自然语言处理
week-2

凌震华

2024年3月7日

- Morphology (形态学)
- Edit Distance (编辑距离)
- Language Model (语言模型)

Morphology

- Morphology (形态学) is the study of the ways that words are built up from smaller meaningful units called morphemes (语素)
- We can usefully divide morphemes into two classes
  - Stems (词干): The core meaning-bearing units
  - Affixes (词缀): Bits and pieces that adhere to stems to change their meanings and grammatical functions

English Morphology

- We can also divide morphology up into two broad classes
  - Inflectional (屈折): affix doesn't change word-class (walk, walking)
  - Derivational (派生): meaning change, word class change (clue, clueless; compute, computerization)

Light Weight Morphology

- Sometimes you just need to know the stem of a word and you don't care about the structure.
- In fact you may not even care if you get the right stem, as long as you get a consistent string.
- This is stemming (词干还原)... it most often shows up in IR (Information Retrieval, 信息检索) applications

Stemming for Information Retrieval

- Run a stemmer on the documents to be indexed
- Run a stemmer on users' queries
- Match
  - This is basically a form of hashing, where you want collisions.

Porter Stemmer

- No lexicon needed
- Basically a set of staged sets of rewrite rules that strip suffixes (后缀)
- Handles both inflectional and derivational suffixes
- Doesn’t guarantee that the resulting stem is really a stem (see first bullet)
- Lack of guarantee doesn’t matter for IR

Porter Stemmer

- Example
  - Computerization
    - ization -> -ize computerize
    - ize -> ε computer

- Many open-source implementations in C, C++, Java, Perl, Python, etc on the web
  - http://tartarus.org/martin/PorterStemmer/

- Morphology (形态学)
- Edit Distance (编辑距离)
- Language Model (语言模型)

How similar are two strings?

- Spell correction
  - The user typed "graffe"

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides (核苷酸)

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

    -**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---
    T**AG**-**CTATCAC**--**GACC**GC**C**--**GG**T**CGA**TT**TGCCC**GAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- The minimum edit distance (最小编辑距离) between two strings
- Is the minimum number of editing operations
  - Insertion (插入)
  - Deletion (删除)
  - Substitution (替代)
- Needed to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment** (对齐):

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

Alignment in Computational Biology

- Given a sequence of bases

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- An alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

- Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

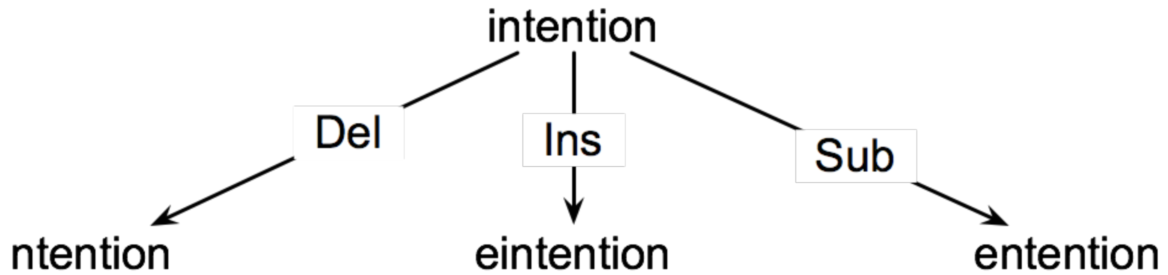
- Evaluating Machine Translation and speech recognition

```
R Spokesman confirms    senior government adviser was shot
H Spokesman said    the senior            adviser was shot dead
              S      I              D                        I
```

- Named Entity (命名实体) Extraction and Entity Coreference (指代)
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stanford University President John Hennessy

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we' re transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we' re trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Minimum Edit as Search

- But the space of all edit sequences is huge!
  - We can’t afford to navigate naïvely
  - Lots of distinct paths wind up at the same state.
    - We don’t have to keep track of all of them
    - Just the shortest path to each of those revisited states.

Defining Min Edit Distance

- For two strings
  - X of length n
  - Y of length m
- We define $D(i,j)$
  - the edit distance between $X[1..i]$ and $Y[1..j]$
    - i.e., the first i characters of X and the first j characters of Y
  - The edit distance between X and Y is thus $D(n,m)$

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming** (动态规划): A tabular computation of D(*n,m*)
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Defining Min Edit Distance (Levenshtein)

- Initialization

  D(i,0) = i

  D(0,j) = j

- Recurrence Relation:

  For each i = 1…M

  For each j = 1…N

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

- Termination:

  D(N,M) is distance

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Computing alignments

- Edit distance isn’t sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”(追踪)
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment

The Edit Distance Table

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

MinEdit with Backtrace

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**	
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9	
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10	
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11	
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10	
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8	
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7	
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8	
#	**0**	1	2	3	4	5	6	7	8	9	
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**	

Adding Backtrace to Minimum Edit Distance

- Base conditions:
  `D(i,0) = i        D(0,j) = j`

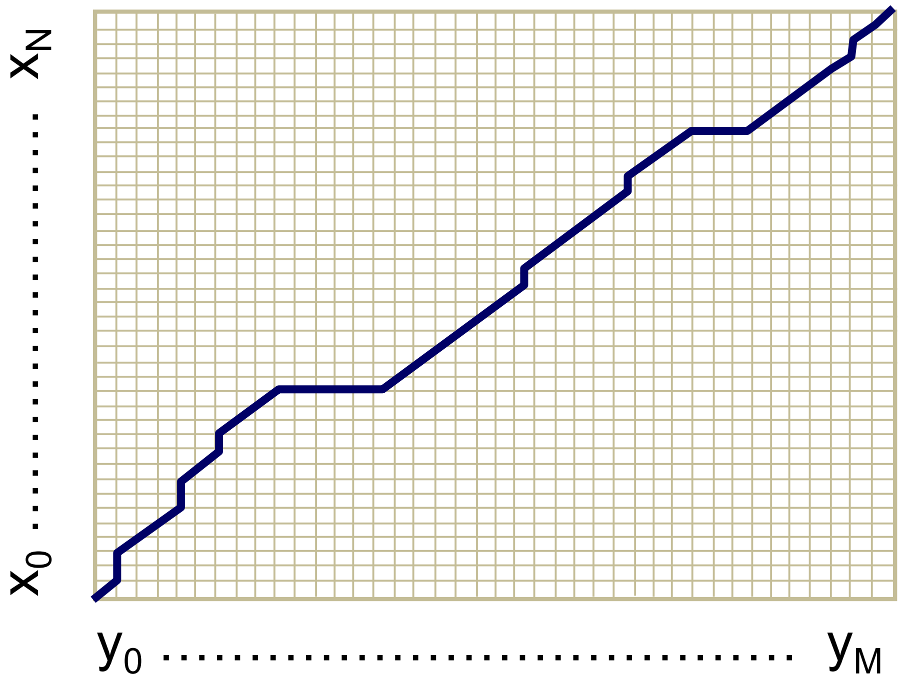
  Termination:
  `D(N,M) is distance`
- Recurrence Relation:

```
For each  i = 1…M
     For each  j = 1…N
```

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

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

The Distance Matrix



Every non-decreasing path

from (0,0) to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed
of optimal subalignments

Result of Backtrace

- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Performance

- Time:

  O(nm)

- Space:

  O(nm)

- Backtrace

  O(n+m)

Weighted Edit Distance

- Why would we add weights to the computation?
  - Spell Correction: some letters are more likely to be mistyped than others
  - Biology: certain kinds of deletions or insertions are more likely than others

Confusion matrix for spelling errors

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Position of letters on keyboard

Weighted Min Edit Distance

- Initialization:

```
D(0,0) = 0
D(i,0) = D(i-1,0) + del[x(i)];     1 < i ≤ N
D(0,j) = D(0,j-1) + ins[y(j)];     1 < j ≤ M
```

- Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) + del[x(i)] \\ D(i,j-1) + ins[y(j)] \\ D(i-1,j-1) + sub[x(i),y(j)] \end{cases}$$

- Termination:

```
D(N,M) is distance
```

- Morphology (形态学)
- Edit Distance (编辑距离)
- Language Model (语言模型)

Language Modeling

- We want to compute P(w1,w2,w3,w4,w5…wn), the probability of a sequence
- Alternatively we want to compute P(w5|w1,w2,w3,w4): the probability of a word given some previous words
- The model that computes P(W) or P(wn|w1,w2…wn-1) is called the language model (语言模型)

Computing P(W)

- How to compute this joint probability:
  - P(“the” ,” other” ,” day” ,” I” ,” was” ,” walking” ,” along” ,” and” ,” saw” ,” a” ,” lizard”)
- Intuition: let’ s rely on the Chain Rule of Probability

The Chain Rule

- Recall the definition of conditional probabilities

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

- Rewriting:

$$P(A \wedge B) = P(A \mid B)P(B)$$

- More generally

$$P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A,B)P(D \mid A,B,C)$$

- In general

$$P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1,x_2)\ldots P(x_n \mid x_1\ldots x_{n-1})$$

The Chain Rule

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

P(“the big red dog was”)=

P(the)*P(big|the)*P(red|the big)*P(dog|the big red)
*P(was|the big red dog)

Very Easy Estimate

- How to estimate?
  - P(you | the river is so wide that)

$$P(\text{you} \mid \text{the river is so wide that}) = \frac{\text{Count(the river is so wide that you)}}{\text{Count(the river is so wide that)}}$$

Very Easy Estimate

- According to Google those counts are 7/204.
  - Search for fixed strings "the river is so wide that" and "the river is so wide that you"

Unfortunately

- There are a lot of possible sentences
- In general, we'll never be able to get enough data to compute the statistics for those long prefixes
- P(lizard|the,other,day,I,was,walking,along,and, saw,a)

Markov Assumption

- Make the simplifying assumption
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)
- Or maybe
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)
- Or maybe… You get the idea.

Markov Assumption

So for each component in the product replace with the approximation (assuming a prefix of N)

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram (2元语法) version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating bigram probabilities

- The Maximum Likelihood (最大似然) Estimate

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

An example

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

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

Maximum Likelihood Estimates

- The maximum likelihood estimate of some parameter of a model M from a training set T
  - Is the estimate that maximizes the likelihood of the training set T given the model M
- Suppose the word *Chinese* occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be "Chinese"
- MLE estimate is 400/1000000 = .004
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that "Chinese" will occur 400 times in a million word corpus.

Berkeley Restaurant Project Sentences

- *can you tell me about any good cantonese restaurants close by*
- *mid priced thai food is what i’m looking for*
- *tell me about chez panisse*
- *can you give me a listing of the kinds of food that are available*
- *i’m looking for a good place to eat breakfast*
- *when is caffe venezia open during the day*

Raw Bigram Counts

- Out of 9222 sentences: Count(col | row)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw Bigram Probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	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

Bigram Estimates of Sentence Probabilities

- P(<s> I want english food </s>) =

p(i|<s>) x p(want|I) x p(english|want) x p(food|english) x p(</s>|food)

=.000031

Kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

- World knowledge
- Syntax
- Discourse

Sentence Generation

- Generate random sentences:
- Choose a random bigram <s>, w according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

<s> I
I want
want to
to eat
eat Chinese
Chinese food
food </s>

Shakespeare

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams: so, 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

The Wall Street Journal is Not Shakespeare

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Why?

- Why would anyone want the probability of a sequence of words?
- Typically because of

$$P(S \mid X) = \frac{P(X \mid S)P(S)}{P(X)}$$

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary (封闭词汇) task
- Often we don’ t know this
  - Open vocabulary (开放词汇) task
  - Out Of Vocabulary = OOV words (集外词)
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Evaluation

- We train parameters of our model on a **training set** (训练集)
- How do we evaluate how well our model works?
- We look at the models performance on some new data
- This is what happens in the real world; we want to know how our model performs on data we haven’ t seen
- So a **test set** (测试集). A dataset which is different than our training set

Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a speech recognizer
  - Run recognition, get word error rate (WER) for A
  - Put model B in speech recognition, get word error rate for B
  - Compare WER for A and B
  - **Extrinsic evaluation** (外在评测)

Difficulty of extrinsic evaluation of N-gram models

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic evaluation** (内在评测), an approximation called **perplexity** (困惑度)
  - But perplexity is a poor approximation unless the test data looks **just** like the training data

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

$$\begin{aligned}\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}\end{aligned}$$

- Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

- For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

A Different Perplexity Intuition

- How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9' : pretty easy
- How hard is recognizing (30,000) names at Microsoft. Hard: perplexity = 30,000
- Perplexity is the weighted equivalent branching factor provided by your model

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Lesson 1: overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models, adapt to test set, etc

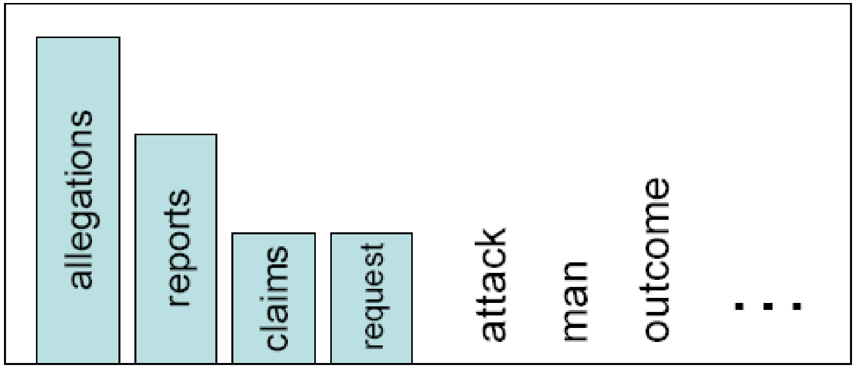
Lesson 2: zeros or not?

- Zipf's Law (齐夫定律)
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! no counts at all for the vast bulk of things we want to estimate!
  - Some of the zeroes in the table are really zeros But others are simply low frequency events you haven't seen yet. After all, **ANYTHING CAN HAPPEN**!
  - How to address?
- Answer:
  - Estimate the likelihood of unseen N-grams!

Smoothing (平滑) is like Robin Hood: Steal from the rich and give to the poor (in probability mass)

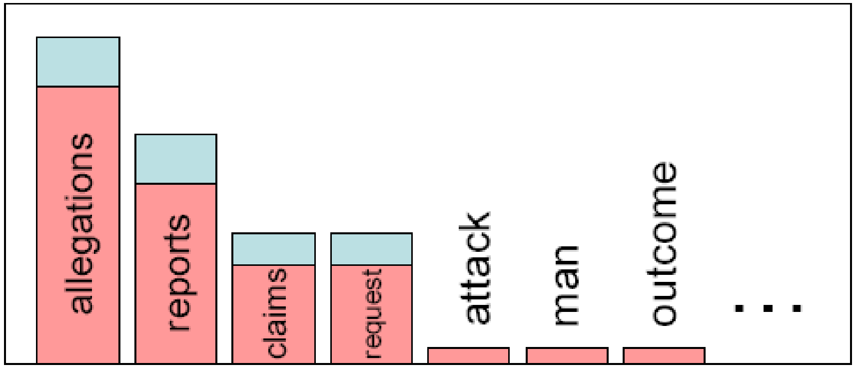
- We often want to make predictions from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Very important all over NLP, but easy to do badly!

Laplace smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple

- MLE estimate: $P(w_i) = \frac{c_i}{N}$

- Laplace estimate: $P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$

- Reconstructed counts: $c_i^* = (c_i + 1)\frac{N}{N + V}$

Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

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

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Big Changes to Counts

- C(want to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for "chinese food" =.10!!! A 10x reduction
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge.

Better Discounting Methods

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell
- Is to use the count of things we’ve seen once to help estimate the count of things we’ve never seen

Good-Turing

- Imagine you are fishing
  - There are 8 species: carp 鲤鱼, perch 河鲈, whitefish 白鱼, trout 鳟鱼, salmon 鲑鱼, eel 鳗鱼, catfish 鲶鱼, bass 鲈鱼

- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel

    = 18 fish (tokens)

    = 6 species (types)

Good-Turing

- Now how likely is it that next species is new (i.e. catfish or bass)

There were 18 distinct events... 3 of those represent singleton species

3/18

Good-Turing Intuition

- Notation: N_x is the frequency-of-frequency-x
  - So N_{10}=1, N_1=3, etc
- To estimate total number of unseen species
  - Use number of species (words) we've seen once
  - $c_0^* = c_1 N_1 / N_0$ $p_0 = N_1/N$
- All other estimates are adjusted (down) to give probabilities for unseen

$$c_x^* = (c_x + 1)\frac{N_{x+1}}{N_x}$$

Good-Turing Intuition

- Notation: N_x is the frequency-of-frequency-x
  - So $N_{10}=1$, $N_1=3$, etc
- To estimate total number of unseen species
  - Use number of species (words) we' ve seen once
  - $c_0^* = c_1 N_1/N_0$ $p_0 = N_1/N$ $p_0=N_1/N=3/18$

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

- All other estimates are adjusted (down) to give probabilities for unseen

$c_1^* = (1+1)\ 1/3 = 2/3$

$$c_x^* = (c_x + 1)\frac{N_{x+1}}{N_x}$$

Bigram frequencies of frequencies and GT re-estimates

AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Backoff (回退) and Interpolation (内插)

- Another really useful source of knowledge
- If we are estimating:
  - trigram $p(z|xy)$
  - but $c(xyz)$ is zero
- Use info from:
  - Bigram $p(z|y)$
- Or even:
  - Unigram $p(z)$
- How to combine the trigram/bigram/unigram info?
  - See Section 4.6/4.7

Backoff vs. Interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram

- **Interpolation**: mix all three

Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n)$$

How to set the lambdas?

- Use a **held-out** corpus
- Choose lambdas which maximize the probability of some held-out data
  - i.e. fix the N-gram probabilities
  - Then search for lambda values
  - That when plugged into previous equation
  - Give largest probability for held-out set
  - Can use EM to do this search

Backoff bigram probs

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don’t use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Language Modeling Toolkits

- SRILM
- CMU-Cambridge LM Toolkit
- These toolkits are publicly available
- Can use it to get N-gram models
- Lots of parameters (need to know the theory!)
- Standard N-gram format: ARPA language model (see Section 4.8)

Google N-Gram Release

August 2006

AUG

3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

Pros and cons of N-gram models

- Really easy to build, can train on billions and billions of words
- Smoothing helps generalize to new data
- Only work well for word prediction if the test corpus looks like the training corpus
- Only capture short distance context

"Smarter" LMs can address some of these issues, but they are order of magnitudes slower…

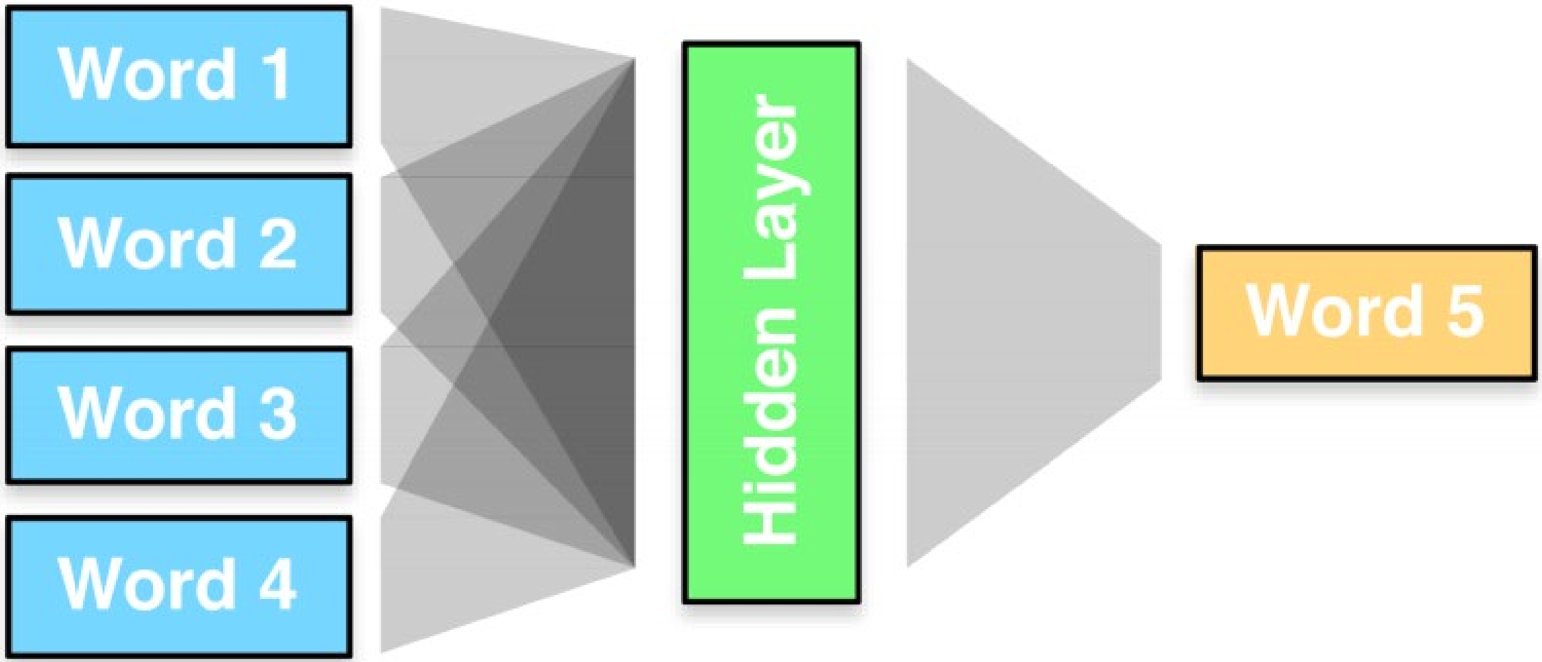
Neural Networks

- Non-linear classification

- Prediction: forward propagation
  Vector/matrix operations + non-linearities

- Training: backpropagation + stochastic gradient descent

Neural Language Model



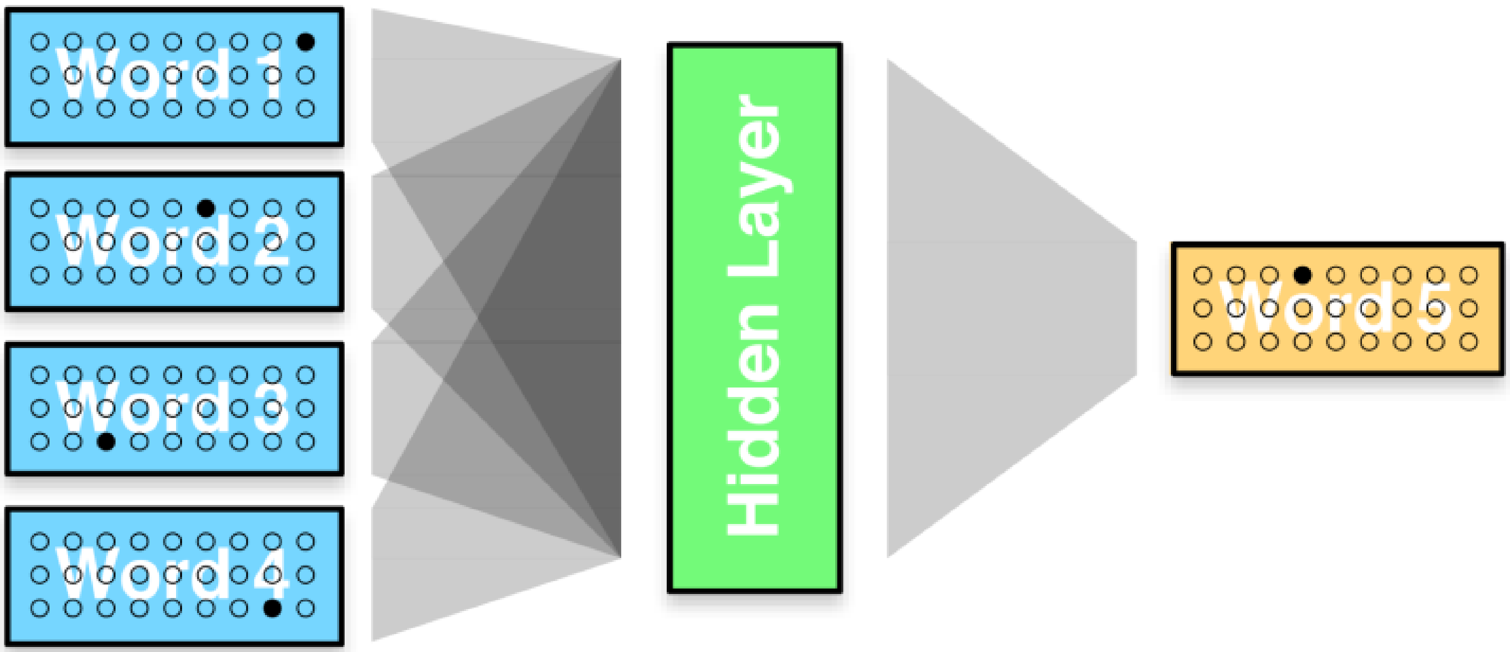
Representing Words

- "one hot vector"

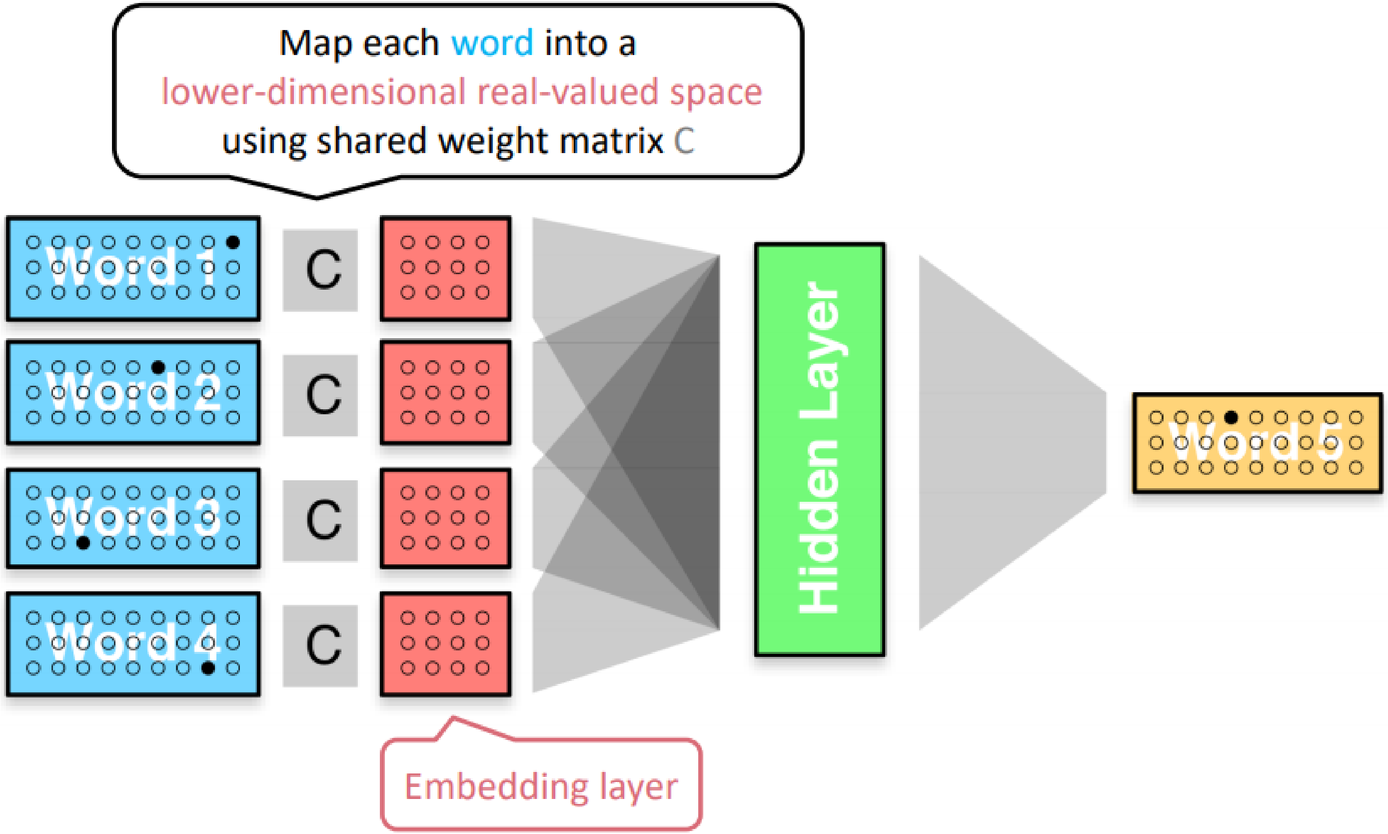
```
dog = [ 0, 0, 0, 0, 1, 0, 0, 0 …]
cat = [ 0, 0, 0, 0, 0, 0, 1, 0 …]
eat = [ 0, 1, 0, 0, 0, 0, 0, 0 …]
```

- That's a large vector! practical solutions:
  - limit to most frequent words (e.g., top 20000)
  - cluster words into classes
    - WordNet classes, frequency binning, etc.

Direct prediction model

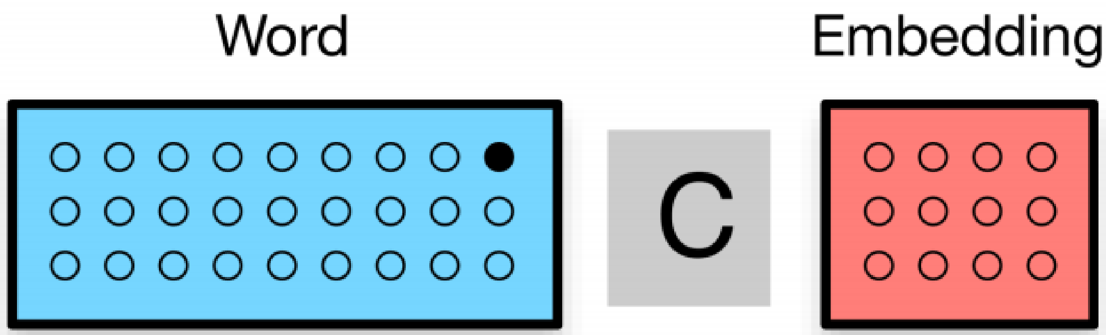


FF Neural Language Model



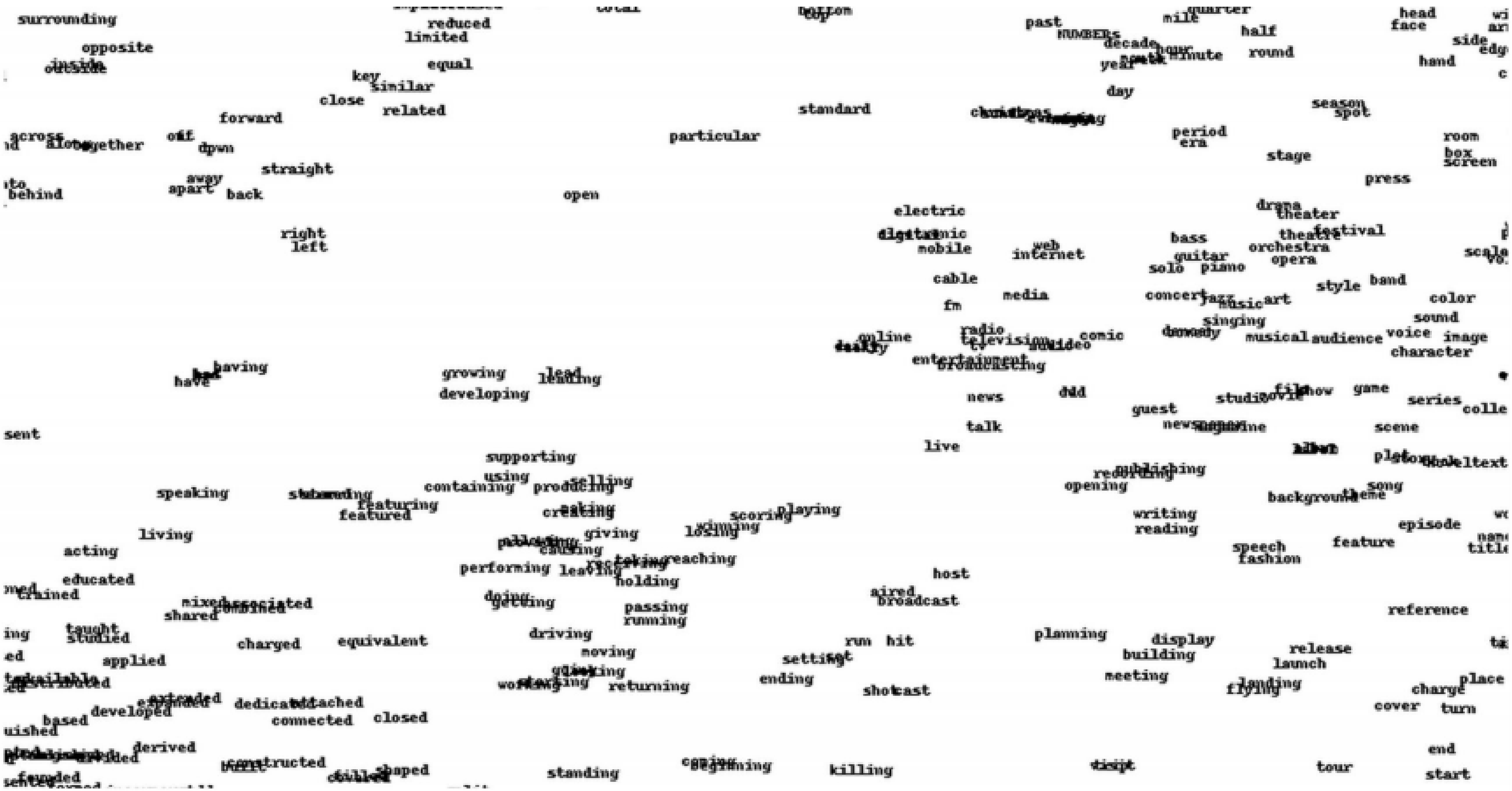
Bengio et al. 2003

Word Embeddings (词嵌入/词向量)



- Neural language models produce word embeddings as a by product
- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings are useful features in many NLP tasks

Word Embeddings Illustrated



Neural Language Models in Practice

- More expensive to train than n-grams!
- But yielded dramatic improvement in hard extrinsic tasks
  - speech recognition
  - and more recently machine translation
- Key practical issue
  - softmax requires normalizing over sum of scores for all possible words
- Deeper dive later in the course!