ML for Human Vision and Language

MSc Artificial Intelligence

Lecturer: Tejaswini Deoskar
t.deoskar@uu.nl

UiL-OTS, Utrecht University

Block 1, 2019

Parts-of-speech

word classes, lexical classes, grammatical classes, lexical tags, ...

POS tagging (inference): *to determine the POS tag for a particular instance (token) of a word in context.*

EX	VBD	JJ	NN	IN	DT	NN
There	was	still	lemonade	in	the	bottle

- Various granularities: fine-grained tags/coarse-grained tags

POS tagging: Why do we care?

- first step towards syntactic analysis; simpler and faster than full syntactic parsing
- POS tagging task also helps introduce useful techniques: Hidden Markov models(HMMs) or Recurrent Neural networks (RNNs), which are used for many other sequence labelling or sequence modelling tasks.
- A variety of (AI-related) applications are sequence labelling/sequence modelling tasks : robot actions, music cognition/generation, etc.
- POS tags can be useful features in e.g. text classification, authorship identification, text-to-speech etc.

Other tagging or sequence labelling tasks

- Named Entity Recognition (NER) e.g., label words as belonging to persons (PER), organizations (ORG), locations(LOC), or none of the above:
  * Donald/PER Trump/PER tweeted/NON from/NON the/NON White/LOC House/LOC today/NON ./NON

Other tagging or sequence labelling tasks

- Named Entity Recognition (NER) e.g., label words as belonging to persons (PER), organizations (ORG), locations(LOC), or none of the above:
  - Donald/PER Trump/PER tweeted/NON from/NON the/NON White/LOC House/LOC today/NON ./NON
- Information field segmentation: Given semi-structured text (classified advertisement, bibiography entry), identify which words belong to which fields (price/ size/ location, author /title/ year)
  - 3BR/SIZE flat/TYPE in/NON Utrecht/LOC ,/NON near/LOC centraal/LOC station/LOC ./NON Bright/FEAT ,/NON well/FEAT maintained/FEAT ...

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT						

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT	JJ					

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT	JJ	NN	VBZ	...		

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT	JJ	NN	VBZ	...		
Intended:	DT	RB	VBG	NNS	IN	DT	NN

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT	JJ	NN	VBZ	...		
Intended:	DT	RB	VBG	NNS	IN	DT	NN

- One incorrect tagging choice might have unintended effects:
- No lookahead: choosing the "most probable" tag at one stage might lead to highly improbable choice later.

Problems with bi-gram (N-gram) tagging

	The	still	smoking	remains	of	the	campfire
Bigram:	DT	JJ	NN	VBZ	...		
Intended:	DT	RB	VBG	NNS	IN	DT	NN

- One incorrect tagging choice might have unintended effects:
- No lookahead: choosing the "most probable" tag at one stage might lead to highly improbable choice later.
- We want to find the **overall most likely** tagging sequence, given the bigram (n-gram) frequencies. This is what the Hidden Markov Model (HMM) approach achieves.

Finding the most likely sequence

- Formalising the model
- Expressing in terms of Hidden Markov Model (HMM)
- Viterbi algorithm to find the best (tag) sequence
- Other sequence models

Stochastic tagging: Formalizing the problem

A sentence is a sequence of n words $(w_1\, w_2 \ldots w_n)$: w_1^n

POS tags for the sentence is a sequence of n tags $(t_1\, t_2 \ldots t_n)$: t_1^n

Stochastic tagging: Formalizing the problem

A sentence is a sequence of n words $(w_1\, w_2 \ldots w_n)$: w_1^n

POS tags for the sentence is a sequence of n tags $(t_1\, t_2 \ldots t_n)$: t_1^n

We want to find the most probable tag sequence t_1^n, given the word sequence w_1^n

$$\hat{t_1^n} = \underset{t_1^n}{\arg\max}\; P(t_1^n \mid w_1^n)$$

Using Bayes' Rule

$$\hat{t_1^n} = \arg\max_{t_1^n} P(t_1^n \mid w_1^n) = \arg\max_{t_1^n} \frac{P(w_1^n \mid t_1^n)\ P(t_1^n)}{P(w_1^n)}$$

- Ignore the denominator, since best tag sequence does not depend on prob. of word sequence

$$\hat{t_1^n} = \arg\max_{t_1^n} P(w_1^n \mid t_1^n)\ P(t_1^n)$$

- $P(t_1^n)$: prior probability of the tag sequence
- $P(w_1^n \mid t_1^n)$: likelihood of data

Simplifying assumptions

- Estimating these probabilities from corpus data will be hard (sparsity)

  $P($"Time flies like an arrow"$|$NN VB IN DT NN$)$

  $P($NN VB IN DT NN$)$

- Make simplyfing assumptions for each term

Simplifying assumptions (Independence assumptions)

For prior term :

$$P(t_1^n) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

- The probability of a tag is independent of all but the previous tag (bi-grams of tags)

Simplifying assumptions (Independence assumptions)

For prior term :

$$P(t_1^n) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

- The probability of a tag is independent of all but the previous tag (bi-grams of tags)

For likelihood term:

$$P(w_1^n|t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)$$

- The probability of a word depends only on its POS tag
  - ∗ Independent of other words in the sentence
  - ∗ Independent of other POS tags

Bi-gram tagger

$$\hat{t_1^n} = \arg\max_{t_1^n} \; P(w_1^n \mid t_1^n) \; P(t_1^n)$$

$$\hat{t_1^n} \approx \arg\max_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i) \; P(t_i|t_{i-1})$$

Estimating the parameters of the model from a corpus

Specific probabilities of $P(t_i|t_{i-1})$ and $P(w_i|t_i)$ are estimated from a tagged (supervised) corpus:

Estimating the parameters of the model from a corpus

Specific probabilities of $P(t_i|t_{i-1})$ and $P(w_i|t_i)$ are estimated from a tagged (supervised) corpus:

Tag Transition probabilities :

$$P(t_i|t_{i-1}) = \frac{count(t_{i-1}, t_i)}{count(t_{i-1})}$$

- A noun but not a verb is very likely to follow a determiner or adjective:

  That/DT flight/NN
  The/DT yellow/JJ hat/NN

- Example estimated from Brown corpus:

$$P(\text{NN} \mid \text{DT}) = \frac{count(\text{DT, NN})}{count(\text{DT})} = \frac{56,509}{116,454} = 0.49$$

Estimating the parameters of the model from a corpus

Word likelihood probabilities :

$$P(w_i|t_i) = \frac{count(t_i, w_i)}{count(t_i)}$$

- If the tag is VBZ (3 Person Singular Present Tense), then 'is' is very likely to be the verb.

[1] Example from J&M 2 edition 5.5

Estimating the parameters of the model from a corpus

Word likelihood probabilities :

$$P(w_i|t_i) = \frac{count(t_i, w_i)}{count(t_i)}$$

- If the tag is VBZ (3 Person Singular Present Tense), then 'is' is very likely to be the verb.
- Example estimated from Brown corpus:

$$P(\text{is} \mid \text{VBZ}) = \frac{count(\text{VBZ,is})}{count(\text{VBZ})} = \frac{10,073}{21,627} = 0.47$$

[1]

[1] Example from J&M 2 edition 5.5

What can we do with this model - I?

- Compute the probability of a tagged sentence, if we know the parameters.

$$P(w_1^n, t_1^n) = \prod_{i=1}^{n} P(t_i \,|\, t_{i-1})\; P(w_i \,|\, t_i)$$

- Comparison to a Language Model:
  - ∗ a LM gives us the probability of a sentence, whereas this model here gives us the probability of a *tagged* sentence.
- (Caveat: A start $< s >$ and stop $< /s >$ symbol is added to beginning and end of sentences.)

What can we do with this model - II?

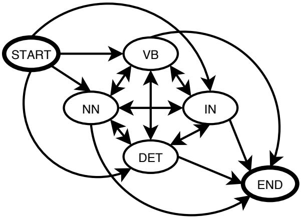
- Tagging: use the model to find the best tag sequence t_1^n for an untagged sentence w_1^n

$$\hat{t_1^n} = \arg\max_{t_1^n} P(t_1^n \mid w_1^n)$$

  * thus, Hidden Markov Model
  * **Markov** because of Markov assumption (tag only depends on immediately previous tag)
  * **Hidden** because we only observe the words/emissions; the tags/states are hidden (or latent) variables.

Probabilistic Finite State Machine

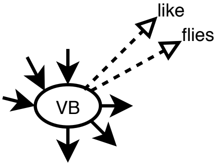
- Sentences are generated by walking through states in a graph. Each state represents a tag.
- Each state emits an output (a word)



transition probability

Prob. of moving from state s to s' :

$$P(t_i = s' | t_{i-1} = s)$$



emission probability

Prob of emitting w from state s :

$$P(w_i = w | t_i = s)$$

Hidden Markov Model (HMM)

HMM is a very general model for sequences.

The elements of an HMM:

- a set of states (here: the tags)
- an output alphabet (here: words)
- initial state (here: beginning of sentence)
- state transition probabilities (here: $p(t_i \mid t_{i-1})$)
- symbol emission probabilities (here: $p(w_i \mid t_i)$)

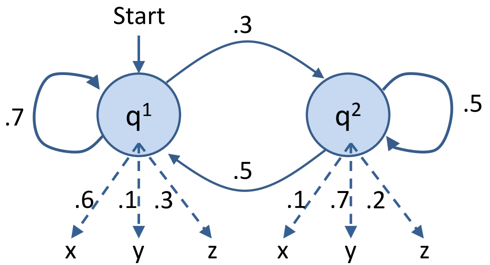
Notation

- Sequence of observations over time $o_1, o_2, \ldots o_T$
  - here, words in sentence
- Vocabulary size V of possible observations
- Set of possible states $q^1, q^2, \ldots q^N$
  - here, tags (*see note on notation on next slide*)
- A, an N X N matrix of transition probabilities
  - a_{ij} : the prob. of transitioning from state i to j
- B, an N X V matrix of output probabilities
  - $b_i(o_t)$: the prob. of emitting o_t from state i

Note on notation

- J&M use $q_1, q_2, \dots q_N$ for set of states, but also use $q_1, q_2, \dots q_T$ for state sequence over time
- We'll use q^i for state names, and q_t for state at time t
- So, we could have $q_t = q^i$, meaning : the state we're in at time t is q^i

HMM toy example with new notation



- States $\{q^1, q^2\}$ (or $\{<s>, q^1, q^2\}$)
- Output alphabet $\{x, y, z\}$

Adapted from Manning & Schuetze, Fig 9.2

Transition and Output Probabilities

- Transition matrix $A : a_{ij} = P(q^j | q^i)$

	q^1	q^2
$< s >$	1	0
q^1	0.7	0.3
q^2	0.5	0.5

Transition and Output Probabilities

- Transition matrix $A : a_{ij} = P(q^j|q^i)$

	q^1	q^2
$< s >$	1	0
q^1	0.7	0.3
q^2	0.5	0.5

- Output matrix $B : b_i(o) = P(o|q^i)$ for output o

	x	y	z
q^1	0.6	0.1	0.3
q^2	0.1	0.7	0.2

- $\lambda = (A, B)$ are the parameters of our HMM

Joint Probability of (states, outputs)

- Using the new notation, given state sequence $Q = (q_1 \ldots q_T)$ and output sequence $O = (o_1 \ldots o_T)$, we have:

$$P(O, Q \mid \lambda) = \prod_{t=1}^{T} P(o_t \mid q_t)\ P(q_t \mid q_{t-1})$$

$$P(O, Q \mid \lambda) = \prod_{t=1}^{T} b_{q_t}(o_t)\ a_{q_{t-1} q_t}$$

Search for the best tag sequence : Viterbi

- For any specific tag sequence t_1^n, it is easy to calculate

$$P(w_1^n, t_1^n) = \prod_{i=1}^{n} P(t_i|t_{i-1}) P(w_i|t_i)$$

- So why can't we enumerate all possible t_1^n sequences, compute their probability, and choose the best one?

The	still	smoking	remains
DT	RB	NN	NNS
	JJ	VBG	VBZ
	$\vdots$	$\vdots$	$\vdots$

Enumeration won't work

- Suppose we have c possible tags for each of the n words in a sentence?
- How many possible tag sequences?

Enumeration won't work

- Suppose we have c possible tags for each of the n words in a sentence?
- How many possible tag sequences?
- There are c^n possible tag sequences: the number grows exponentially in the length n
- For all but sentences of small length, too many sequences to enumerate: $c = 10$

  10 word sent. $\Rightarrow 10,000,000,000$ tag sequences
  15 word sent. $\Rightarrow 1,000,000,000,000,000$ tag sequences

Overview: HMM for sequence tagging

- Given a sentence $O = o_1 \dots o_T$, with tags $Q = q_1 \dots q_T$, we want to compute $P(O, Q)$ as

$$P(O, Q) \quad = \prod_{t=1}^{T} \; P(o_t | q_t) \; P(q_t \mid q_{t-1})$$

$$P(w_1^n, t_1^n) \quad = \prod_{i=1}^{n} \; P(w_i | t_i) \; P(t_i \mid t_{i-1})$$

- But we want to find $\arg\max_Q P(Q|O)$ without enumerating all possible Q
  - ∗ User Viterbi algorithm to store partial computations

Finding the best sequence : Viterbi

- The **Viterbi** algorithm finds the best sequence without explicitly enumerating all sequences.
- Example of a dynamic programming or memoisation algorithm.
- Efficient, exact inference.
- stores partial results in a chart to avoid recomputing them.

Viterbi High level picture

Viterbi intuition: the best path of length t ending in state q **must** include the best path of length $t-1$ to the previous state. (t now a time step, not a tag)

Viterbi High level picture

Viterbi intuition: the best path of length t ending in state q **must** include the best path of length $t-1$ to the previous state. (t now a time step, not a tag)

- thus, we do not have to calculate the full best path at every time step!

High level picture:

- Find the best path of length $t-1$ to each state.
- Consider extending each of those by 1 step, to state q
- Take the best of those options as the best path to state q

Viterbi Intuition

$< s >$	one	dog	bit	$< /s >$
$< s >$	CD	NN	NN	$< /s >$
	NN	VB	VBD	
	PRP			

Viterbi Intuition

$<s>$	one	dog	bit	$</s>$
$<s>$	CD	NN	NN	$</s>$
	NN	VB	VBD	
	PRP			

- Suppose we have already computed
  1. The best tag sequence for $<s> \ldots$ bit that ends in NN.
  2. The best tag sequence for $<s> \ldots$ bit that ends in VBD.

Viterbi Intuition

$< s >$	one	dog	bit	$< /s >$
$< s >$	CD	NN	NN	$< /s >$
	NN	VB	VBD	
	PRP			

- Suppose we have already computed
  1. The best tag sequence for $< s > \ldots$ bit that ends in NN.
  2. The best tag sequence for $< s > \ldots$ bit that ends in VBD.
- Then, the best **full** sequence would be either
  - $*$ sequence (1) extended to include $< /s >$, or
  - $*$ sequence (2) extended to include $< /s >$.

Viterbi Intuition

$< s >$	one	dog	bit	$< /s >$
$< s >$	CD	NN	NN	$< /s >$
	NN	VB	VBD	
	PRP			

- Suppose we have already computed
  1. The best tag sequence for $< s > \ldots$ bit that ends in NN.
  2. The best tag sequence for $< s > \ldots$ bit that ends in VBD.
- Then, the best **full** sequence would be either
  - sequence (1) extended to include $< /s >$, or
  - sequence (2) extended to include $< /s >$.
- Similarly, to get the best tag sequence for $< s > \ldots$ bit that ends in NN, we could extend one of
  1. The best tag sequence for $< s > \ldots$ dog that ends in NN.
  2. The best tag sequence for $< s > \ldots$ dog that ends in VB.

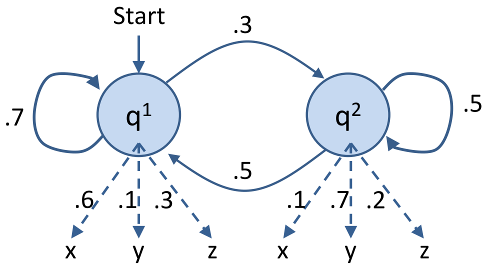
Viterbi Algorithm

- Use a chart N X T to store partial results as we go
  - $v(j, t)$ is the probability of the best state sequence for $o_1 \dots o_t$ that ends in state j
- Fill in columns from left to right, with

$$v(j, t) = \max_{i=1}^{N} \; v(i, t-1) \,.\, a_{ij} \,.\, b_j(o_t)$$

- Store a backtrace to show, for each cell, which state at $t-1$ we came from.

Toy example



- States $\{q^1, q^2\}$ (or $\{<s>, q^1, q^2\}$)
- Output alphabet $\{x, y, z\}$

Adapted from Manning & Schuetze, Fig 9.2

Toy example

- Suppose $O = x\, z\, y$.
- Our initially empty table:

	o_1=x	o_2=z	o_3=y
q^1			
q^2			

Filling the first column

	o_1=x	o_2=z	o_3=y
q^1	.6		
q^2	0		

$$v(1,1) = a_{<s>1} \cdot b_1(x) = (1)(.6)$$

$$v(2,1) = a_{<s>2} \cdot b_2(x) = (0)(.1)$$

Starting the second column

	o_1=x	o_2=z	o_3=y
q^1	.6		
q^2	0		

$$v(1,2) = \max_{i=1}^{N} v(i,1) \cdot a_{i1} \cdot b_1(z)$$

$$= \max \begin{cases} v(1,1) \cdot a_{11} \cdot b_1(z) = (.6)(.7)(.3) \\ v(2,1) \cdot a_{21} \cdot b_1(z) = (0)(.5)(.3) \end{cases}$$

Starting the second column

	o_1=x	o_2=z	o_3=y
q^1	.6	← .126	
q^2	0		

$$v(1,2) = \max_{i=1}^{N} v(i,1) \cdot a_{i1} \cdot b_1(z)$$

$$= \max \begin{cases} v(1,1) \cdot a_{11} \cdot b_1(z) = (.6)(.7)(.3) \\ v(2,1) \cdot a_{21} \cdot b_1(z) = (0)(.5)(.3) \end{cases}$$

Finishing the second column

	o_1=x	o_2=z	o_3=y
q^1	.6	.126	
q^2	0		

$$v(2,2) = \max_{i=1}^{N} v(i,1) \cdot a_{i2} \cdot b_2(z)$$

$$= \max \begin{cases} v(1,1) \cdot a_{12} \cdot b_2(z) = (.6)(.3)(.2) \\ v(2,1) \cdot a_{22} \cdot b_2(z) = (0)(.5)(.2) \end{cases}$$

Finishing the second column

	o_1=x	o_2=z	o_3=y
q^1	.6	.126	
q^2	0	.036	

$$v(2,2) = \max_{i=1}^{N} v(i,1) \cdot a_{i2} \cdot b_2(z)$$

$$= \max \begin{cases} v(1,1) \cdot a_{12} \cdot b_2(z) = (.6)(.3)(.2) \\ v(2,1) \cdot a_{22} \cdot b_2(z) = (0)(.5)(.2) \end{cases}$$

Third column

	o_1=x	o_2=z	o_3=y
q^1	.6	.126	.00882
q^2	0	.036	.02646

$$v(1,3) = max_{i=1}^{N} \; v(i,2) \; a_{i1} \; b_1(y)$$

$$v(2,3) = max_{i=1}^{N} \; v(i,2) \; a_{i2} \; b_2(y)$$

Best Path

	o_1=x	o_2=z	o_3=y
q^1	.6	.126	.00882
q^2	0	.036	.02646

- Choose best final state: $\max_{i=1}^{N} v(i,T)$
- Follow backtraces to find best full sequence: $q^1q^1q^2$

Viterbi pseudocode

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$$
$$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$; termination step
$$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

What else?

- find the best tags for a sentence (decoding), and get $P(O, Q|\lambda)$ (Viterbi algorithm)

What else?

- find the best tags for a sentence (decoding), and get $P(O, Q|\lambda)$ (Viterbi algorithm)
- Compute the likelihood $P(O|\lambda)$, i.e., the probability of a sentence regardless of tags (a language model) (Forward algorithm)

$$\alpha(j,t) = \sum_{i=1}^{N} \alpha(i, t-1) \, . \, a_{ij} \, . \, b_j(o_t)$$

What else?

- find the best tags for a sentence (decoding), and get $P(O, Q|\lambda)$ (Viterbi algorithm)
- Compute the likelihood $P(O|\lambda)$, i.e., the probability of a sentence regardless of tags (a language model) (Forward algorithm)

$$\alpha(j,t) = \sum_{i=1}^{N} \alpha(i, t-1) \; . \; a_{ij} \; . \; b_j(o_t)$$

- Learn the best set of parameters $\lambda = (A, B)$, given only an *unannotated* corpus of sentences (unsupervised estimation) (Forward-backward algorithm) (not covered)

Maximum Entropy Markov Model (MEMM)

- HMMs : compute likelihood (observation word conditioned on tags)

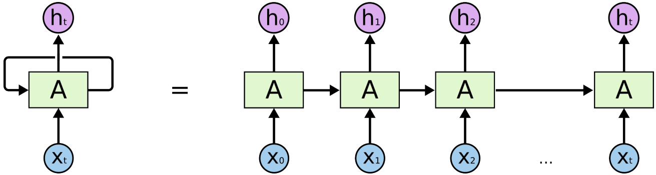
$$\hat{t_1^n} = \arg\max_{t_1^n} \prod_i P(w_i|t_i)P(t_i|t_{i-1})$$

- MEMM : Directly compute the posterior $P(t_1^n|w_1^n)$ (tags conditioned on observation words)

$$\hat{t_1^n} = \arg\max_{t_1^n} \prod_i P(t_i|w_i, t_{i-1})$$

Recurrent Neural Networks (RNNs)

- RNNs are used to model sequences : they are extremely good at language modelling and sequence labelling
- Traditional neural networks do not have *persistence*: presented with a new input, they forget the old one; RNNs solve this problem by having loops



- Input to the hidden layer is augmented with the activation value of the hidden layer from a previous point in time

Computation at the hidden layer

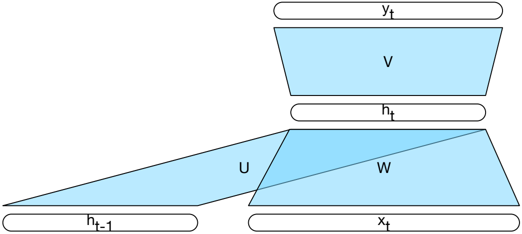
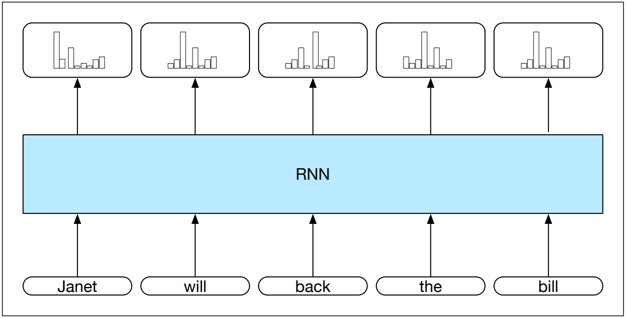


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

- A new set of weights U connect the hidden layer from the previous timestep to the current hidden layer
- U determines how the network makes use of past context in calculating the current output
- Just like other weights, these are also trained with backpropagation

$$h_t = g(Uh_{t-1} + Wx_t)$$

POS tagging as sequence labelling with RNN



- inputs are pre-trained word embeddings
- RNN block: input, hidden, output layer + U, W, V weight matrices
- output at each time step is distribution over POS tagset generated by softmax layer

Sequence labelling with RNNs and variants

- sequence of probabiliity distributions (from softmax) can be combined with a tag-level language model, and selecting the most likely tag sequence
- Bi-directional RNNs, LSTMs, bi-LSTMs
- Sequence-to-sequence models, Transformers.

-
-

-
-

Sequence modelling

- Language Modelling: Sentence probability $P(w_1 \dots w_n)$; Guess the next word: $P(w_i \mid w_1 \dots w_{i-1})$
- POS tagging
- In general, these are insufficient models of language, because language has long distance dependencies

  The train that my friends are arriving on from Paris tomorrow is cancelled.

Long distance dependencies in language

- Syntactic dependencies (Number/Gender agreement):

The **boy** in the house **was** watching television.
The **boys** in the house **were** watching television

She is planning to go to the conference **herself**
He is planning to go to the conference **himself**

Long distance dependencies in language

- Syntactic dependencies (Number/Gender agreement):

  The **boy** in the house **was** watching television.
  The **boys** in the house **were** watching television

  She is planning to go to the conference **herself**
  He is planning to go to the conference **himself**

- Semantic dependencies:

  The **bird** next to the large oak tree near the grocery store on the corner **flies** rapidly.
  The **man** next to the large oak tree near the grocery store on the corner **talks** rapidly.

Supertagging

- Like POS tagging, but tags are more complex, and more numerous
- Instead of 45-200 POS tags, 1000-2000 supertags
- Supertags capture syntactic information
- Supertags can be based on one of many "strongly lexicalised" grammar formalisms
  - Combinatory Categorial Grammar (Steedman, 1996, 2000)
  - Type-logical Grammars (Moortgat & Moot, 2002)
  - Tree-adjoining grammars (Bangalore & Joshi 1999)

Combinatory Categorial Grammar (CCG)

- Categorial grammar CG is one of the oldest grammar formalisms (Ajdukiewicz, 1935; Bar-Hillel, 1953; Lambek 1958)
- Various flavours of CG now available: type-logical CG, algebraic pre-group grammars (Lambek), CCG
- CCG is now an established linguistic formalism (Steedman, 1996, 2000)
  * syntax; semantics; prosody; information structure; wide-coverage parsing; incremental parsing, generation, etc.

CCG slides from Stephen Clark

Combinatory Categorial Grammar (CCG)

- CCG is a strongly lexicalised grammar

Combinatory Categorial Grammar (CCG)

- CCG is a strongly lexicalised grammar
- An elementary syntactic structure - a lexical category - is assigned to each word in a sentence.

  walked : $S \setminus NP$ *(give me an NP to my left and I return a sentence)*

Combinatory Categorial Grammar (CCG)

- CCG is a strongly lexicalised grammar
- An elementary syntactic structure - a lexical category - is assigned to each word in a sentence.

  walked : $S \setminus NP$ *(give me an NP to my left and I return a sentence)*

- A semantic interpretation is also assigned to each word

  walked : $\lambda x.WALK(x)$

- A small number of rules define how categories can combine - rules are called combinators

CCG Lexical Categories

- Atomic categories : S, N, NP, PP (not many more) (sentence, noun, noun phrase, prepositional phrase)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode 'subcategorisation' information
  - intransitive verb: $S \setminus NP$: *walked*
  - transitive verb: $(S \setminus NP)/NP$: *respected*
  - ditransitive verb : $((S \setminus NP)/NP)/NP$: *gave*
  - np-pp : $((S \setminus NP)/PP)/NP$: *put*

A Simple CCG Derivation

$$\begin{array}{ccc}
\textit{interleukin} - \textit{10} & \textit{inhibits} & \textit{production} \\
\hline
NP & (S\backslash NP)/{\color{blue}NP} & {\color{blue}NP} \\
\cline{2-3}
 & \multicolumn{2}{r}{>} \\
 & \multicolumn{2}{c}{S\backslash NP}
\end{array}$$

A Simple CCG Derivation

$$
\begin{array}{ccc}
\textit{interleukin} - \textit{10} & \textit{inhibits} & \textit{production} \\
\hline
NP & (S\backslash NP)/{\color{blue}NP} & {\color{blue}NP} \\
\cline{2-3}
 & \multicolumn{2}{r}{>} \\
 & \multicolumn{2}{c}{S\backslash NP}
\end{array}
$$

$>$ forward application

A Simple CCG Derivation

$$
\begin{array}{ccc}
\textit{interleukin} - \textit{10} & \textit{inhibits} & \textit{production} \\
\hline
NP & (S\backslash NP)/NP & NP \\
\cline{2-3}
 & \multicolumn{2}{r}{>} \\
 & \multicolumn{2}{c}{S\backslash NP} \\
\hline
\multicolumn{3}{r}{<} \\
\multicolumn{3}{c}{S}
\end{array}
$$

$>$ forward application

$<$ backward application

Function Application Rule Schemata

- Forward ($>$) and backward ($<$) application:

$$\begin{array}{llcll} X/Y & Y & \Rightarrow & X & (>) \\ Y & X\backslash Y & \Rightarrow & X & (<) \end{array}$$

A Simple CCG Derivation

$$\frac{[_{NNP}\ \text{John}\ ]}{\textbf{NP}} \quad \frac{[_{VBD}\ \text{baked}\ ]}{\textbf{S}\backslash\textbf{NP}/\textbf{NP}/\textbf{NP}} \quad \frac{[_{NNP}\ \text{Mary}]}{\textbf{NP}} \quad \frac{[_{DT}\ \text{a}\ ]}{\textbf{NP}/\textbf{N}} \quad \frac{[_{NN}\ \text{cake}]}{\textbf{N}}$$

A Simple CCG Derivation

$$
\begin{array}{ccccc}
[_{NNP}\ \text{John}\ ] & [_{VBD}\ \text{baked}\ ] & [_{NNP}\ \text{Mary}] & [_{DT}\ \text{a}\ ] & [_{NN}\ \text{cake}] \\
\hline
\mathbf{NP} & \mathbf{S\backslash NP/NP/NP} & \mathbf{NP} & \mathbf{NP/N} & \mathbf{N} \\
\cline{2-3} \cline{4-5}
& \multicolumn{2}{r}{>} & \multicolumn{2}{r}{>} \\
& \multicolumn{2}{c}{\mathbf{S\backslash NP/NP}} & \multicolumn{2}{c}{\mathbf{NP}}
\end{array}
$$

A Simple CCG Derivation

$$
\begin{array}{ccccc}
[_{NNP}\ \text{John}\ ] & [_{VBD}\ \text{baked}\ ] & [_{NNP}\ \text{Mary}] & [_{DT}\ \text{a}\ ] & [_{NN}\ \text{cake}] \\
\overline{\mathbf{NP}} & \overline{\mathbf{S\backslash NP/NP/NP}} & \overline{\mathbf{NP}} & \overline{\mathbf{NP/N}} & \overline{\mathbf{N}} \\
 & \multicolumn{2}{c}{\overline{\qquad\mathbf{S\backslash NP/NP}\qquad}{>}} & \multicolumn{2}{c}{\overline{\qquad\mathbf{NP}\qquad}{>}} \\
 & \multicolumn{4}{c}{\overline{\qquad\qquad\mathbf{S\backslash NP}\qquad\qquad}{>}} \\
\multicolumn{5}{c}{\overline{\qquad\qquad\qquad\mathbf{S}\qquad\qquad\qquad}{<}}
\end{array}
$$

'John baked Mary a cake'

baked : $((S \backslash NP) / NP) / NP$

$\lambda x.\lambda y.\lambda z.$ z baked y for x

CCG lexicon

Syntactic combinatory potential for every word

bake :

$(S \backslash NP) / NP / NP$
$(S \backslash NP) / PP / NP$
$(S \backslash NP) / NP$
$(S \backslash NP) / NP$
$(S \backslash NP) / S[to]$
$(S \backslash NP) / PP$
……

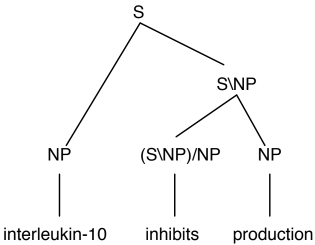
Statistical model

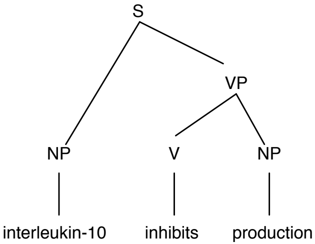
baked Mary a cake
baked a cake for Mary
baked a cake
baked to cheer himself up
baked for a charity

CCG Bank (English): > 1300 categories, 566 for verbs

Classical Categorial Grammar

- Only has application, and is context-free
- CCG has more combinators that make it "mildly" context sensitive.





Extraction out of a Relative Clause

$$\frac{The}{NP/N} \quad \frac{company}{N} \quad \frac{which}{(NP\backslash NP)/(S/NP)} \quad \frac{Microsoft}{NP} \quad \frac{bought}{(S\backslash NP)/NP}$$

Extraction out of a Relative Clause

$$\begin{array}{ccccc} \textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\ \hline NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\ & & & \overline{S/(S\backslash NP)}^{>\mathbf{T}} & \end{array}$$

$>$ Type-raising

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
 & & & \overline{S/(S\backslash NP)}^{\,>\mathbf{T}} & \\
 & & & \multicolumn{2}{c}{\overline{\qquad S/NP \qquad}^{\,>\mathbf{B}}}
\end{array}
$$

$>\mathbf{T}$ Type-raising

$>\mathbf{B}$ forward composition

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
 & & & \xrightarrow{\quad\quad} >\mathbf{T} & \\
 & & & S/(S\backslash NP) & \\
 & & & \multicolumn{2}{c}{\xrightarrow{\quad\quad\quad\quad\quad\quad} >\mathbf{B}} \\
 & & & \multicolumn{2}{c}{S/NP} \\
 & & \multicolumn{3}{c}{\xrightarrow{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad} >} \\
 & & \multicolumn{3}{c}{NP\backslash NP}
\end{array}
$$

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\overline{NP/N} & \overline{N} & \overline{(NP\backslash NP)/(S/NP)} & \overline{NP} & \overline{(S\backslash NP)/NP} \\
\multicolumn{2}{c}{\overline{}{>}} & & \overline{}{>}\mathbf{T} & \\
\multicolumn{2}{c}{\color{blue}{NP}} & & S/(S\backslash NP) & \\
 & & & \multicolumn{2}{c}{\overline{}{>}\mathbf{B}} \\
 & & & \multicolumn{2}{c}{S/NP} \\
 & & \multicolumn{3}{c}{\overline{}{>}} \\
 & & \multicolumn{3}{c}{NP\backslash \color{blue}{NP}} \\
\multicolumn{5}{c}{\overline{}{<}} \\
\multicolumn{5}{c}{NP}
\end{array}
$$

Forward Composition and Type-Raising

- Forward composition ($>_B$):

$$X/Y \quad Y/Z \Rightarrow X/Z \quad (>_B)$$

- Type-raising ($\mathbf{T}$):

$$X \Rightarrow T/(T \setminus X) \quad (>_T)$$

$$X \Rightarrow T \setminus (T/X) \quad (<_T)$$

- Extra combinatory rules increase the weak generative power to mild context-sensitivity

Non-constituents in CCG (Right Node Raising)

$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & {\color{blue}NP} \\
\overline{S/(S\backslash NP)}^{>\mathbf{T}} & & & \overline{S/(S\backslash NP)}^{>\mathbf{T}} & & \\
\multicolumn{2}{c}{\overline{\qquad S/NP \qquad}^{>\mathbf{B}}} & & \multicolumn{2}{c}{\overline{\qquad S/NP \qquad}^{>\mathbf{B}}} & \\
\multicolumn{5}{c}{\overline{\qquad\qquad S/{\color{blue}NP} \qquad\qquad}^{<\Phi>}} & \\
\multicolumn{6}{c}{\overline{\qquad\qquad\qquad S \qquad\qquad\qquad}^{>}}
\end{array}
$$

$> T$ type-raising

$> B$ forward composition

Why CCG/other categorial/tag grammars

- Formally, generative capacity matches human languages: mild-context sensitivity
- Transparent (one-to-one) mapping between syntax and semantics
- CCG emerging as dominant framework for “deep” semantic tasks like question-answering, textual Entailment.
- Supertaggers (English) currently 95% accuracy, almost at par with POS taggers
  - $*$ 1200 lexical category types in CCGBank (compare to 45 POS tags in PennTreeBank)