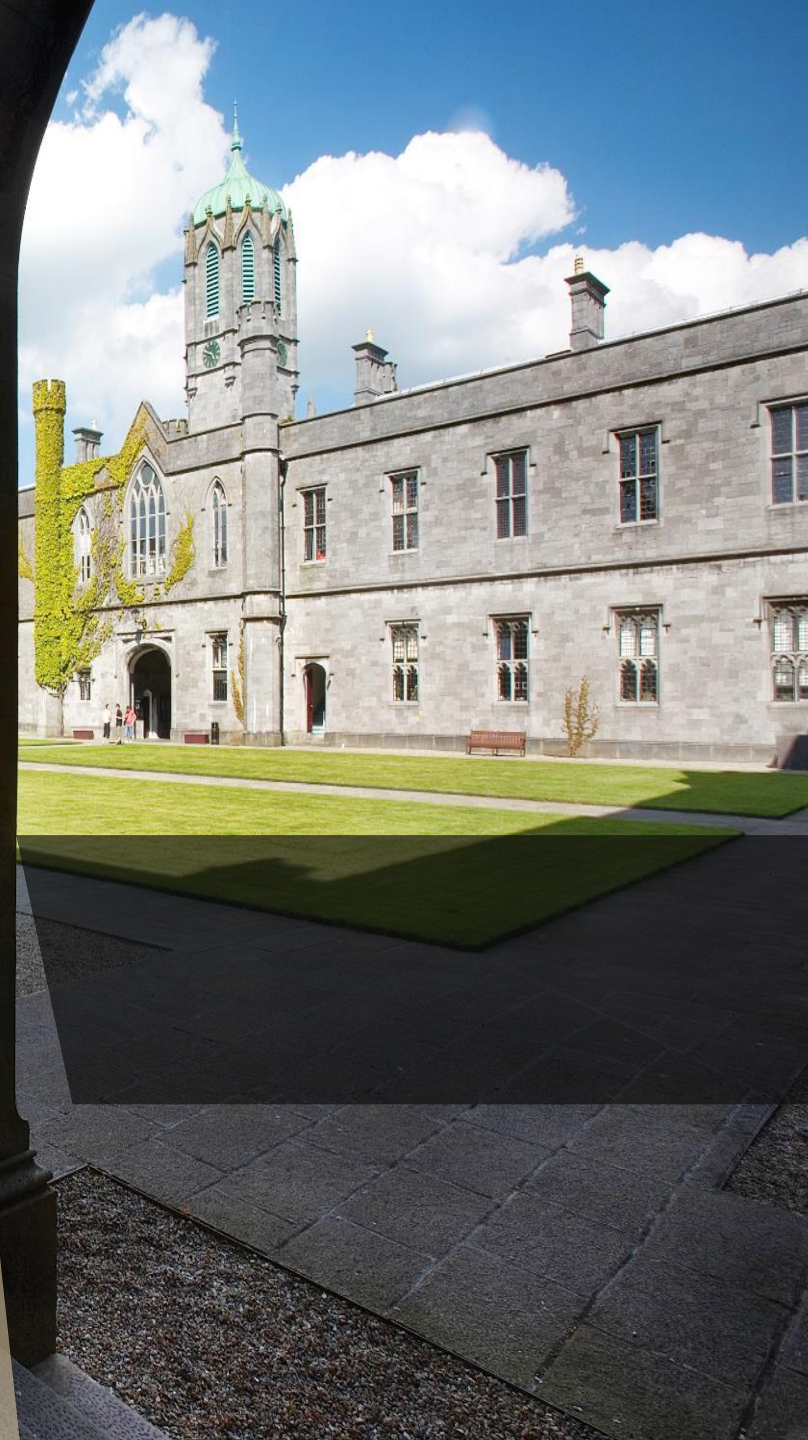
Introduction to NLP

Tagging and Hidden Markov Models

Dr. John McCrae

Data Science Institute, NUI Galway



Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Probability

We create probability over events **p(A)**

Combinations of events **p(A∩B)**

Events given other events **p(A|B∩C)**

Probability of a sentence

p("I like green eggs") is the probability of the intersection of 4 events:

$$\mathbf{p(w_1=\text{"I"} \cap w_2=\text{"like"} \cap w_3=\text{"green"} \cap w_4=\text{"eggs"})}$$

This can be hard to estimate for long sentences

Conditional Probabilities

We can express this in terms of conditional probabilities

p("I like green eggs") =
p("eggs"|"I like green") ×
p("green"|"I like") ×
p("like"|"I") ×
p("I")

N-gram approximations

We can express this in terms of conditional probabilities

p("I like green eggs") ≈
p("eggs"|"green") ×
p("green"|"like") ×
p("like"|"I") ×
p("I")

Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Syntax

Goal of a language model is to predict whether a sentence is in English (or another language)

Syntax is important

Correct:	*Happy children play*	(**Adj Noun Verb)**
Incorrect:	*Children happy play*	**(Noun Adj Verb)**

Trees next week

Tags as probabilities

We now have two kinds of events:

$$p(w_1=\text{"happy"} \cap w_2=\text{"children"} \cap w_3=\text{"play"} \cap t_1=\text{Adj} \cap t_2=\text{Noun} \cap t_3=\text{Verb})$$

This is not possible to calculate for long sentences => Approximation

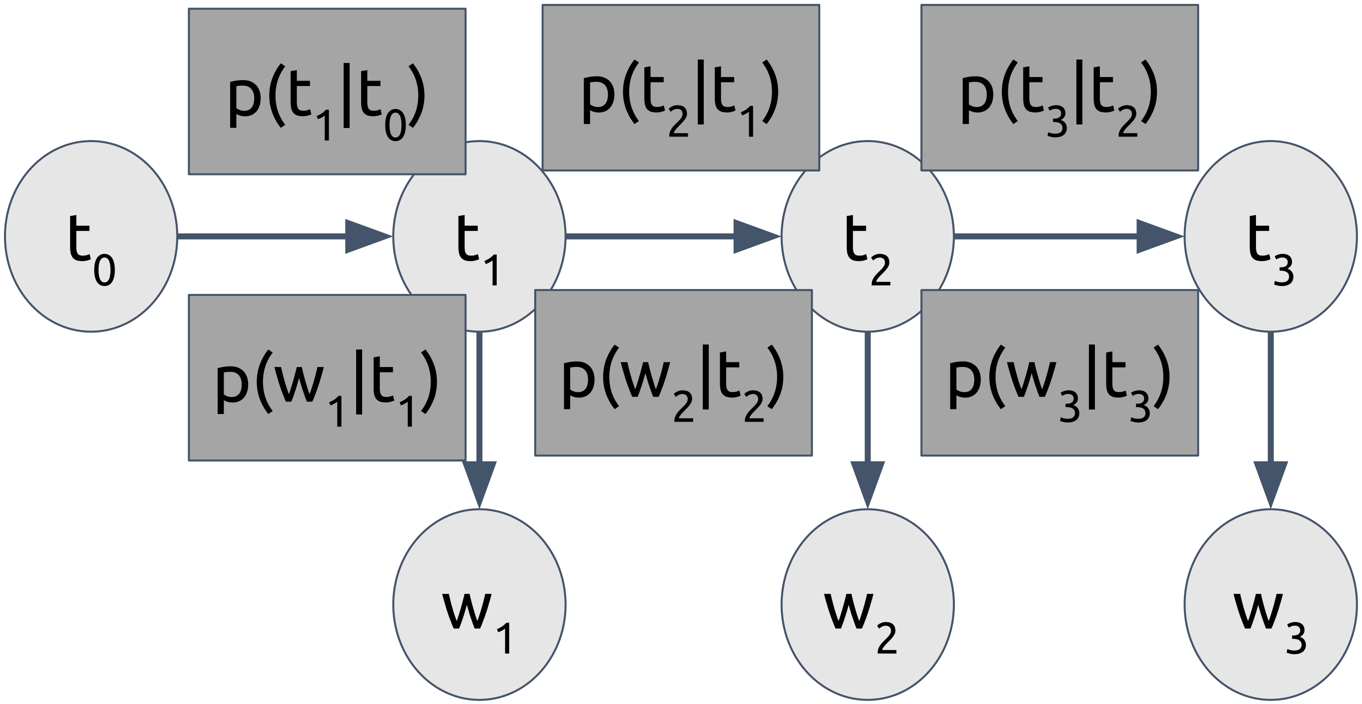
Hidden Markov Model (without approximation)

$$\begin{gathered}
p(w_1,w_2,w_3,t_1,t_2,t_3) = \\
p(w_3|t_3,t_2,t_1,w_1,w_2) \times \\
p(t_3|t_2,t_1,w_1,w_2) \times \\
p(w_2|t_2,t_1,w_1) \times \\
p(t_2|t_1,w_1) \times \\
p(w_1|t_1) \times \\
p(t_1)
\end{gathered}$$

Hidden Markov Model (with approximation)

$$p(w_1,w_2,w_3,t_1,t_2,t_3) \approx$$
$$p(w_3|t_3) \times$$
$$p(t_3|t_2) \times$$
$$p(w_2|t_2) \times$$
$$p(t_2|t_1) \times$$
$$p(w_1|t_1) \times$$
$$p(t_1)$$

Hidden Markov Model



Hidden Markov Model

The general form for this is:

$$p(w_1, \ldots, w_n, t_1, \ldots, t_n) \approx \prod_{i=1,\ldots,n} p(t_i|t_{i-1})p(w_i|t_i)$$

We introduce a variable t_0 which always has the same value ('START')

HMM Example

For "John likes Mary", the probability of tagging it as "NOUN VERB NOUN" is

$$\mathbf{p(w_1{=}John, w_2{=}likes, w_3{=}Mary, t_1{=}N, t_2{=}V, t_3{=}N) = p(N|V) \times p(V|N) \times p(N|Start) \times p(John|N) \times p(likes|V) \times p(Mary|N)}$$

Stochastic Transition Matrices

	V	**N**
Start	0.2	0.8
V	0.2	0.6
N	0.6	0.4

	John	***likes***	***Mary***
V	0.2	0.6	0.2
N	0.4	0.2	0.4

HMM Example

$$p(w_1{=}\text{John}, w_2{=}\text{likes}, w_3{=}\text{Mary}, t_1{=}N, t_2{=}V, t_3{=}N) =$$
$$p(N|V) \times p(V|N) \times p(N|\text{Start}) \times$$
$$p(\text{John}|N) \times p(\text{likes}|V) \times p(\text{Mary}|N) =$$
$$0.6 \times 0.6 \times 0.8 \times 0.4 \times 0.6 \times 0.4 = 0.028$$

Formal Definition

An HMM is a 4-tuple $\mu=(S,K,A,B)$

Set of states (tags)	$S=\{s_1(=\sigma),\ldots,s_N\}$
Output alphabet	$K=\{k_1,\ldots,k_M\}$
State transition probabilities	$A=\{a_{ij}\}\ i,j \in S$
Symbol Emission probabilities	$B=\{b_{ij}\}\ i \in S, j \in K$
State sequences (hidden)	$T=\{t_1, \ldots, t_T\}\ y_T \in S$
Output sequence (observed)	$W=\{w_1,\ldots,w_T\}\ x_T \in K$

NUI Galway
OÉ Gaillimh

Three fundamental problems for HMMs

1. What is the probability of a sequence given an observation and a model?
   a. What is **P(DET N VBZ DET N|the cat chases the mouse, μ)**?

2. What is the probability of an observation given the model?
   a. What is **P(the cat chases the mouse|μ)**?

3. What is the model that maximizes the likelihood of the observed data and known sequences?
   a. What μ maximizes **P(the cat chases the mouse, DET N VBZ DET N|μ)**?
   b. What μ maximizes **P(the cat chases the mouse|μ)**?

Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Tagging

Tagging is the process of assigning (exactly) one class to (each) word

Part-of-Speech Tagging

This	*is*	*an*	*example*
DT	VBZ	DT	NN

Named Entity Recognition

in	*New*	*York*	*City*	*yesterday*
O	B	I	I	O

HMMs for Tagging

How do we use HMMs for tagging?

$$\arg\max_{t_1,\ldots,t_n} p(t_1,\ldots,t_n|w_1,\ldots,w_n)$$

Example

If *S={V,N}*, what is the most likely tagging of "cats drink"?

p(N,N,cats,drink) = p(N|Start) × p(N|N) × p(cats|N) × p(drink|N)

p(N,N,cats,drink)	0.023
p(N,V,cats,drink)	0.034
p(V,N,cats,drink)	0.017
p(V,V,cats,drink)	0.021

Complexity is $\mathcal{O}(TN^T)$

Example

What about three words, e.g., "*cats drink milk*"?

p(N,N,N,cats,drink,milk) = p(N|Start) × p(N|N) × p(N|N)
p(cats|N) × p(drink|N) × p(milk|N)

p(V,N,N,cats,drink,milk) = p(V|Start) × p(N|V) × p(N|N)
p(cats|V) × p(drink|N) × p(milk|N)

Dynamic Programming

a.k.a., caching
a.k.a., DRY
a.k.a., remembering the results of previous calculations

NUI Galway
OÉ Gaillimh

Recursive algorithm for HMMs

$$\pi_{s,i} = \max_{s' \in S} p(t_i|s')p(w_i|t_i)\pi_{s',i-1}$$

$$\max p(w_1, \ldots, w_n, t_1, \ldots, t_n) = \max \pi_{t_n,n}$$

Still exponential complexity

Example (Python)

```
def pi(s, i):
    if i == 0 && s == “start”:
    return 1.0
  else if i == 0:
    return 0.0
  return max(a(s,t) * b(w[i],s) * pi(t, i -1)
                  for t in states)
```

NUI Galway
OÉ Gaillimh

Dynamic programming (Python)

```
def pi(s, i):
  if i == 0 && s == “start”:
    return 1.0
  else if i == 0:
    return 0.0
  else if mem[s,i]:
    return mem[s,i]
  else:
    p = max(a(s,t) * b(w[i],s) * pi(t, i -1)
            for t in states)
    mem[s,i] = p
    return p
```

NUI Galway
OÉ Gaillimh

Viterbi algorithm

Set $\pi_{s,0}$=0 except for $\pi_{Start,0}$=1

Set y_s = []

For i from 1 to T

  For s ∈ S

    Set $\pi_{s,i} = \max_{t \in S} \pi_{t,i-1} p(s|t) p(w_i|s)$

   Append *t* to y_s

Return y_s + s where s

  maximizes $\pi_{s,T}$

Can be more efficiently implemented using only two vectors for $\pi_{s,i}$ and $\pi_{s,i-1}$

Viterbi Algorithm Example - Table

	John	*cat*	*has*	*a*
N	0.3	0.3	0.1	0.3
V	0.1	0.1	0.7	0.1

	p(N|.)	**p(V|.)**
N	0.7	0.3
V	0.5	0.5
Start	0.9	0.1

"John has"

NUI Galway
OÉ Gaillimh

Viterbi Algorithm Example

Previous state (i=0)

Start	N	V
1.0	0.0	0.0
[]	[]	[]

Next state (i=1)

Start	N	V
0.0	max(0.9 × 0.3, 0.0, 0.0) = 0.27	max(0.1 × 0.1, 0.0, 0.0) = 0.01
	[Start]	[Start]

NUI Galway
OÉ Gaillimh

Viterbi Algorithm Example

Previous state (i=1)

	N	V
	0.27	0.01
	[Start]	[Start]

Next state (i=2)

N

max(0.27 × 0.7 × 0.1,
0.01 × 0.5 × 0.1) = 0.019

[Start, N]

V

max(0.27 × 0.3 × 0.7,
0.01 × 0.5 × 0.7) = 0.057

[Start,N]

NUI Galway
OÉ Gaillimh

Viterbi Algorithm Example

Final state:

As $\pi_{V,2} > \pi_{N,2}$
return [Start, N, V]

Complexity is $\mathcal{O}(TN^2)$

Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

HMM as a language model

Recall the law of total probability

$$p(A = a) = \sum_{b} p(A = a \cap B = b)$$

Hidden Markov Models can also work as language models

$$p(w_1, \ldots, w_n) = \sum_{t_1 \in S, \ldots, t_n \in S} p(w_1, \ldots, w_n, t_1, \ldots, t_n)$$

Calculating HMM LM probability

We can follow same principle of calculating all values

$$p(w_1, \ldots, w_n) = \sum_{t_1 \in S} \cdots \sum_{t_n \in S} \prod_{i=1,\ldots,n} p(t_i|t_{i-1})p(w_i|t_i)$$

Direct calculation of this is exponential.

Instead we modify the Viterbi algorithm.

Forward algorithm

Set $\alpha_{s,0}$=0 except for $\alpha_{Start,0}$=1

For i from 1 to T

For s ∈ S

Set $\alpha_{s,i} = \sum_{t \in S} \alpha_{t,i-1} p(s|t) p(w_i|s)$

Return $\Sigma\alpha_{s,T}$

$\alpha_{s,i}$ is the sum of probabilities up to state *s* at word *i*

NUI Galway
OÉ Gaillimh

Forward algorithm Example

Previous state (i=0)

Start	N	V
1.0	0.0	0.0

Next state (i=1)

Start	N	V
0.0	$0.9 \times 0.3 + 0.0 + 0.0 = 0.27$	$0.1 \times 0.1 + 0.0 + 0.0 = 0.01$

Forward Algorithm Example

Previous state (i=1)

N	**V**
0.27	0.01

Next state (i=2)

N

$0.27 \times 0.7 \times 0.1 +$
$0.01 \times 0.5 \times 0.1 = 0.019$

V

$0.27 \times 0.3 \times 0.7 +$
$0.01 \times 0.5 \times 0.7 = 0.060$

Final **p = 0.019 + 0.060 = 0.0796**

Backward Algorithm

Alternatively we can calculate probabilities backwards:

$$\beta_{s,i} = P(t_i = s, w_{t+1} \dots w_T | \mu)$$

$$\beta_{s,T} = 1$$

$$\beta_{s,i} = \sum_{t \in S} \beta_{t,i+1} p(t|s) p(w_{i+1}|t)$$

These backward probabilities can be calculated using a dynamic programming approach much as for the forward variables.

Forward-Backward Procedure

If we combine these forward and backward variables, we can find the probability of any single tag

$$P(T_i = s, W|\mu) = \alpha_{s,i}\beta_{s,i}$$

Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Supervised Learning

If we now the tags for some set of words we can directly obtain the probabilities by counting

$$p(s_i|s_j) = \frac{c(t_{i-1} = s_j, t_i = s_i)}{\sum_{s'} c(t_{i-1} = s_j, t_i = s')}$$

$$p(w|s) = \frac{c(w_i = w, t_i = s)}{c(t_i = s)}$$

Supervised Learning Example

V	**N**	**V**	**O**	**N**	**O**	**N**
Do	you	like	green	eggs	and	ham

N	**N**	**V**	**N**	**O**
Mr	green	eggs	you	on

N	**V**	**O**	**N**
you	are	like	ham

Counts

	N	V	O
Do	0	1	0
You	3	0	0
Like	0	1	1
Green	1	0	1
Eggs	1	1	0
And	0	0	1
Ham	2	0	0
Mr	1	0	0
on	0	0	1
are	0	1	0

	N	V	O
N	1	3	2
V	2	0	2
O	3	0	0
Start	2	1	0

Probabilities

	N	V	O
Do	0/8	1/4	0/4
You	3/8	0/4	0/4
Like	0/8	1/4	1/4
Green	1/8	0/4	1/4
Eggs	1/8	1/4	0/4
And	0/8	0/4	1/4
Ham	2/8	0/4	0/4
Mr	1/8	0/4	0/4
on	0/8	0/4	1/4
are	0/8	1/4	0/4

	N	V	O
N	1/6	3/6	2/6
V	2/4	0/4	2/4
O	3/3	0/3	0/3
Start	2/3	1/3	0/3

Unsupervised Learning

The **unsupervised** case is where we don't have tags

In the supervised case we set probabilities using observed counts

$$p(t|s) = \frac{c(t, s)}{c(s)}$$

In the unsupervised count we can instead use **expected counts**

$$p^*(t|s) = \frac{E[c(t, s)]}{E[c(s)]}$$

Expected Counts

If we throw a coin 4 times, we **expect** to see 2 heads on average

$$E[c(heads)] = \sum_{i=1}^{4} p(heads) = 2$$

We can say the same about our tags, e.g.,

$$E[c(s)] = \sum_{i} P(t_i = s)$$

Unsupervised probabilities

This means we can calculate probabilities in an unsupervised setting by simply using expectancies

$$p^*(s_i|s_j) = \frac{E[c(t_{i-1} = s_j, t_i = s_i)]}{\sum_{s'} E[c(t_{i-1} = s_j, t_i = s'))])}$$

$$p^*(w|s) = \frac{E[c(w_i = w, t_i = s_i)]}{E[c(t_i = s))])}$$

Updating probabilities

Problem: we need the transmission and emissions probabilities, *A, B*, in order to calculate $p^*(s_i|s_j)$ and $p^*(w|s)$, which we then use to update *A, B*! This is a chicken and egg problem.

Solution: Start with random A and B and iterate until convergence

NB. It can be a bit mind-blowing that this actually works, it can be shown formally why, but this is beyond the scope here

Baum-Welch Algorithm

1. Initialize A and B at random
2. Repeat until convergence
   a. Calculate $p^*(s_i|s_j)$ and $p^*(w|s)$ using A,B
   b. Update $A \leftarrow p^*(s_i|s_j)$, $B \leftarrow p^*(w|s)$

This algorithm is a special case of the Expectation-Maximization (E-M) algorithm

A detailed (non-examinable) walkthrough of the Baum-Welch algorithm is here: https://colab.research.google.com/drive/1iDWy-5T-84eD61LhLdL7vwvmfhaYNg7J?usp=sharing

Unsupervised learning

Finding the model that maximizes the likelihood of the data involves estimating the parameters of the HMM. This problem can be formulated as follows. Given a set of states *S*, an output alphabet T and observation *W*, compute the parameters *a* and b of the HMM so that the probability of the observed data is maximized:

$$\mu = argmax_{a,b} P(W|\mu)$$

Learning the model's parameters

There is no known analytic method to select the μ that maximizes the above formula.

We can use an iterative hill-climbing algorithm known as the **Baum-Welch algorithm** that is a special case of the so called **Expectation-Maximization** algorithm.

Expectation-Maximization Algorithm

Chicken and egg problem: To estimate the model we need counts of random variables that we can not directly observe.

The E-M Algorithm applies the following procedure:

1. Initialize the model
2. Estimate the counts
3. Re-compute the model given the estimated counts
4. Repeat 2 and 3 until convergence

Baum-Welch Algorithm

Recall our previous definition of forward and backward variables, and let

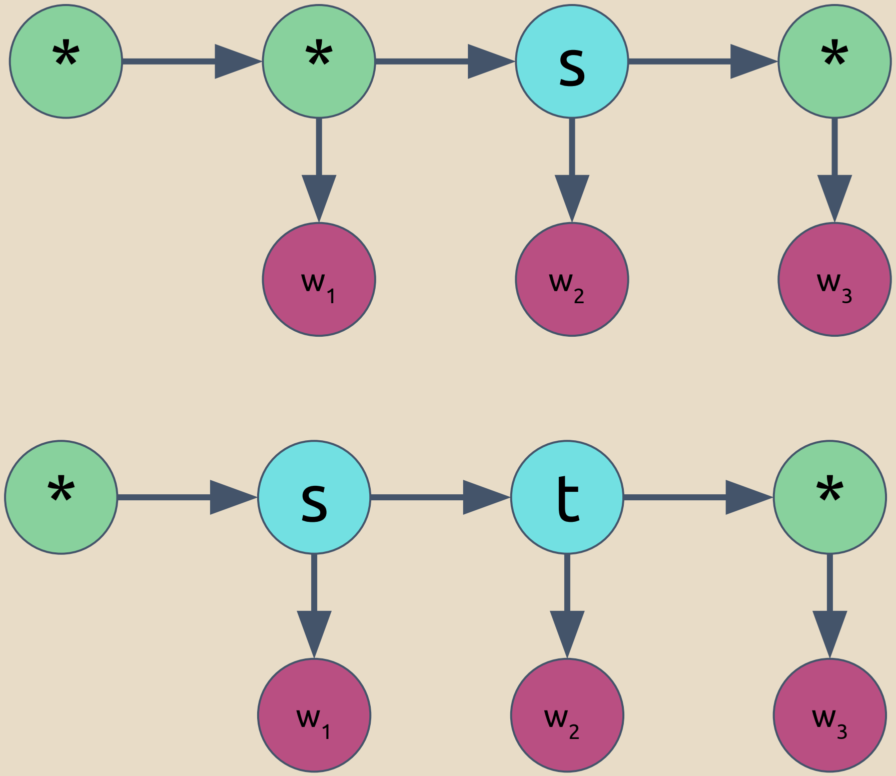
$$\gamma_i(t) = P(t_i = t | W, \mu) = \frac{\alpha_{t,i}\beta_{t,i}}{\sum_{s \in S} \alpha_{s,i}\beta_{s,j}}$$

(Probability of each state)

$$\xi_i(s,t) = P(t_i = s, t_{i+1} = t | W, \mu) = \frac{\alpha_{s,i} a_{s,t} \beta_{t,i+1} b_{t,w_{i+1}}}{\sum_{s' \in S} \sum_{t' \in T} \alpha_{s',i} a_{s',t'} \beta_{t',i+1} b_{t',w_{i+1}}}$$

(Transition probability)

Gamma and Xi



$\gamma_i(s)$ is the probability of all models where the state is $t_i=s$

$\xi_i(s)$ is the probability of all models where the state is $t_{i-1}=s$ and $t_i=t$

Baum-Welch Algorithm

The most likely estimators of our probabilities are thus:

$$a^*_{s,t} = \frac{\sum_{i=1}^{T-1} P(t_i = s, t_{i+1} = t | W, \mu)}{\sum_{i=1}^{T-1} P(t_i = s | W, \mu)} = \frac{\sum_{i=1}^{T-1} \xi_i(s,t)}{\sum_{i=1}^{T-1} \gamma_i(s)}$$

$$b^*_{s,w} = \frac{\sum_{i=1}^{T} \mathbf{1}(w_i = w)\gamma_i(s)}{\sum_{i=1}^{T} \gamma_i(s)}$$

$$\mathbf{1}(w_i = w) = \begin{cases} 1 & \text{if } w_i = w \\ 0 & \text{otherwise} \end{cases}$$

NUI Galway
OÉ Gaillimh

Baum-Welch for Speech Recognition

In practice, Baum-Welch is most useful as a semi-supervised method

Some tags are observed, some not

Most frequently in speech recognition:

- Train the system on known data
- Adapt to an individual speaker

Semi-supervised Baum-Welch

Semi-supervised Baum-Welch, combines the supervised and unsupervised modes.

We use the expected counts for the unlabelled data and the observed counts for the labelled data, e.g.,

$$p^*(w|s) = \frac{E[c(w_i = w, t_i = s_i)] + c(w_i = w, t_i = s_i)}{E[c(t_i = s)]) + c(t_i = s)}$$

Similarly for emission probabilities

Example of semi-supervised Baum-Welch

We may have some corpus with WAV files and IPA Transcription

> iz ðɪs ɹaɪt

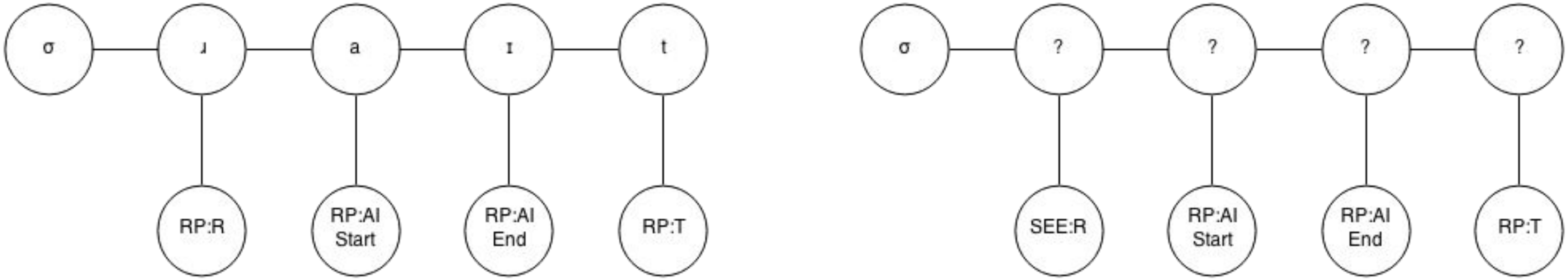
And wave transcription (not shown here)

We may then come across a speaker who pronounces it like this (R-labialization)

> iz ðɪs ʊaɪt

Example of semi-supervised Baum-Welch

By Baum-Welch training we can recognise `SEE:R` as coming from the class 'ɹ'



Overview

Revision

Hidden Markov Model

Tagging

HMM Language Models

Learning Hidden Markov Models

Summary

Summary

Syntax is important for understanding sentences
Sequence model is a good model for:

- Part-of-speech
- Named Entity Recognition
- Speech Recognition

Sequence model needs to be approximated, HMM is most widely used approximation

$$p(t_1, \ldots, t_n, w_1, \ldots, w_n) \approx \prod p(t_i|t_{i-1})p(w_i|t_i)$$

There are an exponential number of *hidden states,* so naive application of this is exponential

Summary

Dynamic programming approaches enable polynomial time calculation of

- Most likely sequence (and its probability)
- Total probability of a sequence (language model)

HMMs are frequently learnt in a supervised setting (with known tags)

In some applications (speech recognition) unsupervised learning is applied

- Baum-Welch algorithm is a special case of expectation-maximization

Lab of this Week

Exercises on POS tagging using NLTK

QA