Sequence labelling

IN4080
Natural Language Processing

Yves Scherrer

Classification tasks

One label/class per document

- Document classification
- Text classification
- Sequence classification
- E.g. sentiment analysis, language identification, hate speech detection, …

One label/class per token in a sequence

- Sequence labelling
- Token classification
- E.g. part-of-speech tagging, named entity recognition, span identification

Part-of-speech tagging

Profits soared at Boeing Co. , easily topping forecasts on Wall Street , as their CEO Alan Mulally announced first quarter results .



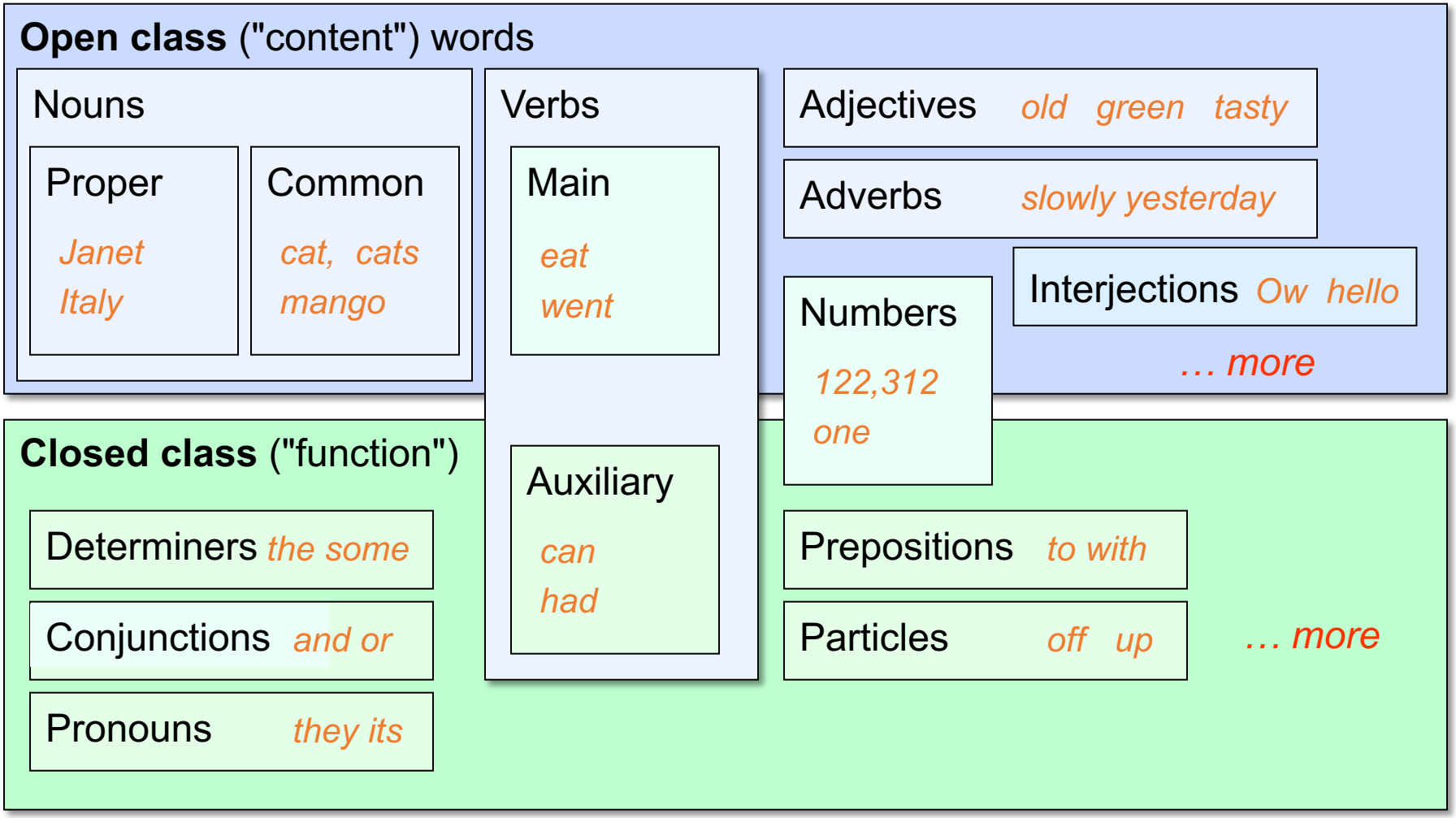
Profits/NOUN soared/VERB at/ADP Boeing/PROPN Co./PROPN ,/PUNCT easily/ADV topping/VERB forecasts/NOUN on/ADP Wall/PROPN Street/PROPN ,/PUNCT as/SCONJ their/DET CEO/PROPN Alan/PROPN Mulally/PROPN announced/VERB first/ADJ quarter/NOUN results/NOUN ./PUNCT

Part-of-speech tagging

Verticalized format:

Profits	NOUN
soared	VERB
at	ADP
Boeing	PROPN
Co.	PROPN
,	PUNCT
easily	ADV
topping	VERB
forecasts	NOUN
on	ADP
Wall	PROPN
Street	PROPN
,	PUNCT
as	SCONJ
their	DET
CEO	PROPN
Alan	PROPN
Mulally	PROPN
announced	VERB
first	ADJ
quarter	NOUN
results	NOUN
.	PUNCT

What are parts-of-speech?



From Jurafsky & Martin slides

Part-of-speech tagging

Why is POS tagging useful?

- Search for syntactic patterns in a corpus
- Information extraction: focus on content words
- Text-to-speech (how do we pronounce “lead”?)
- As a preprocessing step for other types of analysis

Sequence labelling

Goal:

- Predict a label for each element of a sequence (i.e., for each word of a sentence)

Easy cases:

- Unambiguous words: words seen together with a single label during training

Difficult cases:

- Ambiguous words: words seen together with several labels during training
- Unknown words: words not seen during training

Ambiguity of tags (English)

- Roughly 15% of word types are ambiguous
  - Hence 85% of word types are unambiguous
  - *Janet* is always PROPN, *hesitantly* is always ADV
  - But those 15% tend to be very common
- So ~60% of word tokens are ambiguous
- E.g., *back*
  - *earnings growth took a back/ADJ seat*
  - *a small building in the back/NOUN*
  - *a clear majority of senators back/VERB the bill*
  - *enable the country to buy back/PART debt*
  - *I was twenty-one back/ADV then*
- State-of-the-art tagging accuracy is around 97%
  - But most frequent class baseline obtains around 92%

Another task: Named entity recognition

Named entities:

- Person names
- Place names
- Date and time expressions
- Organization names

Citing high fuel prices, United Airlines said Friday it has increased fares by $6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR Corp., immediately matched the move, spokesman Tim Wagner said. United, a unit of UAL Corp., said the increase took effect Thursday and applies to most routes where it competes against discount carriers, such as Chicago to Dallas and Denver to San Francisco.

Another task: Named entity recognition

Named entities often cover several words.
They can be represented with named brackets:

Citing high fuel prices, [United Airlines]$_{ORG}$ said [Friday]$_{TIME}$ it has increased fares by $6 per round trip on flights to some cities also served by lower-cost carriers. [American Airlines]$_{ORG}$, a unit of [AMR Corp.]$_{ORG}$, immediately matched the move, spokesman [Tim Wagner]$_{PERS}$ said. [United]$_{ORG}$, a unit of [UAL Corp.]$_{ORG}$, said the increase took effect [Thursday]$_{TIME}$ and applies to most routes where it competes against discount carriers, such as [Chicago]$_{LOC}$ to [Dallas]$_{LOC}$ and [Denver]$_{LOC}$ to [San Francisco]$_{LOC}$.

This looks different from POS-tagging: not all words have labels, and we didn't have brackets before.

Named entities

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Figure 8.5 A list of generic named entity types with the kinds of entities they refer to.

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

Figure 8.6 Examples of type ambiguities in the use of the name *Washington*.

J&M, SLP 3rd ed., ch. 17

BIO encoding

- B = Beginning of a new named entity
- I = Inside an existing named entity
- O = Outside of any named entity

Citing/O high/O fuel/O prices/O ,/O United/B-ORG Airlines/I-ORG said/O Friday/B-TIME it/O …

… such/O as/O Chicago/B-LOC to/O Dallas/B-LOC and/O Denver/B-LOC to/O San/B-LOC Francisco/I-LOC ./O

This looks exactly like a POS tagging task now.

Why do we need to distinguish B- and I- tags?

BIO encoding

The BIO encoding can be used for any **span annotation** task.

- Span = labeled sequence of consecutive words
- Non-nested brackets

Other examples:

- Chunking
  - $[\text{He}]_{\text{NP}}$ $[\text{reckons}]_{\text{VP}}$ $[\text{the current account deficit}]_{\text{NP}}$ $[\text{will narrow}]_{\text{VP}}$ $[\text{to}]_{\text{PP}}$ $[\text{only \$ 1.8 billion}]_{\text{NP}}$ $[\text{in}]_{\text{PP}}$ $[\text{September}]_{\text{NP}}$.
- Slots in dialogue systems
  - I'd like to see the showtimes for $[\text{Silly Movie 2.0}]_{\text{movie_name}}$ at the $[\text{movie house}]_{\text{object_location_type}}$.

Sequence labelling models

- Hidden Markov Models
  - A probabilistic generative model
  - Combination of Naïve Bayes and n-gram language models
- Discriminative sequence models
  - Extension of logistic regression

Hidden Markov Models

Naïve Bayes for sequence labeling

A first (and probably stupid) idea:

- Look at one word at a time (no context)
- Predict the most probable tag according to training data

Prediction function:

- $\hat{y} = \arg\max_{y \in Y}(P(y) \cdot P(x|y))$

Estimate probabilities from training data:

- $P(x|y) = \frac{Count(x,y)}{Count(y)}$ $P(y) = \frac{Count(y)}{N}$
- Let's use add-one smoothing for both likelihoods and priors

Let's try!

Training corpus:

dogs/N eat/V fish/N

cats/N eat/V mice/N

cats/N like/V fish/N

dogs/N fish/V

Compute priors:

$$P(N) = \frac{7+1}{11+2} \quad P(V) = \frac{4+1}{11+2}$$

Compute likelihoods:

$$P(\text{dogs}|N) = \frac{2+1}{7+6} \quad P(\text{dogs}|V) = \frac{0+1}{4+6}$$

$$P(\text{cats}|N) = \frac{2+1}{7+6} \quad P(\text{cats}|V) = \frac{0+1}{4+6}$$

$$P(\text{eat}|N) = \frac{0+1}{7+6} \quad P(\text{eat}|V) = \frac{2+1}{4+6}$$

$$P(\text{like}|N) = \frac{0+1}{7+6} \quad P(\text{like}|V) = \frac{1+1}{4+6}$$

$$P(\text{fish}|N) = \frac{2+1}{7+6} \quad P(\text{fish}|V) = \frac{1+1}{4+6}$$

$$P(\text{mice}|N) = \frac{1+1}{7+6} \quad P(\text{mice}|V) = \frac{0+1}{4+6}$$

Let's try!

Test sentence: dogs like fish

$$\hat{y} = \arg\max_{y \in Y}(P(y) \cdot P(x|y))$$

Computations:

$$P(N|\text{dogs}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{dogs}) = \frac{5}{13} \cdot \frac{1}{10} = \frac{5}{130} \qquad \hat{y} = N$$

$$P(N|\text{like}) = \frac{8}{13} \cdot \frac{1}{13} = \frac{8}{169} \qquad P(V|\text{like}) = \frac{5}{13} \cdot \frac{2}{10} = \frac{10}{130} \qquad \hat{y} = V$$

$$P(N|\text{fish}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{fish}) = \frac{5}{13} \cdot \frac{2}{10} = \frac{10}{130} \qquad \hat{y} = N$$

Results are as expected for this example

Let's try!

Test sentence: cats fish fish

$$\hat{y} = \arg\max_{y \in Y}(P(y) \cdot P(x|y))$$

Computations:

$$P(N|\text{cats}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{cats}) = \frac{5}{13} \cdot \frac{1}{10} = \frac{5}{130} \qquad \hat{y} = N$$

$$P(N|\text{fish}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{fish}) = \frac{5}{13} \cdot \frac{2}{10} = \frac{10}{130} \qquad \hat{y} = N$$

$$P(N|\text{fish}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{fish}) = \frac{5}{13} \cdot \frac{2}{10} = \frac{10}{130} \qquad \hat{y} = N$$

For ambiguous words, this approach doesn't work well because it doesn't distinguish between different contexts/positions

Let's try!

Test sentence: dogs like bears

$$\hat{y} = \arg\max_{y \in Y}(P(y) \cdot P(x|y))$$

Computations:

$$P(N|\text{dogs}) = \frac{8}{13} \cdot \frac{3}{13} = \frac{24}{169} \qquad P(V|\text{dogs}) = \frac{5}{13} \cdot \frac{1}{10} = \frac{5}{130} \qquad \hat{y} = N$$

$$P(N|\text{like}) = \frac{8}{13} \cdot \frac{1}{13} = \frac{8}{169} \qquad P(V|\text{like}) = \frac{5}{13} \cdot \frac{2}{10} = \frac{10}{130} \qquad \hat{y} = V$$

$$P(N|\text{bears}) = \frac{8}{13} \cdot \frac{1}{13} = \frac{8}{169} \qquad P(V|\text{bears}) = \frac{5}{13} \cdot \frac{1}{10} = \frac{5}{130} \qquad \hat{y} = N$$

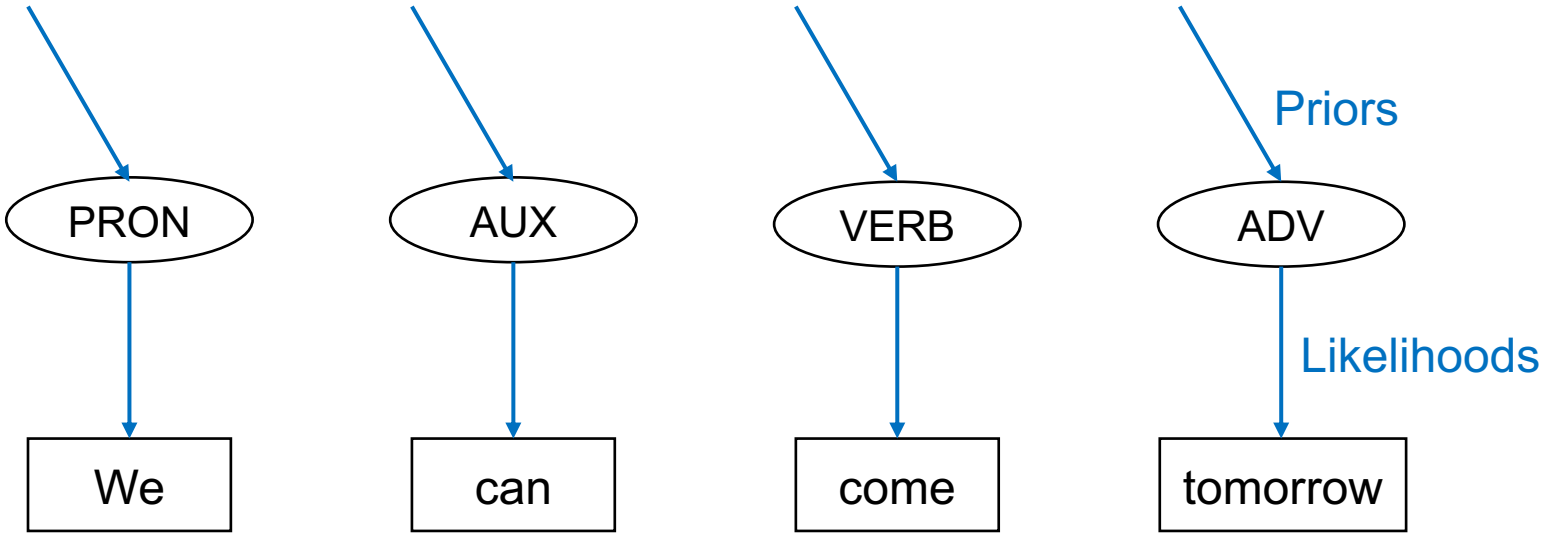
Unknown words are handled reasonably well with simple add-one smoothing

What about ambiguous words?

- Our Naive Bayes model is a **unigram model**
- It labels each word independently of its context:
  - We/PRON **can/AUX** come/VERB tomorrow/ADV ./PUNCT
  - The/DET trash/NOUN **can/AUX** is/VERB empty/ADJ ./PUNCT
  - Tomorrow/ADV empty/ADJ come/VERB **can/AUX** ./PUNCT

- It cannot handle ambiguous words by design
  - We have to change that design…

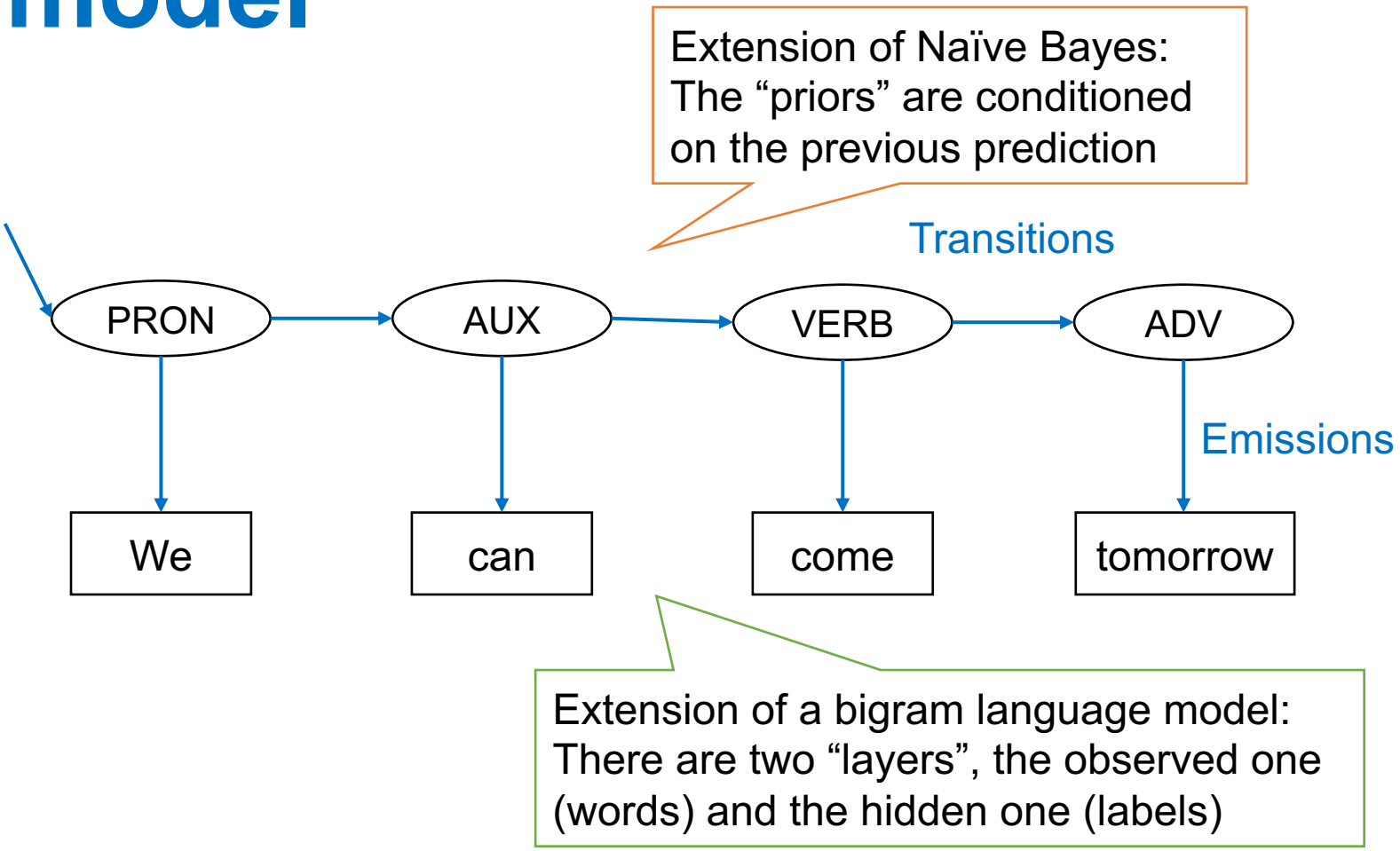
Naïve Bayes



Bigram language model



Bigram hidden Markov model



A simple Hidden Markov Model

Recall the Naïve Bayes prediction function:

$$\hat{y}_i = \arg\max_{y_i \in Y}(P(y_i) \cdot P(x_i|y_i))$$

Priors Likelihoods

We can just replace the prior probability by a conditional probability:

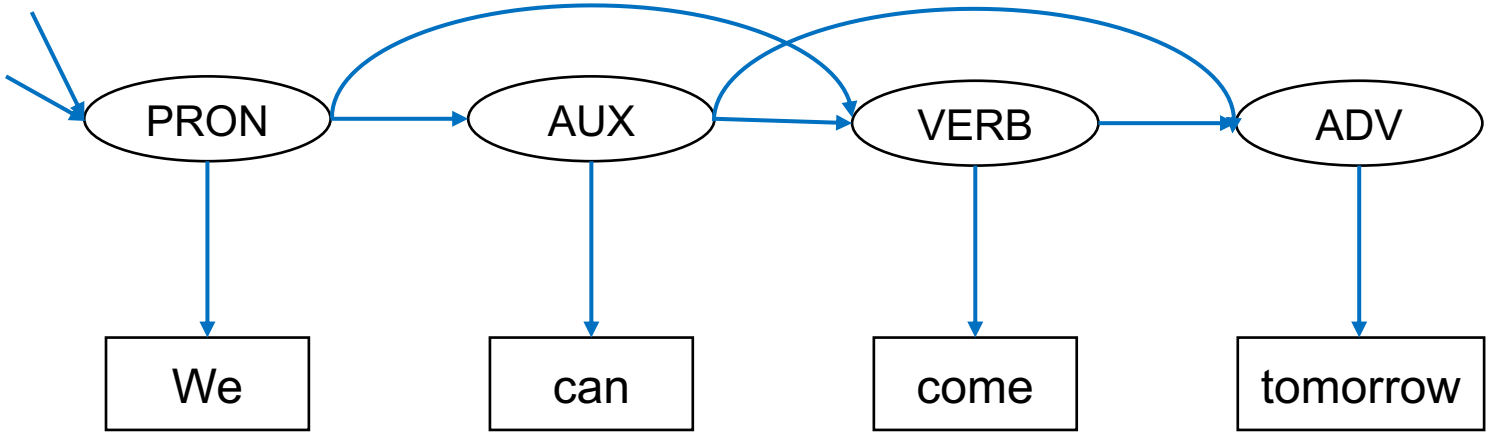
$$\hat{y}_i = \arg\max_{y_i \in Y}(P(y_i|y_{i-1}) \cdot P(x_i|y_i))$$

Transitions Emissions

Bigram language model: $P(x_i|x_{i-1})$

- This is a simple **Hidden Markov model**.

Trigram HMM



$$\hat{y}_i = \arg\max_{y_i \in Y} (P(y_i | y_{i-2}, y_{i-1}) \cdot P(x_i | y_i))$$

Trigram language model: $P(x_i | x_{i-2}, x_{i-1})$

A Hidden Markov Model

Training (without smoothing):

- For each x, y in training data:
  - Compute and store $P_E(x|y) = \frac{Count(x,y)}{Count(y)}$
- For each bigram y_{i-1}, y_i in training data:
  - Compute and store $P_T(y_i|y_{i-1}) = \frac{Count(y_{i-1},y_i)}{Count(y_{i-1})}$

Testing/Prediction (first try):

- For each sentence in test data:
  - For each position i in sentence:
    - Compute $\hat{y}_i = \arg\max_{y_i \in Y}(P_T(y_i|y_{i-1}) \cdot P_E(x_i|y_i))$
    - Store $\hat{y}_i$ to make it accessible for the next position

We will need to apply a trick for the first word of each sentence…

Example Training

Training corpus:

/* dogs/N eat/V fish/N
/* cats/N eat/V mice/N
/* cats/N like/V fish/N
/* dogs/N fish/V

Trick: add a dummy label * at the beginning of each sentence

Smoothed emissions:

$$P_E(\text{dogs}|N) = \frac{2+1}{7+6} \quad P_E(\text{dogs}|V) = \frac{0+1}{4+6}$$

$$P_E(\text{cats}|N) = \frac{2+1}{7+6} \quad P_E(\text{cats}|V) = \frac{0+1}{4+6}$$

$$P_E(\text{eat}|N) = \frac{0+1}{7+6} \quad P_E(\text{eat}|V) = \frac{2+1}{4+6}$$

$$P_E(\text{like}|N) = \frac{0+1}{7+6} \quad P_E(\text{like}|V) = \frac{1+1}{4+6}$$

$$P_E(\text{fish}|N) = \frac{2+1}{7+6} \quad P_E(\text{fish}|V) = \frac{1+1}{4+6}$$

$$P_E(\text{mice}|N) = \frac{1+1}{7+6} \quad P_E(\text{mice}|V) = \frac{0+1}{4+6}$$

Smoothed transitions:

$$P_T(N|N) = \frac{0+1}{4+2}$$

$$P_T(V|N) = \frac{4+1}{4+2}$$

$$P_T(N|V) = \frac{3+1}{3+2}$$

$$P_T(V|V) = \frac{0+1}{3+2}$$

$$P_T(N|*) = \frac{4+1}{4+2}$$

$$P_T(V|*) = \frac{0+1}{4+2}$$

Example – Prediction

Test sentence: dogs like fish

$$\hat{y}_i = \arg\max_{y_i \in Y}(P_T(y_i|y_{i-1}) \cdot P_E(x_i|y_i))$$

Computations:

	dogs	like	fish
N	$\frac{5}{6} \cdot \frac{3}{13} = 0.192$	$\frac{1}{6} \cdot \frac{1}{13} = 0.013$	$\frac{4}{5} \cdot \frac{3}{13} = 0.185$
V	$\frac{1}{6} \cdot \frac{1}{10} = 0.017$	$\frac{5}{6} \cdot \frac{2}{10} = 0.167$	$\frac{1}{5} \cdot \frac{2}{10} = 0.040$
	$\hat{y} = N$	$\hat{y} = V$	$\hat{y} = N$

Still works, as before

Another example

Test sentence: cats fish fish

$$\hat{y}_i = \arg\max_{y_i \in Y}(P_T(y_i|y_{i-1}) \cdot P_E(x_i|y_i))$$

Computations:

	cats	fish	fish
N	$\frac{5}{6} \cdot \frac{3}{13} = 0.192$	$\frac{1}{6} \cdot \frac{3}{13} = 0.038$	$\frac{4}{5} \cdot \frac{3}{13} = 0.185$
V	$\frac{1}{6} \cdot \frac{1}{10} = 0.017$	$\frac{5}{6} \cdot \frac{2}{10} = 0.167$	$\frac{1}{5} \cdot \frac{2}{10} = 0.040$
	$\hat{y} = N$	$\hat{y} = V$	$\hat{y} = N$

Seems to work fine for ambiguous words too!

Computing the probabilities

- Now, we compute one argmax for each position:

$$\widehat{y_i} = \arg\max_{y_i \in Y}(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}))$$

- What if we want to get the joint probability of words and predicted labels?

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y}(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}))$$

  - Replace the argmax by max
  - Multiply the probabilities of each position

- We haven't done this multiplication yet...
  - It doesn't change the predictions, but it will become important later...

Example – Prediction

Test sentence: dogs like fish

Computations:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y} (P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}))$$

	dogs	like	fish
N	$\frac{5}{6} \cdot \frac{3}{13} = 0.192$	$0.192 \cdot \frac{1}{6} \cdot \frac{1}{13} = 0.0025$	$0.0321 \cdot \frac{4}{5} \cdot \frac{3}{13} = 0.0059$
V	$\frac{1}{6} \cdot \frac{1}{10} = 0.017$	$0.192 \cdot \frac{5}{6} \cdot \frac{2}{10} = 0.0321$	$0.0321 \cdot \frac{1}{5} \cdot \frac{2}{10} = 0.0013$
	$\hat{y} = N$	$\hat{y} = V$	$\hat{y} = N$

Probabilities get smaller and smaller as we proceed, but the predictions remain the same

What's the problem with this approach?

What we compute now:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

Check all labels for one position

- What's the problem here?

What we really should compute:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \max_{y_{1..n} \in Y^n} \prod_{i=1}^{n} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

Check all combinations of labels for the entire sentence

Example – Prediction



Test sentence: fish dogs like cats

Computations:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y} (P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}))$$

	fish	dogs	like	cats
N	$\frac{5}{6} \cdot \frac{3}{13} = \mathbf{0.192}$	$0.192 \cdot \frac{1}{6} \cdot \frac{3}{13} = 0.0074$	$0.016 \cdot \frac{4}{5} \cdot \frac{1}{13} = \mathbf{0.00098}$	$0.00098 \cdot \frac{1}{6} \cdot \frac{3}{13} = 0.000038$
V	$\frac{1}{6} \cdot \frac{2}{10} = 0.033$	$0.192 \cdot \frac{5}{6} \cdot \frac{1}{10} = \mathbf{0.016}$	$0.016 \cdot \frac{1}{5} \cdot \frac{2}{10} = 0.00064$	$0.00098 \cdot \frac{5}{6} \cdot \frac{1}{10} = \mathbf{0.000082}$
	N	V	N	V

Unintuitive labelling – the transitions overpower the emissions

Example – Prediction



Test sentence: fish dogs like cats

Computations:

This probability is much higher than the one we found for N V N V: 0.000082

	fish	dogs	like	cats
N	$\frac{5}{6} \cdot \frac{3}{13} = 0.192$	$0.192 \cdot \frac{1}{6} \cdot \frac{3}{13} = 0.0074$	$0.0074 \cdot \frac{1}{6} \cdot \frac{1}{13} = 0.00009$	$0.0012 \cdot \frac{4}{5} \cdot \frac{3}{13} = 0.00022$
V	$\frac{1}{6} \cdot \frac{2}{10} = 0.033$	$0.192 \cdot \frac{5}{6} \cdot \frac{1}{10} = 0.016$	$0.0074 \cdot \frac{5}{6} \cdot \frac{2}{10} = 0.0012$	$0.0012 \cdot \frac{1}{5} \cdot \frac{1}{10} = 0.000024$
	N	N	V	N

We choose the non-optimal solution here

What's the problem?

What we actually compute:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

- Greedy/approximate inference/decoding

What we really should compute:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \max_{y_{1..n} \in Y^n} \prod_{i=1}^{n} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

- Exact inference/decoding

The results are not the same...

What's the problem?

What we actually compute:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \prod_{i=1}^{n} \max_{y_i \in Y} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

- Example: sentence of 20 words, 17 labels (UPOS)
- $20 \cdot 17 = 340$ computations

What we really should compute:

$$P(x_{1..n}, \widehat{y_{1..n}}) = \max_{y_{1..n} \in Y^n} \prod_{i=1}^{n} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

- Example: sentence of 20 words, 17 labels (UPOS)
- $17^{20} = \sim 4\,000\,000\,000\,000\,000\,000\,000$ computations! That looks bad...

Exact inference

$$P(x_{1..n}, \widehat{y_{1..n}}) = \max_{y_{1..n} \in Y^n} \prod_{i=1}^{n} \left(P_E(x_i|y_i) \cdot P_T(y_i|y_{i-1}) \right)$$

- That's one big computation for the whole sentence
- This computation is intractable
  - But there are two tricks!
  - The **Viterbi algorithm** includes these two tricks

The Viterbi algorithm

- **Trick 1:** Decomposing the product
  - Proceed from left to right, one word at a time
  - Store the results of all computations so that you don't have to recompute them later

- **Trick 2:** The Markov assumption
  - The prediction formula for y_i only depends on y_{i-1}, not on all labels back to y_0
  - But there is only one best path towards y_{i-1}, so we can skip all other, non-optimal paths to y_{i-1}



Andrea/Andrew Viterbi, *1935

The Viterbi algorithm

- A nice example of dynamic programming
- One of the classic NLP algorithms
- But we won't present the details here:
  - Viterbi is covered in IN2110
  - State-of-the-art sequence labelling models don't use it

Comparison:

- Greedy decoding:
  - $20 \cdot 17 = 340$ computations
- Exact decoding, brute force:
  - $17^{20} = \sim 4\,000\,000\,000\,000\,000\,000\,000\,000$ computations
- Exact decoding with Viterbi algorithm:
  - $20 \cdot (17^2) = 5780$ computations

Discriminative sequence labeling

Decoding strategies



Matthew Honnibal, 2013:

> *"Unless you really, really can't do without an extra 0.1% of accuracy, you probably shouldn't bother with any kind of search strategy, you should just use a greedy model."*
>
> https://explosion.ai/blog/part-of-speech-pos-tagger-in-python
>
> *"[Viterbi] search can only help you when you make a mistake. It can prevent that error from throwing off your subsequent decisions, or sometimes your future choices will correct the mistake. [... A modern greedy] model is so good straight-up that your past predictions are almost always true. So you really need the planets to align for search to matter at all."*

Different ways to look at the context

Option 1: Look at the previous tag decisions:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_i, y_{i-1})$$

- HMM

Option 2: Look at the neighboring words:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_{i-1}, x_i, x_{i+1})$$

- Logistic regression

Let's look at this option next!

Option 3: Combine the two:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_{i-1}, x_i, x_{i+1}, y_{i-1})$$

- CRF, MEMM

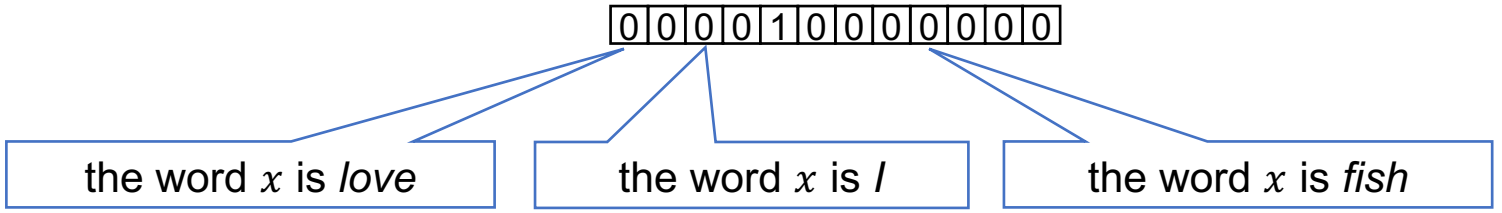
Predicting one label per word

Naive Bayes: $\hat{y} = \arg\max_{y \in Y}(P(y) \cdot P(x|y))$

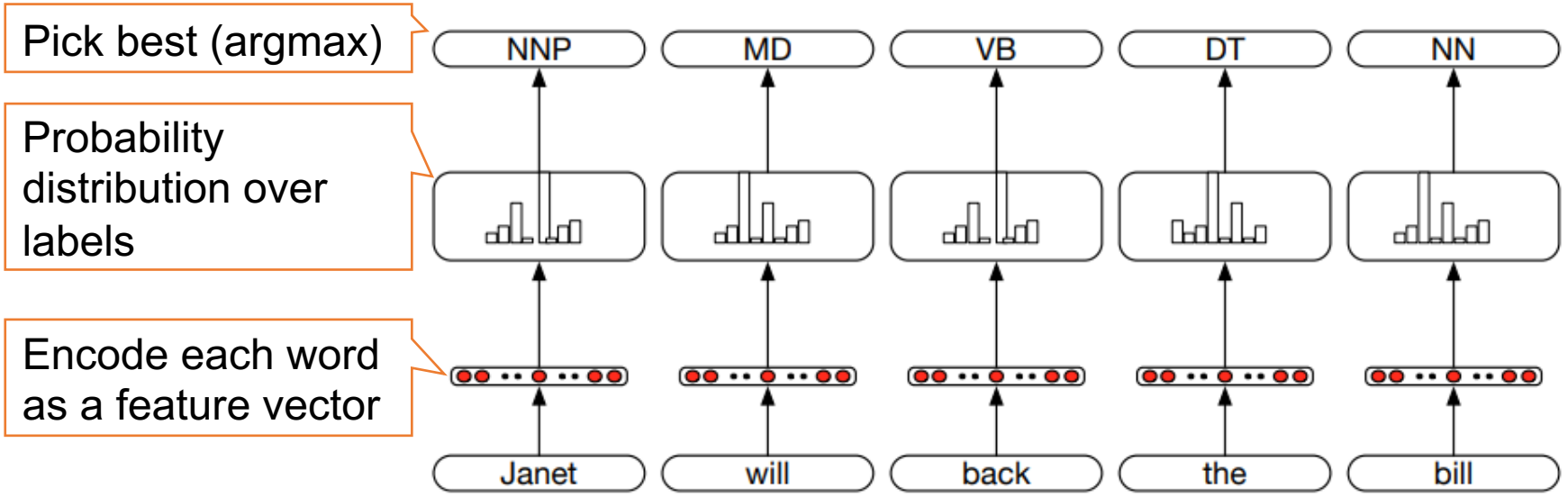
- x is a single word, so $P(x|y)$ is a single value (no bag-of-words decomposition)

Logistic regression: $\hat{y} = \arg\max_{y \in Y}(\boldsymbol{w}^{(y)} \cdot \boldsymbol{f}^{(x)})$

- (Let's skip the softmax for simplicity)
- x is a single word, so $\boldsymbol{f}^{(x)}$ is (typically) a **one-hot vector**:



A LR unigram model



One-hot encoding: 0 0 0 0 1 0 0 0 0 0 0 0

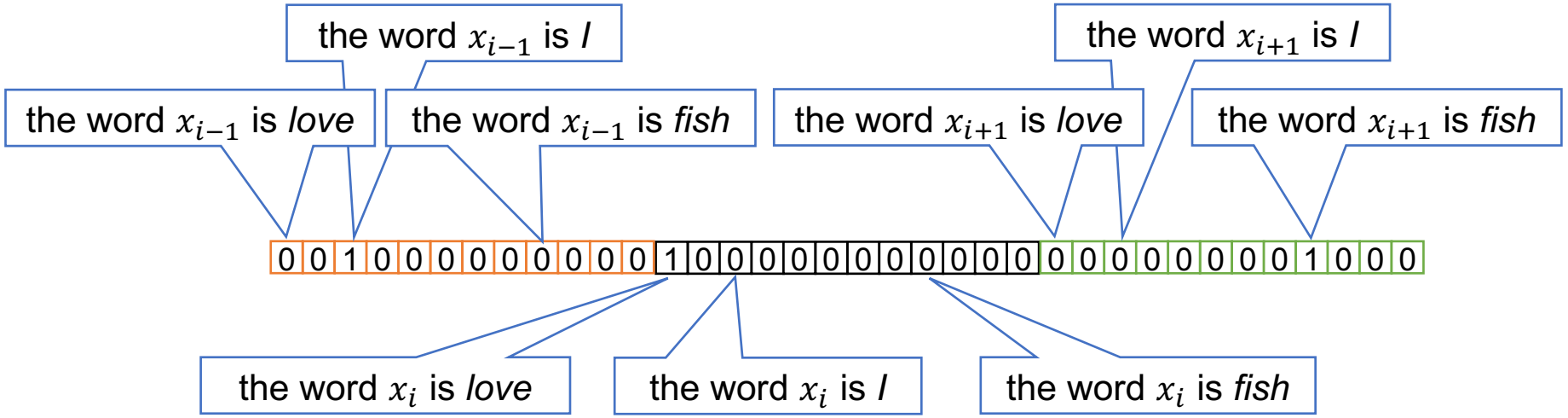
the word x is *love*

the word x is *I*

the word x is *fish*

Include information from context words

$$\boldsymbol{f}^{(\boldsymbol{x})} = \boldsymbol{f}^{(\boldsymbol{x_{i-1}})} \oplus \boldsymbol{f}^{(\boldsymbol{x_i})} \oplus \boldsymbol{f}^{(\boldsymbol{x_{i+1}})}$$

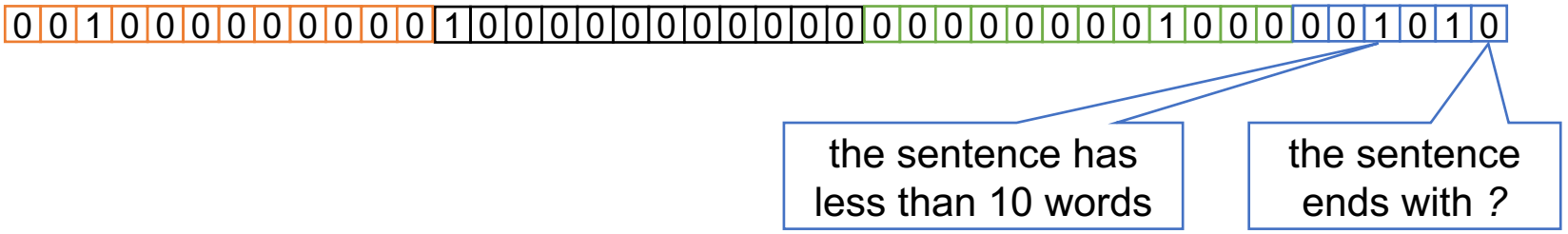


Example: *love* in the context *I* _ *fish*

Context word features

$$\boldsymbol{f}^{(x)} = \boldsymbol{f}^{(x_{i-1})} \oplus \boldsymbol{f}^{(x_i)} \oplus \boldsymbol{f}^{(x_{i+1})} \oplus \boldsymbol{f}^{(x^*)}$$

Using additional sentence-level features $\boldsymbol{f}^{(x^*)}$:



Include information from context words

$$f^{(x)} = f^{(x_{i-1})} \oplus f^{(x_i)} \oplus f^{(x_{i+1})}$$

With fixed-length pretrained word vectors:



LSA-based, or word2vec vectors:

```
said 0.0552 -0.0181 -0.0561 0.0086 -0.0203 -0.1505 -0.0029 0.0232 -0.0645 -0.2647
more -0.0588 -0.0566 -0.0397 0.0094 0.0356 0.0346 0.0027 -0.0144 0.0238 -0.0855
page -0.0702 0.0289 0.0413 0.0366 0.1205 -0.1255 -0.0043 0.0970 0.1616 -0.0868
he 0.0468 -0.0502 0.0332 -0.1099 -0.0402 -0.0179 -0.0137 -0.0941 0.0077 0.0397
your -0.1001 -0.0906 -0.0496 -0.0332 0.0295 -0.1788 0.0294 -0.1279 0.0420 0.1440
will -0.0368 0.0231 -0.1271 -0.1208 -0.0330 -0.0648 -0.0633 -0.0936 0.0096 -0.0760
its -0.0665 -0.0693 0.0478 -0.0544 -0.0050 0.0355 0.0339 0.0049 0.0325 -0.1453
so -0.0154 -0.0020 -0.0725 -0.0519 -0.0176 -0.0549 -0.0281 0.0256 0.0891 0.0902
were 0.0995 -0.0703 -0.1248 0.0915 -0.0324 -0.0358 0.0974 0.0636 -0.0521 0.0291
had 0.1112 -0.1055 0.0189 -0.0002 0.0389 -0.0208 0.0428 0.0665 -0.0183 -0.0010
also -0.0764 0.0451 -0.0883 0.0223 -0.2671 0.0404 0.0121 -0.0349 0.0515 -0.1087
```

How to handle unknown words?

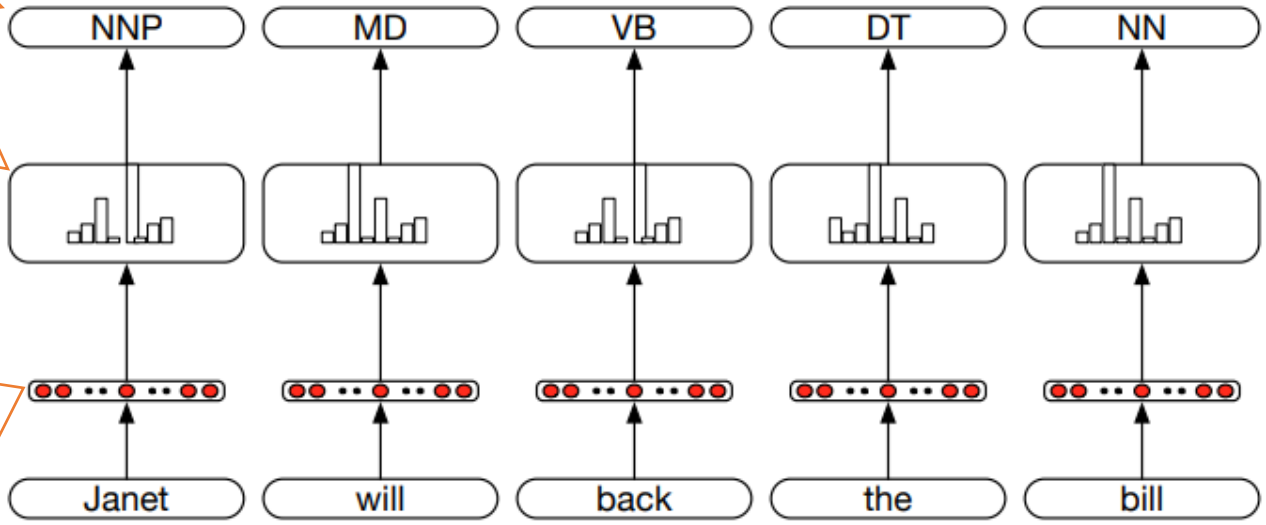
- Reserve one value in the vector for all unknown/rare words
- Add features for word shapes
  - Contains a particular prefix/suffix
  - Contains a number
  - Contains upper-case letters
- If using pretrained vectors: simulate a word vector on the basis of similar-looking words
- Split sentence into subwords
  - No more 1-to-1 mapping between tokens and labels

A LR unigram model

Pick best (argmax)

Probability distribution over labels

Encode each word and its context as a feature vector of arbitrary (but fixed) length



Different ways to look at the context

Option 1: Look at the previous tag decisions:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_i, y_{i-1})$$

- HMM

Option 2: Look at the neighboring words:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_{i-1}, x_i, x_{i+1})$$

- Logistic regression

Option 3: Combine the two:

$$\hat{y}_i = \arg\max_{y_i \in Y} P(y_i | x_{i-1}, x_i, x_{i+1}, y_{i-1})$$

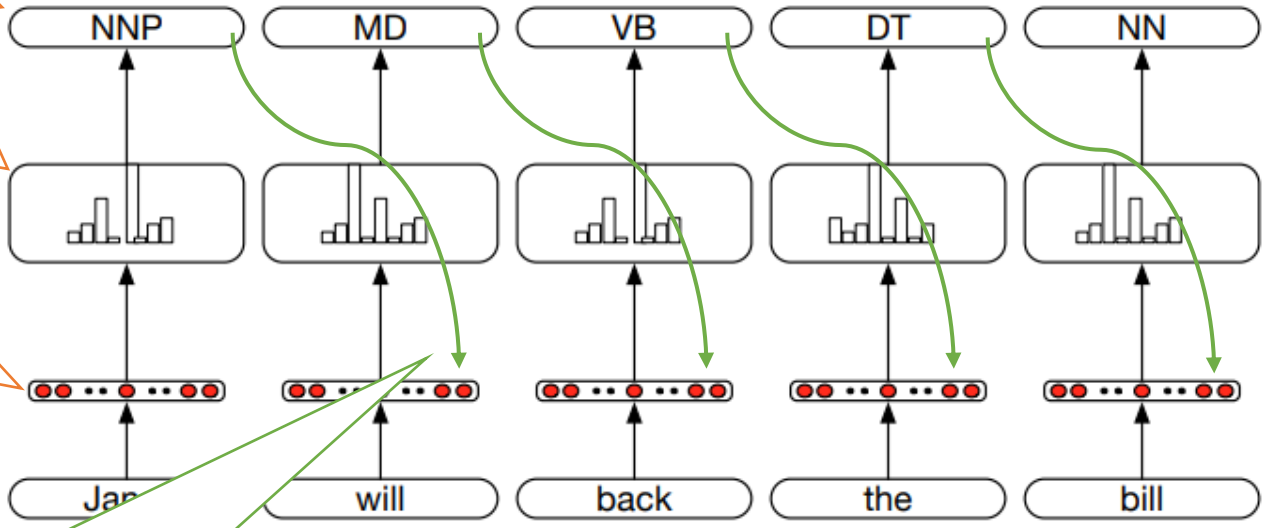
- Greedy LR, CRF, MEMM

A LR model with greedy decoding

Pick best (argmax)

Probability distribution over labels

Encode each word as a feature vector



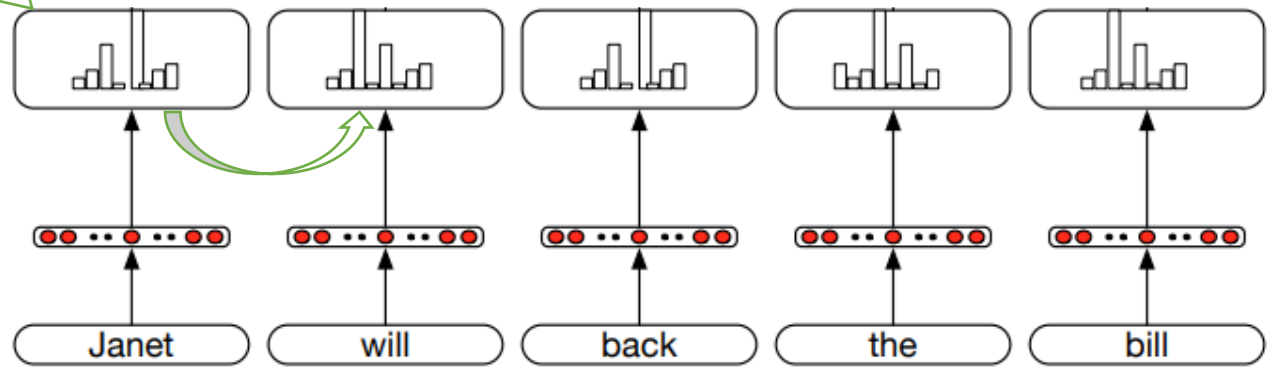
Greedy decoding: The label predicted at the previous position is fed as a one-hot vector into the feature vector of the next word.

$$f^{(x)} = f^{(x_i)} \oplus f^{(y_{i-1})}$$



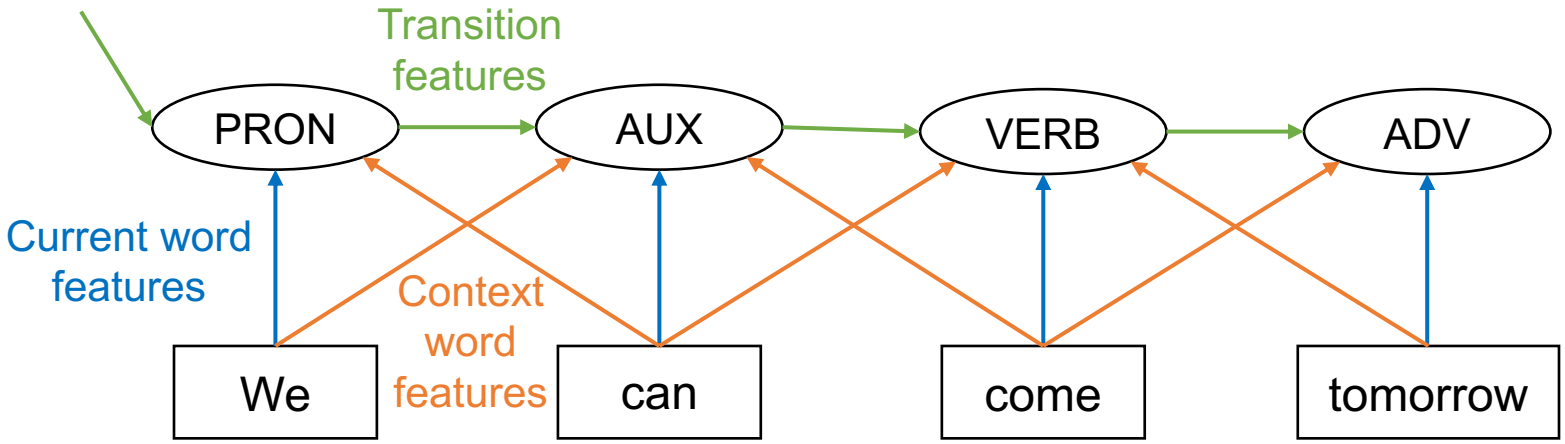
Viterbi decoding for LR models: CRF, MEMM

Viterbi decoding: The labels are only predicted after reaching the end of the sentence. This is known as **CRF layer (conditional random field)** in discriminative models.



The probability distribution over labels at the previous position is taken into account when computing current label probabilities.

Context and transition features



Training a greedy LR sequence model

Training involves prediction:

- Take one training instance $\boldsymbol{x}_{1\ldots n}$
- Predict the label probabilities using the current weight vectors
- For each position $1 \ldots n$:
  - Update the weight vector of the correct label
  - Update the weight vectors of the incorrect labels
- Continue “until tired”

Conditional random fields

Prediction formula without softmax:

$$S(x_{1..n}, \widehat{y_{1..n}}) = \max_{y_{1..n} \in Y^n} \left(\sum_{i=1}^{n} (w_T(y_i, y_{i-1}) + w_E(y_i) \cdot f_E(x_i)) \right)$$

Apply softmax (exponentiation and normalization):

$$P(\widehat{y_{1\ldots n}} | x_{1\ldots n}) = \max_{\boldsymbol{y}_{1\ldots n} \in Y^n} \left(\frac{\exp\left(\sum_{i=1}^{n+1} \left(\boldsymbol{w}_T(y_i, y_{i-1}) + \boldsymbol{w}_E(y_i) \cdot \boldsymbol{f}_E(\boldsymbol{x}, i)\right)\right)}{Z(\boldsymbol{w}_E, \boldsymbol{w}_T, \boldsymbol{x})} \right)$$

- where $Z(\boldsymbol{w}_E, \boldsymbol{w}_T, \boldsymbol{x})$ is called the **partition function**:

$$Z(\boldsymbol{w}_E, \boldsymbol{w}_T, \boldsymbol{x}) = \sum_{\boldsymbol{z}_{1\ldots n} \in Y^n} \exp\left(\sum_{i=1}^{n+1} \left(\boldsymbol{w}_T(z_i, z_{i-1}) + \boldsymbol{w}_E(z_i) \cdot \boldsymbol{f}_E(\boldsymbol{x}, i)\right) \right)$$

This model is called **conditional random field** (CRF).

Maximum Entropy Markov models (MEMMs)

Maximum Entropy Markov Models are a simpler alternative to CRFs:

- Compute the Z normalization term per word
- More efficient to train, but less performant

CRF

$$P(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) = \max_{y_{1\ldots n}\in Y^n} \frac{\exp\left(\sum_{i=1}^{n+1}\left(\boldsymbol{w}_T(y_i, y_{i-1}) + \boldsymbol{w}_E(y_i)\cdot \boldsymbol{f}_E(\boldsymbol{x}, i)\right)\right)}{Z(\boldsymbol{w},\boldsymbol{x})}$$

$$Z(\boldsymbol{w},\boldsymbol{x}) = \sum_{z_{1\ldots n}\in Y^n} \exp\left(\sum_{i=1}^{n+1}\left(\boldsymbol{w}_T(z_i, z_{i-1}) + \boldsymbol{w}_E(z_i)\cdot \boldsymbol{f}_E(\boldsymbol{x}, i)\right)\right)$$

MEMM

$$P(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) = \max_{y_{1\ldots n}\in Y^n} \left(\prod_{i=1}^{n+1} \frac{\exp\left(\boldsymbol{w}_T(y_i, y_{i-1}) + \boldsymbol{w}_E(y_i)\cdot \boldsymbol{f}_E(\boldsymbol{x}, i)\right)}{Z(\boldsymbol{w},\boldsymbol{x})}\right)$$

$$Z(\boldsymbol{w},\boldsymbol{x}) = \sum_{z_i\in Y} \exp(\boldsymbol{w}_T(z_i, y_{i-1}) + \boldsymbol{w}_E(z_i)\cdot \boldsymbol{f}_E(\boldsymbol{x}, i))$$

Models

- Unigram classification models:
  - Naïve Bayes
  - Logistic regression with context word features
- Sequence models with greedy decoding:
  - Hidden Markov model
  - Logistic regression with transition features
- Sequence models with exact/Viterbi decoding:
  - Hidden Markov model
  - CRF, MEMM

Combined approach: use greedy decoding to speed up training, but use Viterbi decoding for best prediction accuracy

Reading

- Jurafsky & Martin, chapter 17
  - HMM, Viterbi algorithm, CRF

- Jurafsky & Martin, appendix A (not required)
  - Implementation details of the Viterbi algorithm