Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11-711 · Fall 2020

Lecture 5: Language modeling

Emma Strubell

Announcements

- Recitation this week will be a P1 Q&A with Han
- P1 bakeoff: No pre-trained word embeddings or BERT. Out-of-the box linguistic annotations (e.g. pos tags), tf-idf, etc. are fine.

Probabilistic language models

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?
  - Machine translation:
    P(*high winds tonight*) > P(*large winds tonight*)

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?
  - Machine translation:
    P(*high winds tonight*) > P(*large winds tonight*)
  - Spelling correction:
    P(*I'll be five minutes late*) > P(*I'll be five minuets late*)

Probabilistic language models

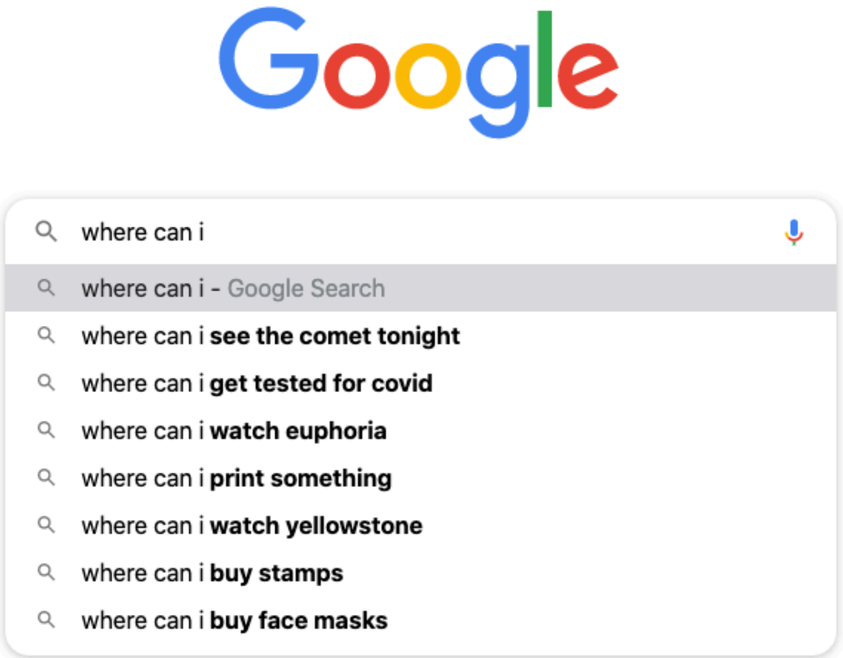
- Today's goal: assign a probability to a sentence. Why?
  - Machine translation:
    P(*high winds tonight*) > P(*large winds tonight*)
  - Spelling correction:
    P(*I'll be five minutes late*) > P(*I'll be five minuets late*)
  - Speech recognition:
    P(*I saw a van*) > P(*eyes awe of an*)

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?
  - Machine translation:
    P(*high winds tonight*) > P(*large winds tonight*)
  - Spelling correction:
    P(*I'll be five minutes late*) > P(*I'll be five minuets late*)
  - Speech recognition:
    P(*I saw a van*) > P(*eyes awe of an*)
  - Summarization, question answering, …

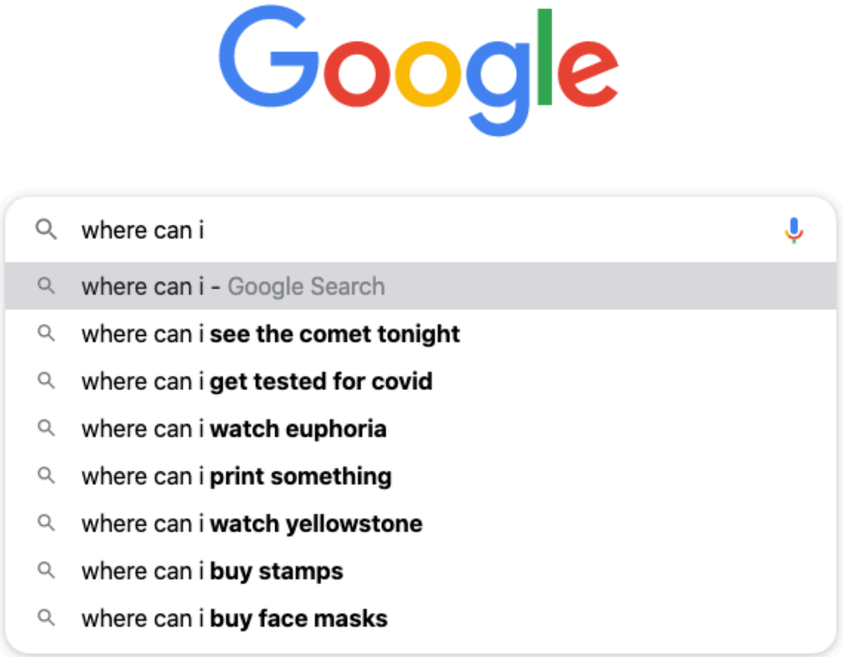
Probabilistic language models

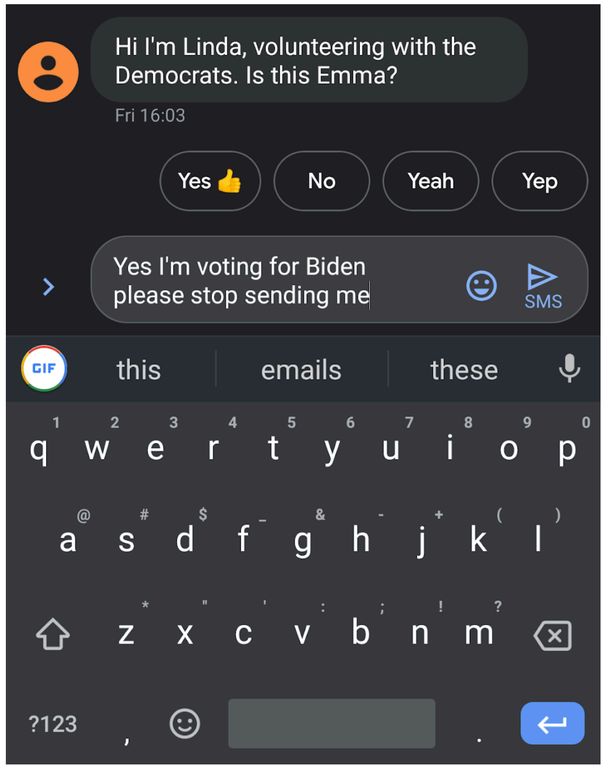
- Today's goal: assign a probability to a sentence. Why?



Probabilistic language models

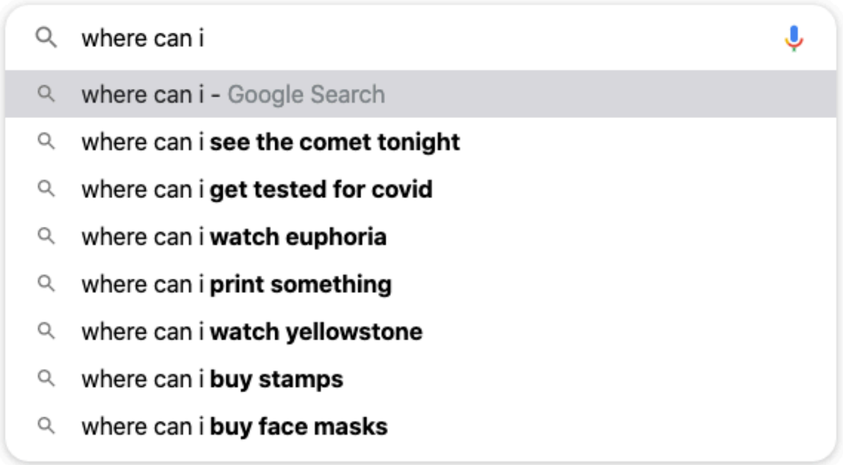
- Today's goal: assign a probability to a sentence. Why?

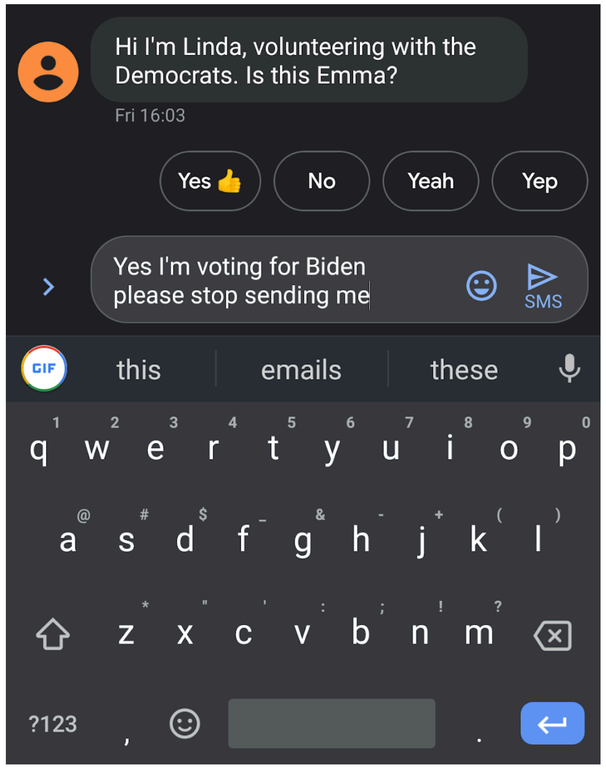




Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?

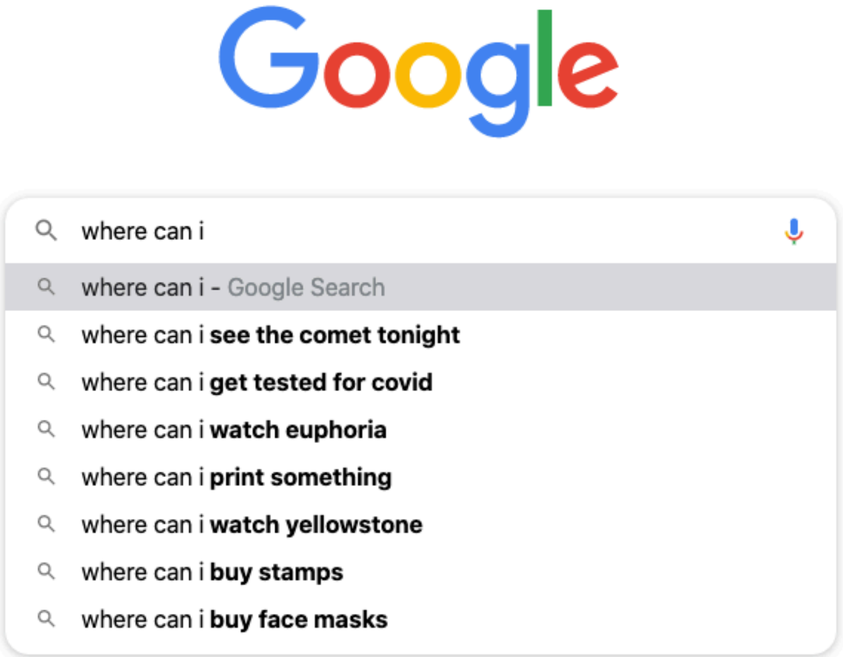


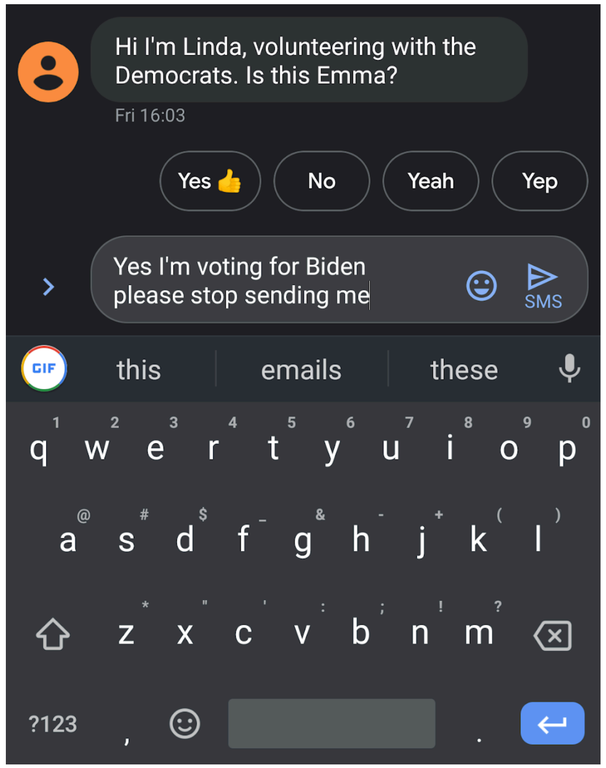


language model

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?



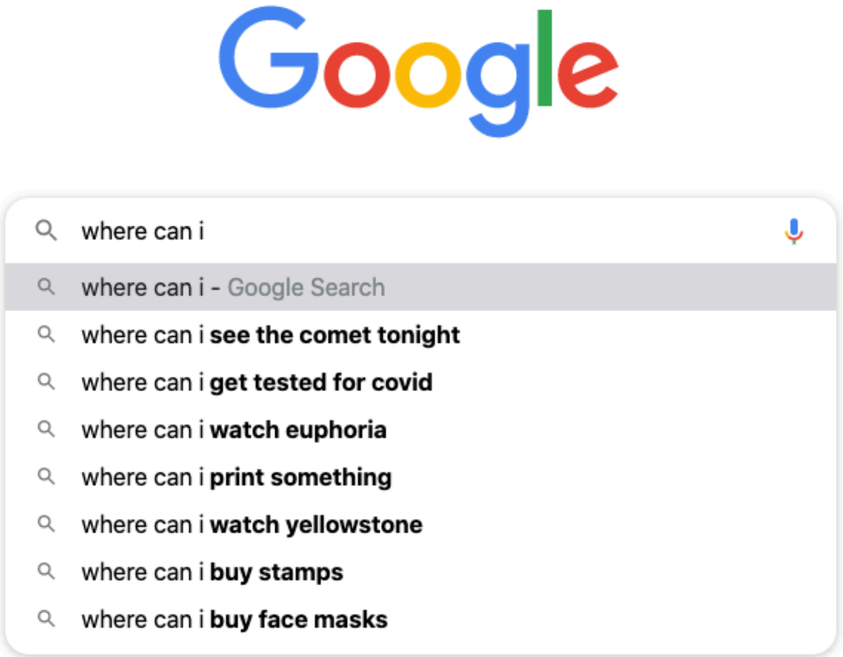


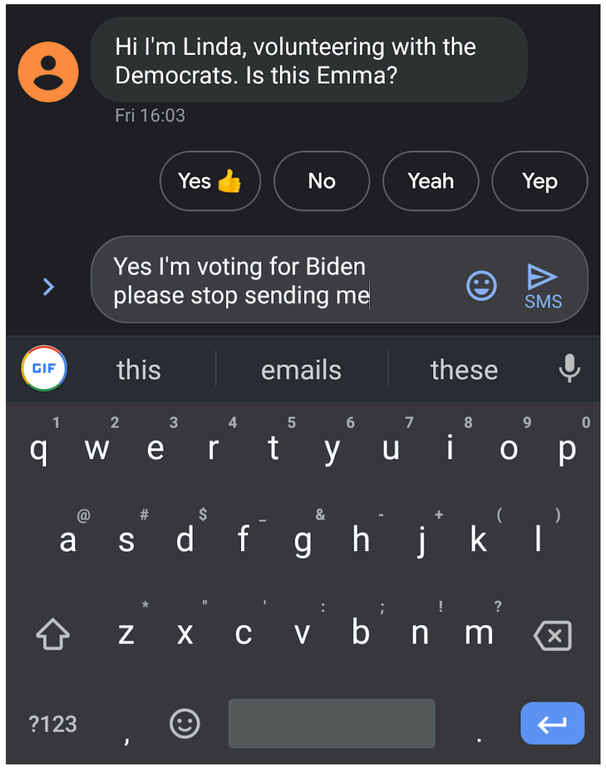
Kannan et al. 2016

language model

Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?





Kannan et al. 2016

language model

- and since ~2018, state-of-the-art vector representations of text.

Probabilistic language models

Probabilistic language models

- Goal: compute the probability of a sentence (or sequence of words):

$$P(\mathbf{w}) = P(w_1, w_2, w_3, \ldots, w_n)$$

Probabilistic language models

- Goal: compute the probability of a sentence (or sequence of words):

$$P(\mathbf{w}) = P(w_1, w_2, w_3, ..., w_n)$$

- Related task: probability of the next word:

$$P(w_5 \mid w_4, w_3, w_2, w_1)$$

Probabilistic language models

- Goal: compute the probability of a sentence (or sequence of words):

$$P(\mathbf{w}) = P(w_1, w_2, w_3, ..., w_n)$$

- Related task: probability of the next word:

$$P(w_5 \mid w_4, w_3, w_2, w_1)$$

- A model that computes either of these is called a **language model** (or **LM**).

How to compute P(w)?

How to compute P(w)?

- Want to compute the joint probability:

$$P(its, water, is, so, transparent, that)$$

How to compute P(w)?

- Want to compute the joint probability:

$$P(its, water, is, so, transparent, that)$$

- Intuition: use chain rule of probability

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

How to compute P(w)?

- Want to compute the joint probability:

$$P(its, water, is, so, transparent, that)$$

- Intuition: use chain rule of probability

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

- Chain rule in general:

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

How to compute P(w)?

How to compute P(w)?

- Want to compute the joint probability:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

How to compute P(w)?

- Want to compute the joint probability:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

- For example:

$$P(\mathit{its}, \mathit{water}, \mathit{is}, \mathit{so}, \mathit{transparent}, \mathit{that}) =$$
$$P(\mathit{its}) \cdot P(\mathit{water} \mid \mathit{its}) \cdot P(\mathit{is} \mid \mathit{its\ water})$$
$$\cdot P(\mathit{so} \mid \mathit{its\ water\ is}) \cdot P(\mathit{transparent} \mid \mathit{its\ water\ is\ so})$$

How to estimate P(w_i)?

How to estimate P(w_i)?

- Can we just count and divide (relative frequency estimate)?

How to estimate P(w_i)?

- Can we just count and divide (relative frequency estimate)?

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{\text{count}(\textit{its water is so transparent that the})}{\text{count}(\textit{its water is so transparent that})}$$

How to estimate P(w_i)?

- Can we just count and divide (relative frequency estimate)?

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{\text{count}(\textit{its water is so transparent that the})}{\text{count}(\textit{its water is so transparent that})}$$

- No! Too many (infinitely) possible sentences.

How to estimate P(w_i)?

- Can we just count and divide (relative frequency estimate)?

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{\text{count}(\textit{its water is so transparent that the})}{\text{count}(\textit{its water is so transparent that})}$$

- No! Too many (infinitely) possible sentences.
- Too sparse: we'll never observe enough data to estimate.

Markov assumption

Markov assumption

- Make the simplifying assumption:

$$P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{that})$$

Markov assumption

- Make the simplifying assumption:

$$P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{that})$$

- Or maybe:

$$P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{transparent that})$$

Markov assumption

Andrei Markov
1856 – 1922

- Make the simplifying assumption:

$$P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{that})$$

- Or maybe:

$$P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{transparent that})$$

- **Markov assumption**: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.

Markov assumption

- **Markov assumption**: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.

$$P(w_1, w_2, w_3, \ldots, w_n) \approx \prod_i P(w_i \mid w_{i-1}, \ldots, w_{i-k})$$

Markov assumption

- **Markov assumption**: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.

$$P(w_1, w_2, w_3, \ldots, w_n) \approx \prod_i P(w_i \mid w_{i-1}, \ldots, w_{i-k})$$

- In other words, approximate each component of the product:

$$P(w_i \mid w_{i-1}, \ldots, w_2, w_1) \approx P(w_i \mid w_{i-1}, \ldots, w_{i-k})$$

Simplest case: Unigram model

Simplest case: Unigram model

- **Unigram model**:

$$P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i)$$

Simplest case: Unigram model

- **Unigram model**:

$$P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i)$$

- Example sentences generated by a unigram model trained on financial news:

fifth an of futures the an incorporated a a the inflation most dollars quarter in is mass

thrift did eighty said hard ‘m july bullish

that or limited the

Bigram model

Bigram model

- **Bigram model**: condition on the previous word

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

Bigram model

- **Bigram model**: condition on the previous word

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

- Example sentences generated by a bigram model trained on financial news:

texaco rose one in this issue is pursuing growth in a boiler house said mr. gurria mexico 's motion control proposal without permission from five hundred fifty five yen

outside new car parking lot of the agreement reached

this would be a record november

N-gram models

N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …

N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

N-gram models

N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.

N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:

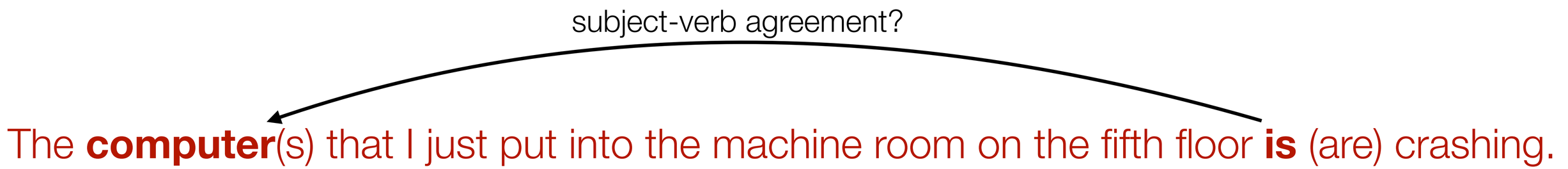
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

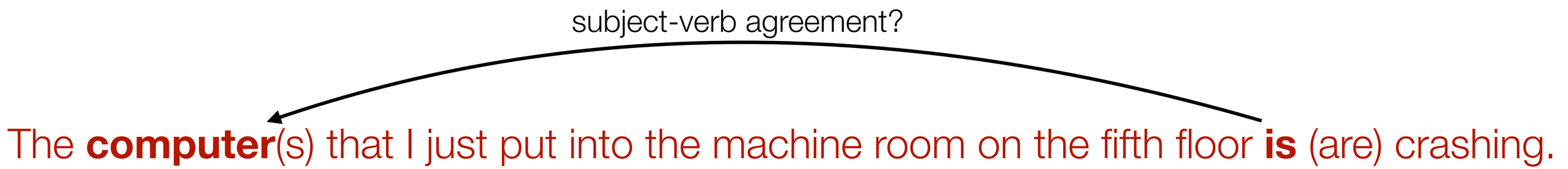
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:



N-gram models

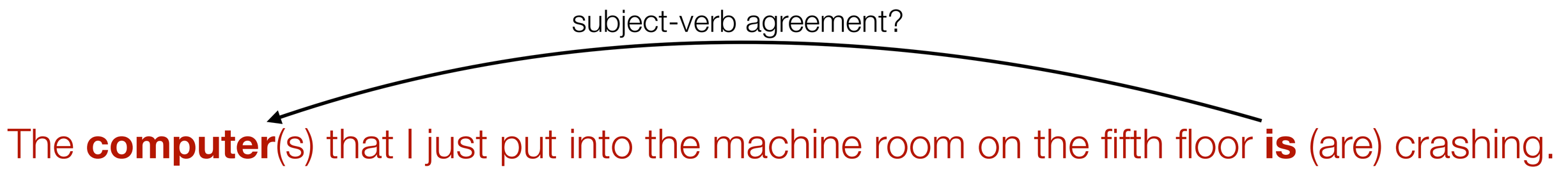
- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:



  - But, n-grams require accounting for V^n events. V=10^4, n=7 → 10^{28} events.

N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:



  - But, n-grams require accounting for V^n events. V=10^4, n=7 → 10^{28} events.

- Another example of **bias-variance tradeoff**

Estimating bigram probabilities

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \qquad P(Sam \mid \langle s \rangle) = \frac{1}{3} = \text{???}$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \qquad P(Sam \mid \langle s \rangle) = \frac{1}{3} = \text{???}$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \qquad P(Sam \mid am) = \frac{1}{2} = \text{???}$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \qquad P(Sam \mid \langle s \rangle) = \frac{1}{3} = 0.33$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \qquad P(Sam \mid am) = \frac{1}{2} = \text{???}$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \qquad P(Sam \mid \langle s \rangle) = \frac{1}{3} = 0.33$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \qquad P(Sam \mid am) = \frac{1}{2} = 0.5$$

Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \qquad P(Sam \mid \langle s \rangle) = \frac{1}{3} = 0.33 \qquad P(do \mid I) = \frac{1}{3} = 0.33$$

$$P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \qquad P(Sam \mid am) = \frac{1}{2} = 0.5 \qquad P(am \mid I) = \frac{2}{3} = 0.67$$

A bigger example

Berkeley Restaurant Project sentences

A bigger 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 venecia open during the day

Raw bigram counts

	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 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

Raw bigram probabilities

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

Raw bigram probabilities

- Normalize by unigrams:

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

Raw bigram probabilities

- Normalize by unigrams:

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

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

Raw bigram probabilities

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

- 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

$$
\begin{aligned}
P(\langle s\rangle\ I\ want\ english\ food\ \langle /s\rangle) = \ & P(I \mid \langle s\rangle) \\
& \times P(want \mid I) \\
& \times P(english \mid want) \\
& \times P(food \mid English) \\
& \times P(\langle /s\rangle \mid food) \\
& = 0.000031
\end{aligned}
$$

Bigram estimates of sentence probabilities

$$
\begin{aligned}
P(\langle s\rangle\ I\ want\ english\ food\ \langle /s\rangle) = \ & P(I \mid \langle s\rangle) \\
& \times P(want \mid I) \\
& \times P(english \mid want) \\
& \times P(food \mid English) \\
& \times P(\langle /s\rangle \mid food) \\
& = 0.000031
\end{aligned}
$$

- In practice:

$$\log P(I \mid \langle s\rangle) + \log P(want \mid I) + \log P(english \mid want) + \ldots$$

What kinds of knowledge?

- P(english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

What kinds of knowledge?

- P(english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

world knowledge

What kinds of knowledge?

- P(english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

world knowledge

grammar (infinitive verb)

What kinds of knowledge?

- P(english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

world knowledge

grammar (infinitive verb)

ungrammatical

What kinds of knowledge?

- P(english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

world knowledge

grammar (infinitive verb)

ungrammatical

people like to talk about themselves

Evaluating language models

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard
- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard
- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:
  - Does it assign higher probability to “real” or “frequently observed” sentences compared to “ungrammatical” or “rarely observed” sentences?

Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard
- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:
  - Does it assign higher probability to “real” or “frequently observed” sentences compared to “ungrammatical” or “rarely observed” sentences?
  - **Perplexity**: A bad approximation for future performance.

Perplexity: Intuition

Perplexity: Intuition

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

Perplexity: Intuition

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

I always order pizza with cheese and _____

The 33rd president of the U.S. was _____

I saw a _____

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1E-100

Perplexity: Intuition

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

  I always order pizza with cheese and _____

  The 33rd president of the U.S. was _____

  I saw a _____

  - Unigrams are terrible at this.

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1E-100

Perplexity: Intuition

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

  I always order pizza with cheese and _____

  The 33rd president of the U.S. was _____

  I saw a _____

  - Unigrams are terrible at this.

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1E-100

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.

Perplexity: Intuition

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

  I always order pizza with cheese and _____

  The 33rd president of the U.S. was _____

  I saw a _____

  - Unigrams are terrible at this.

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1E-100

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.
- Compute per-word log likelihood on held out data: $\sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$

Perplexity: Intuition

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

  I always order pizza with cheese and _____

  The 33rd president of the U.S. was _____

  I saw a _____

  - Unigrams are terrible at this.

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
…	
fried rice	0.0001
…	
and	1E-100

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.
- Compute per-word log likelihood on held out data:

tokens in held out data

$$\sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

Perplexity

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1) \qquad \text{ppl}(\mathbf{w}) = 2^{-\frac{\ell(w)}{M}}$$

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

$$\text{ppl}(\mathbf{w}) = 2^{-\frac{\ell(w)}{M}}$$

entropy: avg negative log likelihood per word

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

$$\text{ppl}(\mathbf{w}) = 2^{-\frac{\ell(w)}{M}}$$

entropy: avg negative log likelihood per word

avg # bits required to encode each word under this model

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

$$\text{ppl}(\mathbf{w}) = 2^{-\frac{\ell(w)}{M}}$$

entropy: avg negative log likelihood per word

avg # bits required to encode each word under this model

- How confused (*perplexed*) is the model on test data?

Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest *P*(*sentence*).
- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)$$

$$\text{ppl}(\mathbf{w}) = 2^{-\frac{\ell(w)}{M}}$$

entropy: avg negative log likelihood per word

avg # bits required to encode each word under this model

- How confused (*perplexed*) is the model on test data?
- Minimizing perplexity is the same as maximizing probability.

Lower perplexity = better model

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, financial news (Wall Street Journal)

n-gram order	unigram	bigram	trigram
perplexity	962	170	109

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, financial news (Wall Street Journal)

n-gram order	unigram	bigram	trigram
perplexity	962	170	109

RANK	MODEL	TEST PERPLEXITY ↓	VALIDATION PERPLEXITY	PARAMS	EXTRA TRAINING DATA	PAPER	CODE	RESULT	YEAR
1	**GPT-3** (Zero-Shot)	20.5		175000M	✓	Language Models are Few-Shot Learners			2020
2	**BERT-Large-CAS**	31.3	36.1	395M	✗	Language Models with Transformers			2019
3	**GPT-2**	35.76		1542M	✓	Language Models are Unsupervised Multitask Learners			2019
4	**Mogrifier LSTM + dynamic eval**	44.9	44.8	24M	✗	Mogrifier LSTM			2019

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Lower perplexity = better model?

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

1 gram	Months the my and issue of year foreign new exchange’s september were recession exchange new endorsed a acquire to six executives
2 gram	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
3 gram	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

Zero probability bigrams

Zero probability bigrams

	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

Zero probability bigrams

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: M = 884,647 tokens; V = 29,066 types

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: M = 884,647 tokens; V = 29,066 types
  - 300,000 bigram types out of V^2 = 884 million possible bigrams

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: M = 884,647 tokens; V = 29,066 types
  - 300,000 bigram types out of V^2 = 884 million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: M = 884,647 tokens; V = 29,066 types
  - 300,000 bigram types out of V^2 = 884 million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)

training set:

…denied the allegations

…denied the reports

…denied the claims

…denied the request

test set:

…denied the offer

…denied the loan

Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: M = 884,647 tokens; V = 29,066 types
  - 300,000 bigram types out of V^2 = 884 million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)

training set:

…denied the allegations

…denied the reports

…denied the claims

…denied the request

test set:

…denied the offer

…denied the loan

$$P(offer \mid denied\ \ the) = 0$$

Smoothing!

Smoothing!

- As with counts for text classification, can use **smoothing** to improve generalization.

Smoothing!

- As with counts for text classification, can use **smoothing** to improve generalization.
- Intuition: steal some probability mass from observed n-grams, distribute the wealth.

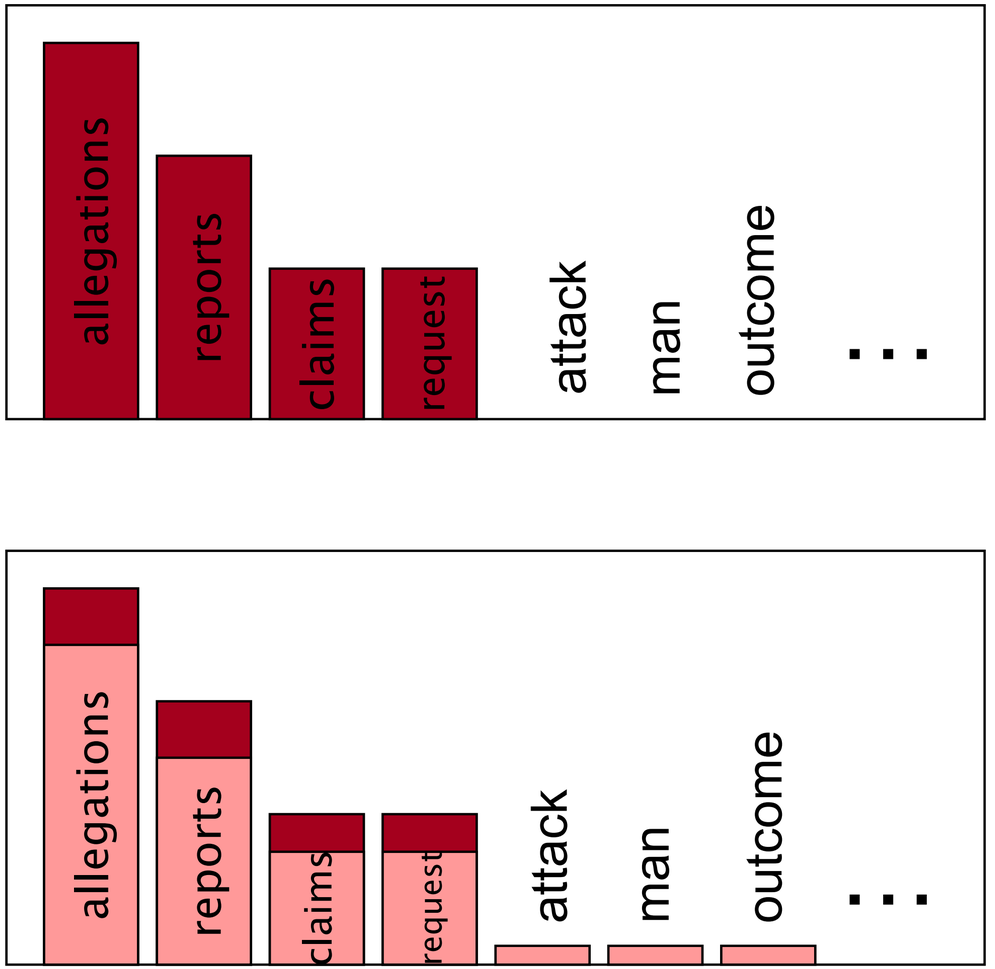


Figure from Dan Klein

Smoothing!

- As with counts for text classification, can use **smoothing** to improve generalization.
- Intuition: steal some probability mass from observed n-grams, distribute the wealth.
- Can use add-one (Laplace) smoothing:

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

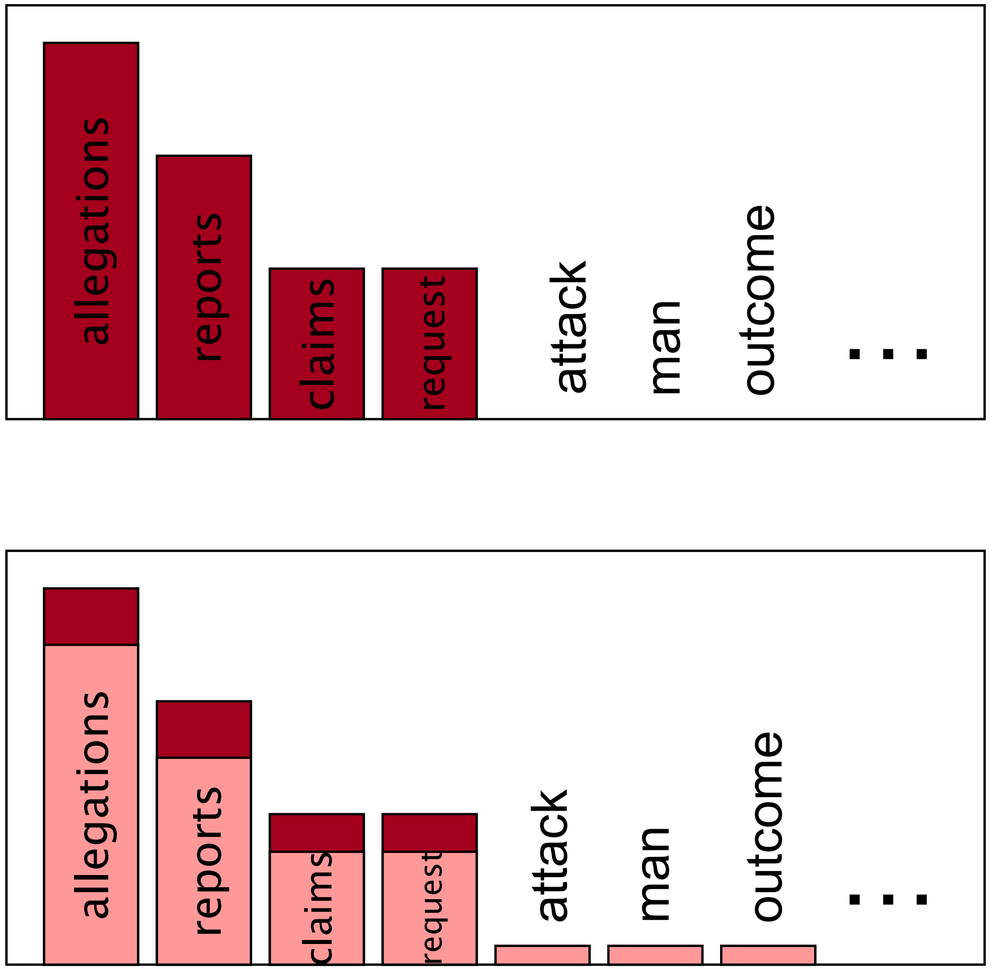


Figure from Dan Klein

Laplace smoothing for bigrams?

$$c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$$

Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

Laplace smoothing for bigrams?

- Compare "reconstituted" counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

	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

Laplace smoothing for bigrams?

- Compare "reconstituted" counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

	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	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

Laplace smoothing for bigrams?

- Compare "reconstituted" counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

	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

discount: $d_c = \dfrac{c^*}{c}$

	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

Laplace smoothing for bigrams?

- Compare "reconstituted" counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

	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

discount: $d_c = \dfrac{c^*}{c}$

$d_c(want\ \ to) = 0.39$

	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

Laplace smoothing for bigrams?

- Compare "reconstituted" counts with original: $c^*(w_{i-1}, w_i) = \dfrac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V}$

	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

discount: $d_c = \dfrac{c^*}{c}$

$d_c(want\ to) = 0.39$

$d_c(chinese\ food) = 0.1$

	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

Backoff and interpolation

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.
- Interpolation works better

Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.
- Interpolation works better
- State-of-the-art: Extended interpolated Kneser-Ney smoothing

Linear interpolation

Linear interpolation

- Simple interpolation:

$$\begin{aligned}\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = &\lambda_1 P(w_i \mid w_{i-1}, w_{i-2}) \\ &+ \lambda_2 P(w_i \mid w_{i-1}) \\ &+ \lambda_3 P(w_i)\end{aligned}$$

Linear interpolation

- Simple interpolation:

$$\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 P(w_i \mid w_{i-1}, w_{i-2}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$$

$$\sum_i \lambda_i = 1$$

Linear interpolation

- Simple interpolation:

$$\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 P(w_i \mid w_{i-1}, w_{i-2}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i)$$

$$\sum_i \lambda_i = 1$$

- Better: λ_i depends on specific context

$$\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 \left(w_{i-2}^{i-1}\right) P(w_i \mid w_{i-1}, w_{i-2}) + \lambda_2 \left(w_{i-2}^{i-1}\right) P(w_i \mid w_{i-1}) + \lambda_3 \left(w_{i-2}^{i-1}\right) P(w_i)$$

Dealing with unseen words

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
    - Choose a fixed vocabulary of size *V* in advance

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
  - Choose a fixed vocabulary of size *V* in advance
  - Convert any token in training set not in *V* to <UNK>

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
  - Choose a fixed vocabulary of size *V* in advance
  - Convert any token in training set not in *V* to <UNK>
  - Estimate probabilities of <UNK> just like any other word in the training set.

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
  - Choose a fixed vocabulary of size *V* in advance

    Threshold words by frequency
  - Convert any token in training set not in *V* to <UNK>
  - Estimate probabilities of <UNK> just like any other word in the training set.

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
  - Choose a fixed vocabulary of size *V* in advance

    Threshold words by frequency
  - Convert any token in training set not in *V* to <UNK>
  - Estimate probabilities of <UNK> just like any other word in the training set.
- Another solution: sub-word representations (e.g. FastText word embeddings)

Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; *V* is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: <UNK>
  - Choose a fixed vocabulary of size *V* in advance

    Threshold words by frequency
  - Convert any token in training set not in *V* to <UNK>
  - Estimate probabilities of <UNK> just like any other word in the training set.
- Another solution: sub-word representations (e.g. FastText word embeddings)

skiing = `{^skiing$, ^ski, skii, kiin, iing, ing$}`

Huge, web-scale n-grams

- Google n-gram release:

All Our N-gram are Belong to You

Thursday, August 3, 2006

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

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.

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

Huge, web-scale n-grams

- Google n-gram release:

All Our N-gram are Belong to You

Thursday, August 3, 2006

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

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.

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:     1,024,908,267,229
Number of sentences:     95,119,665,584
Number of unigrams:          13,588,391
Number of bigrams:          314,843,401
Number of trigrams:         977,069,902
Number of fourgrams:      1,313,818,354
Number of fivegrams:      1,176,470,663
```

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

Huge, web-scale n-grams

- Google n-gram release:

All Our N-gram are Belong to You

Thursday, August 3, 2006

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

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.

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
Number of unigrams:         13,588,391
Number of bigrams:         314,843,401
Number of trigrams:        977,069,902
Number of fourgrams:     1,313,818,354
Number of fivegrams:     1,176,470,663
```

⋮

```
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
serve as the industrial 52
serve as the industry 607
serve as the info 42
```

⋮

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

Scaling to huge, web-scale n-grams?

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative
- Computational efficiency:

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative
- Computational efficiency:
  - Efficient data structures like tries.

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative
- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative
- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.
  - Don't use strings: Huffman coding to stuff many words into two bytes.

Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren't informative
- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.
  - Don't use strings: Huffman coding to stuff many words into two bytes.
  - Quantization: store probabilities in 4-8 bits rather than 32 bit float.

Language modeling toolkits

- SRILM

  http://www.speech.sri.com/projects/srilm/

- KenLM

  https://kheafield.com/code/kenlm/

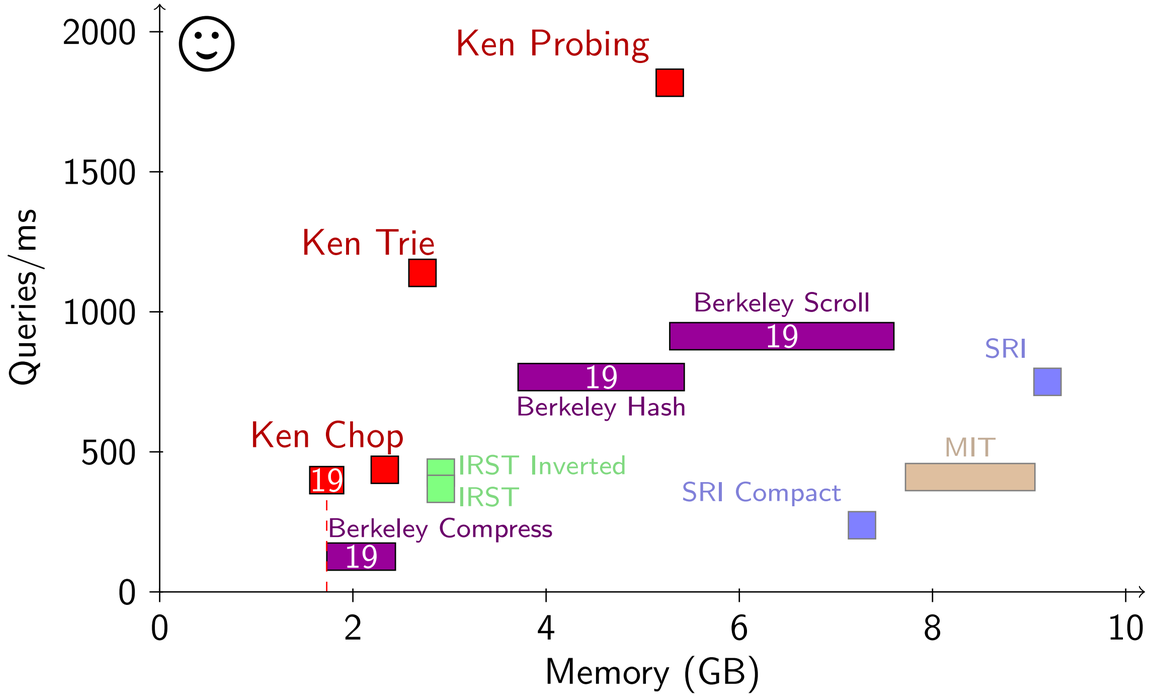
Language modeling toolkits

- SRILM

  http://www.speech.sri.com/projects/srilm/

- KenLM

  https://kheafield.com/code/kenlm/



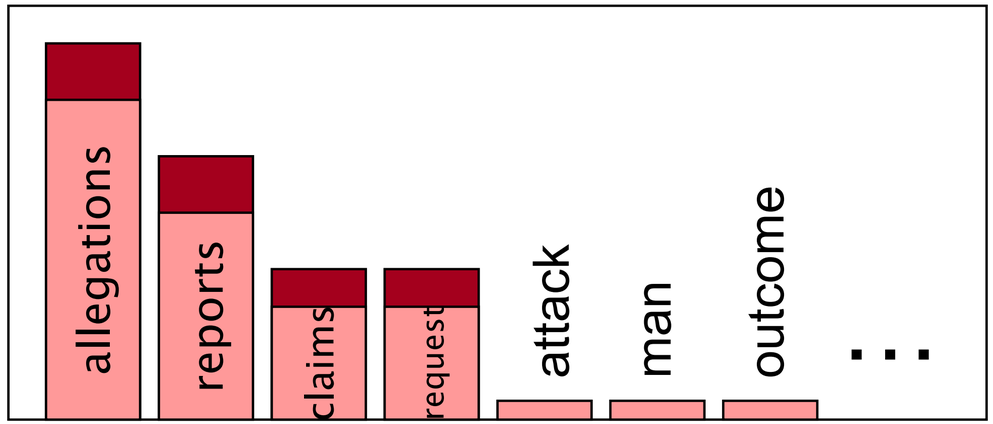
Backoff and interpolation

- Sometimes it helps to use *less* context. Intuition: Condition on less context for contexts you haven't learned much about.
- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.
- Interpolation works better
- State-of-the-art: **Extended interpolated Kneser-Ney smoothing**

Absolute discounting

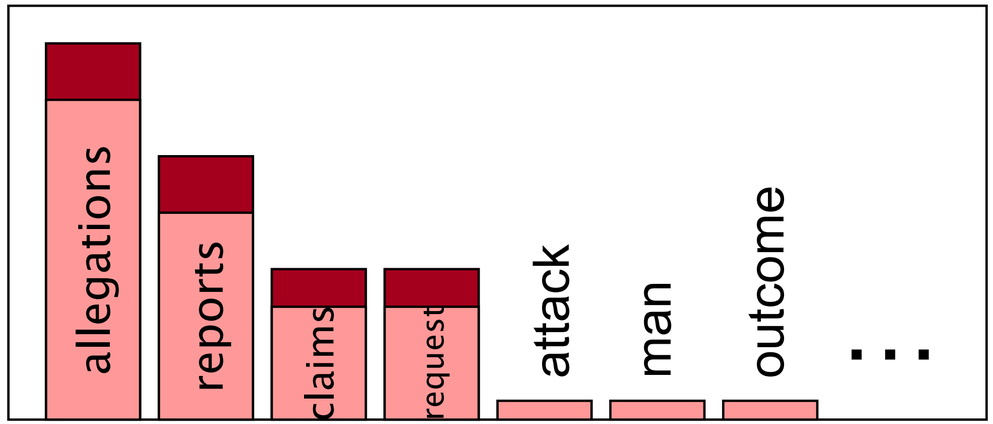
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare



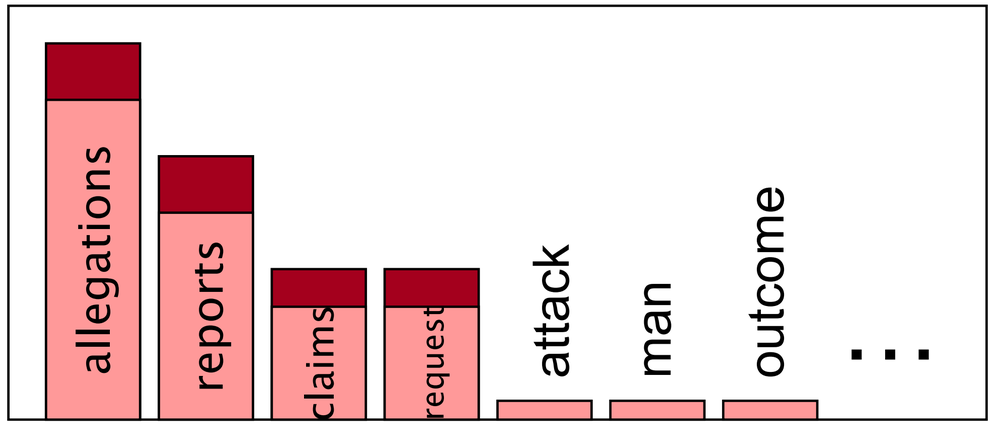
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare
- How much to subtract?



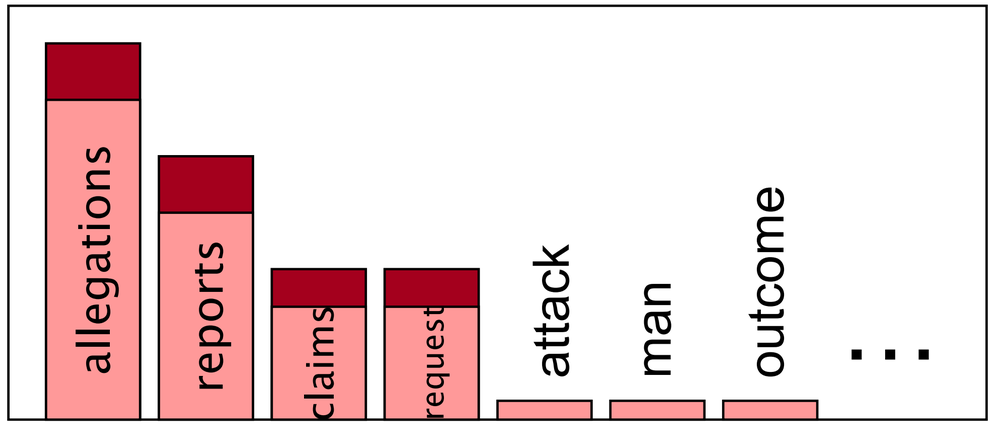
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare
- How much to subtract?
  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?



Absolute discounting

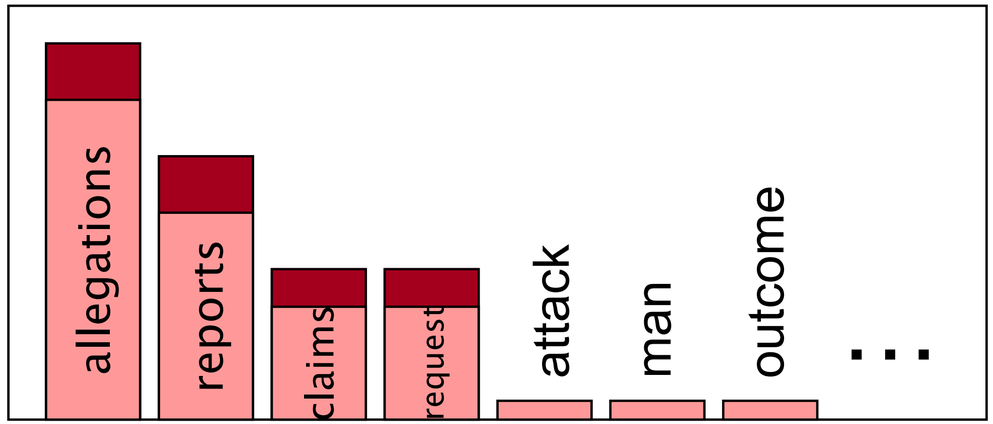
- Removing counts from frequent n-grams, rather than adding to rare
- How much to subtract?
  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?



Bigram count in training	0	1	2	3	4	5	6	7	8	9
Bigram count in held-out	0.0000270	0.448	1.25	2.24	3.23	4.21	5.23	6.21	7.21	8.26
Difference	-0.000027	0.552	0.75	0.76	0.77	0.79	0.77	0.79	0.79	0.74

Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare
- How much to subtract?
  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?



Bigram count in training	0	1	2	3	4	5	6	7	8	9
Bigram count in held-out	0.0000270	0.448	1.25	2.24	3.23	4.21	5.23	6.21	7.21	8.26
Difference	-0.000027	0.552	0.75	0.76	0.77	0.79	0.77	0.79	0.79	0.74

0.75

Absolute discounting interpolation

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount *d* from each count

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount *d* from each count
  - (Maybe keep a couple extra values of *d* for counts 1 and 2.)

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount *d* from each count
  - (Maybe keep a couple extra values of *d* for counts 1 and 2.)
- **Absolute discounting interpolation**:

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount d from each count
  - (Maybe keep a couple extra values of d for counts 1 and 2.)
- **Absolute discounting interpolation**:

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

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount d from each count
  - (Maybe keep a couple extra values of d for counts 1 and 2.)
- **Absolute discounting interpolation**:

$$P(w_i \mid w_{i-1}) = \overbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}^{\text{discounted bigram}} + \lambda(w_{i-1})P(w_i)$$

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount d from each count
  - (Maybe keep a couple extra values of d for counts 1 and 2.)
- **Absolute discounting interpolation**:

$$P(w_i \mid w_{i-1}) = \underbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}_{\text{discounted bigram}} + \underbrace{\lambda(w_{i-1})}_{\text{interpolation weight}} P(w_i)$$

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount *d* from each count
  - (Maybe keep a couple extra values of *d* for counts 1 and 2.)
- **Absolute discounting interpolation**:

$$P(w_i \mid w_{i-1}) = \underbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}_{\text{discounted bigram}} + \underbrace{\lambda(w_{i-1})}_{\text{interpolation weight}} \underbrace{P(w_i)}_{\text{unigram}}$$

Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount *d* from each count
  - (Maybe keep a couple extra values of *d* for counts 1 and 2.)
- **Absolute discounting interpolation**:

$$P(w_i \mid w_{i-1}) = \underbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}_{\text{discounted bigram}} + \underbrace{\lambda(w_{i-1})}_{\text{interpolation weight}} \underbrace{P(w_i)}_{\text{unigram}}$$

- But, should we really use the unmodified unigram $P(w_i)$?

Kneser-Ney Smoothing

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading _____ ?

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___glasses___ ?

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*
- Instead of modeling $P(w_i)$: "How likely is w_i?"

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*
- Instead of modeling $P(w_i)$: "How likely is w_i?"
- $P_{continuation}(w_i)$: "How likely is w_i to appear as a novel *continuation*?"

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*
- Instead of modeling $P(w_i)$: "How likely is w_i?"
- $P_{continuation}(w_i)$: "How likely is w_i to appear as a novel *continuation*?"
  - For each word, count the number of bigram types it completes:

Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can't see without my reading ___Kong___ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*
- Instead of modeling $P(w_i)$: "How likely is w_i?"
- $P_{continuation}(w_i)$: "How likely is w_i to appear as a novel *continuation*?"
  - For each word, count the number of bigram types it completes:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

Kneser-Ney Smoothing

- How many times does w_i appear as a novel continuation:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- How many times does w_i appear as a novel continuation:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalize by the total number of word bigram types:

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

Kneser-Ney Smoothing

- How many times does w_i appear as a novel continuation:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalize by the total number of word bigram types:

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

- To get an estimate for the probability that w_i will appear:

$$P_{continuation}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

Kneser-Ney Smoothing

- How many times does w_i appear as a novel continuation:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalize by the total number of word bigram types:

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

- To get an estimate for the probability that w_i will appear:

$$P_{continuation}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

- A frequent word (Kong) occurring mostly in one context (Hong) will have a low continuation probability.

Kneser-Ney Smoothing

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

- where λ is a normalizing constant to distribute the discounted probability mass:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

- where λ is a normalizing constant to distribute the discounted probability mass:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

- where λ is a normalizing constant to distribute the discounted probability mass:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

normalized discount

Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

- where λ is a normalizing constant to distribute the discounted probability mass:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left|\{w : c(w_{i-1}, w) > 0\}\right|$$

normalized discount

word types that can follow w_{i-1}
= # word types that we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing

Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

where:

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuation\_count}(\cdot) & \text{for lower orders} \end{cases}$$

Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

  where:

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuation\_count}(\cdot) & \text{for lower orders} \end{cases}$$

- Continuation count: number of unique single word contexts for $\cdot$

Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

  where:

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuation\_count}(\cdot) & \text{for lower orders} \end{cases}$$

- Continuation count: number of unique single word contexts for $\cdot$
- **Modified Kneser-Ney smoothing**: Uses three different discounts for n-grams with counts 1, 2, and 3+ [Chen and Goodman 1998].

n-gram language model challenges

n-gram language model challenges

- Bias/variance trade-off

n-gram language model challenges

- Bias/variance trade-off
- Generalization

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____
- Context size

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**. Next class.

n-gram language model challenges

- Bias/variance trade-off
- Generalization

  I'll have a _____ Give me the _____

  I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**. Next class.
  - Better empirical performance.

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**. Next class.
  - Better empirical performance.
  - Provide state-of-the-art dense word representations.

n-gram language model challenges

- Bias/variance trade-off
- Generalization

I'll have a _____ Give me the _____

I'd like to order the _____ I'll take one _____

- Context size

The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**. Next class.
  - Better empirical performance.
  - Provide state-of-the-art dense word representations.
  - But: a lot more computation. Slower to train and infer.

Announcements

- Recitation this week will be a P1 Q&A with Han
- P1 bakeoff: No pre-trained word embeddings or BERT. Out-of-the box linguistic annotations (e.g. pos tags), tf-idf, etc. are fine.