N-gram Language Modeling

Tanel Alumäe

Exercise

- Try to complete the following sentences:
  - They are leaving in about fifteen …
  - Dog ate my …
  - The Panama-registered tanker was carrying 136,000 tons of …
  - North Korea has agreed to send a delegation to next month's Winter Olympics in …
  - Officials from the two nations met …

Answers

- Try to complete the following sentences:
  - They are leaving in about fifteen **minutes**
  - Dog ate my **homework**
  - The Panama-registered tanker was carrying 136,000 tons of **oil**
  - North Korea has agreed to send a delegation to next month's Winter Olympics in **South Korea**
  - Officials from the two nations met **face to face**

Human ability to predict the “future”

- Clearly, at least some of us have the ability to predict future words in an utterance.
- How?
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge

Applications

- Why should a computer need the ability to predict words, given the preceding words?
  - Machine translation
    P("*high winds tonight*") > P("*large winds tonight*")
  - Spelling correction
    The study was conducted mainly be John Black
  - Speech recognition
    - English: P(*I saw a van*) >> P(*eyes awe of an*)
    - Estonian P(*kas sa tuled täna*) >> (*kassa tuled täna*)
  - Optical character recognition
  - Language identification, authorship detection, etc

Language modeling

- Fundamental tool in NLP
- Main idea:
  - Some words are more likely than others to follow each other
  - You can predict fairly accurately that likelihood
- In other words, you can build a language model

N-grams

- N-Grams are sequences of tokens.
- The N stands for how many terms are used
  - Unigram: 1 term
  - Bigram: 2 terms
  - Trigrams: 3 terms
- You can use different kinds of tokens
  - Character based n-grams
  - Word-based n-grams
  - Morph-based n-grams
- N-Grams give us some idea of the context around the token we are looking at

N-gram models of language

- A language model is a model that lets us compute the probability, or likelihood, of a sentence *S*, *P(S)*
- N-Gram models use the previous *N-1* words in a sequence to predict the next word
  - unigrams, bigrams, trigrams,…
- How do we construct or train these language models?
  - Count frequencies in very large corpora
  - Determine probabilities

Simple N-grams

- Assume a language has *V* word types in its lexicon, how likely is word *x* to follow word y?
  - Simplest model of word probability: *1/V*
  - Alternative 1: estimate likelihood of *x* occurring in new text based on its general frequency of occurrence estimated from a corpus (unigram probability)
    - *popcorn* is more likely to occur than *unicorn*
  - Alternative 2: condition the likelihood of *x* occurring in the context of previous words (bigrams, trigrams,…)
    - *mythical unicorn* is more likely than *mythical popcorn*

Computing the probability of a word sequence

- Compute the product of component conditional probabilities?
  - P(*the mythical unicorn*) = P(*the*) P(*mythical|the*) P(*unicorn|the mythical*)
- The longer the sequence, the less likely we are to find it in a training corpus
  - P(*Most biologists and folklore specialists believe that in fact the mythical unicorn horns derived from the narwhal*)
- Solution: approximate using n-grams

Bigram model

- Approximate $P(w_n | w_1^{n-1})$ by $P(w_n | w_{n-1})$
  - P(*unicorn*|*I want to see the mythical*) by P(*unicorn*|*mythical*)
- Markov assumption: the probability of a word depends only on the probability of a limited history
- Generalization: the probability of a word depends only on the probability of the n previous words
  - Trigrams, 4-grams, …
  - The higher n is, the more data needed to train
  - The higher n is, the sparser the matrix.
  - Leads us to backoff models

Using N-grams

- For N-gram models
  - $P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-1})$
  - $P(w_{k-1}, w_k) = P(w_{k-1}) P(w_k|w_{k-1})$
  - E.g.: P(*mythical unicorn*) = P(*mythical*) P(*unicorn*|*mythical*)
  - By the Chain Rule we can decompose a joint probability, e.g.
    $$P(w_{1,} w_{2,} \ldots, w_k) = P(w_1) P(w_2|w_1) \ldots P(w_k|w_1 w_2 \ldots w_{k-1})$$
  - Then we'll apply the bigram approximation:
    $$P(w_{1,} w_{2,} \ldots, w_k) \approx P(w_1) P(w_2|w_1) \ldots P(w_k|w_{k-1})$$
  - The probability of the sentence is thus the product of all it's bigrams
  - Typically, the 1st word is conditioned on a special sentence start symbol <s>, and a special sentence end symbol </s> probability is included as well

$$P(\langle s \rangle I\, saw\, the\, mythical\, unicorn \langle /s \rangle) \approx P(I|\langle s \rangle) P(saw|I) \ldots P(unicorn|mythical) P(\langle /s \rangle|unicorn)$$

The chain rule

- In general, for bigrams:

$$P(w_1, w_2, \ldots, w_k) \approx \prod_i P(w_i | w_{i-1})$$

- Or more generally, for N-grams, typically N=3 or N=4

$$P(w_1, w_2, \ldots, w_k) \approx \prod_i P(w_i | w_{i-1} .. w_{i-N+1})$$

- In general this is an insufficient model of language
  - because language has long-distance dependencies:
    - “The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing.”
- But we often N-gram models work surprisingly good

Estimating bigram probabilities

- Maximum likelihood estimate:
- ***C*** stand for ***count*** in a text corpus

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w)}{C(w_{i-1})}$$

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

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

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

Real corpus example

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

	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

- i.e., “i *want*” occurs 827 times

Raw bigram probabilities

- Find bigram maximum likelihood estimates
- First, compute *C(w)*:

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

- Then, find $$P(w_i|w_{i-1})=\frac{C(w_{i-1},w)}{C(w_{i-1})}$$

	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

Example of a bigram sentence probability

- P(i want to eat chinese food) =
- P(I|<s>) 0.25
- × P(want|I)
- × P(to|want)
- × P(eat|to)
- × P(chinese|eat)
- × P(food|chinese)
- × P(</s>|food) 0.68

	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

Different knowledge in N-grams

- Probabilities seem to capture “syntactic” facts, “world knowledge'”
  - eat is often followed by a noun phrase
  - British food is not too popular

- P(british|eat) = .0001
- P(chinese|eat) = .021
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Practical issues

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

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

Google N-Gram Release, August 2006

- 4-grams from the corpus:
  - `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`

Language model evaluation

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a training set.
- We test the model’s performance on data we haven’t seen.
  - A test set is an unseen dataset that is different from our training set, totally unused.
  - An evaluation metric tells us how well our model does on the test set.

Perplexity

- The Shannon Game:
- How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

- A better language model is the one which assigns a **higher probability to the word that actually occurs**
- Perplexity is the inverse probability of the test set, normalized by the number of words

$$Perplexity(w_1, w_{2,\ldots}, w_k) = \sqrt[k]{\frac{1}{P(w_1, w_{2,\ldots}, w_k)}}$$

Perplexity as a branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Lower perplexity

- Lower perplexity = higher likelihood of test data = better model
- Perplexities of N-grams on English data
  - 38M for training, 1.5M for test

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Zero probability problem

- A few N-grams occur with high frequency
- Many N-grams occur with low frequency
- You can quickly collect statistics on the high frequency events
- You might have need an very very large corpus to get valid statistics on low frequency N-grams
- Some of the zeroes in the N-gram statistics are really zeros but others are simply low frequency events you haven't seen yet. How to address?

Zero probability problem

- Training set:
  - … denied the allegations
  - … denied the reports
  - … denied the claims
  - … denied the request
  - P(“offer” | denied the) = 0
- Test set
- … denied the offer
- … denied the loan

Bigrams with zero probability mean that we will assign 0 probability to the test set!
And hence we cannot compute perplexity (can’t divide by 0)!

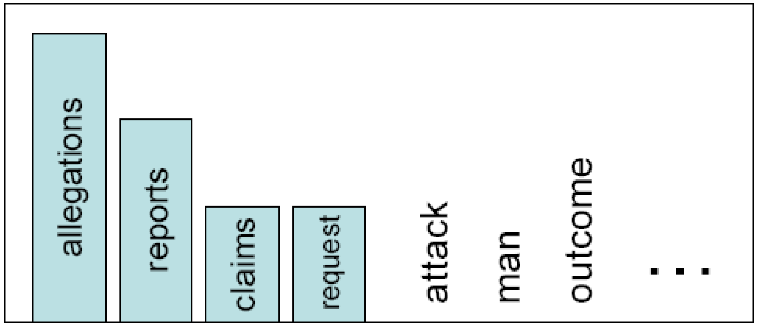
Smoothing

- Every n-gram training matrix is sparse, even for very large corpora (Zipf’s law)
- Solution: estimate the likelihood of unseen n-grams
- Problems: how do you adjust the rest of the corpus to accommodate these ‘phantom’ n-grams?
- Methods to handle this are called **smoothing**.

Smoothing intuition

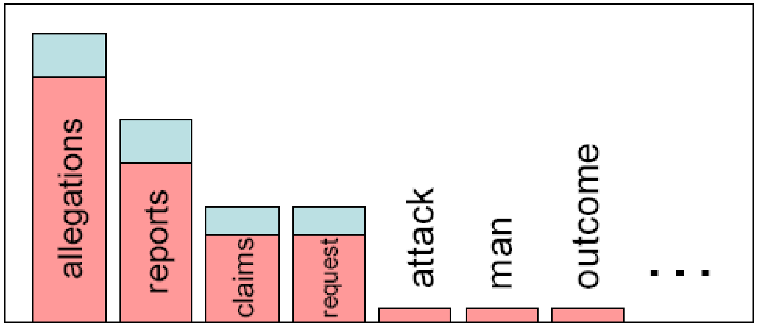
- Smoothing is like Robin Hood: steal from the rich and give to the poor
  - We often want to make predictions from sparse statistics:

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!

Add-one smoothing

- Also called **Laplace smoothing**
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- Maximum likelihood (unsmoothed) estimate:

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w)}{C(w_{i-1})}$$

- Add-one estimate *(V* is the size of the vocabulary)*:*

$$P_{add-one}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w) + 1}{C(w_{i-1}) + V}$$

Add-one smoothing on the **Berkeley Restaurant Corpus**

	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

	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

	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

Exercise

- Corpus:

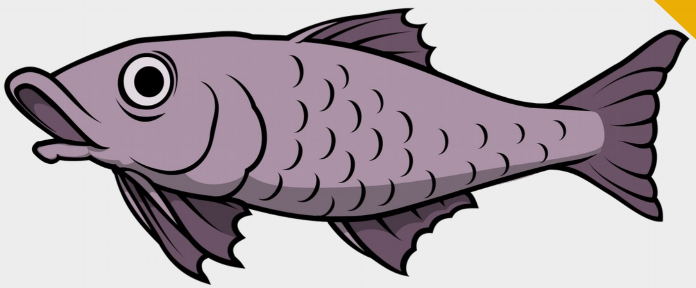
```
<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

<s> I do not like green
eggs and Sam </s>
```

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w)}{C(w_{i-1})}$$

$$P_{add-one}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w)+1}{C(w_{i-1})+V}$$

- Calculate:
  - *P(Sam|am)* using maximum likelihood
  - *P(Sam|am)* using add-one smoothing?
- Include <s> and </s> in your counts just like any other token

Witten-Bell smoothing: intuiton



- Imagine:
  - You go fishing
  - You read from a poster next to the lake that there are 5 fish species living in the lake: pike, perch, trout, bream and catfish
  - After very successful 2 hours, you have caught 5 perches, 2 pike and one bream
  - What's the (unigram) probability that the next fish you'll catch will be a catfish?
  - Assuming that the info on the poster is correct
- Maximum likelihood estimate: 0
- Add-one estimate: (0+1)/(8+5) = 1/13 = 0.077

Witten-Bell discounting

- Continuing the fishing example:..
  - A zero count for catfish means that you haven't seen it yet…but every fish type in the bag was unseen once…so…
- How many times did we catch a fish for the first time? Once for each caught fish species (T=3)
- Estimate total probability of unseen fish as:

$$\frac{T}{N+T}=\frac{3}{8+3}=0.27$$

- View training corpus as series of events, one for each fish (N) and one for each new type (T)
- We can divide the probability mass equally among unseen fish types
  - 0.27/2=**0.14**
  - This is the Witten-Bell probability estimate that the next fish will be catfish
  - Discount values for Witten-Bell are much more reasonable than Add-One

Backoff and Interpolation

- If we don't haven't seen a trigram $w_1 w_2 w_3$ in training data, we can instead check how many times $w_2 w_3$ occurred
  - "Peter eats pizza" – what is the probability of "pizza" after "Peter eats", if "Peter eats pizza" was seen 0 times in training data?
  - Is it the same as for trigram "unicorn monkey pizza" that is also seen 0 times?
  - To get a better probability estimate, we will check whether "eats pizza" is probable
- There are two methods to use lower-order context: backoff and interpolation

Linear interpolation

Simple linear interpolation:

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

The weight must sum to one:

$$\sum_i \lambda_i = 1$$

Backoff

- *Katz* backoff:

$$P_{\mathrm{BO}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

- In the above:
  - P^* is a discounted probability estimate
  - α is a backoff factor for this n-gram
- Intuition:
  - If the higher order n-gram (e.g. trigram) is seen more than a certain number of times, we will trust it and use a probability estimate based on it
  - Otherwise, we'll use a lower order probability estimate
  - But we have to scale the lower order probability estimates using $\alpha<1$, otherwise the lower order probability is too optimistic

Absolute discounting

- Absolute discounting subtracts a fixed discount *d* from each count

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$$

- The first term is the discounted bigram (with d=0.75, for example)
- Second term is unigram with an interpolation weight λ

Kneser Ney smoothing

- Consider the sentence:
  I can't see without my reading ____
- The word *glasses* seems to be much more likely in this context than the word *Francisco*
- However, *Francisco* is more common than *glasses* in the training corpus
- We would like to capture the intuition that *Francisco* occurs almost always after the word *San*, while *glasses* has much wider distribution

Kneser-Ney smoothing, II

- Instead of *P(w)*: “How likely is w”
- $P_{CONTINUATION}(w)$: “How likely is w to appear as a novel continuation?

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}|$$

  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen
  - A frequent word (*Francisco*) occurring in only one context (*San*) will have a low continuation probability
- To turn this into probability, we have to normalize by the number of word bigram **types**

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|}$$

Kneser-Ney smoothing, III

- Final equation for Kneser-Ney bigrams:

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

- The normalizing constant λ is specific to each word:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)}|\{w : C(w_{i-1}w) > 0\}|$$

  - The first term is a normalizing constant
  - The second term is the number of word types that can follow w_{i-1}

Questions?