Natural Language Processing

Chapter 2: Regular Expressions, Text-Normalization, Edit-Distance

Regular Expression: algebraic notation for characterizing a set of strings

Symbols:

Symbol	Description	Example
[ ]	Matches any single character that is within the brackets	[abc] (matches 'a', 'b', 'c')
[-]	Defines a range of characters to match	[a-z] (matches any lowercase letter)
[^]	Matches any character not in the brackets	[^abc] (matches any character except 'a', 'b', 'c')
|	Matches either the expression before or the expression after	a|b|c

Quantifiers:

Quantifier	Description	Example
*	Matches the preceding character 0 or more times	a* (matches '', 'a', 'aa', etc.)
+	Matches the preceding character 1 or more times	a+ (matches 'a', 'aa', etc.)
?	Matches the preceding character 0 or 1 times (i.e., makes it optional)	a? (matches '', 'a')
{n}	Matches the preceding character exactly n times	a{3} (matches 'aaa')
{n,}	Matches the preceding character n or more times	a{2,} (matches 'aa', 'aaa', etc.)

{n,m}	Matches the preceding character between n and m times	a{2,3} (matches 'aa', 'aaa')

Anchors:

Symbol	Description	Example
^	Matches the start of the line	^a (matches 'a' at the start of the line)
$	Matches the end of the line	a$ (matches 'a' at the end of the line)
\b	Matches the word boundary (where a word character is not followed or preceded by another word-character)	\ba (matches 'a' at the start of a word)
\B	Matches the non-word boundary (where a word character is followed or preceded by another word-character)	a\B (matches 'a' not at the end of a word)

Text Normalization

Basic apllications:

- Segmenting/tokenizing words
- Normalizing word formats
- Segmenting sequences

Lemma: same stem + part of spech + rough word sense
example:
cat and **cats** = same lemma

Wordform: the full inflected surface form
example:
cat and **cats** = different wordforms

Stemming: crude chopping of affixes (language dependent)

Stemming for English - **Porter's Stemmer**

Step 1a	Step 2 (for long stems)
sses → ss caresses → caress ies → i ponies → poni ss → ss caress → caress s → ø cats → cat	ational → ate relational→ relate izer → ize digitizer → digitize ator → ate operator → operate …
Step 1b	**Step 3 (for longer stems)**
(*v*)ing → ø walking → walk sing → sing (*v*)ed → ø plastered → plaster …	al → ø revival → reviv able → ø adjustable → adjust ate → ø activate → activ …

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity
over-generalization		under-generalization	

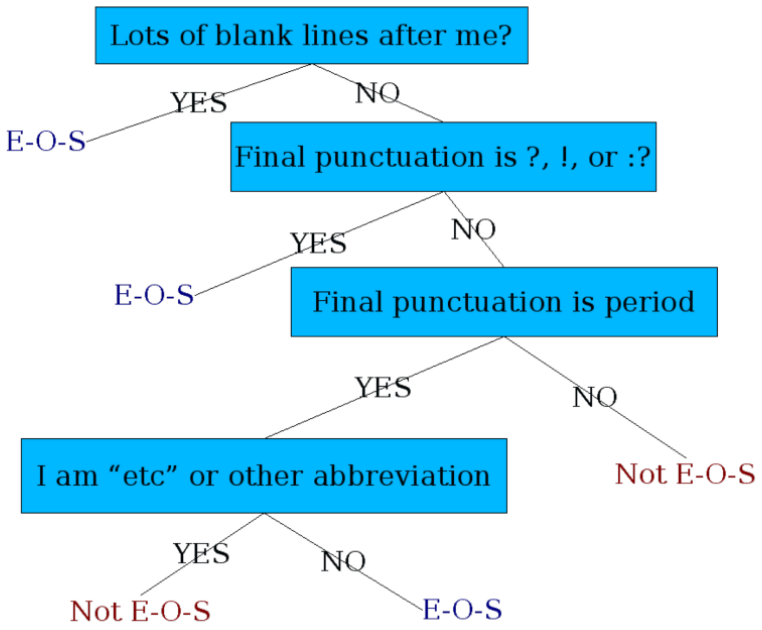
26

Sentence Segmentation

!, ? are relatively unambiguous

Period"." is quite **ambiguous**: sentence boundary, abbreviations, numbers

- Build binary classifier
- Decision Tree



Edit Distance and Alignment

Spell correction, Computational Biology

Minimum Edit Distance between two strings ais the minimum number of editing operations

- insertion
- deletion
- substitution

Levensthein distance

- costs: each operation → 1
- substitution → cost = 2

MinEditDistance:

→ shortest weighted path

D[i,j] = minEditDistance where i is rows and j is columns

→ minEditDistance(X, Y) = D[n, m]

Approach: **Dinamic Programming** - compute for smaller i,j and computer larger D[i,j] on prevous computed samller values

```
# Recurrence relation:
for each row i from 1 to n do
    for each column j from 1 to m do
        D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                      D[i−1,j−1] + sub-cost(source[i], target[j]),
                      D[i,j−1] + ins-cost(target[j]))
# Termination
return D[n,m]
```

up

diag

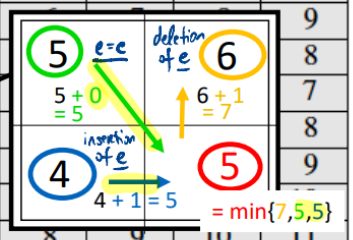
down

example:

j ⇨

i ⇩

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5				9
i	1	2	3	4	5	6				8
n	2	3	4	5	6	7				7
t	3	4	5	6	7	8				8
e	4	3	4	5	6	7				9
n	5	4	5	6	7	8				
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8



$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

Chapter 3: Language Modeling with N-Grams

Motivation: Simple prediction, machine translation, speel correction, speech recognition

---

Language Model: assignment of (joint) probabilities to sequence of words

$$\begin{aligned} P(w_1, w_2, \ldots, w_n) = P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

or (equivalently) modeling conditional probabilities (e.g for predicting next word)

$$P(w_n|w_{1:n-1})$$

MLE based probability estimation: counting instances of sequences in corpora

$$P(\textit{the}|\textit{its water is so transparent that}) = \frac{C(\textit{its water is so transparent that the})}{C(\textit{its water is so transparent that})}$$

Problem: language is creative, combinatorial explosion → **only few instances** of given sentences→ sparsity, poor estimates

---

N-gram Models

Solutions: restrict chain rule to Markov order N

→ N-gram models

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

if n=1: HMM, unigram model

if n=2: bigram model

if n=3: trigram model, etc

- N=2: Bigram model:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

MLE for Bigram model. Count C(w_n-1 w_n) of bigram in corpus:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

MLE for N-gram model

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\ w_n)}{C(w_{n-N+1:n-1})}$$

example bigram:

not on training data

p(chinese | want)

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray. Probabilities capture syntactic, pragmatic and cultural aspects of language

$$\begin{aligned} P(\texttt{<s> i want english food </s>}) \\ &= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\ &\quad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= = .000031 \end{aligned}$$

N-gram Models

- extending to trigram, 4-gram, 5-gram for better quality → not sufficient because language has long-term dependencies
- further aspect: **probabilities should be modelled depending on current usage context**

Evaluation / Comparing N-Gram models

Best: **extrinsic eval** (on actual app with **human eval**): costly ←

Intrinsic: in data only

- split data into Training, Validation and test Sets (80-10-10)
- Simple performance measure: model M1 is better than M2 if P(TeSet|M1) > P(TsSet|M2)

Another measure: Perplexity PP of testSet (low PP = better model)

$$\boxed{\mathrm{PP}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}}$$

how surprised are you by the result

good model → low PP

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

↑ bigger Prob → less PP

$$\underset{\text{(for a bigram model)}}{=} \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

- **Perplexity** = weighted average branching factor of language / corpus.
- **Branching factor** = how many different words can follow any word or N-Gram

---

example: string of N digits, unigram model with uniform P=1/10

$$\mathrm{PP}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = 10$$

if one digit is much more likely →
branching factor is still 10 but PP is smaller

if 0 is much more likely than the other digits, we expect to get much more 0's. Less PP if this happens

10

Corpus Dependence: Unseen combinations/ unknown words

Occurence Out of Vocab (OOV) words:

- approach 1 (preferred): choose fixed vocab, replace with <unk>; count all of them as 1 word
- approach 2: replace all infrequent words with <unk> then use app 1

Black Swan Pradox: unseen words in training have probality 0

Laplace Smoothing

add-1-smoothing (Laplace smoothing): each word gets a prior porbability of 1/|V|

- for unigram models

Shifting probability mass from what I observed to what i didn't but its still possible

MLE $P(w_i) = \frac{c_i}{N}$ → $P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$ → by adding V to the denominator, we are reducing the P(w_i)

corresponding to an adjusted ("discounted") count $\boxed{c_i^* = (c_i + 1)\frac{N}{N+V}}$ → $P_{\text{Laplace}}(w_i) = \frac{c_i^*}{N}$ ($< c_i$)

- for bigram models

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \rightarrow P^*_{\text{Laplace}}(w_n|w_{n-1}) =$$

$$= \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

(adjusted count: $\boxed{c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}}$)

13

- also possible: make prior stronger by adding k instead of 1 (choose k with DevSet)

$$\boxed{P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}}$$

Backoff

- simple smoothing → too much probability mass shifted to zero counts → sharp changes in probabilities / counts.
  → simple smoothing techniques alone often do not work well

Other solution: use less context:

- **Backoff**: use trigram if evidence sufficient, if not (e.g. true counts for trigram are small or zero) use bigram, if bigram evidence too low use unigram
  - reduce the n

probability theory correctness: distribution of probability mass needs to be adjusted by discounting: (e.g. take away mass from trigrams and add it to bigram etc.): **Katz Backoff**:

add it to bigram etc.). Katz Backoff:

$$P_{\mathrm{BO}}(w_n|w_{n-N+1:n-1}) = \begin{cases} P^*(w_n|w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \underbrace{\alpha(w_{n-N+1:n-1})}_{\text{correcting factor because this needs to be a probability distribution}} P_{\mathrm{BO}}(w_n|w_{n-N+2:n-1}), & \text{otherwise.} \end{cases}$$

16

Interpolation

mix all three

- Other solution: **Interpolation**: mix all three:

$$\begin{aligned} \hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 \underbrace{P(w_n|w_{n-2}w_{n-1})}_{\text{trigram}} \\ &+ \lambda_2 \underbrace{P(w_n|w_{n-1})}_{\text{bigram}} \\ &+ \lambda_3 \underbrace{P(w_n)}_{\text{unigram}} \end{aligned} \qquad \sum_i \lambda_i = 1$$

Weighted Interpolation: lambdas depend on context: the higher the true counts, the larger lambda

$$\begin{aligned} \hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2:n-1})P(w_n) \\ &+ \lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) \\ &+ \lambda_3(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1}) \end{aligned}$$

Absolute Discounting

subtract a little from a count to save probability mass for the zeros

$$P_{\mathrm{AbsoluteDiscounting}}(w_i|w_{i-1}) = \underbrace{\frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)}}_{\substack{\text{discounted bigram} \\ d \approx 0.75}} + \underbrace{\lambda(w_{i-1})}_{\substack{\text{Interpolation} \\ \text{weight}}} \underbrace{P(w_i)}_{\text{unigram}}$$

better way: **Kneser-Ney smoothing** (often delivers best performance)

- Example: continuation of sentence

  *I can't see without my reading*_____?

  since *Hong Kong* is a frequent bigram, $P(Kong) > P(glasses)$
  → in a unigram model, select *Kong* instead of *glasses*

intuition: instead of P(w) use the number of different (bigram) contexts that w has appeared in TrainingSet

- now Pcontinuation(Kong) < Pcontinuation(glasses)

→ interpolated Kneser-Ney smoothing:

$$P_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\mathrm{CONTINUATION}}(w_i)$$

$\lambda(w_{i-1})$ is a normalizing constant ←→ distribute the probability mass we've discounted

$$\lambda(w_{i-1}) = \underbrace{\frac{d}{\sum_v C(w_{i-1}v)}}_{\substack{\text{the normalized discount} \\ = d/C(w_{i-1})}} \underbrace{|\{w : C(w_{i-1}w) > 0\}|}_{\substack{\text{The number of word types that can follow } w_{i-1} \\ = \text{\# of word types we discounted} \\ = \text{\# of times we applied normalized discount}}}$$

Stupid Backoff

- on very large corpora (web-scale): instead of full Kneser-Ney, give up idea of correct probabilities and do **simple backoff to lower order N-Gram**

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

does not have to sum to one

with constant $\lambda \approx 0.4$

referring to this as S instead of P (because it is not a probability)

Entropy ←→ Perplexity

- Entripy of length n word sequence

$$H(w_{1:n}) = H(w_1, w_2, \ldots, w_n) = -\sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

- per-word entropy (entropy rate) - divide by 1/n
- entropy rate of the whole language L: (lim)

$$\begin{aligned} H(L) &= \lim_{n\to\infty} \frac{1}{n} H(w_1, w_2, \ldots, w_n) \\ &= -\lim_{n\to\infty} \frac{1}{n} \sum_{W \in L} p(w_1, \ldots, w_n) \log p(w_1, \ldots, w_n) \end{aligned}$$

- language perceived as a stochastic Markov process: if stationary and ergodic → process converges to a limit distribution

ibution →

$$\boxed{H(L) = \lim_{n\to\infty} -\frac{1}{n} \log p(\underline{w_1 w_2 \ldots w_n})}$$

one long enough sequence

→ instead of **averaging over all possible sequences, it suffices to take long enough sequence** (large enough corpus)

→ if $H(W) \approx -\frac{1}{N}\log P(w_1 w_2 \ldots w_N)$

we have

$$\boxed{\text{Perplexity}(W) = 2^{H(W)}}$$

unit of information: bit

2 binary combinations

→ perplexity is related to the average information which is entropy

$$\begin{aligned}\text{Perplexity}(W) &= 2^{H(W)} \\ &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}\end{aligned}$$

Cross Entropy

if we do not know true *p* but instead approximate it with some simpler distribution m, we can use cross entropy as upper limit for entropy:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log m(w_1,\ldots,w_n)$$

because for all m, p $\quad H(p) \leq H(p,m)$

m instead of p, again:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n)$$

Chapter 4: Naive Bayes and Sentiment Classification

Repetition Naive Bayes:

likelihood for model parameters

$$\overbrace{p(\mathcal{D}|\Theta)}^{\text{likelihood for model parameters}} = \prod_{n=1}^{N} p(x^{(n)}, y^{(n)}|\theta, \pi)$$

generative classifier

$$= \prod_{n=1}^{N} \overbrace{p(x^{(n)}|y^{(n)}, \theta)}^{\text{class conditional density}}\overbrace{p(y^{(n)}|\pi)}^{\text{class prior}}$$

= class-likelihood

"naïve" assumption

$$= \prod_{n=1}^{N} \prod_{v=1}^{V} p(x_v^{(n)}|y^{(n)}, \theta) p(y^{(n)}|\pi)$$

categorical class priors

$$= \prod_{n=1}^{N} \prod_{c=1}^{C} \prod_{v=1}^{V} p(x_v^{(n)}|\theta_{vc})^{\mathbb{1}(y^{(n)}=c)} \prod_{c'=1}^{C} \pi_{c'}^{\mathbb{1}(y^{(n)}=c')}$$

- **bag of words model**: position in document / word sequence does not matter in determining feature(s) for a word
- **naïve assumption**: conditional independence of features

$$\log p(\mathcal{D}|\Theta) = \sum_{c=1}^{C} \sum_{v=1}^{V} \sum_{\{n|y^{(n)}=c\}} \log p(x_v^{(n)}|\theta_{vc}) + \sum_{c=1}^{C} N_c \log \pi_c$$

- Simple MLE solution for π_c:

$0 = \partial/\partial\pi_c \log p(\mathcal{D}|\Theta)$ + Lagrange multiplier $\Longrightarrow$

$\pi_c^{MLE} = \frac{N_c}{N}$ as usual for a categorical class prior

Jurafsky notation:

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

Jurafsky notation (v = „i"):

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

Add One Smoothing (Laplace smoothing)

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

- Using the trained classifier (trained == model parameters have been determined) on a new unseen document x is simple:

$$argmax_c \; p(y = c|x, \Theta_{MAP/MLE})$$

$c=1|x \Rightarrow \frac{(x|c=1) \times (c=1)}{(x)}$

$$= argmax_c \; p(x|y = c, \Theta_{MAP/MLE}) * p(y|\Theta_{MAP/MLE})$$

↳ no need for the denominator since its gonna be the same for every class

Very Simple Sentiment Classifiers

- whole documents of **individual sentences**:
  - classify in two (**positive, negative**) or three (**positive, neutral, negative**) classes of sentiment
- **unk words**: ignore
- **stop words**: maybe remove them
- term-frequencies for words also possible: binary features (1 present, 0 not)
  - **Binary NB often works better for sentiment analysis**
- Dealing with Negations: prepend the prefix NOT to every word after a token of logical negation: didn't like → didn't NOT_like
- further possibility: work with only two features per doc:
  - **words with (a priori) positive sentiment**
  - **words with (a priori) negative sentiment**
  - (a priori) word sentiment: from sentiment lexica
    - + - admirable, beautiful, etc
    - - awful, bad, ...

- having learned the $\theta_{vc} = P(w_v|c)$, computing the probability of a sentence S in our model is easy:

$$P(S = (w_{v_1}, w_{v_2}, \ldots, w_{v_{|S|}})|\theta, c) = \prod_{i=1}^{|S|} P(w_{v_i}, |\theta, c) = \prod_{i=1}^{|S|} \theta_{v_i c} = \prod_{v=1}^{V} \theta_{vc}^{N_{vc}}$$

↳ unigram model

- example: for a two class sentiment classifier (using term-frequencies or binary word features) the class likelihoods for a sentence are

higher prob.

$$P(\text{"I love this fun film"}|+) = 0.1 \times 0.1 \times 0.01 \times 0.05 \times 0.1 = 0.0000005$$
$$P(\text{"I love this fun film"}|-) = 0.2 \times 0.001 \times 0.01 \times 0.005 \times 0.1 = .0000000010$$

assuming

w	P(w|+)	P(w|-)
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...	...	...

"love" is much more positive than negative

13

Classifiers: Error / Performance Measures: Two Classes

problem: class disbalance

Accuracy:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

All

		Actual	
		y=1	y=0
Predicted	y=1	TP	FP (type I error)
	y=0	FN (type II error)	TN

how much of the predicted positives are in fact positive

Precision (positive predictive value):

$$PREC = \frac{TP}{TP + FP}$$

predicted Positive

how many positives is the system able to recall

Recall (sensitivity, true positive rate):

$$REC = \frac{TP}{TP + FN}$$

actual positive

Specifity (true negative rate):

$$TNR = \frac{TN}{FP + TN}$$

same as recall but for negatives

False Negative Rate (miss rate):

$$FNR = \frac{FN}{TP + FN}$$

1 - Recall

False Positive Rate (fall out):

$$FPR = \frac{FP}{FP + TN}$$

1 - Specificity

F1 Score (harmonic mean of Recall and Precision):

$$F1 = \frac{2 * PREC * REC}{PREC + REC}$$

Classifiers: Error / Performance Measures: >2 Classes

- usually: use average of one vs rest (macro averaging): example:

Accuracy:

$$ACC = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c + TN_c}{TP_c + FP_c + FN_c + TN_c} = \frac{1}{C}\sum_{c=1}^{C} ACC_c$$

- possible: weighted approach (e.g. using class-priors / inverse class priors to emphasize importance of frequent / infrequent classes): example:

Accuracy:

$$ACC = \sum_{c=1}^{C} \pi_c \, ACC_c$$

- micro-averaging μ vs macro-averaging M example:

Precision$_\mu$

$$PREC = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C} TP_c + FP_c}$$

Precision$_M$:

$$PREC = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FP_c} = \frac{1}{C}\sum_{c=1}^{C} PREC_c$$

- example

example

system output \ gold labels	urgent	normal	spam	
urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
	$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$\text{precision} = \frac{8}{8+11} = .42$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$\text{precision} = \frac{60}{60+55} = .52$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$\text{precision} = \frac{200}{200+33} = .86$

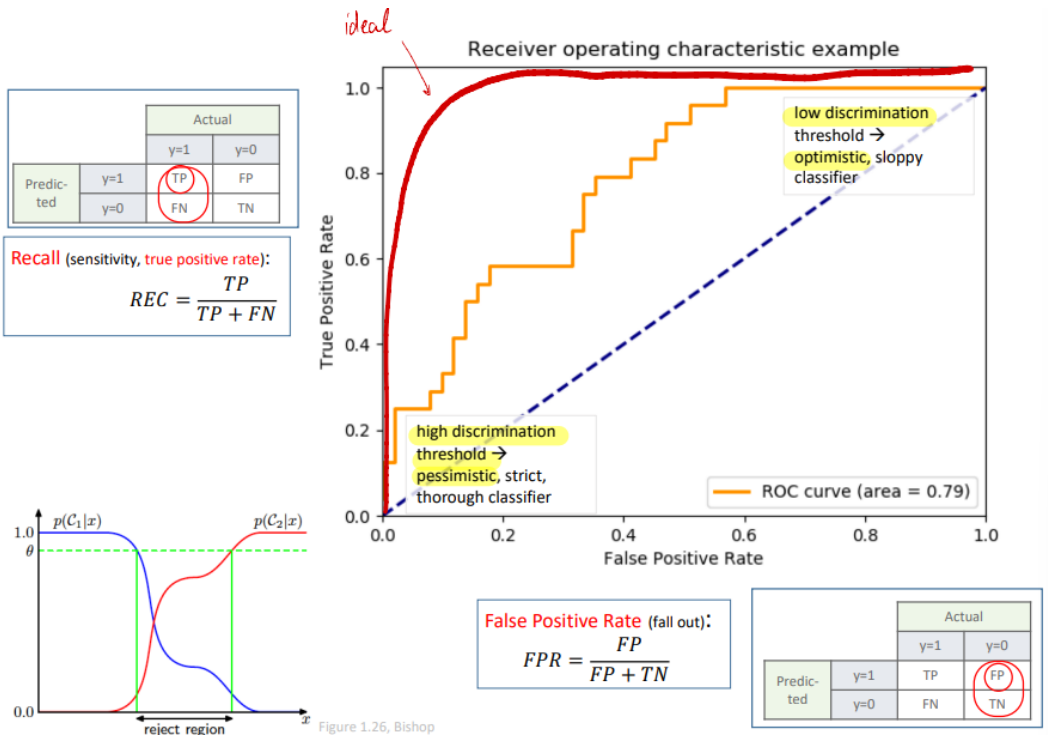
Pooled

	true yes	true no
system yes	268	99
system no	99	635

$\text{microaverage precision} = \frac{268}{268+99} = .73$

$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = .60$

Classifiers: Error / Performance Measures: ROC / AUC



Statistical Significance Testing

Compare two classifiers: F1 score on test set x

- Null Hypothesis H0

  - Null Hypothesis H_0:
    "result 'A better than B' was achieved by chance" =
    "typically, A is not better" = $\delta(x) \leq 0$
    → if X is random variable over all conceivable test-sets, under H_0 (== small $\delta(x)$) we expect $P(\delta(X) > \delta(x)|H_0)$ to be large (sic!) and for a large $\delta(x)$ (== $\neg H_0 = H_1$) we expect $P(\delta(X) > \delta(x)|H_0)$ to be small
    → reject H_0 ←→ p-Value $P(\delta(X) > \delta(x)|H_0) < 0.05$.

- Since the p-value (0.15) is greater than the significance level (0.05), we fail to reject the null hypothesis.
- Performance measures are rarely normally distributed à paired T-test not applicable; furthermore: test data is scarce à use bootstrap testing (can be applied to any performance measure)

Bootstrap Testing:

Bootstrap testing is a resampling technique used for estimating the distribution of a statistic by resampling with replacement from the observed data.

Feature Selection

rank features according to their disciminative power:

- Entropy-based (Iinformation Gain)

$$
\begin{aligned}
G(w) = \quad & -\sum_{i=1}^{C} P(c_i) \log P(c_i) \\
& + P(w) \sum_{i=1}^{C} P(c_i|w) \log P(c_i|w) \\
& + P(\bar{w}) \sum_{i=1}^{C} P(c_i|\bar{w}) \log P(c_i|\bar{w})
\end{aligned}
$$

- GINI Index

$$
G(w) = 1 - \sum_{i=1}^{C} P(c_i|w)^2
$$

- or select skew direction in feature space: e.g. via PCA, pPCA, Factor Analysis, SVD

Chapter 5 - Logistic Regression

- Generative classifier: $\hat{y} = \underset{y}{\operatorname{argmax}}\, P(y|x) = \underset{y}{\operatorname{argmax}}\, P(x|y)P(y)$

  vs. discriminative classifier: $\hat{y} = \underset{y}{\operatorname{argmax}}\, P(y|x)$

- Logistic Regression: $p(y = 1 \mid x) = \sigma(x^T w + w_0)$

  where x are pattern-vectors or feature-vectors of pattern-vectors and σ is the Sigmoid fct.

- Multi-class Logistic Regression (using softmax fct.):

$$p(\mathbf{y}_k = 1|\mathbf{x}) = \frac{\exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}$$

Features: Example Period Disambiguation

goal: using features of words before ".", decide **End of Sentence (EOS)** or **not EOS**

Training Logistic Regression

- Loss function: $L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$

- Cross Entropy Loss:

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y} \qquad \hat{y} = \sigma(w \cdot x + b)$$

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$
$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

- Cross Entropy Loss for a dataset: $\mathcal{D} = ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)}))$

$$\log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)})$$
$$= -\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

- learning $\theta = w, b$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- stochastic gradient descent

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta), y)$$

- Avoid overfitting → switch from MLE for *w* to MAP: assume priors for *w*: Regularization

- L2-Regularization (Gaussian prior)

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} w_i^2$$

- L1-(Lasso)-Regularization (Laplace prior): enforces sparse *w*-vector

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} |w_i|$$

Multiple classes

$$\begin{aligned} L_{\mathrm{CE}}(\hat{\mathbf{y}}, \mathbf{y}) &= -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\ &= -\log \hat{\mathbf{y}}_c, \quad \text{(where } c \text{ is the correct class)} \\ &= -\log \hat{p}(\mathbf{y}_c = 1|\mathbf{x}) \quad \text{(where } c \text{ is the correct class)} \\ &= -\log \frac{\exp(\mathbf{w_c} \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)} \quad (c \text{ is the correct class}) \end{aligned}$$

Choosing a Classifier

discriminative vs generative discussion

- **Naive Bayes**

    - good for small docs
    - naive conditional independent assumption is pretty unrealistic in most cases
    - cannot deal well with correlated features
    - fast training
- **Logistic regression**
    - can robustly deal well with correlated features (distribute wieights accordingly)
    - may work better in large documents
- Important: good features!

Chapter 6 - Vector Semantics and Embeddings

“meaning” of a word?

- general conceptual relatedness / similarity
- specific forms of conceptural realtions: anatomy, hyponymy
- distinguish between word senses (bank, bank)
- etc

Lemma is **polysemous**: it has multiple semantically unrelated word senses

ex: lemma jaguar: jaguar1 (the cat), jaguar2 (the car) jaguar3 (the Fender guitar model)

homonyms and homographs

write - right: **homophones**

homographs that are not homophones:
bass1 (“fish”) - bass2 (“instrument”)

- (nearly) identical senses of different lemmas: **synonymy**: substitutable one for the other, may replace one another in a sentence
    - principle of contrast: different word forms → at least slightly different meaning

- **Hyponmy**: sub-class - class relation;

  Hypernymy (Superordinate): super-class – class relation;
- **Meronym**: part-of - whole relation;

  Holonym: whole – part-of relation;
- Antonymy: (opposite/resersed direction)

Similarity between words:

- measure via human judgement
- word relatedness (association): words belong to the same semantic field, e.g. semantic field of house = {door, roof, kitchen, ...}

semantic fields ←→ topics in Topic Models (Latent Dirichlet Allocation (LDA))

Semantic Frames: set of words that denote perspectives or participants in a particular type of event.

- **Semantic Roles:** buy or sell → buyer, seller;

Connotation:

- words can have **affective meaning** (connotation):
  sentiment, opinions, evaluations
- **dimensions of affective meaning** (vector representation):
  - **valence**: the pleasantness of the stimulus,
    e.g. happy ←→ sad
  - **arousal**: the intensity of emotion provoked by the stimulus,
    e.g. excited ←→ relaxed
  - **dominance**: the degree of control exerted by the stimulus
    e.g. important, controlling ←→awed, influenced

Sentiment analysis → need to get labels → **extremely hard to get them** → **subjective**

Distributional Semantics:

- “words that occur in similar contexts tend to have similar meanings”
- **distributional semantics**: meaning of a word is computed from the distribution of words around it

Embeddings

combine distributionalist intuition with vector representation idea → **word embeddings**

T-SNE

Map high-dim. patterns *x* to low-dim. patterns *y*:

·dim. patterns $\boldsymbol{x}$ to
patterns $\boldsymbol{y}$:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

"The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i."

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

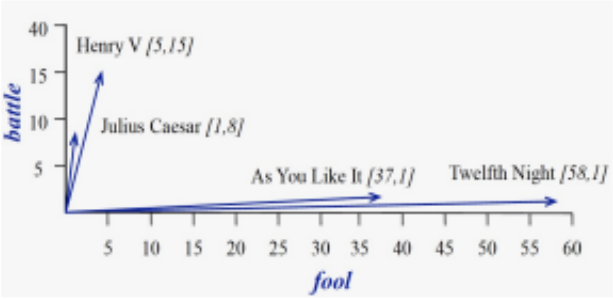
Set $\boldsymbol{y}_i$ minimizing $\boxed{\mathrm{KL}(P \parallel Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}}$

Words and Vectors

- vector space models of semantics: model word by embedding it into a vector space → vector representations of a word: embedding
- semantic similarity of words ↔ similarity of vectors

- |V| x D term (word)-document matrix with D document vectors:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

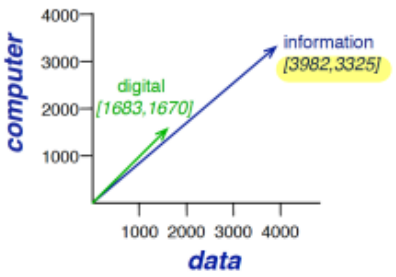


- Information retrieval: query q, document collection $D = \{d_1, d_2, \ldots\}$: find $\{d_i\}$ with high $sim(d_i, q)$. → simple application of tf-idf document (column) vectors : $sim(d_i, q) = \cos(d_i, q) = (||d_i||\,||q||)^{-1}\mathbf{d}_i^T\boldsymbol{q}$

  inner product : $d_i^T q = \|d_i\| \|q\| \cos(d_i, q)$

- words as vectors: e.g. rows of word-document matrix
- more common: word-vectors: rows of |V| x |V| word-word matrix (term-term)
  - matrix elements: feature-values (e.g. counts) of co-occurences in certain contexts

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	
strawberry	0	...	0	0	1	60	19	
digital	0	...	1670	1683	85	5	4	
information	0	...	3325	3982	378	5	13	



Instead of Counts:

TF-IDF, Pointwise Mutual Information, etc.

Similarity: usually cosine similarity

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

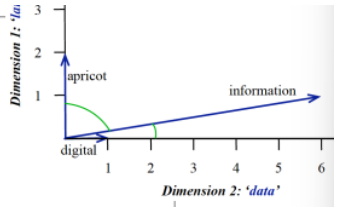
$$\text{sim}_{\text{Cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|}$$

$$\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}\max(v_i,w_i)}$$

$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}(v_i+w_i)}$$

$$\text{dist}_{\text{Jensen-Shannon}}(\vec{v},\vec{w}) = D(\vec{v}\|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}\|\frac{\vec{v}+\vec{w}}{2})$$

JS∈[0,1] → e.g. $\text{sim}_{JS}=1-\text{dist}_{JS}$



KL-divergence: $D(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$

$$JS(P||Q) = D(P\|\frac{P+Q}{2}) + D(Q\|\frac{P+Q}{2})$$

TF-IDF

- term frequencies alone: not very discriminative

- → alternative term-document matrix:
  tf-idf: term frequency in a document $\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$

  * inverse document frequency $\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$

  N : number of documents

  df_t : number of documents in which t occurs

  if word occurs in all documents, not very discriminative (0)

  → matrix element for word t and document d:

  $$\boxed{w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t}$$

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.96

- TF-IDF helps in identifying words that are **important and relevant to a specific document**
- applications: **tf-idf + cossine similarity**
- idea: represent document by centroids of its word vectors, then compute sim(d1, d2) = cos(d1, d2)

Pointwise Mutual Information

- Pointwise Mutual Information (PMI):

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

how often two events x and y occur, compared with what we would expect if they were independent

- PMI between a word w and a word c in context:

$$I(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

← co-occurrence of words (MLE)

← expectation of co-occurrence if words were independent

- $I(w,c) \in [-\infty, \infty]$ but negative values (co-occurrence *less* likely than independent occurrences) not very reliable →
  Positive PMI:

$$\boxed{\mathrm{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)}$$

- Assuming a co-occurrence count matrix F, we have

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$\text{PPMI}_{ij} = \max(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0)$$

example:

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$P(\text{w=information,c=data}) = \frac{3982}{11716} = .3399$$

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

$$P(\text{c=data}) = \frac{5673}{11716} = .4842$$

$$\text{ppmi(information,data)} = \log 2(.3399/(.6575 * .4842)) = .0944$$

PPMI	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Problem: biased towards infrequent terms → P(A) x P(B) tends to be small for infrequent terms

- use PPMI_α:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0) \qquad P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

e.g. with $\alpha = 0.75$ → $P_\alpha(c) > P(c)$ → PPMI smaller

- use Laplace Smoothing (+1 in the counts)

Dense Word-Vectors

in a |V| ×|V| word-word-matrix, each |V|-dim. word vector → **use a more dense, lower dim vectors**.

- fewer params, less overfitting, better modeling

First ideia: dimensionality reduction → SVD, PCA, ...

Modern Ideia: prediction based embeddings (word2vec, GloVe, etc...)

word2vec (skip-gram with Neg Sampling)

- intuition: instead of counting how often each word w occurs near e.g. apricot , train a classifier on a **binary prediction task**: “Is word w likely to show up near apricot ?”
  - "The quick brown fox jumps over the lazy dog.”
  - "fox" as the target word, the objective is to predict the context words (such as "quick", "brown", "jumps", ...)
- use classifier **weights as** dense **embeddings** for words
- use large corpora as “supervised” data sources for classifier

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the regression weights as the embeddings

(Log. Regression) Classifier

$$Similarity(w, c) \approx w \cdot c$$

word w

any word c in context window

vector repre-sentation of w

vector repre-sentation of c

$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

$$P(-|w,c) = 1 - P(+|w,c) = \sigma(-\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(\mathbf{c}\cdot\mathbf{w})}$$

problem: the words are not independent

- assuming independence between context words, we get

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c_i}\cdot\mathbf{w})$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(\mathbf{c_i}\cdot\mathbf{w})$$

problem: long range dependencies aren't take into account

25

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
                c1         c2    w     c3        c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

for each positive example, k negative examples (here k=2)

- sample negative examples from modified unigram distribution:

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

usually with $\alpha = 0.75$ (give rare words higher probability)

- → objective function: $L(\theta) = -\left[\sum_{(w,c)\in +} \log P(+|w,c) + \sum_{(w,c)\in -} \log P(-|w,c)\right]$

for one word - context pair (w,c) and k noise words:

$$L_{CE}(\theta) = -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k} P(-|w,c_{neg_i})\right]$$

$$= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k} \log P(-|w,c_{neg_i})\right]$$

$\theta = \{w, c_{pos}, \{c_{neg_i}\}\}$

- maximize vector similarity of (w,c) drawn from positive examples, minimize for negative examples
- start with random embeddings;
  use
  stochastic gradient descent as usual for log. regression;
  k, dimension d, and window size L: meta parameters
- two different embeddings for each word:
  - **target embedding w (matrix W)**
  - **context embedding c (matrix C)**
- final embeddings: use only w, or add w and c, or stack them

Visualizing Embeddings

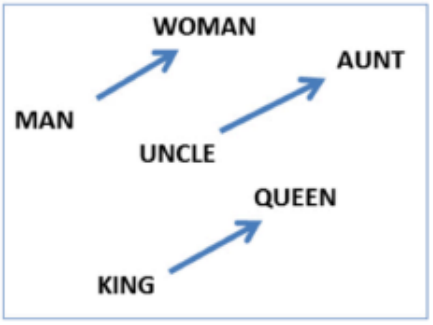
- word clouds, projection tecniques, hierarchical clustering

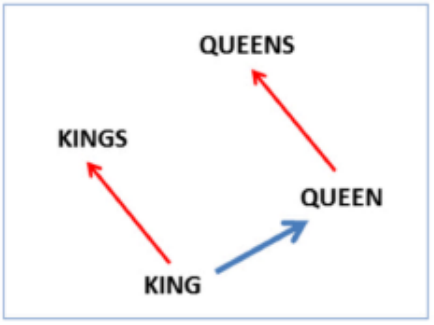
Effect of Window Size in Embeddings

- short → most similar words are semantically similar
- long → most similar words are topically realted

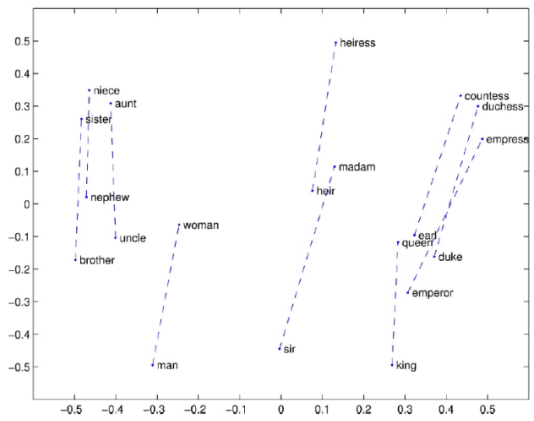
- **first order** co-occurence (**syntagmatic** co-occurrence): words that typically occur near each other: wrote ↔ book, poem
-
second order co-occurrence (**paradigmatic** co-occurrence): words that have similar neighbours: wrote ↔ said, remarked

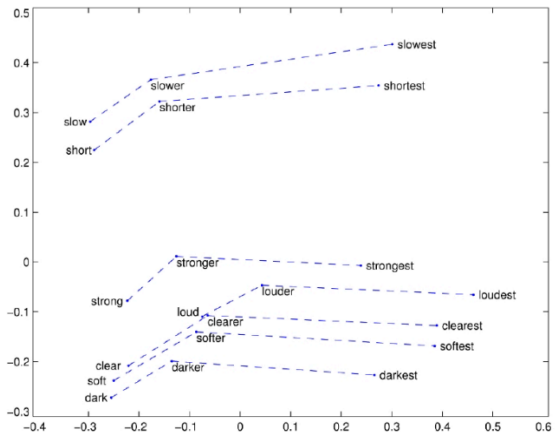
- skip-gram embeddings show semantic and other forms of **compositionality**:





example: “king” - “man” + “woman” ≈ “queen”





Historical Semantics & Cultural Bias

- historical drift, gender bias, racist bias

Evaluation Word Vectors

- compare vector similarity against human similarity judgement of raw word pairs (without sentences)
  - Word-sim-353 (2002):, SimLex-99 (2015), TOEFL dataset (1997)
- compare vector similarity against human similarity judgement of raw word pairs (with sentences)

- Stanford Contextual Word Similarity (SCWS) (2012)
- semantic textual similarity task (2012)

- analogy task: a is to b as c is to d; given a, b, and c: find d.

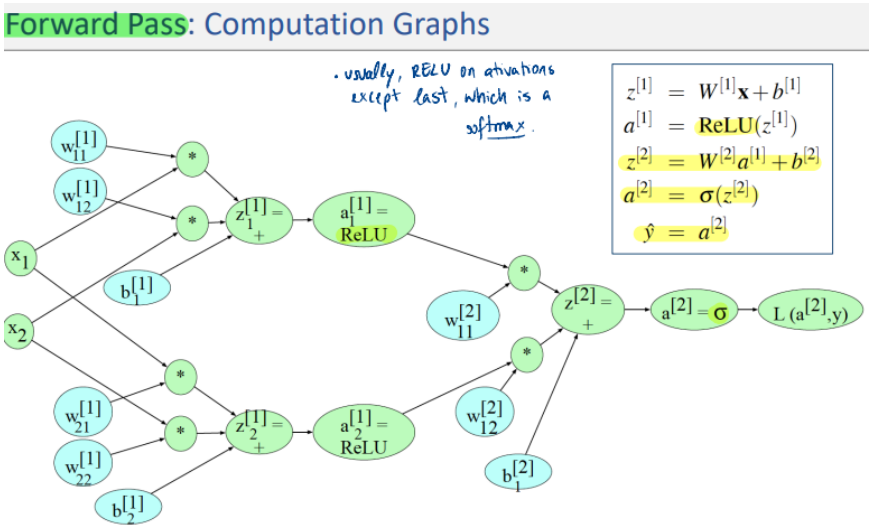
Chapter 7 - Neural Networks and Neural Language Models

Advantages:

- No restrictions due to choice of a particular model; neural networks are universal approximators.
- capable of learning appropriate features and feature representations autonomously, eliminating the need for handcrafting features.
- online learning, transfer learning, etc.

Disadvantages:

- require large amounts of data to function properly.
- computing resource-intensive.
- extremely sub-symbolic, making model parameters difficult to interpret.



- Loss function in general:

$$\boxed{L(\hat{y}, y) = L(f(x;\boldsymbol{\theta}), y)} = \text{How much } f(x) \text{ differs from the true } y$$

- Example for loss function: e.g. for regression case: → MSE

$$\boxed{L_{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)}(\theta) - y^{(i)})^2}$$

- Example for loss function:
  multi-class classification: for a softmax'ed K-neuron output:

$$p(y = k|\mathbf{x}, \theta) = \frac{\exp(a_k(\mathbf{x}, \theta))}{\sum_{k'=1}^{K} \exp a_{k'}(\mathbf{x}, \theta)} = \hat{y}_k(\mathbf{x}, \theta)$$

→ cross entropy:

$$L_{CE}(\theta) = \sum_{n=1}^{N} L_{CEn}(\theta) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_k^{(n)} \log \hat{y}_k(\mathbf{x}^{(n)}, \theta)$$

for a one-hot encoded true $y^{(n)}$

$$= -\sum_{n=1}^{N} \log \hat{y}_{correctclass}(\mathbf{x}^{(n)}, \theta)$$

if $\hat{y} \approx 1$, $\log(1) = 0$
↓
Loss is small

- NN learning: preferrably in (mini-)batches of M training examples: $x^{(n)}$ → forward pass → $\hat{y}^{(n)}(\theta)$

- compute loss $\mathrm{L}(\hat{y}^{(n)}(\theta), y^{(n)})$

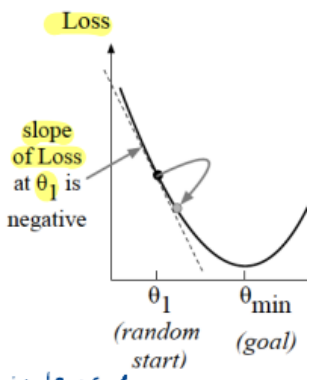
- backward pass / Backpropagation → compute

$$\nabla_\theta L = \sum_{i=1}^{N} \nabla_\theta \, \mathrm{L}(\hat{y}^{(n)}(\theta), y^{(n)})$$

- stochastic gradient descent, ADAM etc.: randomly sample some $(x^{(i)}, y^{(i)})$ (or a whole minibatch) and do smth. similar to

$$\boxed{\theta_{t+1} = \theta_t - \eta \, \nabla \mathrm{L}(\hat{y}^{(n)}(\theta), y^{(n)})}$$

↳ go against the derivate of loss



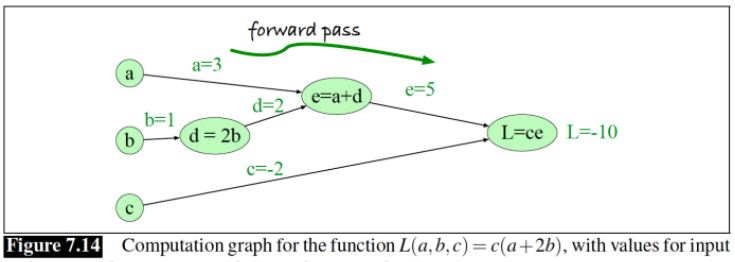


Figure 7.14 Computation graph for the function $L(a,b,c) = c(a+2b)$, with values for input nodes $a = 3$, $b = 1$, $c = -2$, showing the forward pass computation of L.

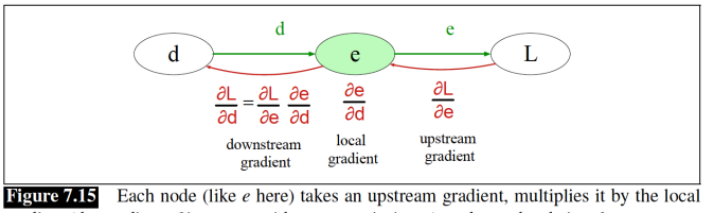
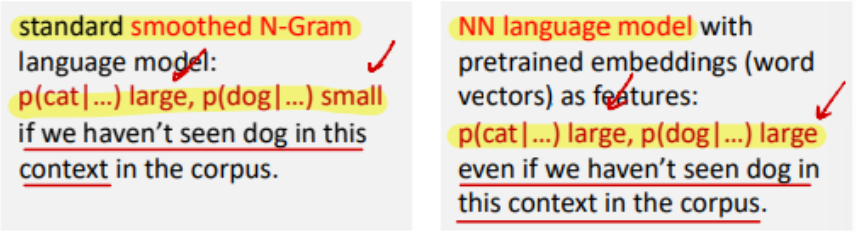


Figure 7.15 Each node (like e here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple inputs.

Application: Language Modelling

- use deep FF NN (later: RNN, LSTMS, Transformers, etc.)
- standard representation for words: **word embeddings**: vectors ∈ ℝ^d → **vector space model of meaning**: word-vectors that are ‚near' in vector space correspond to words with similar or related meaning



Chapter 8 - Sequence Labeling for Parts of Speech and Named Entities

Parts-of-Speech

- (POS, word classes, synstatic categories)
- examples: noun, pronoun, verbm adjective
- important for:
  - language models
  - Named Entity Recognition and Classification

    - Stemming
    - auto-summarization
- POS: based on not primarily categories like “adjective” but rather:
    - **syntactic** categories / functions (e.g. distributional properties (which other words usually in neighborhood)) **and**
    - **morphological** categories/functions (e.g. to carry similar suffixes)
- **closed class** (function words (e.g. of, it); fixed members (e.g. prepositions)) - are always the same

  open class (nouns, verbs, adjectives, adverbs; e.g. new nouns are continually created)

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by, under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that, which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Figure 8.1 The 17 parts of speech in the Universal Dependencies tagset (Nivre et al., 2016a). Features can be added to make finer-grained distinctions (with properties like number, case, definiteness, and so on).

Part-of-Speech-Tags

- **Nouns**:
    - Can occur with determiners (e.g., a goat, its bandwidth)
    - Can take possessives (e.g., husband’s house)
    - May occur in plural (e.g., goats, hounds)
    - Two types: Proper Nouns (specific entities, usually capitalized, e.g., Regina, IBM, Colorado) and Common Nouns (Count Nouns e.g., one goat, two goats, and Mass Nouns e.g., snow, salt, communism)
- **Verbs**:
    - Represent actions, processes, etc.

    - Can be inflected (e.g., eat, eats, eating, eaten)
- **Adjectives**:
    - Represent properties, qualities, etc. (e.g., beautiful, tall, small)
- **Adverbs**:
    - Modify something (e.g., extremely, very, somewhat, slowly, slinkily, delicately, yesterday, Monday)
    - Directional, degree, manner, and temporal adverbs exist
- **Prepositions**:
    - Occur before noun phrases to indicate spatial, temporal, or other relations (e.g., by the house, on time, with gusto, at the gate)
- **Particles**:
    - Occur with verbs to form phrasal verbs with non-compositional meaning (e.g., hand the paper over, turn down equals reject)
- **Determiners**:
    - Especially articles (definite: the; indefinite: a, an), also this, that, etc.
- **Conjunctions**:
    - Join phrases, sentences, clauses
    - Coordinating (e.g., and, or) and subordinating conjunctions (e.g., I thought that you might fail)
- **Pronouns**:
    - Shorthand referring to noun phrase etc.
    - Personal (e.g., you, I, he, she, it), possessive (e.g., your, mine, his, her, its, one's), wh-pronouns (e.g., what, whom, whoever, why)
- **Auxiliary verbs**:
    - Mark semantic features of verbs (e.g., can, do, may, should, are, have)
    - Copula be connects (e.g., he is a duck)
    - Modal verbs (e.g., can, must)
- **Other classes**:

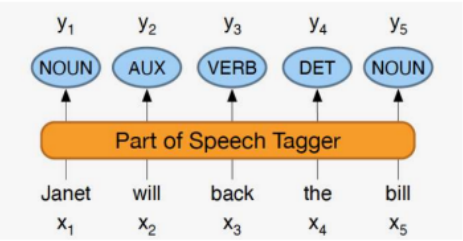
- Interjections (e.g., oh, hey, um, hmmm), negatives (e.g., no, not), politeness markers (e.g., please, thank you), greetings (e.g., hello, goodbye), etc.

Penn Treebank POS Tags

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past partici-ple	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

POS Labelled Corpora

- examples:
  - Brown corpus (1961, 106 words, different genre texts),
  - Wall Street Journal corpus (1989, 106 words),
  - Switchboard corpus (1991, 2∗106 words, telephone conversations)
- some POS tagsets assume tokenization of multipart words
- after tokenization: POS tagging for each words → desambiguation task
  - e.g. *book a flight*, *read a book*
  - not many words ambiguous but ambiguous words are among the most common tokens
- Most frequent POS tag (class) baseline: always predict the most frequent POS tag among the possible POS tags for ambiguous words ↔ state of the art ≈ 0.97



earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** debt
I was twenty-one **back/RB** then

Named Entity Recognition

- **Named Entity**: Anything referred to by a proper name, often extended to temporal or numerical expressions
  - e.g. United Airlines, Friday, Tim Wagner, etc...
- Entity Types: PER, ORG, LOC, ...
- **Named entity recognition:** finding spans of text that constitute NEs + classification
- categorical ambiguities: Washington → PER or LOC?
- use supervised **sequence classifier** such as HMM or RNN, with
  - **BIO** tagging (for n entity types → 2n+1 corresp. BIO classes) or
    - *n* entities require *2n* tags for “B-” and “I-”
    - additional “O” tag which is a single class
  - **IO** tagging (n+1 corresp. IO classes) or
  - **BIOES** tagging (E=end, S=one word span)

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

HMM for POS Tagging

- **Stages**: tags; **observations**: words
- training on labelled data: MLE by counting for A and B separately

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

- POS-Tagging via **Viterbi** algorithm
- First order Markov assumptions for A and B

Result:

$$\underset{t_1 \dots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}}$$

Viterbi:

$$\begin{aligned} v_t(j) &= \max_{q_{1:t-1}} P(q_{1:t-1}, q_t = j, o_{1:t}|\lambda) \\ &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \end{aligned}$$

Optimization: Beam Search

- instead of keeping all possibilities at each column, just concentrate on the β (beam width) most probable ones (prune the possible hidden sequence tree)

Problems

- unknown words. Possible remedies:
  - use morphological features → ___-ed : VBN or VBD most likely
  - deviate from Markov first order

Conditional Random Fields (CRF)

- discriminative sequence processing model: **linear chain CRF**

CRF:

$$\hat{Y} = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} P(Y|X)$$

- CRFs allow for the incorporation of a wide range of features derived from the input sequence, such as word identities, word context, morphological features, etc.

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)}$$

$$p(Y|X) = \frac{1}{Z(X)}$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}}$$

$$\underbrace{F_k(X,Y)}_{\text{global feature}} = \sum_{i=1}^{n} \underbrace{f_k(y_{i-1}, y_i, X, i)}_{\text{local feature}}$$

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

restriction to current and previous tags: „linear chain“ → variants of HMM algorithms (e.g. Viterbi) can be used

- any feature constructed from (y_{i-1}, y_i, X, i) e.g.

$$\mathbb{1}\{x_i = \textit{the},\ y_i = \text{DET}\}$$
$$\mathbb{1}\{y_i = \text{PROPN},\ x_{i+1} = \textit{Street},\ y_{i-1} = \text{NUM}\}$$
$$\mathbb{1}\{y_i = \text{VERB},\ y_{i-1} = \text{AUX}\}$$

- useful for unknown words:
  - word shape features:
    - x: letter, X: uppercase letter; d: number; punctuation
      - example: I.M.F. → X.X.X DC10-30 → XXdd-dd
  - prefix- or suffix-features

x_i contains a particular prefix (perhaps from all prefixes of length ≤ 2)
x_i contains a particular suffix (perhaps from all suffixes of length ≤ 2)
x_i's word shape
x_i's short word shape

example: *well-dressed*:

prefix(x_i) = `w`
prefix(x_i) = `we`
suffix(x_i) = `ed`
suffix(x_i) = `d`
word-shape(x_i) = `xxxx-xxxxxxx`
short-word-shape(x_i) = `x-x`

  - For NER (Name Entity Recognition):
    - words themselves are embeddings
    - POS of word + neighbors
    - Short word-shape: X / Xx / x
    - presence of w_i in a gazetteer

$$\hat{Y} = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} P(Y|X) = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1}, y_i, X, i)$$

- HMM Viterbi formulated in terms of tags t (also denoted as states s) as y and outputs w (also denoted as o) as x:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) P(y_j|y_i) P(x_t|y_j) \quad 1 \le j \le N, 1 < t \le T$$

- Linear chain CRF version of Viterbi

freedom to add new features

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1}, y_t, X, t) \quad 1 \le j \le N, 1 < t \le T$$

Pratical Variations for POS and NER

- Combine rule-based with ML-based (difficult data to label, subjective, hidden meaning, etc.)
- POS-Tagging for morphological rich language (use morphologically rich POS
  tags to be able to deal with unseen variants of words) → 4-10x larger tag set

Chapter 9 - RNNs and LSTMs

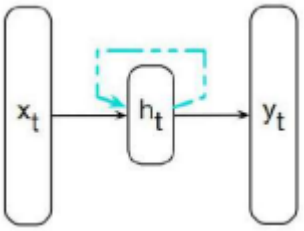
Feedforward Neural Networks

- **NOT temporal**
- assumption of simultaneous access
- Possible "fix": **sliding window approach** → windows must be large → **weights explode**, **No shared weights** for words
- **Fixed size input**: limiting the contextual information and long range dependencies

Recurrent Neural Networks (RNNs)

Definition: Any network that contains a **cycle** within the network connection.

- hidden layer act has **“memory”**
- **result**: value of some unit within the network is directly (or indirectly) dependent (as an additional input) on its own earlier outputs.



Properties:

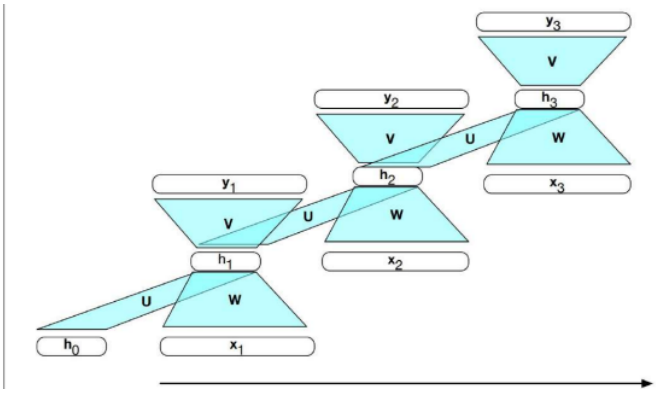
- hidden layer contains a **recurrent connection**
- sequences are processed **one item at a time**
- Hidden layer provides **memory** from the previous time steps
- **No fixed-length limit** on the prior context
- **Shared weight matrices** during inference time

Forward inference

$$\boxed{\begin{aligned} \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \\ \mathbf{y}_t &= f(\mathbf{V}\mathbf{h}_t) \end{aligned}}$$

$$W \in \mathbb{R}^{d_h \times d_{in}}$$
$$U \in \mathbb{R}^{d_h \times d_h}$$
$$V \in \mathbb{R}^{d_{out} \times d_h}$$

Training RNNs



- Backpropagation to adjust W, U and V

- However:
  - Assessing the error *h_t* requires computing its influence on current and future outputs.
- **Solution: Backpropagation Through Time**
  - Two-pass alg for traning
  - First pass - forward inference computing h_t, y_t
  - Saving value of the hidden layer at each timestep
  - Second pass - reverse processing computing gradients
- Alternative: **Unrolling** into feedforward computational graph
  - more computational expensive

RNNs as Language Models

- Predict the next word from current word and previous hidden state
- Advantages:
  - No limited context
  - Easy forward inference
- Making use of word embedding matrix *E* to encode current word

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)
\end{aligned}
$$

- Vector **Vh_t** is a set of **scores** over the vocabulary given the evidence in the hidden state *h*.
- Probability of a word being the next word in the sequence

$$P(w_{t+1} = i | w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

- Probability of the entire sequence:

$$
\begin{aligned}
P(w_{1:n}) &= \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \\
&= \prod_{i=1}^{n} \mathbf{y}_i[w_i]
\end{aligned}
$$

where y_i[w_i] is the probability of the true word w_i at time i

- **Training:** using a corpus of text by **predicting the next word**
- Minimizing **cross-entropy loss:**

$$
L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]
$$

$$
L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]
$$

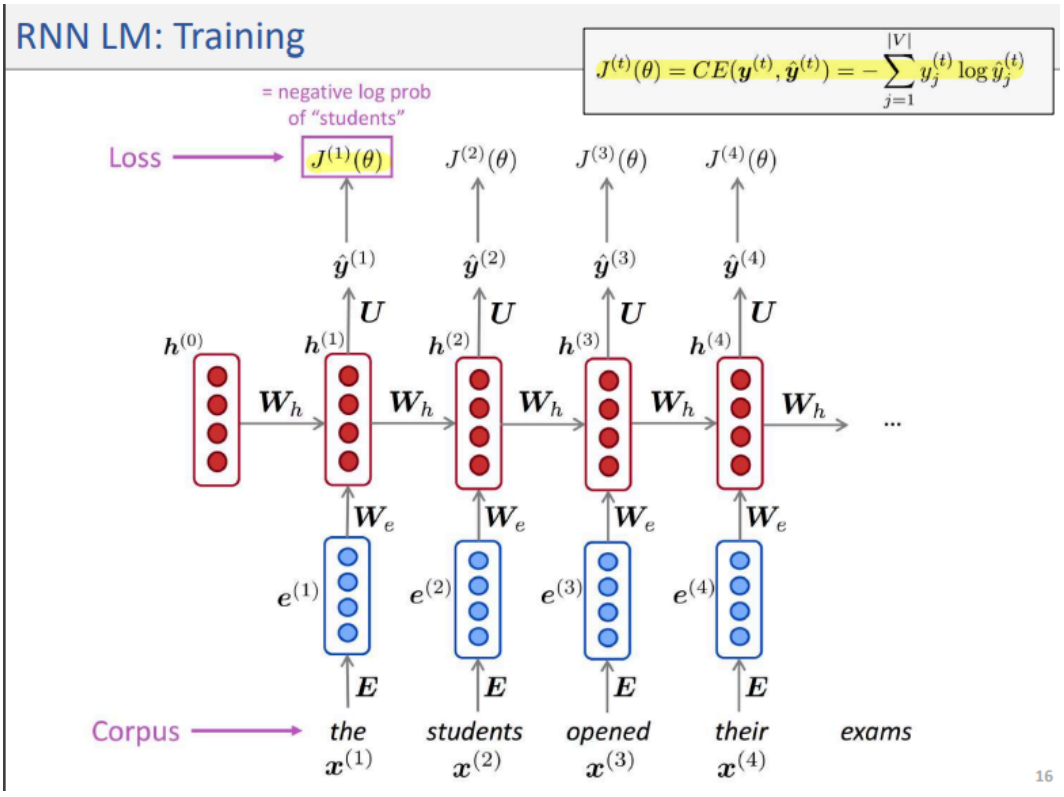
- **Teacher forcing**: input is always the correct sequence to avoid making up improbable sequences

Advantages of RNNs:

- Memory: Able to remember past information through hidden layers.
- Processing Time: Process sequences one item at a time.
- No fixed-length limit: Can handle various lengths of prior context.
- Shared weight matrices: Makes inference time more efficient.

Disadvantages of RNNs:

- Recurrent computation is slow
- All language info is put in the weight matrices
- Inability to handle long-term dependencies: 'vanishing gradient' problem, where contributions of information decay geometrically over time.
- Difficulty with real-time processing: Not naturally suited for tasks that require real-time predictions.



Final loss:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

RNNs for Sequence Labeling

= assigning a label to each element of a sequence

- inputs: word embeddings ; outputs: tag probabilities

RNNs for Sequence Classification

= assigning a single label to a sequence of inputs

- only last h_n is takien as input to FFN
  - idea: h_n is a compressed representation of sequence
- end-to-end training (loss on final task)

RNNs for Text Generation

Basic approach: **Autoregressive generation**

- put <s> in the beggining of the sentence

- take ouput word and use as new input

Stacked RNNs

Output of one layer serves as input to another

- generally better performance than single-layer
- subsequent stacked layers offer more abstraction

Bidirectional RNNs

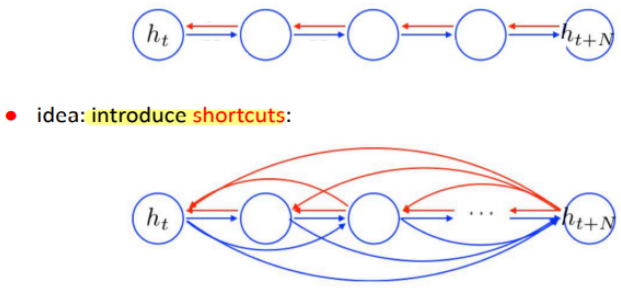
Introduces **right-context** (looking into the **future**)

- **forward** traversal and **reverse** input sequence
- Concatenate both representations before prediction

$$\begin{aligned} \mathbf{h}_t &= [\mathbf{h}_t^f ; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b \end{aligned}$$

RNNs Problems

- Increasing distance is hard to represent ← Bottleneck!
- Hidden state is mostly local → overwritten in every step
- **Bottleneck: Final Hidden State** (especially in seq classification)
- **Vanishing Gradients:** repeated multiplications drive gradient to zero especially if long backpropagation
  - **Idea/Solution:** prevent error backpropagation through all intermediate nodes → introduce **shortcuts**



Long Short-term Memory (LSTM)

Divides the problem into:

- Removing unneeded information
- Adding new information
- Keeping separate states memory **C** and hidden state **H**

Uses **gates** to control flow of information

Gates

- Use additional weights U and W
- Use **input X**, **hidden layer H, memory C**

- Forget gate $\mathbf{f}_t = \sigma(\mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t)$
- Input gate $\mathbf{i}_t = \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t)$
- Output gate $\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)$
- Candidate memory (update candidate) $\mathbf{g}_t = \tanh(\mathbf{U}_g\mathbf{h}_{t-1} + \mathbf{W}_g\mathbf{x}_t)$

- **Forget Gate:** decides the amount of previous memory to retain.
- **Input Gate:** controls how much of the current information is stored in the memory.
- **Output Gate:** determines the amount of memory to transfer to the hidden state.

Problems:

- Process sequences sequentially, limiting parallelization
- Struggle with capturing long-range dependencies

Encoder-Decoder Networks (Seq2Seq Networks)

Models capable of generating contextually appropriate, arbitrary length, output sequences

3 parts:

- Encoder: accepts input sequence
- Context vector c: conveys the essence of the input
- Decoder: accepts input c → generates hidden states and output sequence

Conditional language model

Condition on source text

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)\dots p(y_m|y_1,\dots,y_{m-1},x)$$

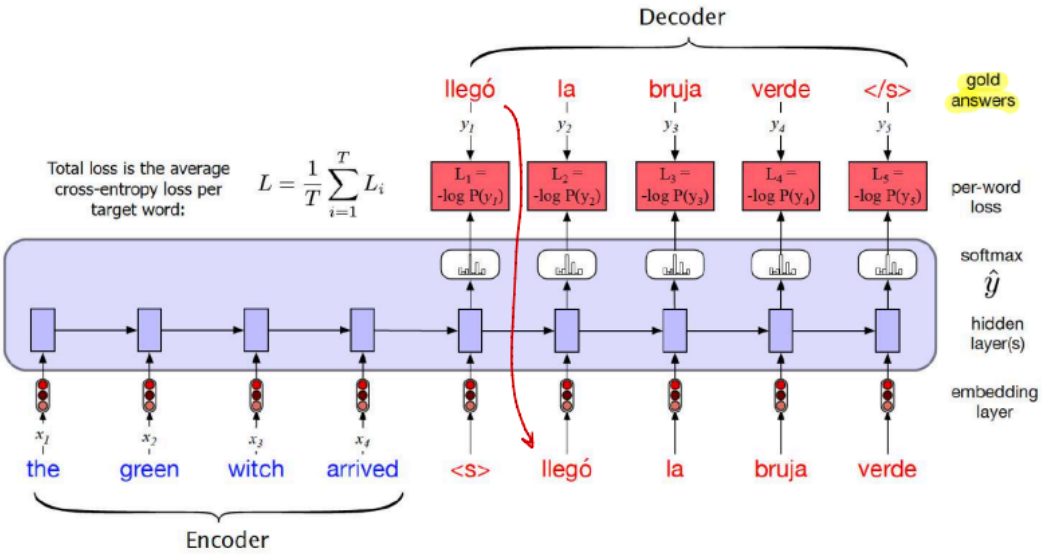
counter „waning of context“ in decoding by making context available in all decoding steps:

decoder encoder

$$\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\boxed{\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})} \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}$$

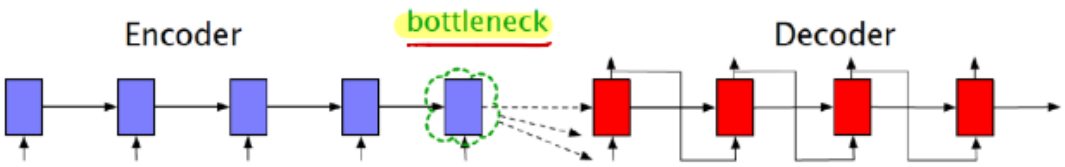
Training

- training data: **source-target pairs**
- **Approach:**
  - Give source text to model
  - train autoregressively to predict next target word
  - Calculate loss for each token
  - Average token losses for each sentence level loss
- **Teacher forcing**
  - Inference: Previous prediction as input for next word
  - Training: Gold target token instead of prediction as input
    - not propagate wrong predictions
    - speed up training



Attention

- Solves the **Bottleneck problem**:
  - **Encoder final state** only info about source sentence for decoder
  - → must encode **ALL information in one state vector**



- **Attention** mechanism: include **all encoder hidden states** in context vector
  - weighted sum of encoder hidden states → **focus on relevant parts in the source text**
  - **context vector is different** for each decoding step

Dot-Product Attention

- Calculate relevance of each encoder state to i-th decoder state:

- similarity score between ***j*-th encoder** and ***i-1*-th decode**r hidden states

$$\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j$$

decoder is one step ahead because of <START> token

and repeat for each encoder state

- **normalize** score and repeat for **each encoder state**

$$\alpha_{ij} = \boxed{\text{softmax}}(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)\ \boxed{\forall j \in e})$$
$$= \frac{\exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j))}{\sum_k \exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}$$

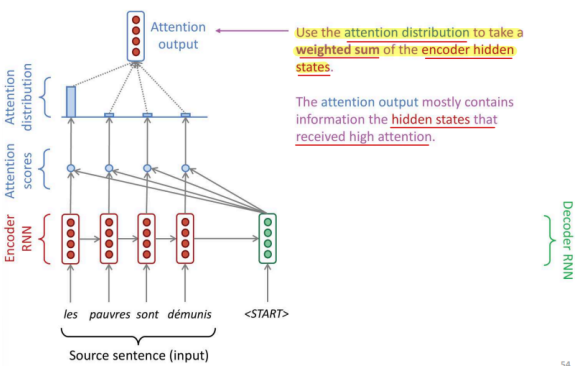
- possibility to include weights in score function
  - weights optimized during training
    - **learn aspects** relevant in current application
    - allow **different dimensions** in encoder and decoder
- Given distribution α , calculate fixed-length context vector for **i-th decoder hidden state**

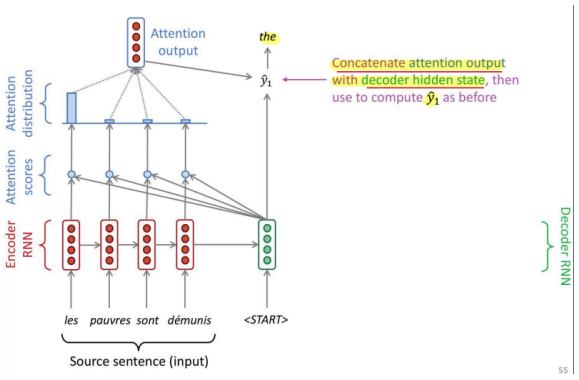
$$\boxed{\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j}$$

weighted hidden states

- Encoder-decoder with **dynamic context vector**

Example:





Dot-Product Attention: Summary

- encoder hidden states $\boldsymbol{h}_1^e, \boldsymbol{h}_2^e, \ldots, \boldsymbol{h}_N^e$ (where each $\boldsymbol{h}_1^e \in \mathbb{R}^h$)
- at time-step t:
  - decoder hidden state: $\boldsymbol{h}_t^d \in \mathbb{R}^h$
  - 1. attention scores: $\boldsymbol{scores}^{(t)} = [\boldsymbol{h}_t^d \boldsymbol{h}_1^e, \boldsymbol{h}_t^d \boldsymbol{h}_2^e, \ldots, \boldsymbol{h}_t^d \boldsymbol{h}_N^e] \in \mathbb{R}^N$
  - 2. attention distribution: $\boldsymbol{\alpha}^{(t)} = \text{softmax}(\boldsymbol{scores}^{(t)}) \in \mathbb{R}^N$
  - 3. attention output (fixed-length context vector): weighted sum of encoder hidden states:

$$\boldsymbol{c}_t = \sum_{i=1}^{N} \alpha_i^{(t)} \boldsymbol{h}_i \in \mathbb{R}^h$$

  - 4. usually: concatenate attention output and decoder hidden state:

$$[\boldsymbol{c}_t; \boldsymbol{h}_t^d] \in \mathbb{R}^{2h}$$

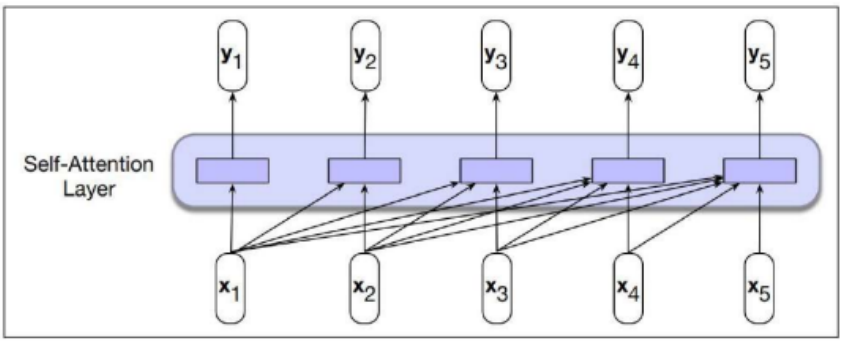
Chapter 10: Transformers and Pretrained Language Models

Transformers Networks

- Mapping sequence of inputs to sequence outputs
- use **Self-Attention** mechanism
- Solves the problems of
  - recurrent connections → information loss, difficulties in training
  - parallelization

Self-Attention Networks

- Self-attention layer:
  - Extract information from **arbitrary length context**
    - computation for each token is independent of the sequence length
  - No passing through intermediary recurrent connections



- For each input: Access to all information from previous inputs = **availability of long term context**
- Computation per node is independent = **parallelizable**

Self-Attention for Transformers

How each input token can act in the attention process:

- as **Query**: current focus of attention
- as **Key**: preceding input being compared to the query
- as **Value**: used to compute the output

Use matrices to project inputs to these roles:

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Using projections, scoring becomes:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

Output calculation is done via the values:

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{v}_j$$

Optimization

- Since **dot products can become too large**, they are instead **scaled** by the square root of the dimensionality of the key vector

$$\boxed{\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}}$$

- Further sped up through **matrix multiplication**

$$\boxed{\mathit{SelfAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}}$$

Note: **Attention is quadratic** w.r.t. length of the input

Masking

- avoid each query being compared with each key, including those that follow the query → we cannot see what we are predicting

N

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

Transformer Blocks

- Basis of the transformer block: **Self-attention layer**
- Additional features:
  - Feedforward layers
    - capture complex, non-linear relationships within the data
  - Residual layers
    - facilitate the flow of information through the model
  - Normalizing layers
    - help stabilize and accelerate the training process
- Transformer blocks have **identical input and output dimensions**, meaning they can be **stacked**

Layer Norm

- used to improve training performance
- based on standard **z-score:**

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

- Optional: **Learnable parameters** for gain and offset

$$LayerNorm = \gamma \hat{\mathbf{x}} + \beta$$

Multihead Attention

- Words can relate to each other in different ways at the same time
- hard to learn this with a single transformer block

Solution:

- **Multihead self-attention layers**
- each layer features a **distinct set of parameters and weights**
- **input is copied** across all heads
- computed **in paralle**l at the same depth of the model
- at the end of a block **reduced** back to the original dimensionality by

  concatenation and linear projection via weight matrix W^0

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 ... \oplus \mathbf{head}_h)\mathbf{W}^O$$
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$
$$\mathbf{head}_i = SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

Positional Encodings

Problem:

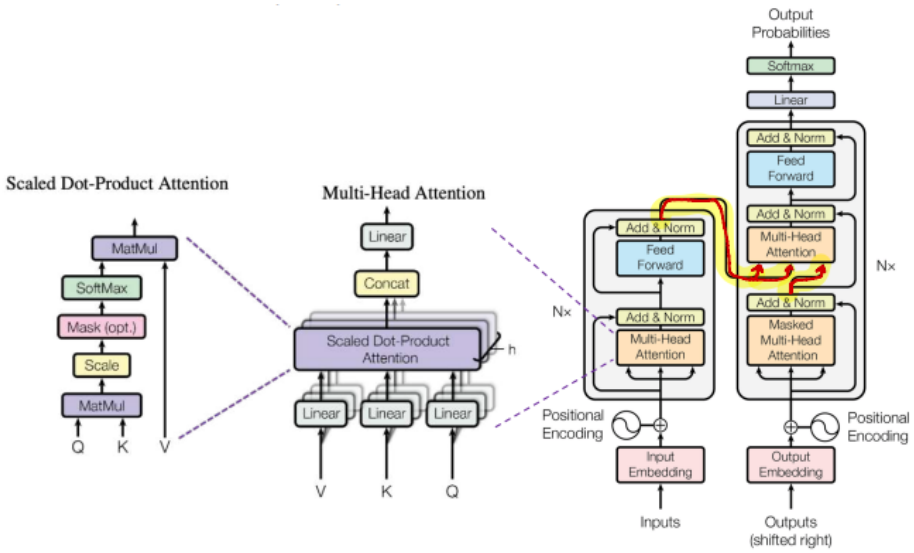
- No notion of relative or absolute position in a sequence

Solution:

- Modify input embeddings with positional encodings

- Example: Randomly initialized, trainable positional encodings
    - Either use absolute positions or a mapping to relative positions

The Transformer



Advantages

- no longer serial computation → **Parallelization**
- new text can be autoregressively generated

Applications

- **Contextual generation**: model has access to full priming context & all own subsequently generated outputs so far
- **Summarization**: full length text + delimiter δ + autoregressive; trained using teacher forcing
- **Sequence labeling**
- **Sequence classification**

Usual Way:

- No direct training of the transformer on the task
- Use **pretrained** transformer
- Add **feedforward layer on top** to finetune on the task

Problems of Greedy Decoding

- decoder uses **greedy decoding**: take most probable word at each step
- Problem: no way to undo decisions
- Solution:
  - use **Beam Search** → on each step of decoder, keep track of the **k most probable partial translations**
  - select k-best options → compute V possible extensions → rank k*V new hypotheses → prune k-best → repeat
  - Ranking hypotheses → Calculate **probability of token given existing sequence**
  - **Length normalization**: avoid lower probability for longer strings

---

Chapter 11 - Fine-Tuning and Masked LMs

- GloVe, Word2Vec → one dense embedding for each word
- however: word tokens can have different aspects in different contexts → **contextual-embeddings**
  - BERT == whole pre-trained NN allowing to take whole context as input at test time
- **Contextual-embeddings**: each word *w* will be represented by a different vector each time it appears in a different context.
- **Pretraining**: learning some sort of representation for words or sentences by processing very large amounts of text → **pretrained language models**
- **Fine-tuning**: using pretrained models + further training the model to perform some down-stream task
- **Transfer learning**: acquiring knowledge from one task or domain and then applying it (transferring it) to a new task.

Bidirectional Transformer Encoders

- for e.g. sequence classification usage of information only from the left context is
  not enough.
- In casual transformer (e.g. for language modelling / NMT decoding): upper triangular portion of **QK** matrix is **masked**.
- In **bidirectional transformer** (e.g. for NMT encoding, sequence classification) no masking → **allow the model to contextualize each token using information from the entire input**.

Original bidirectional encoder model **BERT** (Devlin et al., 2019)

Masking Words

- How to train bidirectional transformers? **Cloze task** (Taylor, 1953): predict the missing elements.
- **Masked Language Modeling (MLM)** (Devlin et al., 2019): the original approach to training bidirectional encoders → mask some words

randomly select a token in train text and use it in one of three ways:

1. replace with **unique vocabulary token [MASK]**.
2. replace with **another token** from the vocabulary, **randomly sampled** based on token unigram probabilities. (confuse the model on purpose)
3. it is left **unchanged**.

Considerations in training

- **too little** masking: too expensive training (you should eventually mask and guess every word);
- **too much masking**: not enough context (quality suffers)

Masking Spans

- **Span**: a contiguous sequence of one or more words selected from a training text, prior to sub-word tokenization.
- SpanBERT
  - The parameters for spans masking are the same as in BERT
  - Span Boundary Objective (SBO)

    - *s* denotes the word before the span and *e* denotes the word after the span, p_x: positional embedding (which word in the spam is currently predicted).

- Main advantages over **BIO-based per-word labelling**:
  - prone to a labeling mismatch problem;
  - naturally accommodate embedded named entities (*United Airlines* and *United Airlines Holding*)

Next Sentence Prediction

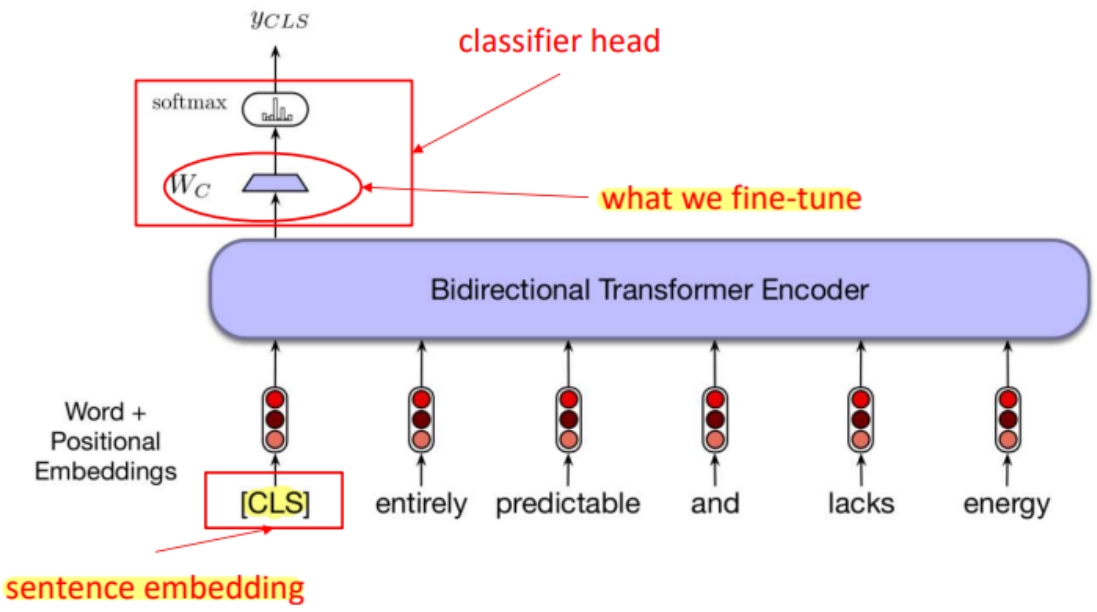
Next Sentence Prediction (NSP): given a pair of sentences, to predict whether a pair consists of a pair of adjacent sentences or a pair of unrelated sentences. → classification

- New tokens: [CLS] - input sentence pair; [SEP] - between sentences and at the end

Transfer Learning through Fine-Tuning

pretrained on big corpora → fine-tuning with additional application-specific parameters of LLMs on labeled data for the application

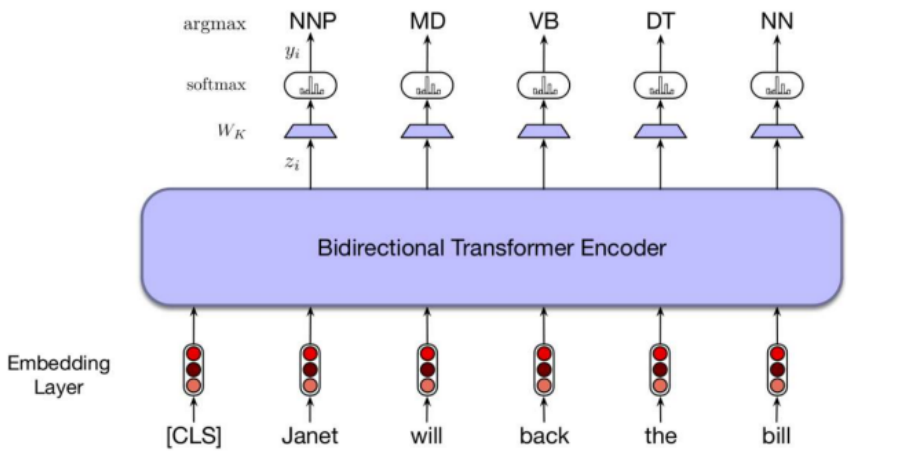
Sequence Classification



Pair-Wise Sequence Classification

- **Natural Language Inference (NLI)** or **recognizing textual entailment**: pair of sentences → classify the relationship between their meanings
- **MultiGenre Natural Language Inference (MultiNLI**) dataset: given a premise and a hypothesis : predict their relationship – entails, contradicts or neutral:

Sequence Labelling



The ideal tagging: [$_{LOC}$ **Mt. Sanitas**] is in [$_{LOC}$ **Sunshine Canyon**]

but WordPiece splits:

```
'Mt', '.', 'San', '##itas', 'is', 'in', 'Sunshine', 'Canyon' '.'
```

Solution: assign BIO tag of the first sub-token to the whole token.

Potential Harms from Language Models

- **Toxic language generation**: non-toxic prompts can nonetheless lead large language models to output hate speech and abuse
- Language Models can be **biased**
- Generation of **misinformation**, phishing, radicalization, and other socially harmful activities
- **Privacy issues**: leak of information from training data

Chapter 13 - Machine Translation and Encoder-Decoder Models

- 1950- ... : early machine translation: rule-based systems based on bilingual dictionaries
- 1990s-2010s: core idea: learn probabilistic model from data
  - find best sentence *y* in target language given a source language sentence
    x → large amounts of parallel data required, detailed
- **Alignment:** correspondence between particular words in translated sentence pair
  - some words may have no counterpart
  - can be **one-to-many** (“fertile” words)
  - or many-to-one
  - or many-to-many (phrase level)

Statistical Models

- extremely complex, lots of important details, lots of feature engineering, human effort to maintain language pairs
- use **Neural Machine Translation (NMT)**

Language Divergences and Typology

Models need to consider **linguistic differences among languages**

- **Word Order** Typology
  - position of verb compared to subject/object
  - adjective before/after noun
  - adverb positions
  - lexical divergences (words with multiple meanings)
  - more than one word (e.g. heaven, sky → himmer (german))
  - Referential density: Languages with implicit pronouns and references

Encoder-Decoder Networks

Generally:

- **Encoder** gets arbitrary length input x_1:n → generates hidden states ***h_e***
- **Context vector *c*** is a function of *h_e,* includes essence of the input
- **Decoder**: gets c, generates ***h_d*** from which **output** is obtained

Can use RNNs / LSTMs, CNNs or Transformer architectures

Advantages and Disadvantages of NMT

Advantages of NMT:

- Compared to SMT,
  - better performance, more fluent, better use of context, etc
  - a single neural network optimized
  - less human engineering effort - no feature engineering

Disadvantages of NMT:

- less interpretable - hard to debug
- difficult to control

Advantages of Attention

- significantly improves NMT performance
  - focus on certain parts of the source
- solves the bottleneck problem
  - allows decoder to look directly at source; bypass bottleneck
- helps with the vanishing gradient
  - provides shortcut to faraway states
- provides some interpretability

Encoder-Decoder with Transformer

Cross-Attention

- **Recap** self-attention: queries, keys and values from previous decoder layer

- **Cross-attention:** keys and values from encoder output

$$\mathbf{Q} = \mathbf{W}^{\mathbf{Q}}\mathbf{H}^{dec[i-1]}; \quad \mathbf{K} = \mathbf{W}^{\mathbf{K}}\mathbf{H}^{enc}; \quad \mathbf{V} = \mathbf{W}^{\mathbf{V}}\mathbf{H}^{enc}$$

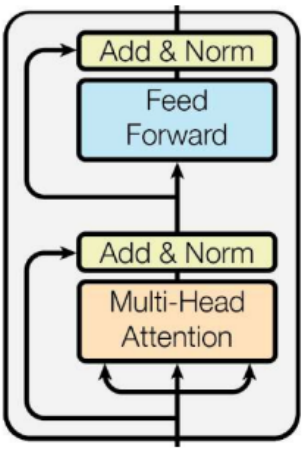
$$\boxed{CrossAttention(\mathbf{Q},\mathbf{K},\mathbf{V}) \;=\; \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\mathbf{V}}$$

Complete Block of Encoder

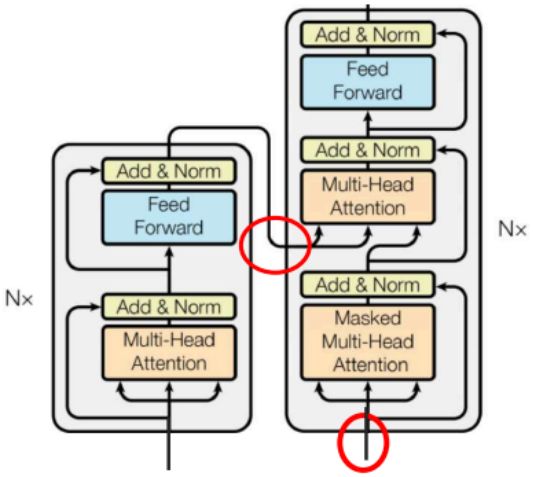
- each block: two "sublayers":
  - unmasked multihead self attention
  - two-layer fully connected feed forward (C)NN with ReLu:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- each sublayer:
  - residual (short-circuit) connection and Layer Norm:
  - Layer Norm(x + Sublayer(x))
  - Layer Norm changes input to have mean 0 and variance 1



Decoder



encoder decoder attention : queries from previous decoder layer; keys & values from encoder output

decoder self attention on previously generated outputs; masked to prevent information flow from future to past (ensure auto-regressive property)

Tokenization

Wordpiece algorithm:

- input: Training corpus and desired vocab size *V*

1. Init lexicon with all characters
2. Repeat until V wordpieces in lexicon
    - train N-gram LM on training corpus using current lexicon
    - New wordpieces by concatenating two from current lexicon

Backtranslation

- Problem: limited amount of parallel data but large monolingual corpus available in target language (e.g. English)
    - **add synthetic data** to parallel corpus

Monte Carlo search: useful for underfitted decoders

- Previous approaches: take most probable word
- Here: **sample words according to probabilities**

Sentence alignment

- Score function to measure **likelihood of two spans being translations** e.g. cosine similarity between multilingual embeddings
- Alignment algorithm to **find alignment** based on scores

Evaluation of MT

- Evaluation along two dimensions:
    - Adequacy / Faithfulness: Preservation of exact meaning
    - Fluency: quality of target language text
- Human evaluators
    - online crowd workers
    - **disagreement** between raters → normalize and exclude outliers

Automatic Evaluation Metrics

Compared to humans: **less expensive but less accurate**

- **BLEU** (Bilingual Evaluation Understudy): compare machine translation to one or more human translations & compute similarity score based on:
  - n-gram precision (usually n=3 or 4)
  - & extra penalties for too short machine translations
  - Purely precision-based
  - disadvantages of BLEU:
    - **many valid translations** exist → poor BLEU score of an otherwise good translation that just differs in n-gram overlap from the available human translations in corpus
    - no synonym awareness
    - reference summary needed

- **Character F-score (chrF)**: rank candidate by character n-gram overlaps
  - chrF used to compare performances of two models → statistical significance test

chrP percentage of character 1-grams, 2-grams, ..., k-grams in the hypothesis that occur in the reference, averaged.

chrR percentage of character 1-grams, 2-grams,..., k-grams in the reference that occur in the hypothesis, averaged.

→ combine precision and recall in weighted F-score
($\beta = 2$ weights recall twice as much as precision)

$$\text{chrF}\beta = (1+\beta^2)\frac{\text{chrP}\cdot\text{chrR}}{\beta^2\cdot\text{chrP}+\text{chrR}}$$

chrF example

- chrF2,2 → 2-grams and $\beta = 2$

  REF: `witness for the past,`
  HYP1: `witness of the past,` chrF2,2 = .86
  HYP2: `past witness` chrF2,2 = .62

- 1. Remove whitespaces

  REF: `witnessforthepast`, (18 unigrams, 17 bigrams)
  HYP1: `witnessofthepast`, (17 unigrams, 16 bigrams)

- 2. Count matching unigrams and bigrams

  unigrams that match: `w i t n e s s f o t h e p a s t` , (17 unigrams)
  bigrams that match: `wi it tn ne es ss th he ep pa as st t`, (13 bigrams)

- 3. Calculate precision and recall

  unigram P: 17/17 = 1 unigram R: 17/18 = .944
  bigram P: 13/16 = .813 bigram R: 13/17 = .765

- 4. Combine to F-Score

$$\begin{aligned} \text{chrP} &= (17/17 + 13/16)/2 = .906 \\ \text{chrR} &= (17/18 + 13/17)/2 = .855 \\ \text{chrF2,2} &= 5\frac{\text{chrP} * \text{chrR}}{4\text{chrP} + \text{chrR}} = .86 \end{aligned}$$

41

- **BERTScore**: no supervised training data → **token-wise cosine similarity** of embedding representations

  - Token in x matched to $\hat{x}$ → recall
  - Tokens in $\hat{x}$ matched to x → precision

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\tilde{x}_j \in \tilde{x}} x_i \cdot \tilde{x}_j$$

$$P_{\text{BERT}} = \frac{1}{|\tilde{x}|} \sum_{\tilde{x}_j \in \tilde{x}} \max_{x_i \in x} x_i \cdot \tilde{x}_j$$

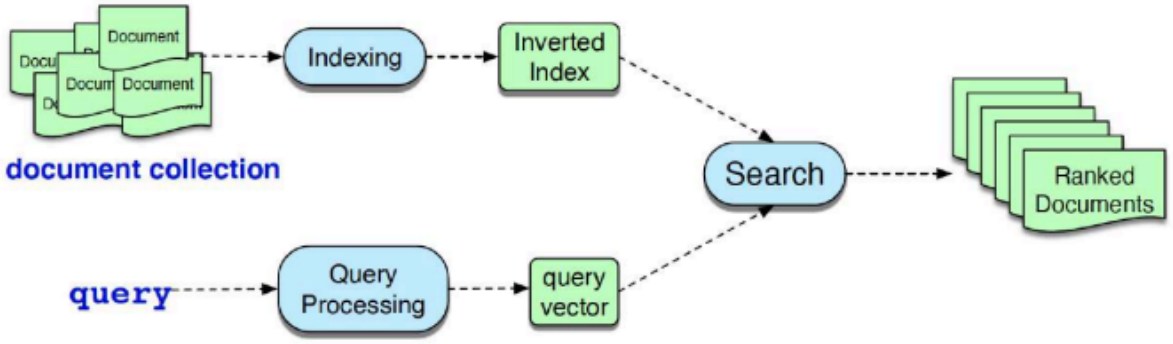
Prevailing problems

- **out-of-vocabulary** words
- **domain mismatch** between train and test data
- maintaining **context** over long text
- low-resource language pairs
- using **common sense** still hard
- NMT picks up **cultural bias** in training data

Chapter 14 - Question Answering and Information Retrieval

- factoid question answering via **Information Retrieval** (i.e. by finding as answers short text segments on the Web or some other collection of documents)

General idea: document collection & query → process index → search → rank documents



Query-document matching

simple approach → **tf-idf + cosine**

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$$

→ document

↳ term

$$\text{idf}_t = \log_{10}\frac{N}{\text{df}_t}$$

N = total docs

df_t → number of docs where t appears

$$\text{tf-idf}(t,d) = \text{tf}_{t,d}\cdot\text{idf}_t$$

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|}$$

$$= \sum_{t\in\mathbf{q}} \frac{\text{tf-idf}(t,q)}{\sqrt{\sum_{q_i\in q}\text{tf-idf}^2(q_i,q)}} \cdot \frac{\text{tf-idf}(t,d)}{\sqrt{\sum_{d_i\in d}\text{tf-idf}^2(d_i,d)}}$$

$$\approx \sum_{t\in q} \frac{\text{tf-idf}(t,d)}{|d|}$$

$d_i \in d$: the tokens d_i in the document d

$q_i \in q$: the tokens q_i in the query q

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

In all Shakespeare plays

Query: sweet love
Doc 1: Sweet sweet nurse! Love?
Doc 2: Sweet sorrow
Doc 3: How sweet is love?
Doc 4: Nurse!

	Document 1					Document 2				
word	count	tf	df	idf	tf-idf	count	tf	df	idf	tf-idf
love	1	0.301	2	0.301	0.091	0	0	2	0.301	0
sweet	2	0.477	3	0.125	0.060	1	0.301	3	0.125	0.038
sorrow	0	0	1	0.602	0	1	0.301	1	0.602	0.181
how	0	0	1	0.602	0	0	0	1	0.602	0
nurse	1	0.301	2	0.301	0.091	0	0	2	0.301	0
is	0	0	1	0.602	0	0	0	1	0.602	0
$\lvert d_1\rvert = \sqrt{.091^2+.060^2+.091^2} = .141$						$\lvert d_2\rvert = \sqrt{.038^2+.181^2} = .185$				

Doc	$\lvert d\rvert$	tf-idf(*sweet*)	tf-idf(*love*)	score
1	.141	.060	.091	1.07
3	.274	.038	.091	0.471
2	.185	.038	0	0.205
4	.090	0	0	0

- Variant: **BM25**

$$\sum_{t\in q} \overbrace{\log\left(\frac{N}{df_t}\right)}^{\text{IDF}} \overbrace{\frac{tf_{t,d}}{k\left(1-b+b\left(\frac{|d|}{|d_{\text{avg}}|}\right)\right)+tf_{t,d}}}^{\text{weighted tf}}$$

k: adjust balance btw. tf and idf

b: control influence of document length normalization

$|d_{avg}|$: length of average document

IR: Inverted Index

dictionary+postings: hash-map of terms (keys) and list of references to documents containing term

Doc 1: Sweet sweet nurse! Love?
Doc 2: Sweet sorrow
Doc 3: How sweet is love?
Doc 4: Nurse!

how {1} → 3 [1]
is {1} → 3 [1]
love {2} → 1 [1] → 3 [1]
nurse {2} → 1 [1] → 4 [1]
sorry {1} → 2 [1]
sweet {3} → 1 [2] → 2 [1] → 3 [1]

document frequency

term frequency

IR: Performance Measures

- Performance measures: IR-system returns ranked list of answer documents
  → Precision and Recall ⇝ Precision@rank , Recall@rank

$$\Rightarrow \text{Precision @ Rank } r = \frac{\#\,R \text{ in First}(r)}{r} \qquad \text{Recall @ Rank } r = \frac{\#\,R \text{ in First}(r)}{\text{Total } \#\,R}$$

Variations:

- Combine Precision@ fixed Recall step size
- **Mean average precision**: in contrast to previous slide: only note precision **where recall actually changes**

IR with Embeddings

obvious idea:

$$h_q = \text{BERT}_Q(\text{q})\,[\text{CLS}]$$
$$h_d = \text{BERT}_D(\text{d})\,[\text{CLS}]$$
$$\text{score}(d,q) = h_q \cdot h_d$$

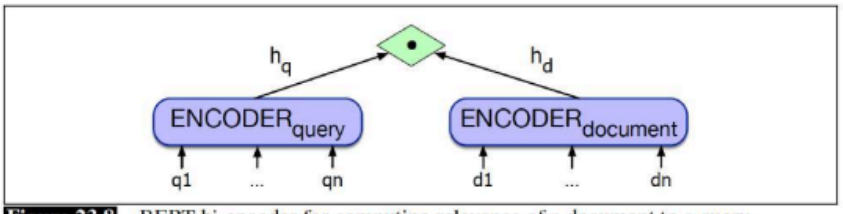


Figure 23.8 BERT bi-encoder for computing relevance of a document to a query.

Retriever and Reader

Retriever → gets relevant docs for question, e.g. by calculating score for questions + doc.

Reader → Extract answer span from doc

→ **<u>Answer Span Extraction</u>**

Span-labelling: *q* (n tokens), document passage *p* (m tokens):
infer
p(a | p,q) for span *a* in *p*

Simplifying assumption: $p(a|p,q) = p_{start}(a_s|p,q)p_{end}(a_e|p,q)$

Entity Linking

- associating a mention in text with some real-world entity represented by a Wikipedia article ("wikification") or as a concept in an ontology (DBPedia, OWL-based...)
- components / stages: **mention detection**, **mention disambiguation**

Classical Approach (Wikification by **TAGME**):

- Give idx to *e_i* to each wiki-page (=entity)
- For each *e_i*, compute **in-link** count from other *e_j* (#links tot his page from other pages)
- **anchor dictionary entry** for page/entity *e_i* : title of *e_i*, all **in-link anchor strings**
- delete anchor strings with low **link probability** *link(a)/freq(a)*

Mention detection:

- for each token sequence (≤ 6 tokens):
  →
  query anchor dictionary → find mention spans in anchor strings

Mention disambiguation:

- for all pages $\mathscr{E}$(a) pointed to by ambiguous anchor span *a*
  - **prior probability** that anchor string *a* points to entity (page) *e* ∈ $\mathscr{E}$(a)

$$\text{prior}(a \to e) \;=\; p(e|a) = \frac{\text{count}(a \to e)}{\text{link}(a)}$$

Problem: favors more common entities

- relatedness score: incorporate other candidate anchor spans

$$\text{relatedness}(a \to X) = \sum_{b \in \mathcal{X}_q \setminus a} \text{vote}(b, X)$$

chinest

- To **answer:** choose entity with the **highest prior probability among those with highest relatedness scores**

Neural Graph-Based Linking

Entity Mention Detection

$$[\mathbf{q}_1 \cdots \mathbf{q}_n] = \text{BERT}([\text{CLS}]q_1 \cdots q_n[\text{SEP}])$$

$$s_{\text{start}}(i) = \mathbf{w}_{\text{start}} \cdot \mathbf{q}_i$$

$$s_{\text{mention}}(t) = \mathbf{w}_{\text{mention}} \cdot \mathbf{q}_t \quad , \quad i < t < j$$

$$s_{\text{end}}(j) = \mathbf{w}_{\text{end}} \cdot \mathbf{q}_j$$

$$p([i,j]) = \sigma\left(s_{\text{start}}(i) + s_{\text{end}}(j) + \sum_{t=i}^{j} s_{\text{mention}}(t)\right)$$

Entity Linking

entity encoding:

$$\mathbf{x}_e = \text{BERT}_{[\text{CLS}]}([\text{CLS}]t(e_i)[\text{ENT}]d(e_i)[\text{SEP}])$$

title of entity (page)

first 128 tokens of entity (page))

linking:

$$s(e,[i,j]) = \mathbf{x}_e^{\cdot}\mathbf{y}_{i,j} \qquad p(e|[i,j]) = \frac{\exp(s(e,[i,j]))}{\sum_{e' \in \mathcal{E}} \exp(s(e',[i,j]))}$$

Knowledge-Based QA from RDF

<u>From RDF-triples (subj+pred+obj)</u>

- Task: Map question to formal relation:
  - e.g. when was Ada born? → birthyear(Ada, ?x)

$$\mathbf{m}_r = \text{BERT}_{\text{CLS}}([\text{CLS}]q_1 \cdots q_n[\text{SEP}])$$

$$s(\mathbf{m}_r, r_i) = \mathbf{m}_r \cdot \mathbf{w}_{r_i}$$

$$p(r_i|q_1, \cdots, q_n) = \frac{\exp(s(\mathbf{m}_r, r_i))}{\sum_{k=1} N_R \exp(s(\mathbf{m}_r, r_k))}$$

Trained representation for each relatipn r_i out of a fixed set of relations (e.g. rdf predicates)

23

By Semantic Parsing

- Map question to logical form (e.g. SQL, FOL, ...)

With Language Models

- Pretraining (fill in masked spans) + Fine-tuning on QA dataset

---

Chapter 15 - Chatbots & Dialogue Systems

Task-oriented agents → Siri, Alexa, ... **vs** **Chatbots** (extended, casual conversations → ChatGPT, Eliza, ...

Eliza

- rule-based via keywords + memory stack to keep conversation
  - e.g. (* YOU * ME) [pattern] → (WHAT MAKES YOU THINK I 3 YOU) [transform]
- main contribution: : transformations of input sentences (P**sycho-therapy**)

Properties of Human Conversation

- **Turns** & turn-taking patterns
- **Speech acts**:
  - Constatives - *answering, claiming, ...*
  - Directives - *advising, asking, ...*
  - Commissives - *promissing, planning, ...*
  - Acknowledgments - *apologizing, greeting, ...*
- **Grounding**: acknowledge understanding (*"Right."*; *"OK"*)
- Subdialogues and Dialogue Structure:

- clarification question, presequences
- Initiative
- Inference and Implicature

Corpus-Based Chatbots

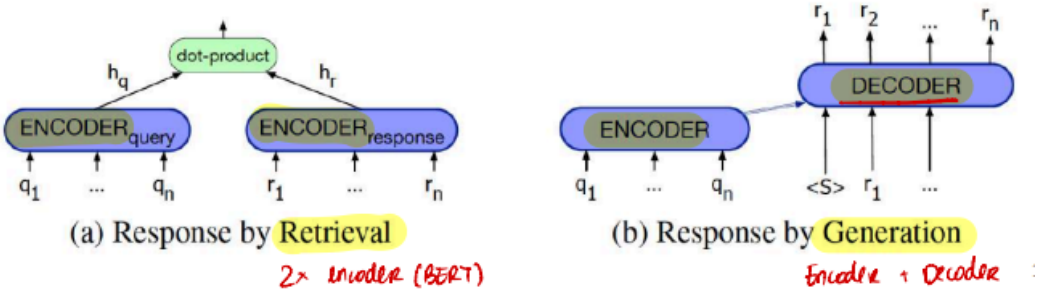
- **Retrieval-based:** Select most probable response from set of predefined responses

$$\text{response}(q,C) = \underset{r \in C}{\text{argmax}} \frac{q \cdot r}{|q||r|}$$

$$\begin{aligned} h_q &= \text{BERT}_Q(\text{q})[\text{CLS}] \\ h_r &= \text{BERT}_R(\text{r})[\text{CLS}] \\ \text{response}(q,C) &= \underset{r \in C}{\text{argmax}}\, h_q \cdot h_r \end{aligned}$$

- **Generation-based:** Generate response from scratch, via stat. patterns learned from corpus

$$\hat{r}_t = \text{argmax}_{w \in V}\, \underbrace{P(w|q, r_1 \ldots r_{t-1})}_{\text{constructed by the decoder}}$$



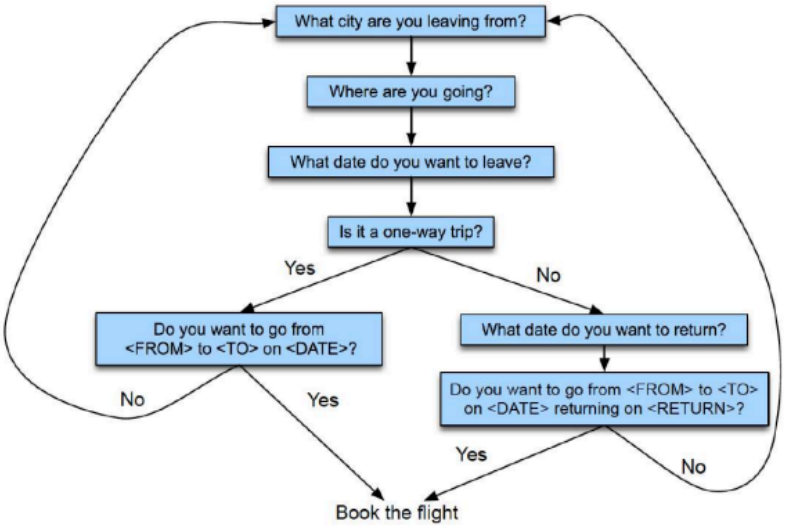
Hybrid Architectures

- combine neural approaches (e.g. Alexa, Siri)
- use **rule-based** if available, else switch to **NN-corpus-based** conversation

Simple Frame-Based Dialogue Systems

e.g. **GUS architecture** → dialogue state (state machine), frames, slots + values, domain ontology

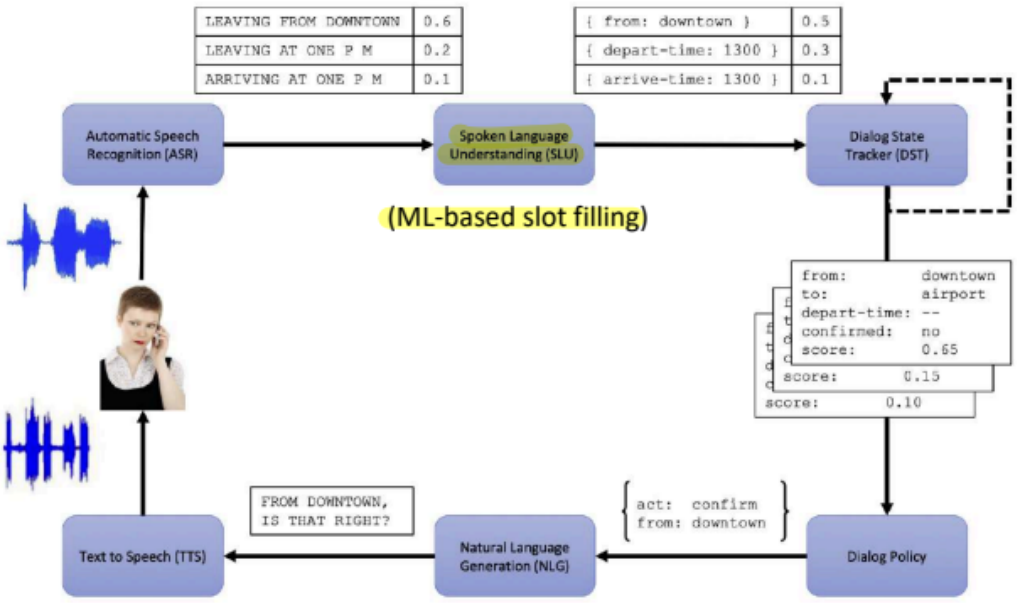
- GUS (also Siri, Alexa etc.): control structure ↔ slot filling in frames
- GUS: **production rules** determine state transitions



- Determine
  - **Domain** (→ classifier),
  - **Intent** (→ classifier),
  - **Slot-Filling** (with hand-written rules - RegEx and Semantic grammar (e.g. ORIGIN → from CITY)

```
Wake me tomorrow at 6
⇨ DOMAIN:  ALARM-CLOCK
  INTENT:  SET-ALARM
  TIME:    2017-07-01 0600-0800
```

Dialogue State Architecture



Dialogue Acts

- Function of turn/sentence (e.g. request(*a=x*, *b=x*))

NN Slot Filling

- Map input words with BERT to embeddings, do sequence labeling → determine BIO-tags

Dialogue State Tracking

- determine current **slot filling status** and user's **dialogue act**
- state tracking and slot filling: can be done jointly
- special case: detecting (user) correction acts

Dialogue Policy

- Decide what dialogue act to generate next

$$\hat{A}_i = \underset{A_i \in A}{\operatorname{argmax}} P(A_i | \text{Frame}_{i-1}, A_{i-1}, U_{i-1})$$

  possible approach: reinforcement learning

- Example policy: confirmation and rejection
  - explicit confirmation

- implicit confirmation
- rejection: progressive prompting
- rejection: rapid re-prompting
- confidence based

Dialogue System Evaluation

- Relative method: Compute 2 chats, rate which is better

$$\text{Slot Error Rate for a Sentence} = \frac{\text{\# of inserted/deleted/subsituted slots}}{\text{\# of total reference slots for sentence}}$$

Chapter 17 - Context-Free Grammars and Constituency Parsing

Constituents - groups of words behaving as a single unit

example: noun phrases → *"Harry the Horse"*

Evidence that these are constituents:

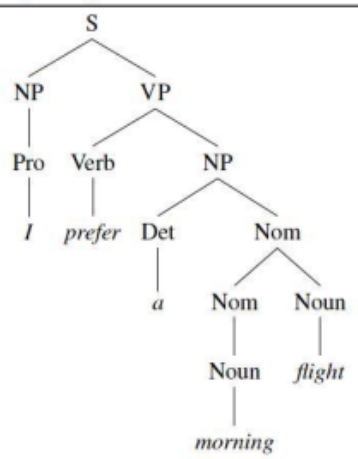
- **can appear together** before verbs while **parts thereof cannot**
- **different positions possible** (e.g. propositional phrase) while parts thereof cannot appear in different positions

Context Free Grammars

Noun →	*flights* | *breeze* | *trip* | *morning* | *flight*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective →	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun →	*me* | *I* | *you* | *it*
Proper-Noun →	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner →	*the* | *a* | *an* | *this* | *these* | *that*
Preposition →	*from* | *to* | *on* | *near*
Conjunction →	*and* | *or* | *but*

The lexicon for $\mathscr{L}_0$.

simple language $\mathcal{L}_0$
for flight booking



[$_S$ [$_{NP}$ [$_{Pro}$ I]] [$_{VP}$ [$_V$ prefer] [$_{NP}$ [$_{Det}$ a] [$_{Nom}$ [$_N$ morning] [$_{Nom}$ [$_N$ flight]]]]]]

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

The grammar for $\mathscr{L}_0$, with example phrases for each rule.

- Context Free (Phrase Structure) Grammar is 4-tuple of

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$, where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$
S	a designated **start symbol** and a member of N

here: terminal symbols are words

Notation:

Capital letters like A, B, and S	Non-terminals
S	The start symbol
Lower-case Greek letters like α, β, and γ	Strings drawn from $(\Sigma \cup N)*$
Lower-case Roman letters like u, v, and w	Strings of terminals

Syntactic Parsing

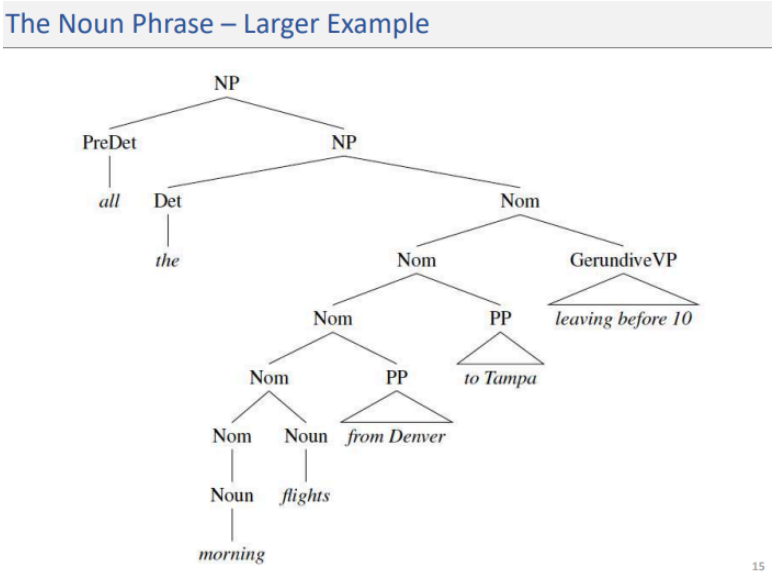
→ given sequence ("string") of words & CF grammar: construct **parse tree**

Examples

- declarative sentences - *S → NP VP*
- imperative sentences - *S → VP*
- yes-no-questions - *S → Aux NP VP*
- Wh-subject-questions - *S → Wh-NP VP*
- etc.

The Noun Phrase

- Proper Nouns, pronouns, *NP → Det Nominal* constructions
- Determiners:
  - Lexical determiners: *a, the, this*, ...
  - Mass nouns / plural nouns: no determiner needed
- Nominal:
  - Simple noun or with pre-head modifiers: cardinal numbers, ordinal numbers, quantifiers, adjectives
  - Post-head modifiers:
    - Prepositional phrase (PP)
    - Non-finite postmodifiers:
      - Gerundive (-ing) form
      - ed form
      - Infinitive form
    - Postnominal relative clause
- Predeterminers: possible before the whole noun phrase



The Verb Phrase

- Example: simple rules
  - *VP → Verb*
  - *VP → Verb NP*
  - *VP → Verb NP PP*
- Example: sentences as complements for verbs (sentential complements)
  - *VP → Verb S*
- Also possible: Verb Phrases (VPs) as complements for verbs
- Subcategorization of verbs:
  - Old approach: transitive verbs, intransitive verbs
  - Modern approach: over 100 **subcategorization frames** (complements) - not every verb supports all frames

Coordinations

Conjoin major phrase types (noun phrases, verb phrases etc.) with **conjunctions** (and, or, but, ...) → coordinate phrases

- *NP → NP and NP* — the flights and the costs
- *Nominal → Nominal and Nominal* — flights and costs
- etc..

Treebanks

- Corpora with parse trees for each sentence. Example: Penn Treebank
  - problem some whole treebank: 17500 rule types

Heads

- each constituent is associated with a lexical **head** (word)
  - For a *NP* it is a *N*, for a *VP* it is a *V*
- Finding heads, if heads are not explicitly specified in rules: nontrivial: parse sentence and apply **head rules**

Grammar Equivalence and Normal Form

- two grammars are **weakly equivalent** if they generate the **same language**
- two grammars are **strongly equivalent** if they generate the **same language and the same phrase structure**

Chomsky Normal Form (CNF)

- if it is **ε-free** and every rule is of the form $A \rightarrow BC$ or $A \rightarrow a$.

$$\left.A \rightarrow B\ C\ D\right\} \Longrightarrow \begin{array}{l} A \rightarrow B\ X \\ X \rightarrow C\ D \end{array}$$

→ Problem:

- Context Free (CF) grammars may have **too little an emphasis on the lexicon** and may have **too many actual non-terminal** productions.

Combinatory Categorial Grammars

- define $\mathscr{C}$, a set of categories for a grammar
  - $\mathscr{A} \subseteq \mathscr{C}$, where $\mathscr{A}$ is a given set of atomic elements (e.g. NP)
  - $(X/Y), (X\backslash Y) \in \mathscr{C}$, if $X, Y \in \mathscr{C}$

- Rules:

  X expects Y on the righ

  $X/Y \;\; Y \;\Rightarrow\; X$ $\quad$ X/Y : function that seeks its argument to the right

  $Y \;\; X\backslash Y \;\Rightarrow\; X$ $\quad$ $X\backslash Y$: function that seeks its argument to the left

  X expects Y on the left

- Lexicon: assignment of words to categories

flight :	N	
Miami :	NP	
cancel :	$(S\backslash NP)/NP$	transitive verb (one object)
give :	$((S\backslash NP)/NP)/NP$	bitransitive verb (two objects)

  - example:

United	*serves*	*Miami*
NP	(S\NP)/NP	NP
	S\NP (>)	
S (<)		

- coordination (meta-)rule : $X \;\; CONJ \;\; X \;\Rightarrow\; X$ $\quad$ <Φ>

We	*flew*	*to*	*Geneva*	*and*	*drove*	*to*	*Chamonix*
NP	(S\NP)/PP	PP/NP	NP	CONJ	(S\NP)/PP	PP/NP	NP
		PP (>)				PP (>)	
	S\NP (>)				S\NP (>)		
	S\NP (<Φ>)						
S (<)							

- more expressive power than CF by:
  - function composition: $X/Y \; Y/Z \Rightarrow X/Z$
    $Y\backslash Z \; X\backslash Y \Rightarrow X\backslash Z$
  - type raising: $X \Rightarrow T/(T\backslash X)$ with T any category
    $X \Rightarrow T\backslash(T/X)$

- example for type raising (T) and function composition (B):
  type-raise NP: $NP \Rightarrow S/(S\backslash NP)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\textit{United}}{\mathrm{NP}}\;\;\dfrac{\textit{serves}}{(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}}}{\mathrm{S}/(\mathrm{S}\backslash\mathrm{NP})}\,{>}\mathbf{T}\;\;\dfrac{\textit{Miami}}{\mathrm{NP}}}{\mathrm{S}/\mathrm{NP}}\,{>}\mathbf{B}}{\mathrm{S}}\,{>}$$

→ left to right parsing (human style) is possible

→ *United serves* as S/NP (not normally considered a constituent)

- from previous slide: type raising allows to derive constituents not normally considered a constituent *United serves* as S/NP

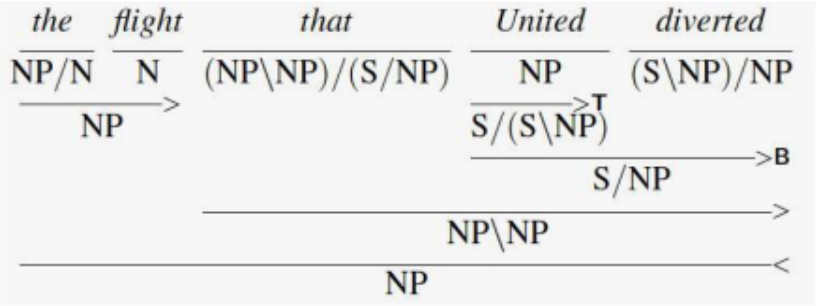
$$\dfrac{\dfrac{\dfrac{\textit{flew}}{(\mathrm{VP/PP})/\mathrm{NP}}\;\;\dfrac{\textit{IcelandAir}}{\mathrm{NP}}}{\mathrm{VP/PP}}\,{>}\;\;\dfrac{\dfrac{\textit{to}}{\mathrm{PP/NP}}\;\;\dfrac{\textit{Geneva}}{\mathrm{NP}}}{\mathrm{PP}}\,{>}}{\mathrm{VP}}\,{>}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{flew}}{(VP/PP)/NP}\;\;\dfrac{\text{IcelandAir}}{NP}}{(VP/PP)\backslash((VP/PP)/NP)}\,{<}\mathbf{T}\;\;\dfrac{\dfrac{\text{to}}{PP/NP}\;\;\dfrac{\text{Geneva}}{NP}}{PP}\,{>}}{VP\backslash(VP/PP)}\,{<}\mathbf{T}}{VP\backslash((VP/PP)/NP)}\,{<}\mathbf{B}}{VP}\,{<}$$

- now possible: coordinate such constituents:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textit{flew}}{(VP/PP)/NP}\;\;\dfrac{\dfrac{\textit{IcelandAir}}{NP}}{(VP/PP)\backslash((VP/PP)/NP)}\,{<}\mathbf{T}\;\;\dfrac{\dfrac{\textit{to}}{PP/NP}\;\;\dfrac{\textit{Geneva}}{NP}}{PP}\,{>}}{VP\backslash(VP/PP)}\,{<}\mathbf{T}}{VP\backslash((VP/PP)/NP)}\,{<}\;\;\dfrac{\textit{and}}{CONJ}\;\;\dfrac{\dfrac{\dfrac{\textit{SwissAir}}{NP}}{(VP/PP)\backslash((VP/PP)/NP)}\,{<}\mathbf{T}\;\;\dfrac{\dfrac{\textit{to}}{PP/NP}\;\;\dfrac{\textit{London}}{NP}}{PP}\,{>}}{VP\backslash(VP/PP)}\,{<}\mathbf{T}}{VP\backslash((VP/PP)/NP)}\,{<}}{VP\backslash((VP/PP)/NP)}\,{<}\Phi{>}}{VP}\,{<}$$

- modelling of long range dependencies is possible



(NP\NP) / (S/NP):
seeks a sentence missing an (NP)-argument to its right,
and transforms it into an NP seeking a missing (NP)-element to its left

---

Syntactic Parsing

- sentence → parse tree
- applications:
  - grammar checking
  - information extraction + question answering
- classic "conflict":
  - **symbolic reasoning** (syntactic + semantic parsing + rules, logic) **vs**
  - **sub-symbolic reasoning** (deep learning)

Ambiguity

→ assign **more than one parse tree** to a sentence

- **attachment ambiguity**: a particular constituent can be attached to the parse tree at more than one place
  - *"I shot an elephant in my pajamas"* — was I in my pajamas or was the elephant in my pajamas
- **coordination ambiguity**: join different sets of phrases by and
  - [old [men and women]] [old men] and [women]

Solution: integrate **statistical info** into parsers to produce most plausible parse tree

---

CYK Algorithm

→ for parsing CF grammars

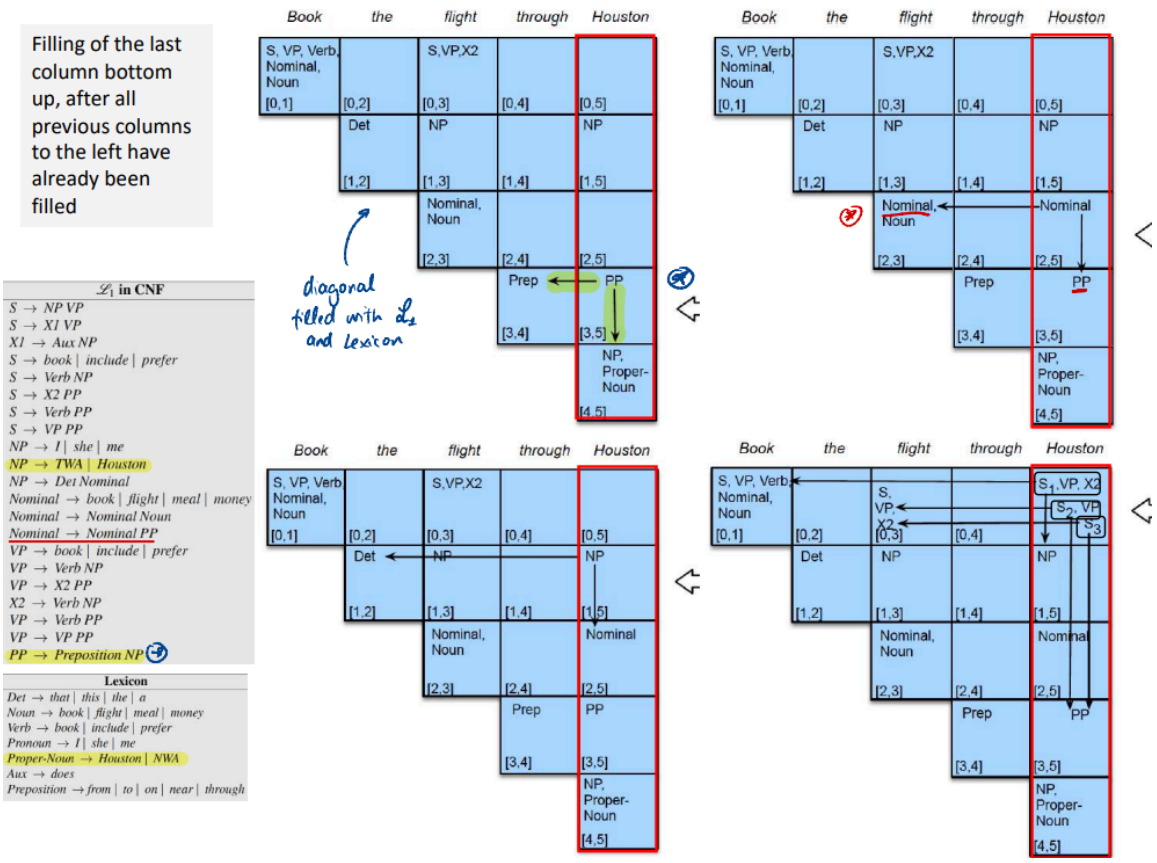
- dynamic programming
- two parts: recognizer + actual parser
- typically: convert to Chomsky Normal Form (CNF) first

Steps:

1. Convert into **Chomsky Normal Form (CNF)** — **only *A → BC, A → w***
   a. **eliminate terminals** from non-pure **right hand sides**
      i. *INF-VP → to VP* - transform in - *INF-VP → TO VP, TO → to*
   b. **eliminate unit productions** *A → B*
   c. iteratively **shorten long productions**
2. **CYK Recognizer** - build table with all possibilities
   a. organize possible rule applications into upper triangle
   b. first super-diagonal (cells [0,1], [1,2], [2,3], ..., [n-1,n]) contain all possible POS of the words of the sentence. (words are the terminals)
   c. other than first super-diagonal: cell [i,j] contains the set of all nonterminals that represent all possible constituents spanning positions *i* through *j* of the sentence

- **work** for each cell (**bottom-up** and **left to right**) from first super diagonal to ultimately the right upper corner (cell [0,n]):
  for each current cell [i,j]: fill it with all A from rules A → B C where B ∈ cell [i,k] (to the left) and C ∈ cell [k,j] (below) for all possible k (i<k<j).
  (for actual parser: also store k, B, C together with all possible multiple occurrences A_n of A in cell [i,j])

Example:



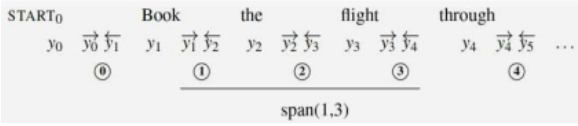
3. **CYK Parser**
   a. trace back all possible rules (productions) *S* → *B C* that led to this Si
   b. For every instance Si of S in cell [0,n] we may get an **exponential number of possible parse trees**
      i. incorporate probabilities
      ii. beam search

Span-Based Neural Constituency Parsing

- CYK produces a lot of possible trees → disambiguate? → (span-based) neural approaches
- neural span-based constituency parsing **(neural CYK)**: assign **scores** to possible trees
  - input: [START] + words + [END] → Encoder → Map back to words + postprocessing layers → one output y per word

score for each labeled constituent span $s(i,j,l)$

$$v(i,j) = [\overrightarrow{y_j} - \overrightarrow{y_i} ; \overleftarrow{y_{j+1}} - \overleftarrow{y_{i+1}}]$$



- integrate labelled span-scores s(i,j,l) into parse tree

$$T = \{(i_t, j_t, l_t) : t = 1, \ldots, |T|\}$$

$$s(T) = \sum_{(i,j,l) \in T} s(i,j,l)$$

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, s(T)$$

- to actually compute argmax:
  - use **greedy**: choose best scoring span each (these spans might not correctly "combine" to a parse tree)
  - or use CYK algorithm idea
- Evaluation

$$\left.\begin{aligned}
\textbf{labeled recall:} &= \sum_s \frac{\#\text{ of correct constituents in hypothesis parse of } s}{\#\text{ of correct constituents in reference parse of } s} \\
\text{or } & \frac{\sum_s \#\text{ of correct constituents in hypothesis parse of } s}{\sum_s \#\text{ of correct constituents in reference parse of } s} \\
\textbf{labeled precision:} &= \sum_s \frac{\#\text{ of correct constituents in hypothesis parse of } s}{\#\text{ of total constituents in hypothesis parse of } s} \\
\text{or } & \frac{\sum_s \#\text{ of correct constituents in hypothesis parse of } s}{\sum_s \#\text{ of total constituents in hypothesis parse of } s}
\end{aligned}\right\} F_1 = \frac{2PR}{P+R}$$

- modern parsers: F1 > 0.9

Chunking (Partial Parsing)

→ identifying and classifying **flat, non-overlapping** segments of a sentence (chunks)

Criteria:

- chunks do not contain other chunks (not nested)
- boundaries: headword + pre-head modifiers

Classification

- sequence labeling (BIO notation) → 2n+1 features per word

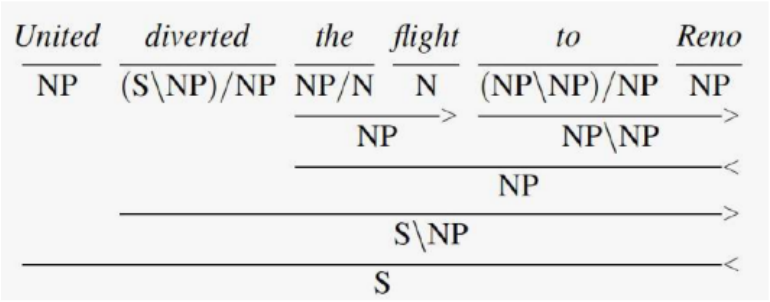
Ground Truth for Chunking

- concentrate on the non-terminals corresponding to the chunks
- b5oundary detection: find headword + all words preceding it

---

CCG Parsing

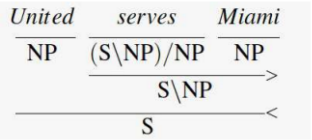
- ambiguity → choice of **lexical rule** determines parse



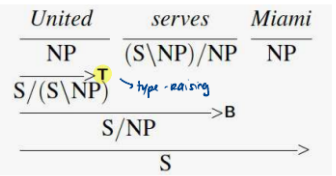
Repetition: Combinatory Categorial Grammars

- $X/Y \; Y \Rightarrow X$ — X/Y : function that seeks its argument to the right
  $Y \; X\backslash Y \Rightarrow X$ — $X\backslash Y$: function that seeks its argument to the left

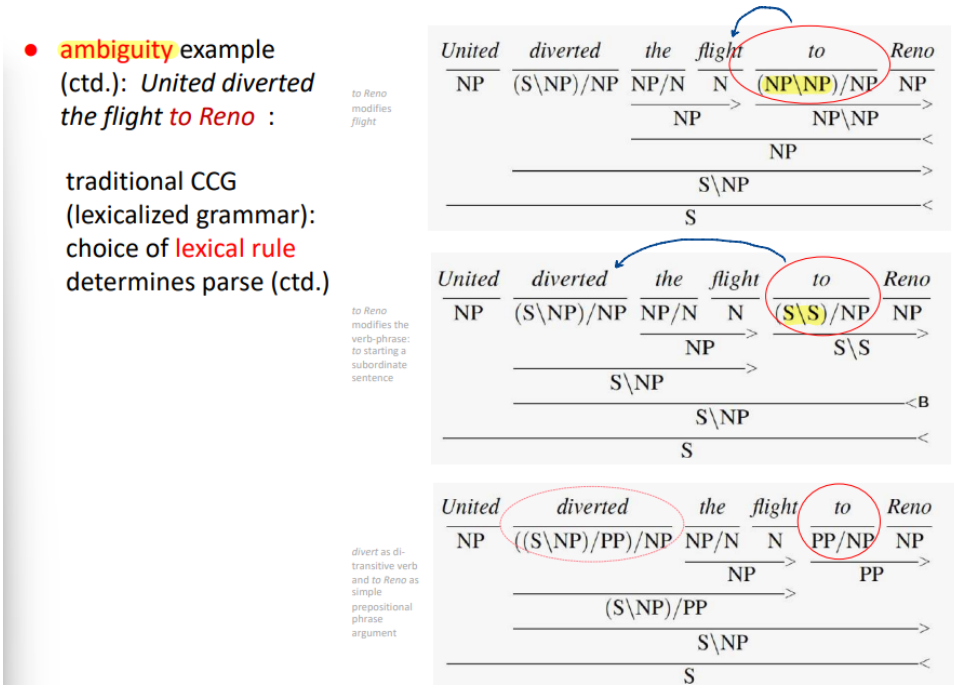
- Lexicon: $flight: N$, $Miami: NP$, $cancel: (S\backslash NP)/NP$ (transitive verb (one object))



- function composition B: $X/Y \; Y/Z \Rightarrow X/Z$, $Y\backslash Z \; X\backslash Y \Rightarrow X\backslash Z$
- type raising T: $X \Rightarrow T/(T\backslash X)$, $X \Rightarrow T\backslash(T/X)$ with T any category



- Ambiguity example



Supertagging

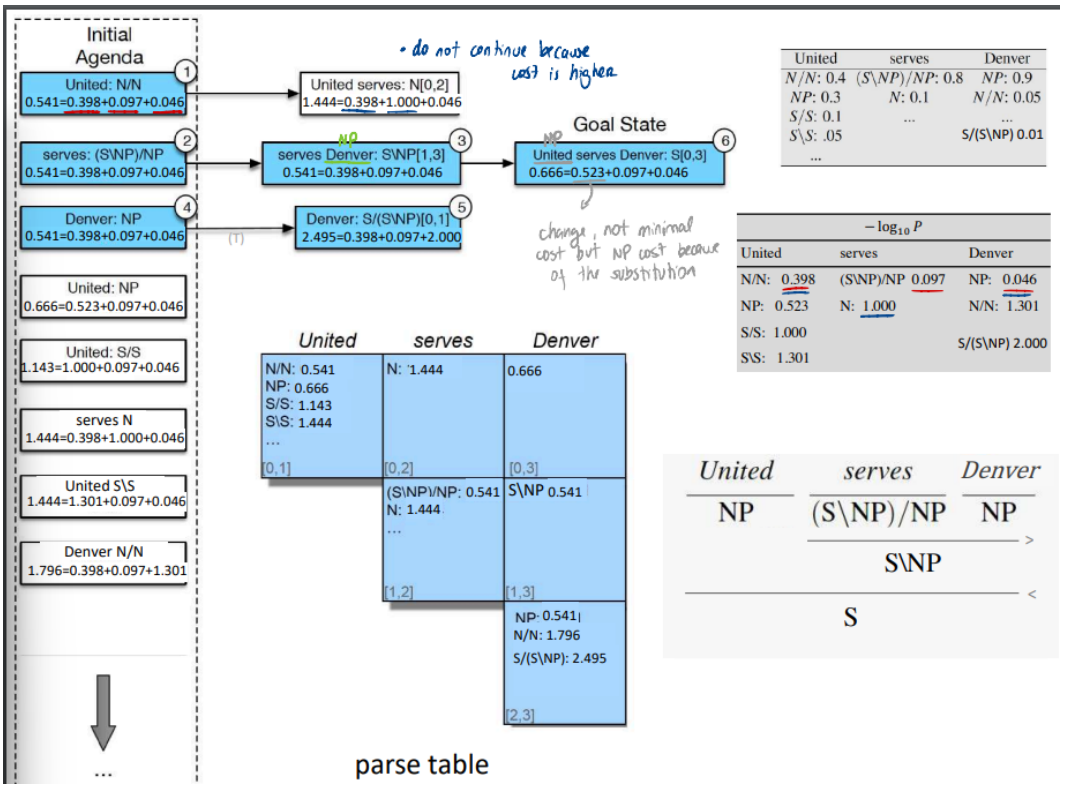
- exploit only the **most likely lexical categories** instead of every possible lexical catergory → less combinatorial explosion
- apply heuristic search algorithms (e.g. A*)

Leveraging A* Algorithm

- A*-algorithm: complete and optimal, informed search algorithm
- iteratively choose next state n with **optimal estimated** cost **f(n)= g(n)+h(n)**
  , where **g(n) current actual cost** to n from start node and

  h(n) non-overestimating(!) estimate to target state from n
- How to compute heuristics f(n)?
  - probability of derivation *D* of a sentence *S* = product of probabilities of supertags *T*
  - take negative log → convert probability to a cost
- *f(n)* = *g(n)* + *h(n)* = inside cost + outside cost:a
  - **g(n) cost of current span** (sub-sequence ("sub-string") of words currently considered)

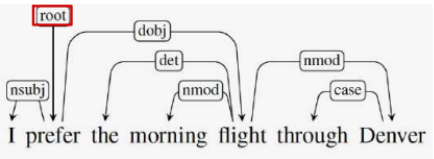
- **h(n) estimated cost for words outside of span**: assume that each of these outside words is assigned most probable supertag (→ never overestimate costs)



Chapter 18 - Dependency Parsing

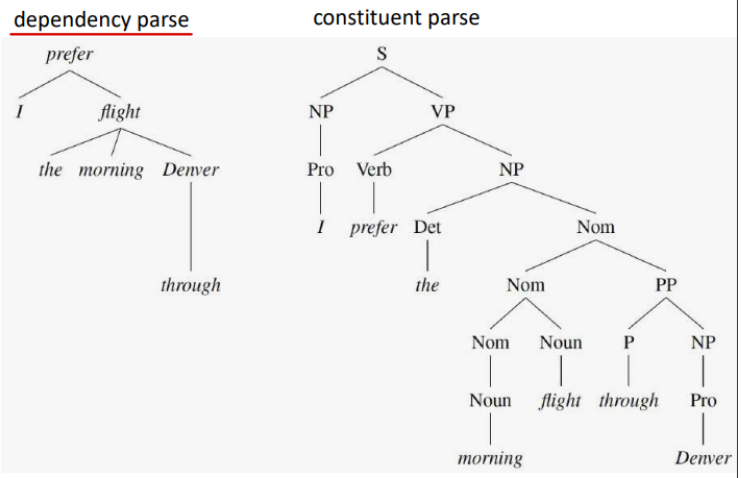
IDEA: **syntactic structure** of a sentence is described solely in terms of the

words + associated set of **directed binary relations** among the words



- relations: between head and dependent
- root: head of entire structure
- every major constituent has a head (e.g. flight, Denver)

simpler and more abstract



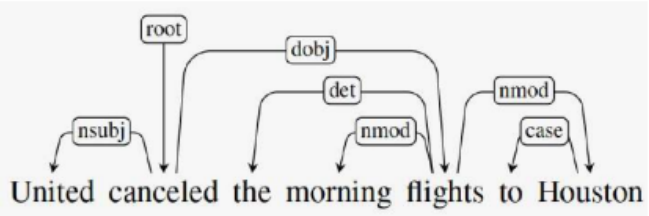
Advantages:

- DP abstracts away from word-order — only information that is necessary
- head-dependent relations → approximation to the semantic relationship

Dependency Relations

- **clausal relations** - describe syntactic roles with respect to a predicate (often a verb)
- **modifier relations** - categorize ways that words that can modify their heads

example:

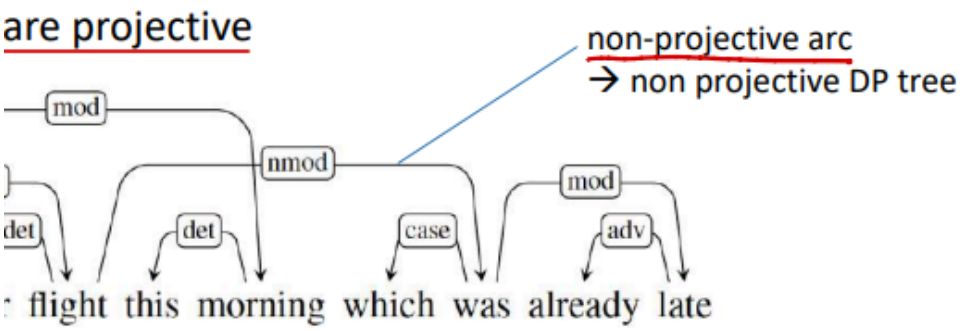


causal: NSUBJ, DOBJ
modifier: NMOD, DET, CASE

Dependency Trees and Projectivity

- DP: resulting graph of relations, often a **dependency tree**
  - single root node (no incoming arcs)
  - each vertex has exactly one incoming arc (except root)
  - unique path from the root node to each vertex

- **Projectivity**
  - arc(h,d) **projective** if path from h to every word w that lies between h and d.
  - **tree is projective if all arcs are projective**



  No connection between “flight” and “morning” or “this”

- **Translate constituent-based parse trees into DP trees**
  - identify **head** and **dependents**
  - mark head child of each node using head rules
  - classify arcs

Transition-Based Dependency Parsing

→ adaptation of Shift-Reduce

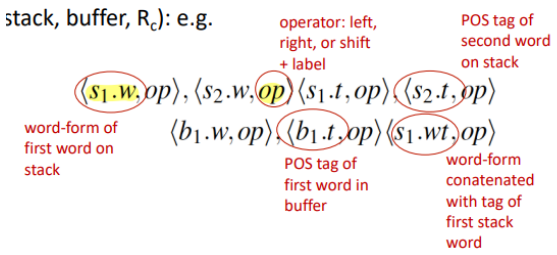
Action	Description	Result
Left Arc	Connects the **top of the stack** to the **word directly beneath it**	The **lower word** is **removed** from the stack
Right Arc	Connects the **second word** on the stack to the **word at the top**	The **top word is removed** from the stack
Shift	**Removes a word from the input buffer**	The **word is pushed onto the stack**

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

- **ambiguity**: there may be more than one path through the configuration space leading to same or different parse
- **greedy:** no going back if it makes the wrong decision
- produces **projective parse trees only:** any such parse tree for a sentence with a fundamentally non-projective DP-structure will contain errors.
- **<u>Oracle Training</u>**
  - with supervised ML — input: (features of) configuration, output: transition operator
  - create training data via simulating parsing
    - **choose left** arc if it produces correct-head dependent relation
    - otherwise **choose right** arc if correct and **all dependents of word at top of stack** assigned
    - else shift

Features

- **basic features** - word forms, lemmas, parts of speech, the head. ...
- **feature templates** for config (stack, buffer, R_c)



- example config of
  United canceled the morning flights to Houston :

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

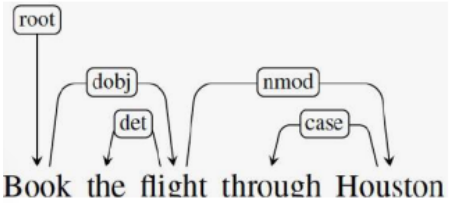
$$\langle s_1.w = flights, op = shift\rangle$$
$$\langle s_2.w = canceled, op = shift\rangle$$
$$\langle s_1.t = NNS, op = shift\rangle$$
$$\langle s_2.t = VBD, op = shift\rangle$$
$$\langle b_1.w = to, op = shift\rangle$$
$$\langle b_1.t = TO, op = shift\rangle$$
$$\langle s_1.wt = flightsNNS, op = shift\rangle$$
$$\langle \underbrace{s_1t.s_2t}_{s_1.t \circ s_2.t} = NNSVBD, op = shift\rangle$$

Alternative: Neural Classifier

- top 2 words on the stack and first word of the buffer
- represent them by their encodings
- concatenate the embeddings
- pass through a softmax to choose a parser action

Alternative Transition Systems

- **arc-eager** (”eager to apply RIGHTARC earlier”)



- LEFT ARC : Assert head-dependent relation between word at front of input buffer and word at top of stack; pop the stack.
- RIGHT ARC : Assert head-dependent relation between word on top of stack and word at front of input buffer; shift the word at front of input buffer to stack.
- SHIFT : Remove word from front of input buffer and push it to stack.
- REDUCE : Pop stack.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

Beam Search

- avoid greedy algo → bad choices cannot be undone

- keep **number of possible configs**
- core **new configs based on previous config and transition applied**

Graph-Based Dependency Parsing

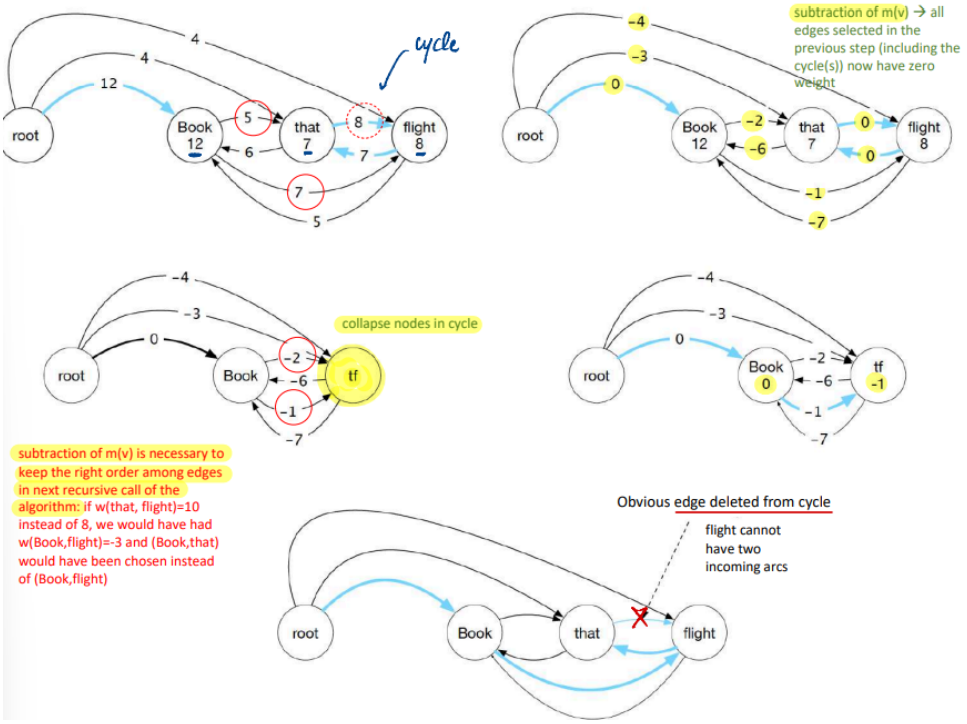
- Parsing a sentence S is search in the space of all possible parse tress for S
- score a tree: from score of tree edges(edge factored approach) — sum them
- Advantages:
  - non-projective tress possible
  - better long range dependencies

Maximum Spanning Tree Parsing

- fully (except root) connected graph
  - vertices → words + ROOT
  - edges: all possible (head, dependent) relations
  - edge weights: edge scores
- search for **maximum spanning tree in graph**

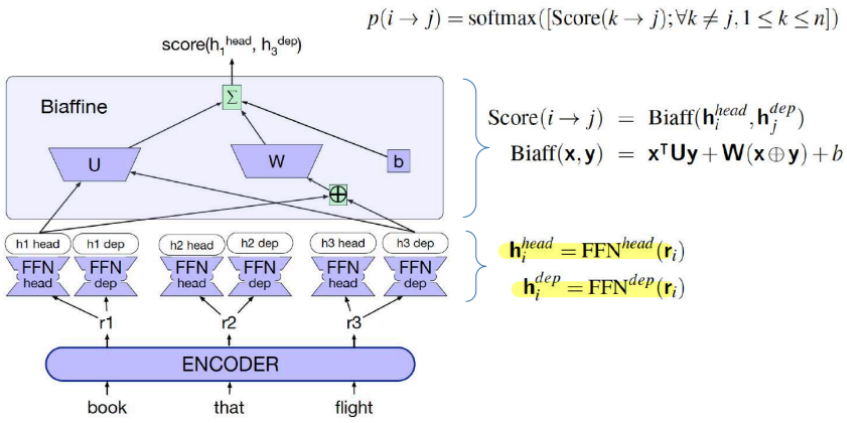
1. for each node v: choose incoming edge with max weight m(v)
2. if result is spanning tree (each node has one incoming edge and no cycles exist): done.
3. else: cycles exist: cleanup:
   - for each vertex v subtract m(v) from all incoming edge's weights. This does not change any MST (only relative choices among in-edges of a node matter)
   - collapse cycle into single (meta-)node

call process again recursively for cleansed graph → result: MST for cleansed graph; expand cycle (meta-)nodes back, breaking cycle by deleting obvious edge



NN for Scoring

words → encoder → contextual embeddings r_i → scores for r_i as a head and as a dep → score(i→j) → softmax



Metrics

- Accuracies: **Labeled Attachment Score** (LAS) or **Unlabeled Attachment Score** (UAS)

Chapter 19 - Logical Representations of Sentence Meaning

text → meaning representation

Desiderata

- **Verifiability**: compare state of affairs described by representation to the state of affairs as modeled in a knowledge base
- **Unambiguity**
- **Canonical form**: distinct inputs that mean the same thing should have the same meaning representation.
- Ability to **support inference and variables**
- **Sufficient Expressiveness**

Model-Theoretic Semantics

- **Model** → a formal construct that stands for the particular state of affairs in the world
  - expressions in a meaning representation leanguage can be mapped to elements in the model (”interpretation”)
    - logical or non-logical (predicate symbols, function symbols) → only one possible representation
- **Domain** → set of represented objects
- **Properties:** denote sets of elements of the domain
- **Relations** denote sets of tuples of elements of the domain

First Order Logic

- **Sound →** ($T \vdash F$ then $T \vDash F$) but **<u>not</u> complete** ($T \vDash F$ then $T \vdash F$)
  - *Note:* $\vdash$ syntatically derived , $\vDash$ logically derived
- Refutation complete
  - $T \wedge \neg F \vDash \square \rightarrow T \wedge \neg F \vdash \square$
- **Resolution calculus** (sound and resolution complete)

$$\frac{\begin{array}{l} C_1^{(1)} \vee C_2^{(1)} \vee \cdots \vee C_n^{(1)} \vee L \\ C_1^{(2)} \vee C_2^{(2)} \vee \cdots \vee C_m^{(2)} \vee \neg L' \end{array}}{\sigma_{L,L'}(C_1^{(1)} \vee C_2^{(1)} \vee \cdots \vee C_n^{(1)} \vee C_1^{(2)} \vee C_2^{(2)} \vee \cdots \vee C_m^{(2)})}$$

disjunction normal form

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y$

B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x, z)]$

C. $\forall x\ Animal(x) \Rightarrow Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosit$

E. $Cat(Tuna)$

F. $\forall x\ Cat(x) \Rightarrow Animal(x)$

$\neg$G. $\neg Kills(Curiosity, Tuna)$

apply negation to prove by contradiction

Eliminate impli
variables, skole

Beyond FOL

- **Second order logic**: allow for **quantifiers over predicates**
- **Modal logic:** represent statements about necessity (square) and possibility (diamond)

Event and State Representations

- **State**: holds for longer periods of time.
- **Event**: more point-like in time

Representations of Time

- **Temporal logic**: events ↔ points or intervals

<u>Reichenbach's reference point</u>

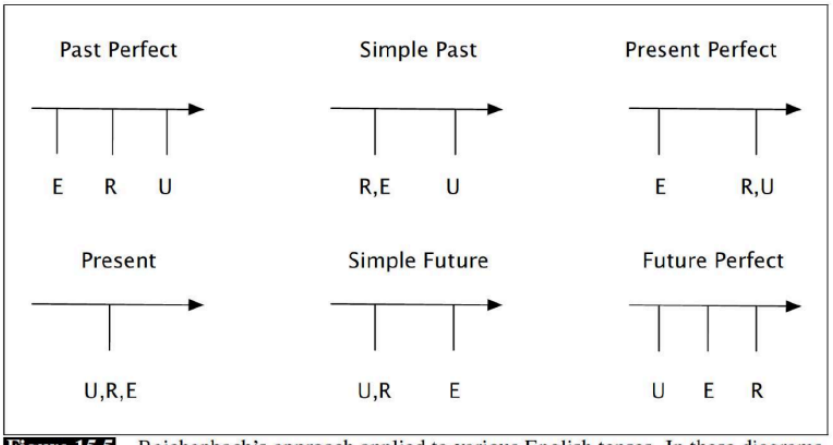


Figure 15.5 Reichenbach's approach applied to various English tenses. In these diagrams, time flows from left to right, **E** denotes the time of the event, **R** denotes the reference time, and **U** denotes the time of the utterance.

Description Logics → Ontologies

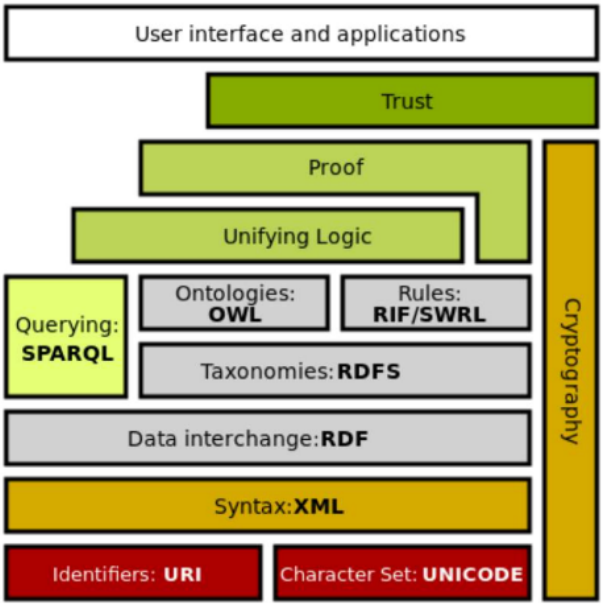
Representation of knowledge about **categories**, **individuals** that belong to those categories, the
relationships among these individuals → OWL, Semantic Web

- **T-Box** ("terminology"): category + relationship principal structure

  A-Box ("assertion"): info about individuals
- **Ontologies**: main relation between categories: subsumption:

  $C \sqsubseteq D$ (C is subsumed by D: every instance of C is also in D)
- **Disjointness** (⊑ **not**)
- **Coverage** (⊑ **or** x y)
- **Roles**
- Subsumption inference
- Instance checking

Semantic Web



Chapter 17 - Information Extraction

Relation Extraction

→ identify unary/binary relations

Databases

- the Semantic Web
  - RDF
  - DBPedia
- Wikipedia
- Freebase
- WordNet
  - hyponym hypernym relation
  - instance-of relation
- domain specific ontologies
  - example: Unified Medical Language System (UMLS)
- TACRED → hand-labeled

Relation Extraction Using Patterns

- **lexico-syntactic** patterns (**Hearst patterns**)
  - if: NP_0 *such as* NP_1
- patterns **using NE types**
  - PER, POSITION of ORG → “Mark, CEO of Google”

⟶ **Hight Precision, Low Recal**

Relation Extraction Using Supervised ML

- 3 sub-classifiers necessary
  - hand-labeling is costly

- **Neural Extraction →** replace subj and obj entities by their NER tag
- “Query” LMs

Semi-Supervised Relation Extraction via Bootstrapping

- learning new rules / patterns from seed rules or seed tuples

example:

- Ryanair has a hub at Charleroi → **seed**: hub(Ryanair,Charleoi)
- find other sentences with Ryanair, hub, Charleroi
- use context of words between entity mentions, words before mention one, word after mention two, NE types of the two mentions, ... →
  extract general patterns
- assign **confidence values** to new tuples to avoid **semantic drift**

Distant Supervision for Relation Extraction

1. relation databases to produce large number of seeds
2. find all sentences in T with two NEs matching one of those tuples
3. for each tuple (e.g. <born-in, Albert Einstein, Ulm>: from all sentences matching the tuple extract conjunctions of features

- very large training sets → rich features
- uses hand-created knowledges → high precision
- large numbers of features simultaneously → no semantic drift
- doesn’t use a labeled training corpus of texts directly → no genre bias

---

Event Extraction

- most events: verbs / VP; also possible: NP
- **classes of events**: actions, states, reporting events (say, report, tell, explain),
  perception events etc.
  and further

sub-classes: example: (class=REPORTING, tense=PAST, aspect=PERFECTIVE)

- approach: supervised ML with IOB tagging

Template Filling

- **template filling task**: detect presence of template and fill slots with slot filler values
- **template recognition** - supervised text classification task
- **role filler extraction** - detect and fill roles

Chapter 22 - Time and Temporal Reasoning

- Temporal logic: events ↔ points or intervals

Allen interval algebra

precedes	meets	overlaps	finished by	contains	starts	equals	started by	during	finishes	overlap-ped by	met by	preceded by
p	m	o	F	D	s	e	S	d	f	O	M	P

Table 1. Allen's thirteen basic relations

Reichenbach‘s Reference Point Approach

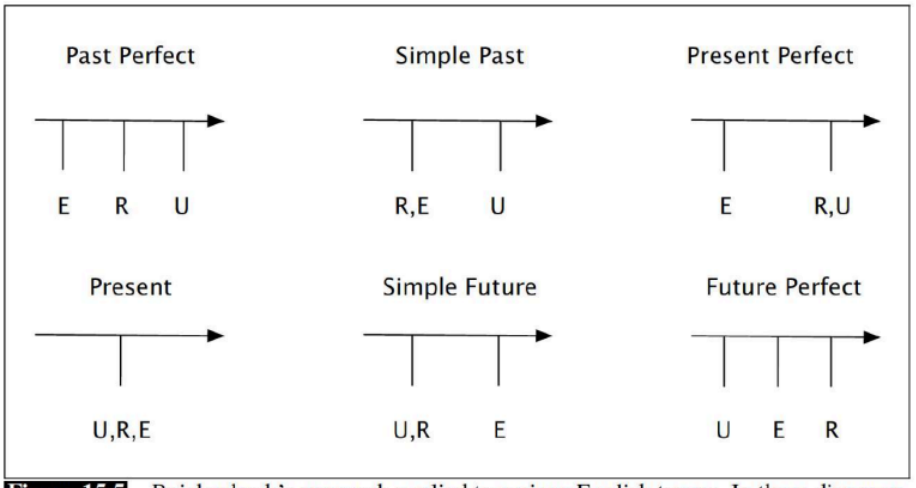


Figure 15.5 Reichenbach's approach applied to various English tenses. In these diagrams, time flows from left to right, **E** denotes the time of the event, **R** denotes the reference time, and **U** denotes the time of the utterance.

Aspects

- **Events** (involves change) vs **states** (do not involve change)
- Stative expressions: state a point in time
- Activity expressions: activity (event) over interval in time
- Accomplishment expressions: interval has definitive end-point
- Achievement expressions: point in time, results in state change

Time Bank Corpus

Time ML:

- EVENT: Refers to events and states.
- TIME: Denotes time points.
- TLINKS: Represents Allen relations between time points.
- ALINKS: Describes aspectual relationships between Events and subevents.
- SLINKS: Indicates facts.

(22.25) **[DCT:11/02/891]**$_1$: Pacific First Financial Corp. **said**$_2$ shareholders **approved**$_3$ its **acquisition**$_4$ by Royal Trustco Ltd. of Toronto for $27 a share, or $212 million. The thrift holding company **said**$_5$ it **expects**$_6$ to **obtain**$_7$ regulatory **approval**$_8$ and **complete**$_9$ the **transaction**$_{10}$ by **year-end**$_{11}$.

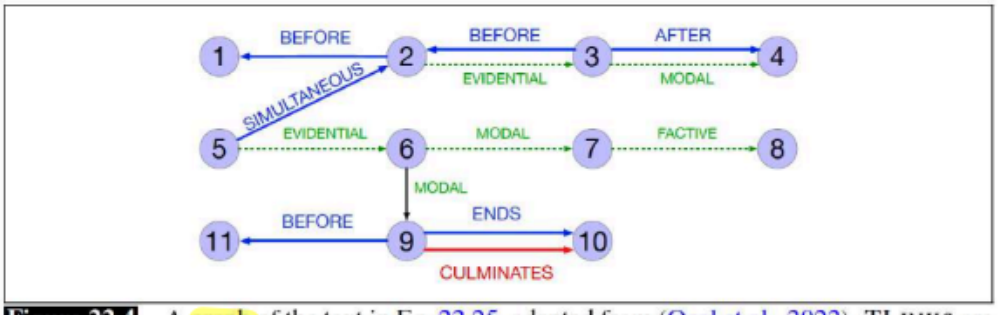


Figure 22.4 A graph of the text in Eq. 22.25, adapted from (Ocal et al., 2022). TLINKS are shown in blue, ALINKS in red, and SLINKS in green.

Extracting Temporal Expressions

- Steps:

BIO sequence tag

usually not simple

1. Extracting **temporal expressions**
2. **Normalizing** these expressions, by converting them to a standard format.
3. **Linking** events to times and extracting time graphs and timelines

- **Absolute** points in time, **durations**, and **relations** between them
- **Temporal expressions**: grammatical constructions with temporal triggers as heads
- supervised sequence labelling with any ML seq. classifier using IOB
- challenge: false positives

Temporal Normalization

- map to **ISO8601 standard**
- Most approaches for temporal normalizations: **rule systems**
- logical temporal **arithmetic**
  - tomorrow = anchor for today + 1
- but: **complexity** of absolute referencing may be high

Temporal Ordering of Event

- **absolute positioning** of events in anchored timeline **or partial ordering** of events (after, before etc.) → Allen temporal logic relations

---

Chapter 18 - Word Senses and WordNet

Gloss - definition of a word sense

Criteria for deciding whether **differing uses** of a word should be represented as **distinct senses**

- independent truth condtions
- different synstatic behaviour
- independent sense relations
- etc.

Zeugma: decide problem “experimentally

WordNet

- database(s) of **lemmas** + **senses** and **relations**
- Supersenses (higher level categories) for synsets)

Word-Sense Disambiguation

WSD: maps words to sense tags.

- with **Contextual Embeddings** → Very simple, but **SOTA: 1-NN classifier with sense embeddings**
    - imputation of unseen senses "(of words s not present in sense-tagged corpus but present in WordNet):

$$\text{if} \quad |S_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|S_{\hat{s}}|} \sum \mathbf{v}_s, \forall \mathbf{v}_s \in S_{\hat{s}}$$

known sense embeddings of words in synset $\hat{s}$ of s

$$\text{else if} \quad |H_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|H_{\hat{s}}|} \sum \mathbf{v}_{syn}, \forall \mathbf{v}_{syn} \in H_{\hat{s}}$$

known sense embeddings of hyponym synsets of synset $\hat{s}$ of s

$$\text{else if} \quad |L_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|L_{\hat{s}}|} \sum \mathbf{v}_{syn}, \forall \mathbf{v}_{syn} \in L_{\hat{s}}$$

known sense embeddings of synsets that have the same super-sense as synset $\hat{s}$ of s

---

Dictionary and Thesaurus Methods

- **Simplified Lesk**: idea: choose word-sense, whose dictionary gloss or dictionary definition has highest word-overlap with context window of target word in sentence
- **Original Lesk**: choose word-sense, whose dictionary definition has highest word-overlap with dictionary definitions of other words in context window of target word in sentence
  - usually performs **worse** than Simplified Lesk