_{i=1}^{L} \underbrace{\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)}_{\text{emissions}}$$



What are the parameters?

- **Transition probabilities** $\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})$
- **Emission probabilities** $\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)$
- Total number of parameters: $O(|\Sigma||\Lambda| + |\Lambda|^2)$

Sampling from an HMM

Two equivalent ways of sampling:

1. Sample the sequence of hidden states using a Markov model $\mathbb{P}(\boldsymbol{y}_i \mid \boldsymbol{y}_{i-1})$; then for each state $\boldsymbol{y}_i$, sample a word from $\mathbb{P}(\boldsymbol{x}_i \mid \boldsymbol{y}_i)$
2. Alternate between sampling the next state and the current word from the current state

Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1,\ldots\boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L|\boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

We'll see that:

- Problems 1 and 2 can be solved with **dynamic programming**
- Problem 3 can be solved by **counting and normalizing**

Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

We'll see that:

- Problems 1 and 2 can be solved with **dynamic programming**
- Problem 3 can be solved by **counting and normalizing**

Dynamic Programming

A central concept applicable to many structured prediction problems

Key idea: use a table data structure to store partial quantities that will be reused many times

Optimal substructure: best solution to a problem relies on best solutions to its (similar-looking) subproblems

Overlapping subproblems: reuse a small number of quantities many times

Examples:

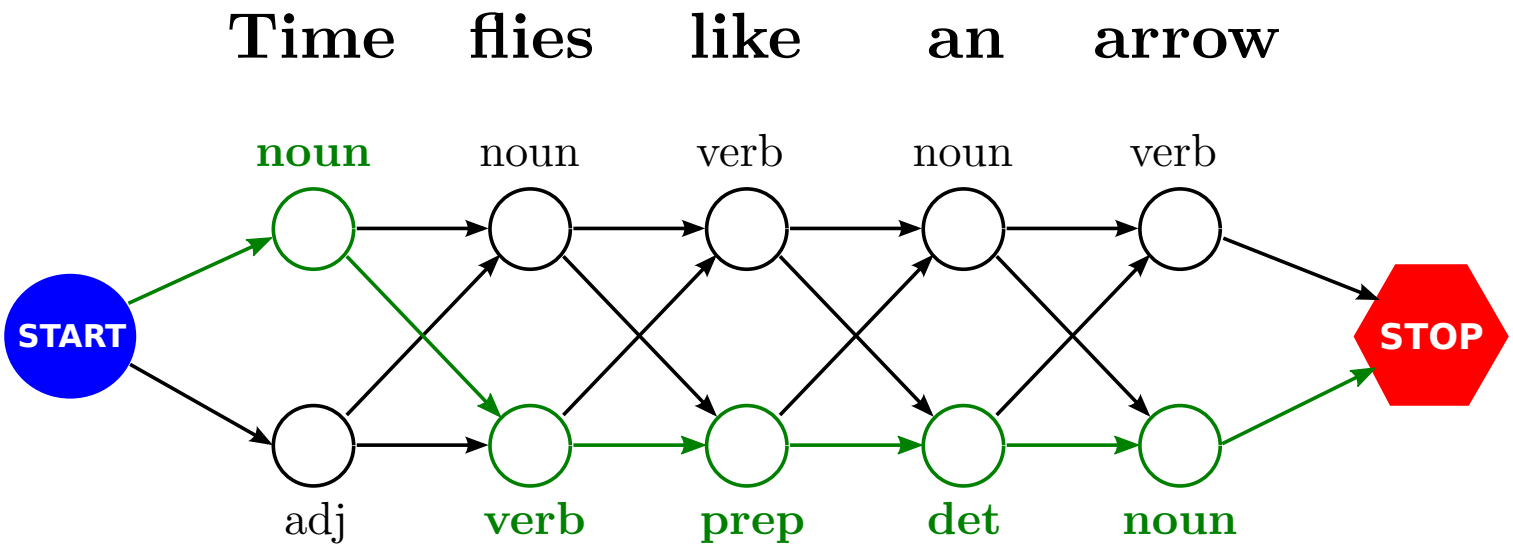
- Viterbi
- Levenshtein distance (a.k.a. edit distance)
- Dijkstra's shortest path algorithm
- Bellman equation in optimal control theory

Problem 1: Most Likely State Sequence

$$
\begin{aligned}
\widehat{\boldsymbol{y}}_1, \dots \widehat{\boldsymbol{y}}_L &= \arg \max_{\boldsymbol{y}_1, \cdots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_L | \boldsymbol{x}) \\
&= \arg \max_{\boldsymbol{y}_1, \cdots \boldsymbol{y}_L} \mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \dots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP}) \\
&= \arg \max_{\boldsymbol{y}_1, \cdots \boldsymbol{y}_L} \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_{i-1}) \times \prod_{i=1}^{L} \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{y}_i)
\end{aligned}
$$

- **Combinatorial problem: need to search over $|\Lambda|^L$ possibilities.**

Viterbi Trellis



- Each word's state depends on the word and on the nearby labels
- Key fact: given adjacent labels, the others do not matter

The Viterbi algorithm

- Key idea: recurrence
- Traverse the sequence left-to-right and compute the best way of reaching the ith word given the possible decisions at the $(i-1)$th word
- Fill a table along the way, then backtrack to recover the best sequence of states

$$V(i, \boldsymbol{y}_i) = \max_{\boldsymbol{y}_{i-1} \in \Lambda} \left(\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{y}_i) \times V(i-1, \boldsymbol{y}_{i-1}) \right)$$

$$\psi(i, \boldsymbol{y}_i) = \arg \max_{\boldsymbol{y}_{i-1} \in \Lambda} \left(\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{x}_i | \boldsymbol{y}_i) \times V(i-1, \boldsymbol{y}_{i-1}) \right)$$

The Viterbi algorithm

input: sequence $x_1, \ldots, x_L$, emission/transition probabilities

Forward pass: incrementally fill the table

$V(1, y_1) = \mathbb{P}(y_1|\text{START}) \times \mathbb{P}(x_1|y_1) \quad \forall y_1 \in \Lambda$

for $i = 2$ **to** L **do**

  for $y_i \in \Lambda$ **do**

    $V(i, y_i) = \max_{y_{i-1}} \mathbb{P}(y_i|y_{i-1}) \times \mathbb{P}(x_i|y_i) \times V(i-1, y_{i-1})$

    $\psi(i, y_i) = \arg\max_{y_{i-1}} \mathbb{P}(y_i|y_{i-1}) \times \mathbb{P}(x_i|y_i) \times V(i-1, y_{i-1})$

  end for

end for

Backward pass: follow backpointers

$\widehat{y}_L = \arg\max_{y_L} \mathbb{P}(\text{STOP}|y_L) \times V(L, y_L)$

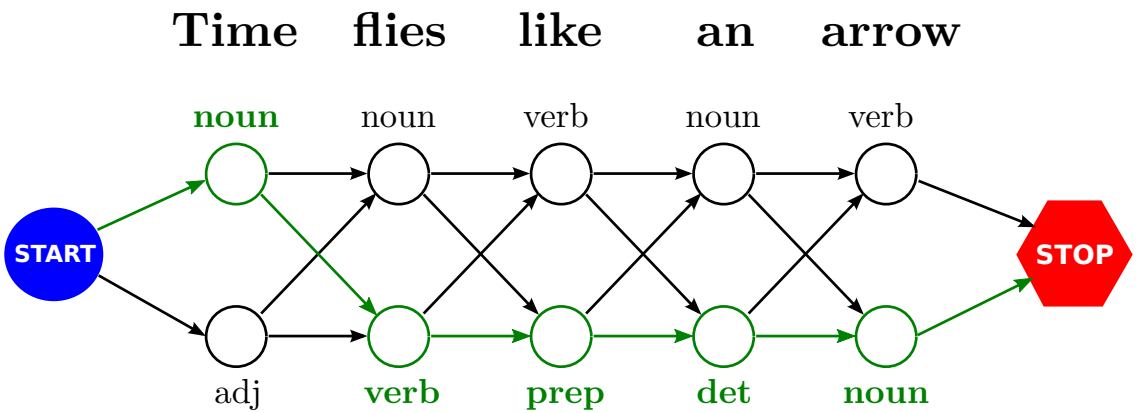
for $i = L - 1$ **to** 1 **do**

  $\widehat{y}_i = \psi(i + 1, \widehat{y}_{i+1})$

end for

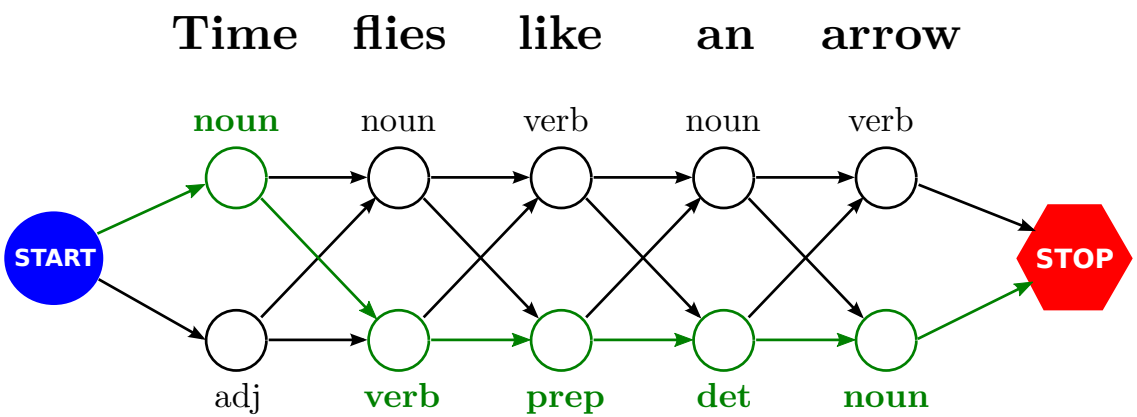
output: the best state sequence $\widehat{y}_1, \ldots, \widehat{y}_L$.

Example – $i = 1$



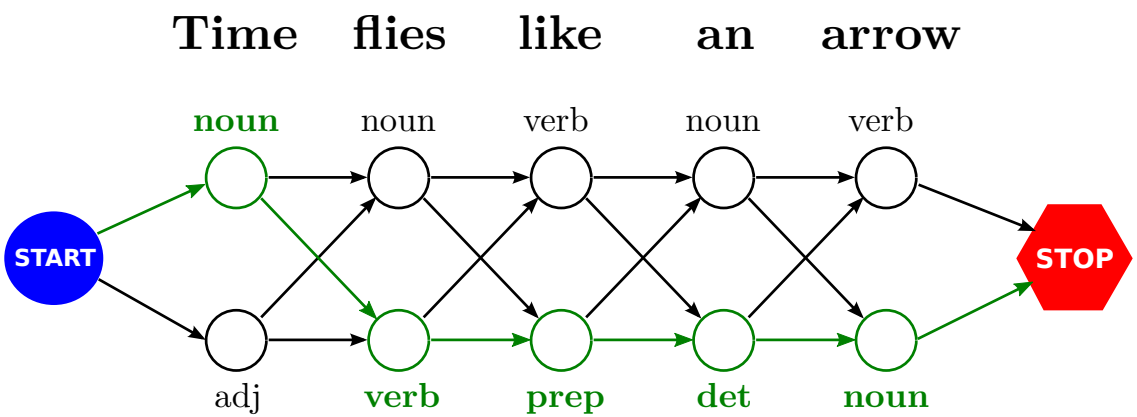
$$
\begin{aligned}
V(1, \text{noun}) &= \mathbb{P}(\text{noun}|\textsc{start}) \times \mathbb{P}(\text{“Time”}|\text{noun}) \\
&= 0.4 \times 0.1 = 0.04 \\
V(1, \text{adj}) &= \mathbb{P}(\text{adj}|\textsc{start}) \times \mathbb{P}(\text{“Time”}|\text{adj}) \\
&= 0.2 \times 0.05 = 0.01
\end{aligned}
$$

Example – $i = 2$



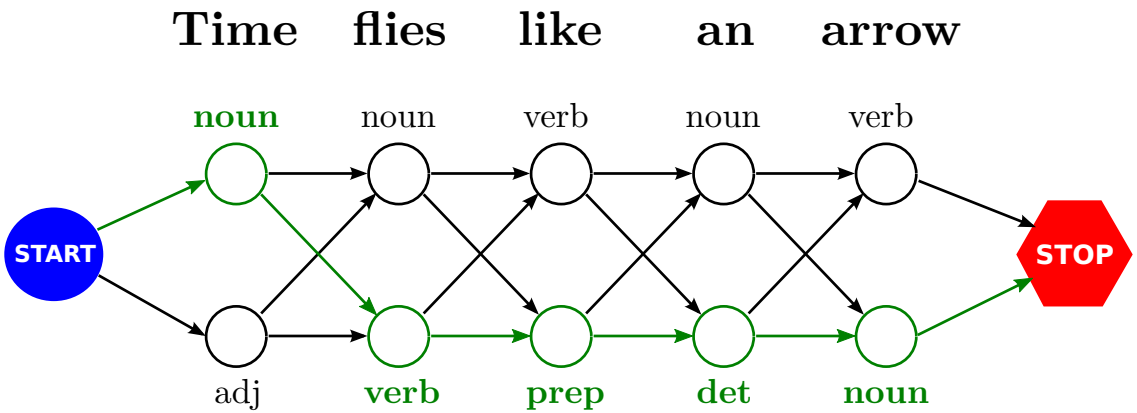
$$
\begin{aligned}
V(2, \text{noun}) &= \max\{\mathbb{P}(\text{noun}|\text{noun}) \times \mathbb{P}(\text{“flies”}|\text{noun}) \times V(1, \text{noun}), \\
&\qquad \mathbb{P}(\text{noun}|\text{adj}) \times \mathbb{P}(\text{“flies”}|\text{noun}) \times V(1, \text{adj})\} \\
&= \max\{0.1 \times 0.1 \times 0.04, 0.8 \times 0.1 \times 0.01\} = 0.0008 \\
\psi(2, \text{noun}) &= \text{adj} \\
V(2, \text{verb}) &= \max\{\mathbb{P}(\text{verb}|\text{noun}) \times \mathbb{P}(\text{“flies”}|\text{verb}) \times V(1, \text{noun}), \\
&\qquad \mathbb{P}(\text{verb}|\text{adj}) \times \mathbb{P}(\text{“flies”}|\text{verb}) \times V(1, \text{adj})\} \\
&= \max\{0.4 \times 0.2 \times 0.04, 0.01 \times 0.2 \times 0.01\} = 0.0032 \\
\psi(2, \text{verb}) &= \text{noun}
\end{aligned}
$$

Example – $i = 3$



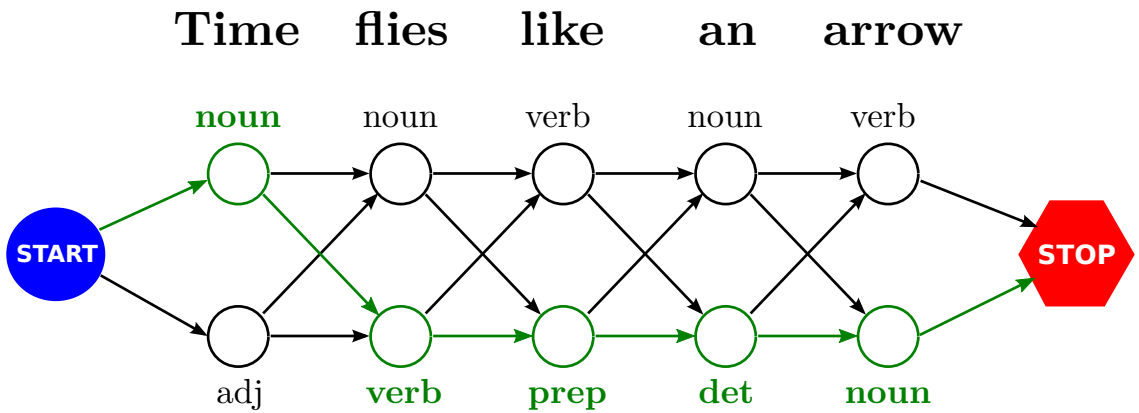
$$
\begin{aligned}
V(3, \text{verb}) &= \max\{\mathbb{P}(\text{verb}|\text{noun}) \times \mathbb{P}(\text{“like”}|\text{verb}) \times V(2, \text{noun}), \\
&\qquad \mathbb{P}(\text{verb}|\text{verb}) \times \mathbb{P}(\text{“like”}|\text{verb}) \times V(2, \text{verb})\} \\
&= \max\{0.6 \times 0.1 \times 0.0008, 0.01 \times 0.1 \times 0.0032\} = 4.8 \times 10^{-5} \\
\psi(3, \text{verb}) &= \text{noun} \\
V(3, \text{prep}) &= \max\{\mathbb{P}(\text{prep}|\text{noun}) \times \mathbb{P}(\text{“like”}|\text{prep}) \times V(2, \text{noun}), \\
&\qquad \mathbb{P}(\text{prep}|\text{verb}) \times \mathbb{P}(\text{“like”}|\text{prep}) \times V(2, \text{verb})\} \\
&= \max\{0.2 \times 0.5 \times 0.0008, 0.2 \times 0.5 \times 0.0032\} = 0.00032 \\
\psi(3, \text{prep}) &= \text{verb}
\end{aligned}
$$

Example – $i = 4$



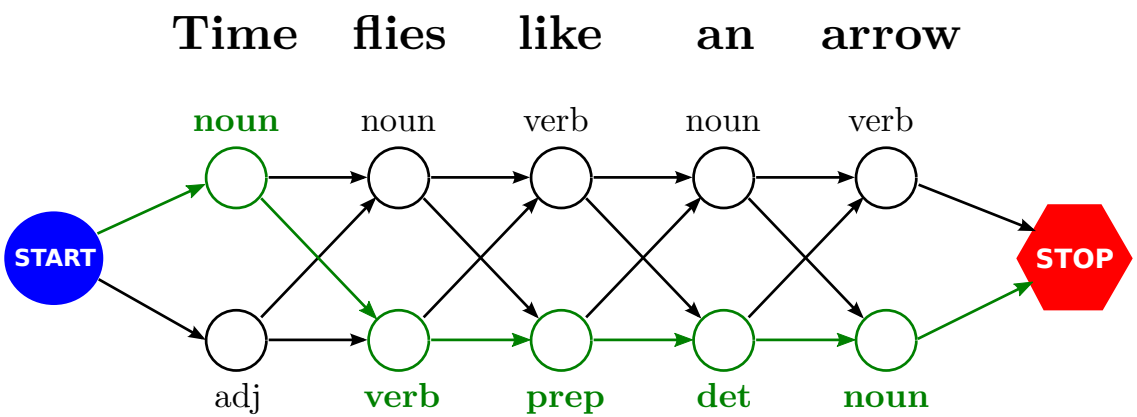
$$
\begin{aligned}
V(4, \text{noun}) &= \max\{\mathbb{P}(\text{noun}|\text{verb}) \times \mathbb{P}(\text{“an”}|\text{noun}) \times V(3, \text{verb}), \\
&\qquad \mathbb{P}(\text{noun}|\text{prep}) \times \mathbb{P}(\text{“an”}|\text{noun}) \times V(3, \text{prep})\} \\
&= \max\{0.4 \times 0.001 \times 4.8 \times 10^{-5}, 0.2 \times 0.001 \times 0.00032\} = 6.4 \times 10^{-8} \\
\psi(4, \text{noun}) &= \text{prep} \\
V(4, \text{det}) &= \max\{\mathbb{P}(\text{det}|\text{verb}) \times \mathbb{P}(\text{“an”}|\text{det}) \times V(3, \text{verb}), \\
&\qquad \mathbb{P}(\text{det}|\text{prep}) \times \mathbb{P}(\text{“an”}|\text{det}) \times V(3, \text{prep})\} \\
&= \max\{0.3 \times 0.5 \times 4.8 \times 10^{-5}, 0.5 \times 0.5 \times 0.00032\} = 8 \times 10^{-5} \\
\psi(4, \text{det}) &= \text{prep}
\end{aligned}
$$

Example – $i = 5$



$$
\begin{aligned}
V(5, \text{verb}) &= \max\{\mathbb{P}(\text{verb}|\text{noun}) \times \mathbb{P}(\text{“arrow”}|\text{verb}) \times V(4, \text{noun}), \\
&\qquad \mathbb{P}(\text{verb}|\text{det}) \times \mathbb{P}(\text{“arrow”}|\text{verb}) \times V(4, \text{det})\} \\
&= \max\{0.3 \times 0.01 \times 6.4 \times 10^{-8}, 0.01 \times 0.01 \times 8 \times 10^{-5}\} = 8 \times 10^{-9} \\
\psi(5, \text{verb}) &= \text{det} \\
V(5, \text{noun}) &= \max\{\mathbb{P}(\text{noun}|\text{noun}) \times \mathbb{P}(\text{“arrow”}|\text{noun}) \times V(4, \text{noun}), \\
&\qquad \mathbb{P}(\text{noun}|\text{det}) \times \mathbb{P}(\text{“arrow”}|\text{noun}) \times V(4, \text{det})\} \\
&= \max\{0.1 \times 0.1 \times 6.4 \times 10^{-8}, 0.9 \times 0.1 \times 8 \times 10^{-5}\} = 7.2 \times 10^{-6} \\
\psi(5, \text{noun}) &= \text{det}
\end{aligned}
$$

Example – Backward Pass



$$
\begin{aligned}
\text{Probability} &= \max\{\mathbb{P}(\text{STOP}|\text{verb}) \times V(5, \text{verb}), \\
&\qquad \mathbb{P}(\text{STOP}|\text{noun}) \times V(5, \text{noun})\} \\
&= \max\{0.2 \times 8 \times 10^{-9}, 0.5 \times 7.2 \times 10^{-6}\} = 3.6 \times 10^{-6} \\
\widehat{\boldsymbol{y}}_5 &= \text{noun} \\
\widehat{\boldsymbol{y}}_4 &= \psi(5, \text{noun}) = \text{det} \\
\widehat{\boldsymbol{y}}_3 &= \psi(4, \text{det}) = \text{prep} \\
\widehat{\boldsymbol{y}}_2 &= \psi(3, \text{prep}) = \text{verb} \\
\widehat{\boldsymbol{y}}_1 &= \psi(2, \text{verb}) = \text{noun}
\end{aligned}
$$

Preventing Underflowing: Computation in the Log-Domain

- You may have noticed that probabilities get smaller and smaller as we go through the several time steps
- This may cause underflows and numerical instability
- Easy solution: instead of multiplying probabilities, sum log-probabilities!

$$\log(\exp(a) \times \exp(b)) = a + b$$

- To add probabilities, implement a log-sum function:

$$\log(\exp(a) + \exp(b)) = a + \log(1 + \exp(b - a))$$

Summing Up: Viterbi

- An instance of a dynamic programming algorithm
- Computes the most likely sequence of tags
- In other words, the structured output that maximizes the total score/probability
- This is called MAP (*maximum a posteriori*) decoding
- Later we'll talk about marginal decoding as a way of computing the structured output that minimizes the total risk

Recap: Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

Recap: Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

Problem 2: Likelihood and State Posteriors

- Likelihood:

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}) &= \sum_{\boldsymbol{y}_1,\dots\boldsymbol{y}_L} \mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \dots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP}) \\ &= \sum_{\boldsymbol{y}_1,\dots\boldsymbol{y}_L} \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1}) \times \prod_{i=1}^{L} \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)\end{aligned}$$

- Posterior state probabilities:

$$\begin{aligned}\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{x}) &= \frac{\sum_{\boldsymbol{y}_1,\dots,\boldsymbol{y}_{i-1},\boldsymbol{y}_{i+1},\dots\boldsymbol{y}_L} \mathbb{P}(\text{START}, \boldsymbol{x}_1, \boldsymbol{y}_1, \dots, \boldsymbol{x}_L, \boldsymbol{y}_L, \text{STOP})}{\mathbb{P}(\boldsymbol{x})} \\ &= \frac{\sum_{\boldsymbol{y}_1,\dots,\boldsymbol{y}_{i-1},\boldsymbol{y}_{i+1},\dots\boldsymbol{y}_L} \prod_{j=1}^{L+1} \mathbb{P}(\boldsymbol{y}_j|\boldsymbol{y}_{j-1}) \times \prod_{j=1}^{L} \mathbb{P}(\boldsymbol{x}_j|\boldsymbol{y}_j)}{\mathbb{P}(\boldsymbol{x})}\end{aligned}$$

- Both involve marginalizing out the hidden states
- Huge summations ($O(|\Lambda|^L)$ terms).

The Forward-Backward Algorithm

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}) &= \sum_{\boldsymbol{y}_1,\cdots\boldsymbol{y}_L}\prod_{i=1}^{L+1}\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})\times\prod_{i=1}^{L}\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)\\ &= \sum_{\boldsymbol{y}_L}\mathbb{P}(\text{STOP}|\boldsymbol{y}_L)\sum_{\boldsymbol{y}_{L-1}}\mathbb{P}(\boldsymbol{y}_L|\boldsymbol{y}_{L-1})\mathbb{P}(\boldsymbol{x}_L|\boldsymbol{y}_L)\ldots\sum_{\boldsymbol{y}_1}\mathbb{P}(\boldsymbol{y}_2|\boldsymbol{y}_1)\mathbb{P}(\boldsymbol{x}_2|\boldsymbol{y}_2)\big(\mathbb{P}(\boldsymbol{y}_1|\text{START})\mathbb{P}(\boldsymbol{x}_1|\boldsymbol{y}_1)\big)\end{aligned}$$

- **Key idea: interleave summations with products!**

The Forward-Backward Algorithm

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}) &= \sum_{\boldsymbol{y}_1,\cdots\boldsymbol{y}_L} \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1}) \times \prod_{i=1}^{L} \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i) \\ &= \sum_{\boldsymbol{y}_L} \mathbb{P}(\text{STOP}|\boldsymbol{y}_L) \sum_{\boldsymbol{y}_{L-1}} \mathbb{P}(\boldsymbol{y}_L|\boldsymbol{y}_{L-1})\mathbb{P}(\boldsymbol{x}_L|\boldsymbol{y}_L) \ldots \sum_{\boldsymbol{y}_1} \mathbb{P}(\boldsymbol{y}_2|\boldsymbol{y}_1)\mathbb{P}(\boldsymbol{x}_2|\boldsymbol{y}_2)\big(\mathbb{P}(\boldsymbol{y}_1|\text{START})\mathbb{P}(\boldsymbol{x}_1|\boldsymbol{y}_1)\big)\end{aligned}$$

- **Key idea: interleave summations with products!**
- **Forward variables**:

$$\alpha(i, \boldsymbol{y}_i) = \sum_{\boldsymbol{y}_{i-1} \in \Lambda} \mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i) \times \alpha(i-1, \boldsymbol{y}_{i-1})$$

The Forward-Backward Algorithm

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}) &= \sum_{\boldsymbol{y}_1,\cdots\boldsymbol{y}_L}\prod_{i=1}^{L+1}\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})\times\prod_{i=1}^{L}\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)\\ &= \sum_{\boldsymbol{y}_L}\mathbb{P}(\text{STOP}|\boldsymbol{y}_L)\sum_{\boldsymbol{y}_{L-1}}\mathbb{P}(\boldsymbol{y}_L|\boldsymbol{y}_{L-1})\mathbb{P}(\boldsymbol{x}_L|\boldsymbol{y}_L)\ldots\sum_{\boldsymbol{y}_1}\mathbb{P}(\boldsymbol{y}_2|\boldsymbol{y}_1)\mathbb{P}(\boldsymbol{x}_2|\boldsymbol{y}_2)\big(\mathbb{P}(\boldsymbol{y}_1|\text{START})\mathbb{P}(\boldsymbol{x}_1|\boldsymbol{y}_1)\big)\end{aligned}$$

- **Key idea: interleave summations with products!**
- **Forward variables**:

$$\alpha(i,\boldsymbol{y}_i) = \sum_{\boldsymbol{y}_{i-1}\in\Lambda}\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1})\times\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i)\times\alpha(i-1,\boldsymbol{y}_{i-1})$$

- **Forward algorithm**: same as Viterbi, but replacing max with $\sum$
- **Backward algorithm**: same rationale, but right-to-left

The Forward-Backward Algorithm

input: sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$, emission/transition probabilities

Forward pass: compute the forward probabilities
$\alpha(1, \boldsymbol{y}_1) = \mathbb{P}(\boldsymbol{y}_1|\text{START}) \times \mathbb{P}(\boldsymbol{x}_1|\boldsymbol{y}_1) \quad \forall \boldsymbol{y}_1 \in \Lambda$
for $i = 2$ **to** L **do**
  for $\boldsymbol{y}_i \in \Lambda$ **do**
    $\alpha(i, \boldsymbol{y}_i) = \sum_{\boldsymbol{y}_{i-1}} \mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i) \times \alpha(i-1, \boldsymbol{y}_{i-1})$
  end for
end for

Backward pass: compute the backward probabilities
$\beta(L, \boldsymbol{y}_L) = \mathbb{P}(\text{STOP}|\boldsymbol{y}_L) \quad \forall \boldsymbol{y}_L \in \Lambda$
for $i = L-1$ **to** 1 **do**
  for $\boldsymbol{y}_i \in \Lambda$ **do**
    $\beta(i, \boldsymbol{y}_i) = \sum_{\boldsymbol{y}_{i+1}} \mathbb{P}(\boldsymbol{y}_{i+1}|\boldsymbol{y}_i) \times \mathbb{P}(\boldsymbol{x}_{i+1}|\boldsymbol{y}_{i+1}) \times \beta(i+1, \boldsymbol{y}_{i+1})$
  end for
end for

output: Posterior unigram probabilities $\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{x}) \propto \alpha(i, \boldsymbol{y}_i) \times \beta(i, \boldsymbol{y}_i)$
Posterior bigram probabilities

$$\mathbb{P}(\boldsymbol{y}_i, \boldsymbol{y}_{i-1}|\boldsymbol{x}) \propto \alpha(i-1, \boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_{i-1}) \times \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{y}_i) \times \beta(i, \boldsymbol{y}_i)$$

Likelihood $\mathbb{P}(\boldsymbol{x}) = \sum_{\boldsymbol{y}_i} \alpha(i, \boldsymbol{y}_i)\beta(i, \boldsymbol{y}_i)$.

Decoding: Dynamic Programming

- The Viterbi and forward-backward algorithms are two instances of dynamic programming algorithms
- Many other problems in NLP (e.g. parsing with context-free grammars) can also be solved with dynamic programming
- Examples: inside-outside, CKY, Eisner's, belief propagation, ...
- We'll see this later in another lecture.

Minimum Risk Decoding

- Recall that the forward-backward algorithm allow us to compute the posterior marginal probabilities:

$$\mathbb{P}(\boldsymbol{y}_i \mid \boldsymbol{x}) \propto \alpha(i, \boldsymbol{y}_i)\beta(i, \boldsymbol{y}_i)$$

- This is called marginal decoding (a.k.a. marginal inference)
- We can now compute, for each position i, the label $\widehat{\boldsymbol{y}}_i$ that maximizes this posterior probability

$$\widehat{\boldsymbol{y}}_i = \arg\max_{\boldsymbol{y}_i} \mathbb{P}(\boldsymbol{y}_i \mid \boldsymbol{x})$$

- This is called minimum risk decoding
- Does this give the same sequence as Viterbi decoding?

Minimum Risk Decoding

- Recall that the forward-backward algorithm allow us to compute the posterior marginal probabilities:

$$\mathbb{P}(\boldsymbol{y}_i \mid \boldsymbol{x}) \propto \alpha(i, \boldsymbol{y}_i)\beta(i, \boldsymbol{y}_i)$$

- This is called marginal decoding (a.k.a. marginal inference)
- We can now compute, for each position i, the label $\widehat{\boldsymbol{y}}_i$ that maximizes this posterior probability

$$\widehat{\boldsymbol{y}}_i = \arg\max_{\boldsymbol{y}_i} \mathbb{P}(\boldsymbol{y}_i \mid \boldsymbol{x})$$

- This is called minimum risk decoding
- Does this give the same sequence as Viterbi decoding? No.

Viterbi or Minimum Risk Decoding?

- They will not, in general, give the same label sequence.
- Sometimes one works better, sometimes the other.
- Posterior decoding can give a label sequence that itself gets zero probability!
- We can motivate each by the **cost** they minimize.

Cost

Imagine the following game:

- The HMM will be given a new sequence $\boldsymbol{x}$, which we must label.
- Our predicted label sequence $\widehat{\boldsymbol{y}}$ will be compared to the true one, $\boldsymbol{y}$.
- Depending on how badly we do, we will pay a fine.
- We want to minimize the expected cost (a.k.a. the risk).
- Our best strategy will depend on how the cost is defined!

All-or-Nothing Cost

- Suppose we pay 1€ if we get the label sequence wrong, i.e., $\widehat{\boldsymbol{y}} \neq \boldsymbol{y}$.
- Otherwise we pay nothing.
- What should we do?
- We should use the most probable whole sequence $\widehat{y}$.
- i.e., we should do **Viterbi decoding**.

Hamming Cost

- Alternately, suppose we pay 0.1€ for every word that we label incorrectly.
- This is more forgiving, and suggests that we focus on reasoning about each word without worrying about the coherence of the whole sequence.
- What should we do?
- We should use the most probable label $\widehat{y}_i$ for each word.
- i.e., we should do **minimum risk decoding**.

Recap: Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

Recap: Three Problems in HMMs

Let $\boldsymbol{x} = \text{START}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_L, \text{STOP}$ denote a word sequence.

1. Given $\boldsymbol{x}$, compute the *most likely sequence of states* $\arg\max_{\boldsymbol{y}_1, \ldots \boldsymbol{y}_L} \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L | \boldsymbol{x})$
2. Given $\boldsymbol{x}$, compute the *likelihood* $\mathbb{P}(\boldsymbol{x})$ and the *posterior state probabilities* $\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x})$
3. Given training sequences of word and states (or just words), *estimate the emission and transition probabilities*.

Problem 3: Estimating the Model Parameters

How to estimate the emission probabilities $\mathbb{P}(x_i|y_i)$ and the transition probabilities $\mathbb{P}(y_i|y_{i-1})$?

We will look at two cases:

1. **Supervised learning:** assumes we have *labeled* training data $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$

2. **Unsupervised learning:** assumes all we have is *unlabeled* training data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$

HMM Supervised Learning

Assumes we have *labeled* training data $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$

Simply use Maximum Likelihood Estimation for the complete data!

- Choose parameters that maximize $\prod_{n=1}^{N} \mathbb{P}(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$
- Closed form solution: **just count events and normalize**
  - How many times two states $\boldsymbol{y}_{i-1}$ and $\boldsymbol{y}_i$ occur consecutively $\Longrightarrow$ **Estimate transition probabilities** $P(\boldsymbol{y}_i \mid \boldsymbol{y}_{i-1})$
  - How many times a state $\boldsymbol{y}_i$ and a symbol $\boldsymbol{x}_i$ occur together $\Longrightarrow$ **Estimate emission probabilities** $P(\boldsymbol{x}_i \mid \boldsymbol{y}_i)$
- This is the "structured" version of Naive Bayes!

HMM Unsupervised Learning

Assumes all we have is *unlabeled* training data $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$

This is a lot harder!

We never get to see any label $\boldsymbol{y}$, so the best we can aim is to learn a good HMM up to a permutation of the labels!

But which objective function should we try to optimize?

Common choice: Maximum Likelihood Estimation with imcomplete data!

- Involves maximizing the marginal likelihood $\prod_{n=1}^{N} \mathbb{P}(\boldsymbol{x}^{(n)}) = \prod_{n=1}^{N} \sum_{\boldsymbol{y}}^{(n)} \mathbb{P}(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$
- However, this is usually prohibitive because it requires marginalizing out $\boldsymbol{y}^{(n)}$
- "Standard" solution: **Expectation-Maximization (EM) algorithm**

Expectation-Maximization (Baum-Welch)

Expectation-Maximization (EM) is a general inference technique, applicable beyond HMMs

- It works for any graphical model mixing observed and latent variables
- Applicable to any generative model with latent variables (discrete or continuous)!
- You might have seen it for mixtures of Gaussians and other clustering algorithms (e.g. soft clustering)
- In the case of HMMs, this is also called the **Baum-Welch algorithm**.

Expectation-Maximization (Baum-Welch)

Recall that if we observed the hidden variables $\boldsymbol{y}$ during training, the parameters could be estimated easily by counting and normalizing

- How many times two states $\boldsymbol{y}_{i-1}$ and $\boldsymbol{y}_i$ occur consecutively $\Longrightarrow$ **Estimate transition probabilities** $P(\boldsymbol{y}_i \mid \boldsymbol{y}_{i-1})$
- How many times a state $\boldsymbol{y}_i$ and a symbol $\boldsymbol{x}_i$ occur together $\Longrightarrow$ **Estimate emission probabilities** $P(\boldsymbol{x}_i \mid \boldsymbol{y}_i)$

Unfortunately we don't observe the $\boldsymbol{y}$'s...

... but **if we knew** the model parameters, we could estimate the posterior marginal probabilities $P(\boldsymbol{y}_i \mid \boldsymbol{x})$ and $P(\boldsymbol{y}_{i-1}, \boldsymbol{y}_i \mid \boldsymbol{x})$, which would give us **fractional counts**

This is a chicken-and-egg problem!

Expectation-Maximization (Baum-Welch)

EM initializes the model parameters randomly

Then it alternates between these two steps:

- E-step: apply the current model to each training sequence $\boldsymbol{x}$ to obtain $P(\boldsymbol{y}_i \mid \boldsymbol{x})$ and $P(\boldsymbol{y}_{i-1}, \boldsymbol{y}_i \mid \boldsymbol{x})$
  - We can do this with the **forward-backward algorithm**!
- M-step: update the model
  - We can do this by counting events (fractional counts) and normalizing!

This is guaranteed to **improve the observed data likelihood at every iteration** (try to prove this at home!)

But MLE with incomplete data is a non-convex problem, so we can get to a local minimum.

Alternatives to EM

Also tackling MLE:

- Gradient descent: tackles MLE directly, also guaranteed to converge to a local minimum, also reutilizes existing toolboxes
- SGD: similar, but does updates after observing each sequence

Method of moments:

- Instead of tackling MLE, try to match observed moments with expected moments under the model
- Collect high-order moments (e.g. triplets of consecutive observed symbols) and solve an equation to obtain the model parameters
- Can be done with **spectral learning**, which boils down to a SVD matrix factorization
- Also doable with **anchor learning**, which boils down to non-negative matrix factorization

Semi-Supervised Learning of HMMs

Can also be done with EM, by combining sequences where labels are observed with sequence where they are not observed.

Boils down to combining true counts with fractional (expected) counts.

Summary of HMMs

- Representation? **Directed sequence model.**
- Assumptions? **Markov assumption on states; words are conditionally independent given the state.**
- Decoding/Inference? **Viterbi/forward-backward algorithms.**
- Learning the parameters? **Maximum likelihood (count and normalize)** for the supervised case, EM for the unsupervised case.

Roadmap: Models for Structured Prediction

Binary/Multi-class	Structured Prediction
Naive Bayes	HMMs
Logistic Regression	?
Perceptron	?
SVMs	?

Related: Class-Based Language Models

- Proposed by Brown et al. (1992)
- Similar to HMMs where the observed symbols are **words** and the states are **word classes**
- Detail: we assume that each word belongs to **one and only one class**
  - i.e. the emission probabilities $P(\boldsymbol{x}_i \mid \boldsymbol{y}_i)$ are hard-clustered around classes; $P(\boldsymbol{x}_i \mid \boldsymbol{y}_i) = 0$ if word $\boldsymbol{x}_i$ does not belong to class $\boldsymbol{y}_i$
- We also assume classes have a hierarchy (from coarse-grained to fine-grained classes)
- Each class is represented with a bit-string, number of bits determines how fine-grained it is
- Classes are learned with agglomerative clustering
- This is known as **Brown clustering**.

Examples of Brown clusters (from Twitter data)

Path	Terms
001010110	never neva nvr gladly nevr #never neverr nver neverrr nevaa
001010111	ever eva evar evr everrr everr everrrr evah everrrrr everrrrrr
01000010	does duz doess does sayeth doez doesss d0es deos

Path	Terms
0100	Monday
010100	Sunday
010101	Friday
0101100	Thursday
01011010	Saturday

(from `http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html`)

Brown clustering

Brown clustering is a way of **representation learning**

It obtains **discrete word representations** (binary vectors) via unsupervised hierarchical clustering

It was extremely popular in NLP before the age of neural networks

We'll look at other ways of learning word representations (continuous ones) in the next class

Outline

Conditional Random Fields (Lafferty et al., 2001)

HMMs have two drawbacks in practical NLP problems:

1. Words being conditionally independent is too strong an assumption
2. Can't have coarser features than words (e.g capitalization, suffixes, etc.) which often help

Conditional Random Fields (Lafferty et al., 2001)

HMMs have two drawbacks in practical NLP problems:

1. Words being conditionally independent is too strong an assumption
2. Can't have coarser features than words (e.g capitalization, suffixes, etc.) which often help

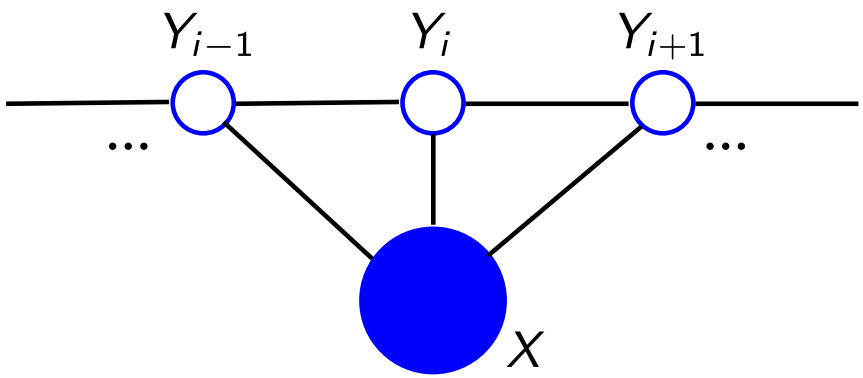
CRFs overcome these drawbacks—they're a structured variant of logistic regression!

Also used in vision (Kumar and Hebert, 2003; Quattoni et al., 2004)

Conditional Random Fields

Similar factorization as HMMs, but globally normalized!

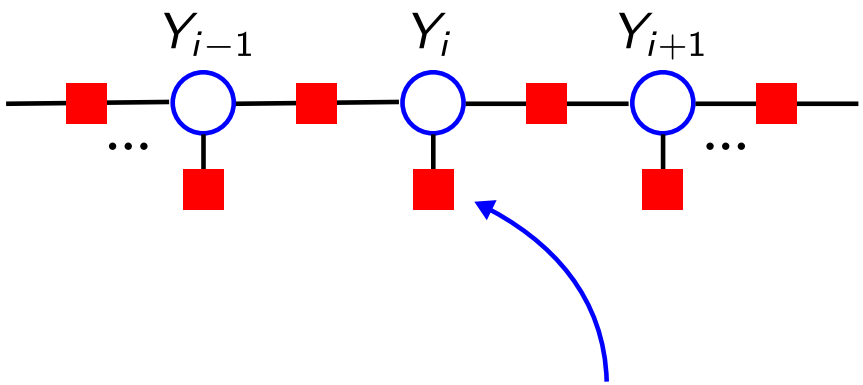
$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{w},\boldsymbol{x})} \exp\left(\sum_i \underbrace{\boldsymbol{w} \cdot \boldsymbol{\phi}_i(\boldsymbol{x}, \boldsymbol{y}_i)}_{\textbf{nodes}} + \sum_i \underbrace{\boldsymbol{w} \cdot \boldsymbol{\phi}_{i,i-1}(\boldsymbol{x}, \boldsymbol{y}_i, \boldsymbol{y}_{i-1})}_{\textbf{edges}} \right)$$



Conditional Random Fields

Similar factorization as HMMs, but globally normalized!

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{w},\boldsymbol{x})} \exp\left(\sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_i(\boldsymbol{x},\boldsymbol{y}_i)}_{\textbf{nodes}} + \sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_{i,i-1}(\boldsymbol{x},\boldsymbol{y}_i,\boldsymbol{y}_{i-1})}_{\textbf{edges}} \right)$$



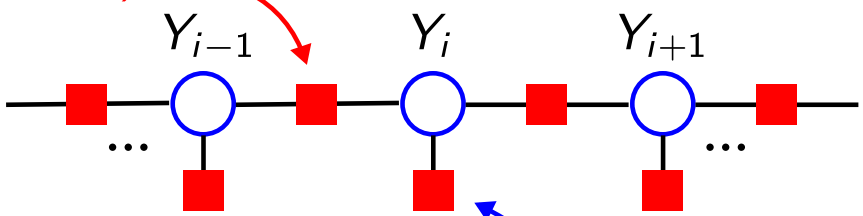
$\psi_i(y_i) := \exp(w \cdot \phi_i(x, y_i))$ (unary potentials)

Conditional Random Fields

Similar factorization as HMMs, but globally normalized!

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{w},\boldsymbol{x})} \exp\left(\sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_i(\boldsymbol{x},\boldsymbol{y}_i)}_{\textbf{nodes}} + \sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_{i,i-1}(\boldsymbol{x},\boldsymbol{y}_i,\boldsymbol{y}_{i-1})}_{\textbf{edges}} \right)$$

$\psi_{i,i-1}(y_i, y_{i-1}) := \exp(w \cdot \phi_{i,i-1}(x, y_i, y_{i-1}))$
(pairwise potentials)



$\psi_i(y_i) := \exp(w \cdot \phi_i(x, y_i))$ (unary potentials)

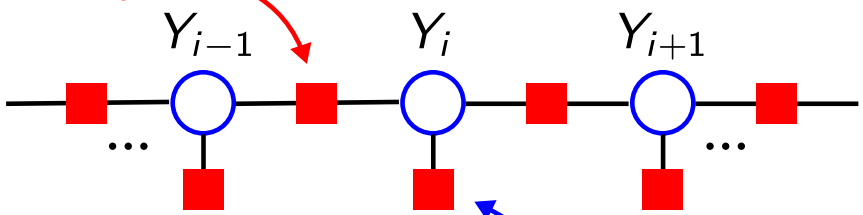
Conditional Random Fields

Similar factorization as HMMs, but globally normalized!

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{w},\boldsymbol{x})} \exp\left(\sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_i(\boldsymbol{x},\boldsymbol{y}_i)}_{\textbf{nodes}} + \sum_i \underbrace{\boldsymbol{w}\cdot\boldsymbol{\phi}_{i,i-1}(\boldsymbol{x},\boldsymbol{y}_i,\boldsymbol{y}_{i-1})}_{\textbf{edges}} \right)$$

$$\propto \prod_i \psi_i(\boldsymbol{y}_i) \prod_i \psi_{i,i-1}(\boldsymbol{y}_i,\boldsymbol{y}_{i-1})$$

$\psi_{i,i-1}(y_i, y_{i-1}) := \exp(w \cdot \phi_{i,i-1}(x, y_i, y_{i-1}))$
(pairwise potentials)



$\psi_i(y_i) := \exp(w \cdot \phi_i(x, y_i))$ (unary potentials)

Learning and Decoding in CRFs

- HMMs are a *generative* model: they model $\mathbb{P}(\boldsymbol{x}, \boldsymbol{y})$
- CRFs are a *discriminative* model: they model $\mathbb{P}(\boldsymbol{y}|\boldsymbol{x})$

Learning and Decoding in CRFs

- HMMs are a *generative* model: they model $\mathbb{P}(\boldsymbol{x}, \boldsymbol{y})$
- CRFs are a *discriminative* model: they model $\mathbb{P}(\boldsymbol{y}|\boldsymbol{x})$
- We can't compute $\mathbb{P}(x)$ with a CRF...
- ... But we can compute the most likely sequence of states and the posterior state probabilities
- **Just use the standard Viterbi and forward-backward algorithms (they work with unnormalized distributions too!)**

Learning and Decoding in CRFs

- Learning the parameters $\boldsymbol{w}$ (with supervision) is done by maximizing *conditional log-likelihood*:

$$\max_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_t \log \mathbb{P}_{\boldsymbol{w}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)})$$

- **A convex optimization problem, typically solved with L-BFGS or stochastic gradient descent**

Recap: Logistic Regression

- Define conditional probability

$$P_{\boldsymbol{w}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}))}{Z_{\boldsymbol{x}}}$$

- Set weights to maximize conditional log-likelihood of training data:

$$\boldsymbol{w} = \arg\max_{\boldsymbol{w}} \sum_t \log P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\min_{\boldsymbol{w}} \sum_t L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$$

- Can find the gradient and run gradient descent (or any gradient-based optimization algorithm)

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \sum_{\boldsymbol{y}'} P_{\boldsymbol{w}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

Structural Decomposition

For conditional random fields, the features and their expectation decompose according to the sequential structure:

$$\begin{aligned}
\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) &= \text{concat}(\boldsymbol{\phi}_{\text{unig}}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\phi}_{\text{big}}(\boldsymbol{x}, \boldsymbol{y})) \\
\boldsymbol{\phi}_{\text{unig}}(\boldsymbol{x}, \boldsymbol{y}) &= \sum_i \boldsymbol{\phi}_i(\boldsymbol{x}, \boldsymbol{y}_i) \\
\boldsymbol{\phi}_{\text{big}}(\boldsymbol{x}, \boldsymbol{y}) &= \sum_i \boldsymbol{\phi}_{i,i-1}(\boldsymbol{x}, \boldsymbol{y}_i, \boldsymbol{y}_{i-1}) \\
\sum_{\boldsymbol{y}'} P_{\boldsymbol{w}}(\boldsymbol{y}'|\boldsymbol{x}) \boldsymbol{\phi}_{\text{unig}}(\boldsymbol{x}, \boldsymbol{y}') &= \sum_i \sum_{\boldsymbol{y}'_i} P_{\boldsymbol{w}}(\boldsymbol{y}'_i|\boldsymbol{x}) \boldsymbol{\phi}_i(\boldsymbol{x}, \boldsymbol{y}'_i) \\
\sum_{\boldsymbol{y}'} P_{\boldsymbol{w}}(\boldsymbol{y}'|\boldsymbol{x}) \boldsymbol{\phi}_{\text{big}}(\boldsymbol{x}, \boldsymbol{y}') &= \sum_i \sum_{\boldsymbol{y}'_{i-1}, \boldsymbol{y}'_i} P_{\boldsymbol{w}}(\boldsymbol{y}'_i, \boldsymbol{y}'_{i-1}|\boldsymbol{x}) \boldsymbol{\phi}_{i,i-1}(\boldsymbol{x}, \boldsymbol{y}'_i, \boldsymbol{y}'_{i-1})
\end{aligned}$$

All quantities can be obtained with the **forward-backward algorithm**!

Conditional Random Fields

- Representation? **Undirected sequence model.**
- Assumptions? **Markov assumption on states; no assumption of the words whatsoever.**
- Decoding/Inference? **Viterbi/forward-backward algorithms.**
- Learning the parameters? **Maximum conditional likelihood (convex optimization).**

Features for POS tagging (Ratnaparkhi, 1999)

Tag trigram	$y_{i-2} \wedge y_{i-1} \wedge y_i$
Tag bigram	$y_{i-1} \wedge y_i$
Word	$x_i \wedge y_i$
Previous word	$x_{i-1} \wedge y_i$
Two words before	$x_{i-2} \wedge y_i$
Next word	$x_{i+1} \wedge y_i$
Two words after	$x_{i+2} \wedge y_i$
Word has upper case	$\textsc{HasUpper}(x_i) \wedge y_i$
Word has digit	$\textsc{HasDigit}(x_i) \wedge y_i$
Word has hyphen	$\textsc{HasHyphen}(x_i) \wedge y_i$
Word prefixes (up to 4 characters)	$\textsc{Prefix}(x_i) \wedge y_i$
Word suffixes (up to 4 characters)	$\textsc{Suffix}(x_i) \wedge y_i$

- **Accuracies are $\sim 97\%$ in the Penn Treebank**

Features for NER (Ratinov and Roth, 2009)

Similar, plus some more features:

- Word shapes: "Leuven" $\rightarrow$ `Aa+`
- Flags for all-digits, all-upper, first-upper
- Same for previous/next word and two words before/after
- Bigram tag features also conjoined with words
- Gazetteer features (e.g. entities found in Wikipedia)

Standard evaluation metric is F_1 *measure* (harmonic mean of precision and recall)

- $F_1 \sim 80\%$ **in the English CoNLL 2003 dataset**

Roadmap: Models for Structured Prediction

Binary/Multi-class	Structured Prediction
Naive Bayes	HMMs
Logistic Regression	CRFs
Perceptron	?
SVMs	?

Outline

Structured Perceptron (Collins, 2002)

- The perceptron algorithm has their structured counterpart too
- The prediction $\widehat{\boldsymbol{y}}$ according to the current model $\boldsymbol{w}$ is now the most likely sequence of states (computed using the Viterbi algorithm)
- Everything else still applies (similar updates, similar mistake bound)

Recap: Perceptron

input: labeled data $\mathcal{D}$
initialize $\boldsymbol{w}^{(0)} = \mathbf{0}$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
  predict $\widehat{\boldsymbol{y}}_i = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}^{(k)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y})$
  if $\widehat{\boldsymbol{y}}_i \neq \boldsymbol{y}_i$ **then**
    update $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)$
    increment k
  end if
until maximum number of epochs
output: model weights $\boldsymbol{w}$

Structured Perceptron

- Representation? **Undirected sequence model.**
- Assumptions? **Markov assumption on states; no assumption of the words whatsoever.**
- Decoding/Inference? **Viterbi.**
- Learning the parameters? **Perceptron updates.**

Outline

Structured Support Vector Machines (Taskar et al., 2003; Altun et al., 2003; Tsochantaridis et al., 2004)

- SVMs have their structured counterpart too
- Similar idea to CRFs, but maximize margin
- No probabilistic interpretation anymore
- Can still be used to compute the most likely sequence of states (using the Viterbi algorithm)
- The loss function is **structured hinge loss**: a convex surrogate to **Hamming loss** (the fraction of incorrect states)

Recap: SVMs

Hinge loss:

$$\begin{aligned} L((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{w}) &= \max\left(0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})\right) \\ &= \left(\max_{\boldsymbol{y}' \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]]\right) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

A subgradient of the hinge is

$$\partial_{\boldsymbol{w}} L((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{w}) \ni \boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}}) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

where

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + [[\boldsymbol{y}' \neq \boldsymbol{y}]]$$

Structured Support Vector Machines

Key idea: replace the 0/1 cost $[[\boldsymbol{y}' \neq \boldsymbol{y}]]$ by a **Hamming cost**:

$$\rho(\boldsymbol{y}', \boldsymbol{y}) = \sum_i [[\boldsymbol{y}'_i \neq \boldsymbol{y}_i]]$$

In other words: the "margin" counts how many labels are incorrect

Cost-augmented decoding:

$$\begin{aligned} \widehat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}') + \rho(\boldsymbol{y}', \boldsymbol{y}) \\ &= \arg\max_{\boldsymbol{y}' \in \mathcal{Y}} \sum_i \left(\boldsymbol{w} \cdot \boldsymbol{\phi}_i(\boldsymbol{x}, \boldsymbol{y}'_i) + [[\boldsymbol{y}'_i, \boldsymbol{y}_i]]\right) + \sum_i \boldsymbol{w} \cdot \boldsymbol{\phi}_{i,i-1}(\boldsymbol{x}, \boldsymbol{y}'_i, \boldsymbol{y}'_{i-1}) \end{aligned}$$

The Hamming cost is **decomposable**! Hence cost-augmented decoding can still be done with the Viterbi algorithm

A subgradient can then be computed as usual via

$$\partial_{\boldsymbol{w}} L((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{w}) \ni \boldsymbol{\phi}(\boldsymbol{x}, \widehat{\boldsymbol{y}}) - \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$$

Structured Support Vector Machines

- Representation? **Undirected sequence model.**
- Assumptions? **Markov assumption on states; no assumption of the words whatsoever.**
- Decoding/Inference? **Viterbi algorithm.**
- Learning the parameters? **Minimize structured hinge loss (convex, non-smooth optimization).**

Roadmap: Models for Structured Prediction

Binary/Multi-class	**Structured Prediction**
Naive Bayes	HMMs
Logistic Regression	CRFs
Perceptron	Structured Perceptron
SVMs	Structured SVMs

Neural Sequence Models

So far, all models discussed are structured but **linear**

But we know that neural networks are more powerful than linear models...

Can we mix a neural network and a CRF?

Neural Sequence Models

So far, all models discussed are structured but **linear**

But we know that neural networks are more powerful than linear models...

Can we mix a neural network and a CRF? Yes.

Neural Sequence Models

Just append a CRF output layer to your favorite neural network!

For example:

- a shared feedforward neural network to each element of the sequence to compute unigram scores
- another shared feedforward to compute bigram scores (or just a label bigram indicator)
- an output layer with a sequential structure using those unigram and bigram scores

Use Forward-Backward to compute the gradient in the last layer; then gradient backpropagation will make all the work for us

Other configurations are possible instead of a shared feedforward: e.g. a CNN, a BILSTM, ... (will learn those later)

Works for a structured SVM output layer too.

Outline

Conclusions

- Sequences are a common form of structure
- Markov models are simple generative models, widely used for language modeling
- Their main limitation is finite memory (more on this later) and data sparsity
- HMMs extend Markov models with a mix of hidden and observed variables
- MAP and marginal inference in HMMs is possible with dynamic programming (Viterbi; Forward-Backward)
- Structured perceptron, conditional random fields, and structured SVMs are more flexible, discriminative, sequential models
- They can still reuse the dynamic programming toolkit of HMMs for inference and learning

Thank you!

Questions?

References I

Altun, Y., Tsochantaridis, I., and Hofmann, T. (2003). Hidden Markov support vector machines. In *Proc. of International Conference of Machine Learning*.

Brants, T. (2000). Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics.

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. of Empirical Methods for Natural Language Processing*.

Kumar, S. and Hebert, M. (2003). Discriminative random fields: A discriminative framework for contextual interaction in classification. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1150–1157. IEEE.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of International Conference of Machine Learning*.

Quattoni, A., Collins, M., and Darrell, T. (2004). Conditional random fields for object recognition. In *Advances in neural information processing systems*, pages 1097–1104.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1):151–175.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin Markov networks. In *Proc. of Neural Information Processing Systems*.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proc. of International Conference of Machine Learning*.

Zhang, T. and Johnson, D. (2003). A robust risk minimization based named entity recognition system. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 204–207. Association for Computational Linguistics.