Outline

- Word Sequences: Language Models or Grammars?
- Counting Words in Corpora
- Simple (Unsmoothed) N-Grams
- Limitations of Simple N-Gram Models
- Smoothing
  - Add-one Smoothing
  - Good-Turing Discounting

Word Sequences: Language Models or Grammars?

- Natural Language texts are not just bags (unordered sets) of words.
  Happy families are all alike; every unhappy family is unhappy in its own way.
  does not mean the same as:
  Happy unhappy unhappy way all family are is its families every; alike own in.
- Word order is essential in conveying meaning
  - ◇ at least in English; less so in some other languages – but never completely irrelevant
- Various approaches can be taken to studying/modelling/exploiting word order in language

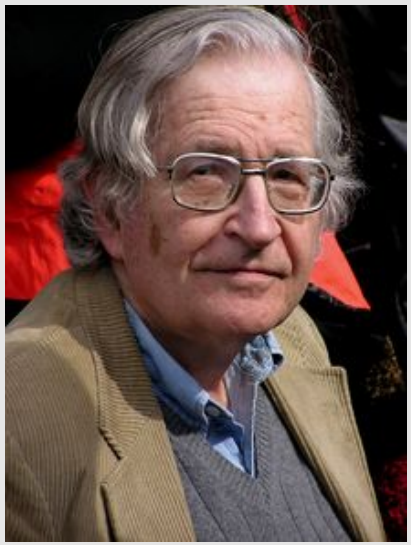
Word Sequences: Language Models or Grammars? (cont.)

- One approach is to observe that
  - ◇ words can be placed into classes (**word classes** or **parts-of-speech**) that predict in what order they can combine with words in other classes to form meaningful expressions
    - E.g. happy is an adjective (ADJ); family is a noun (N)
  - ◇ allowable word orderings can be specified via grammar rules which state permissible sequences of word classes
    - E.g. a noun phrase (NP) can consist of an adjective followed by a noun, but not the reverse. i.e. `NP -> ADJ+ N` but not `NP -> N ADJ`
- Sets of such rules are called a grammar and the study of what word classes there are, what rules there are and how these rules should be expressed is referred to as the study of syntax.

Word Sequences: Language Models or Grammars? (cont.)

- Another approach is to build **statistical models** of the sequences of words or wordforms that actually occur
- Such word-based models are frequently called **language models**, though grammars are model of language as well.
- The debate between advocates of statistical word sequence modelling and advocates of syntax-based modelling has been at the centre of natural language processing since the 1950's.

Word Sequences: Language Models or Grammars? (cont.)



But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of the term.
Noam Chomsky, 1969.

Everytime I fire a linguist our system performance improves.
Fred Jelinek, 1988.

Word Sequences: Language Models or Grammars? (cont.)

- Current view is both should be exploited for their respective strengths.
- We'll begin by looking at word sequence language modelling ...

Word Sequence Modelling

- Suppose we pause at a given word in a sentence
  Happy families are all alike; every unhappy . . .
  and think about what word is likely to come next.

- Impossible to know exactly which word will follow. But some (family, person, child) are more likely than others (is, the, unhappy, zebra).

Word Sequence Modelling (cont.)

- Word sequence models allow us to assign probabilities to which words are likely to follow others in a sentence
- Such models are useful for
  - ◇ speech recognition – deriving word sequences from phone sequences
    - e.g. I ate a nice peach vs I ate an ice beach
  - ◇ augmentative communication systems – assist severely disabled people to communicate
  - ◇ context-sensitive spelling error correction
    - e.g. They are leaving in about fifteen minuets
  - ◇ word completion in limited input devices such as mobile phone keypads

Counting Words in Corpora

- Probabilistic models are based on counting things.
  Need to decide
  - ◇ what to count
  - ◇ where to find the things to count

- Statistical models of language are built from **corpora** (singular **corpus**) – collections of digital texts and/or speech.

- The corpus used to train a model has a significant impact on the nature and potential use of the model.

Example Corpora

Some examples of corpora that have been built for studying language use/building statistical models:

- Brown Corpus – the "first" electronic corpus; assembled at Brown University in 1963-64; 1 million words from 500 written texts; mixture of genres.
- British National Corpus – 100 million words; spoken (transcribed) and written; balanced across a range of sources.
- Gigaword Corpus – a billion words of newswire text
- Switchboard corpus – 2430 telephone conversations/240 hour of speech/3 million words.
- The Web ...

Dickens10

- Dickens10 is a corpus assembled for this lecture.
- It consists of 10 well-known Dickens novels from Project Gutenberg: Bleak House, David Copperfield, Great Expectations, Hard Times, Little Dorritt, Nicholas Nickelby, Oliver Twist, Our Mutual Friend, Pickwick Paper, Tale of Two Cities.
- Some stats:
  - 3,234,061 wordform tokens (case-insensitive, counting punctuation)
  - 38,532 wordform types

Simple N-Gram Models

- Want to derive probabilistic model that will allow us to compute:
  - ◇ probability of entire word sequences (e.g. sentences); or
  - ◇ probability of the next word in a sequence

- Suppose any word is equally likely to appear at any point in a text.
  - ◇ If we assume a vocabulary V of $|V|$ wordforms then the probability of a wordform following any other wordform is $1/|V|$.
  - ◇ E.g. if there are 100,000 wordforms then the probability of any word following another is $1/100,000 = .00001$.

- Of course words are not equally likely to appear at any point. However, we could make the model slightly more sophisticated by assuming words occur with the probability that reflects their frequency of occurrence across the corpus.

Simple N-Gram Models (cont)

- E.g. the following frequencies and associated probabilities are observed in the 3,234,061 word Dickens10 corpus

Word	Frequency	Unigram Probabilty
the	122474	.03787
man	5270	.00163
worst	161	.00005
blunderbus	1	.0000003

- Using a model based on these frequencies we can predict that the word the is much more likely to appear than the word worst

Simple N-Gram Models (cont)

- However, in the sentence beginning: It was the best of times, it was the . . . it is unlikely the next word is the – much more likely to be worst, even though the is $\sim$ 750 times more likely to occur at a randomly chosen point in the corpus than worst

- This suggests that rather than looking at the relative frequencies of individual words we should look at the **conditional probabilities** of words given the preceding words. i.e. the probability of seeing the immediately after the is lower than it is otherwise; and the probability of seeing worst after the is higher.

Simple N-Gram Models (cont)

- Suppose we represent a word sequence of n words as $w_1 \ldots w_n$ or w_1^n.
- If we consider each word occurring in its correct position an independent event the probability of the sequence can be represented as: $P(w_1, w_2, \ldots, w_{n-1}, w_n)$
- Using the chain rule of probability this can be decomposed as:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\cdots P(w_n|w_1^{n-1})$$
$$P(w_1^n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

- Problem: how can we compute probabilities like $P(w_n|w_1^{n-1})$? Cannot count number of times w_n follows $w_1 \ldots w_{n-1}$ – for $n > 4$ or 5 would need a fantastically large corpus.

Simple N-Gram Models (cont)

- Solution (**Markov assumption**): approximate the probability of n-th word in sequence given $n-1$ preceding words by the probability of the nth word given a limited number k of preceding words.
  When $k = 1$ this yield a **bigram** model; when $k = 2$ a **trigram** model.

- The general equation for N-gram approximation to the conditional probability of the next word in a sequence is:
  $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$

- Taking the case where $N = 2$ (bigrams) and substituting this approximation into the equation for computing the probability of a sequence of n words we get:
  $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$

Simple N-Gram Models (cont)

- To train a bigram model using a corpus count the number of times each bigram occurs and divide that count by the number of bigrams that share the first word:

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

where $C(w_{n-1}w_n)$ is the count of the bigram $(w_{n-1}w_n)$.

- This can be simplified since the sum of all bigram counts that start with a given word w_{n-1} is equal to the unigram count for the word w_{n-1}:

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

- This generalises to N-grams of any length:

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

Simple N-Gram Models (cont)

- This approach estimates the N-gram probability by dividing the observed frequency of a sequence by the observed frequency of its prefix – this ratio is called a **relative frequency**.
- Using relative frequencies to estimate probabilities is one example of **Maximum Likelihood Estimation (MLE)** – the resulting parameter estimates maximize the likelihood of the corpus, C, given the model, M (i.e. maximize $P(C|M)$).

Simple N-Gram Models (cont)

- E.g. worst occurs 161 times in the 3,234,061 word Dickens10 corpus.
- What is the probability that worst will occur in another corpus of 3,234,061 words?
- 161/3234061 is not the best estimate of the probability of worst in any situation.
- However, it is the estimate that makes it *most likely* that worst will occur 161 times in a corpus of 3,234,061 words.

Simple N-Gram Models: Bigram Example

- Suppose we wish to calculate the probability of the sentence:
  He poured out a glassful and drank it greedily
- We begin by determining the bigram counts:

	He	poured	out	a	glassful	and	drank	it	greedily
He	0	9	0	19	0	122	20	3	0
poured	0	0	29	2	0	0	0	5	0
out	3	0	0	133	0	112	0	1	0
a	1	0	0	0	6	2	0	0	0
glassful	0	0	0	0	0	1	0	0	0
and	1413	13	126	2154	0	0	35	837	2
drank	0	0	4	11	0	1	0	15	0
it	17	1	304	321	0	206	0	1	1
greedily	0	0	0	0	0	0	0	0	0

Simple N-Gram Models: Bigram Example (cont)

- Given these bigram counts and the unigram counts:

He	32787
poured	93
out	7633
a	57698
glassful	15
and	89219
drank	124
it	36345
greedily	8

we can compute bigram probabilities according to the parameter estimation formula above.

Simple N-Gram Models: Bigram Example (cont)

- Here are the bigram probabilities:

	He	poured	out	a	glassful	and	drank	it	greedily
He	0	.000274	0	.000579	0	.003720	.000609	.000091	0
poured	0	0	.311828	.021505	0	0	0	.053763	0
out	.000393	0	0	.017424	0	.014673	0	.000131	0
a	.000017	0	0	0	.000103	.000035	0	0	0
glassful	0	0	0	0	0	.066667	0	0	0
and	.015837	.000146	.001412	.0241428	0	0	.000392	.009381	.000022
drank	0	0	.032258	.088710	0	.008064	0	.120968	0
it	.000468	.000027	.008364	.008832	0	.005668	0	.000027	.000027
greedily	0	0	0	0	0	0	0	0	0

- e.g. $P(\text{a}|\text{poured}) = \frac{C(\text{poured a})}{C(\text{poured})} = 2/93 = .021505$

Simple N-Gram Models: Bigram Example (cont)

- With these probabilities we can now compute the probability of the whole sentence
  S = He poured out a glassful and drank it greedily
  subject to one further assumption.

- To compute $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$ we need to know $P(He|w_0)$.

- In computing sequence probabilities a special marker for start of sequence is usually introduced. For sentences we are interested in the probability that a word starts a sentence. We denote start of sentence *s*. We can estimate $P(he|*s*)$ by using the bigram count of he preceded by . (full stop).

Simple N-Gram Models: Bigram Example (cont)

- We have:

  $P(\text{He poured out a glassful and drank it greedily}) =$

  $P(he|*s*) \times P(poured|he) \times P(out|poured) \times P(a|out) \times$
  $P(glassful|a) \times P(and|glassful) \times P(drank|and) \times$
  $P(it|drank) \times P(greedily|it) =$

  $.029465 \times .000274 \times .311828 \times .017424 \times .000103 \times$
  $.066667 \times .000392 \times .120968 \times .000027$

- Computing such products can frequently result in underflow

- Solution: Take log of each probability (logprob), sum logprobs, then take the anti-log.

Limitations of Simple N-Gram Models

- Suppose our example sentence

  He poured out a glassful and drank it greedily

  had instead been the perfectly plausible sentence:

  He poured out a glassful and drank it anxiously

- Unfortunately the bigram (it anxiously) does not occur in Dickens10.
  As a result $P(\text{anxiously}|\text{it}) = 0$ and then as a further consequence

  $$P(\text{He poured out a glassful and drank it anxiously}) = 0$$

Limitations of Simple N-Gram Models (cont)

- This example illustrates a number of problems with the n-gram approach
  - To get good estimates we need **very large corpora**: Dickens10 has 38,532 wordform types and therefore potentially $38532^2 \approx 1.5$ billion bigram types.
  - We observe 546546 of these in the corpus – i.e. approximately 4/10000ths of the total possible (put otherwise our training data is very sparse).
  - Since something not yet observed does not necessarily have 0 probability, we need some way of estimating low probability events that does not assign them 0 probability.

Smoothing

- There are a number of approaches to overcome the problem of 0 occurrence bigrams.
- Such approaches – which assign some probability to bigrams unseen in the training data, and possibly also to those seen very few times whose estimates are consequently unreliable – are called **smoothing**.
  - Must also reduce probabilities of seen bigrams so that total probability mass remains constant
- One simple approach to smoothing is to add one to all the counts in the bigram matrix before normalizing them into probabilities.
  - This is known as **add-one** or **Laplace** smoothing
  - Does not work terribly well but is useful to introduce concepts relevant to smoothing and gives a useful baseline

Add-One Unigram Smoothing

- Consider first smoothing unigram counts by adding one.
- The unsmoothed maximum likelihood estimate of the unigram the probability of a word w_x is computed by dividing the count of the word by the count of all word tokens N:

$$P(w_x) = \frac{c(w_x)}{\sum_i c(w_i)} = \frac{c(w_x)}{N}$$

- To adjust the counts add one to each and then multiply by $\frac{N}{N+V}$ where V is the total number word types (i.e. the vocabulary size).
- Where c_i is the unadjusted count for the i-th word, the adjusted count c_i^* is given by:

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

These new counts can be turned into adjusted probabilities p_i^* by normalising by N.

Add-One Unigram Smoothing (cont)

- Alternatively the probabilities can be computed directly from the counts:

$$p_i^* = \frac{c_i + 1}{N + V}$$

- Instead of defining smoothing in terms of adjusted counts, some accounts define it in terms of a **discount** d_c which is the ratio of the discounted counts to the original counts:

$$d_c = \frac{c^*}{c}$$

- The intuition here is that the non-zero counts need to be discounted in order free up the probability mass that is re-assigned to the zero counts.

Add-One Bigram Smoothing

- The approach to bigram smoothing is similar
- Recall that the bigram counts in the unsmoothed case are computed by:

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

- For add-one bigram smoothing we add one to each bigram count and divide by the unigram counts updated by the number of unigram types in the vocabulary V:

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

Add-One Smoothing: Example

- Returning to our earlier example, we begin by adjusting the counts by adding 1 to each.

	He	poured	out	a	glassful	and	drank	it	greedily
He	1	10	1	20	1	123	21	3	1
poured	1	1	30	3	1	1	1	6	1
out	4	1	1	134	1	113	1	1	1
a	1	1	1	1	7	2	1	1	1
glassful	1	1	1	1	1	1	1	1	1
and	1414	14	127	2155	1	1	36	838	3
drank	1	1	5	12	1	1	1	16	1
it	18	1	305	322	1	207	1	1	1
greedily	1	1	1	1	1	1	1	1	1

Add-One Smoothing: Example (cont)

- We adjust the unigram counts by adding $V = 38532$.

He	32787	+	38532	=	71319
poured	93	+	38532	=	38625
out	7633	+	38532	=	46165
a	57698	+	38532	=	96230
glassful	15	+	38532	=	38547
and	89219	+	38532	=	127751
drank	124	+	38532	=	38656
it	36345	+	38532	=	74877
greedily	8	+	38532	=	38540

we can now recompute bigram probabilities according to the parameter estimation formula:

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

Add-One Smoothing: Example (cont)

- The effect on bigram probabilities is striking:

Bigram	Unsmoothed Probability	Add-one Smoothed Probability
poured out	.311828	.000777
a glassful	.000103	.000073
drank greedily	0	.000026

Good-Turing Discounting

- There are a number of much better discounting algorithms than add-one smoothing
- Several rely on the idea that you can use the count of things you've seen once to help estimate the count of things you've never seen (Good-Turing discounting; Witten-Bell discounting; Kneyser-Ney smoothing)
- Good-Turing algorithm based on insight that the amount of probability to be assigned to zero occurrence n-grams can be re-estimated by looking at number of N-grams that occurred once.

Good-Turing Discounting (cont)

- Good-Turing relies on determining N_c, the number of n-grams with frequency c, also called the frequency of frequency c. Formally:

$$N_c = \sum_{x:count(x)=c} 1$$

  E.g. when appplied to bigrams N_0 is the number of bigrams with count 0, N_1 the number of bigrams with count 1, etc.

- The Maximum Likelihood Estimate (MLE) count for N_c is c. Good-Turing replaces this with the smoothed count:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

Good-Turing Discounting (cont)

- Can use this equation to replace MLE counts for all N_i.
- Instead of using this to re-estimate a smoothed count for N_0 use the following as probability of things with zero count (the **missing mass**)

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

where N_1 = count of items seen once in training, N = count of all items seen in training.

- For things with frequency $c > 0$ in training set:

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

Good-Turing Discounting (cont)

- Good-Turing originally used for estimating population of animal species
- Suppose we:
  - are fishing in a lake with 8 species: bass, carp, catfish, eel, perch, salmon, trout, whitefish
  - have caught: 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel.
  - we have not yet seen any catfish or bass.

  What is the probability that the next fish we catch will be a new species (catfish or bass)?

Good-Turing Discounting (cont)

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18}$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{GT}(\text{trout}) = \frac{c^*(\text{trout})}{N} = \frac{.67}{18} = .037$

Since there are two unseen species, the probability of the next fish caught being a catfish is:
$p^*_{GT}(\text{catfish}) = \frac{1}{N_0} \times p^*_{GT}(\text{unseen}) = \frac{1}{2} \times \frac{3}{18} = .085.$

Good-Turing Discounting: Issues

- Estimating how much probability to assign to each unseen N-gram assumes we know N_0.
- Compute this, for bigrams, for example, by calculating V (size of vocabulary = number of words observed in corpus) and setting

$$N_0 = V^2 - T$$

  where T is number of bigram types observed in the corpus.
- The re-estimate for $c*$ is undefined when $N_{c+1} = 0$.
- This can happen quite frequently (e.g. in above example the re-estimation for N_3 is undefined since $N_4 = 0$).

Summary

- The order in sequences of words in language can be modelled either using a grammar or a statistical language model
- Language models estimate probabilities from corpora but infeasible to estimate probabilities of long word sequences due to data sparsity
- Instead use Markov Assumption to approximate the probability of n-th word in sequence based on the k preceding words
  - Typically $k = 1$ (bigram model) or $k = 2$ (trigram model)
- Bigram and trigram probabilities can be estimated from corpora by counting frequency of a sequence and dividing it by the observed frequency of its prefix
  - using these relative frequencies is an example of Maximum Likelihood Estimation (MLE)

Summary (cont.)

- Zero probabilities are common in bigram and trigram estimates due to data sparsity
- Smoothing can help to deal with zero probabilities
- Various approaches
  - Add one smoothing is simple approach
  - Good-Turning discounting is more complex approach which uses observed probabilities