N-gram Language Models

Language model

A statistical **language model** is a probability distribution over sequences of words.

The language model provides context to distinguish between words and phrases that sound phonetically similar. For example, in American English, the phrases "recognize speech" and "wreck a nice beach" sound similar, but mean different things.

A language model is basically a probability distribution over words or word sequences. In practice, a language model gives the probability of a certain word sequence being “valid”. Validity in this context does not refer to grammatical validity at all.

It means that it resembles how people speak (or, to be more precise, write) — which is what the language model learns.

This is an important point: there is no magic to a language model (like other machine learning models, particularly deep neural networks), it is “just” a tool to incorporate abundant information in a concise manner that is reusable in an out-of-sample context.

Data sparsity is a major problem in building language models. Most possible word sequences are not observed in training. One solution is to make the assumption that the probability of a word only depends on the previous *n* words. This is known as an *n*-gram model or unigram model when $n = 1$. The unigram model is also known as the bag of words model.

Estimating the relative likelihood of different phrases is useful in many natural language processing applications, especially those that generate text as an output. Language modeling is used in speech recognition, machine translation, part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition,information retrieval and other applications.

What can a language model do for us?

The abstract understanding of natural language — which is necessary to infer word probabilities from context — can be used for a number of tasks.
Lemmatization or stemming aims to reduce a word to its most basic form, thereby dramatically decreasing the number of tokens.
These algorithms work better if the part-of-speech role of the word is known: a verb's postfixes can be utterly different from a noun's postfixes — hence the rationale for part-of-speech tagging (or POS-tagging), a common task for a language model.
With a good language model, we can perform extractive or abstractive summarization of texts. If we have models for different languages, a machine translation system can be built easily.
Less straightforward use-cases include question answering (with or without context, see the example at the end of the article). Language models can also be used for speech recognition, OCR, handwriting recognition and more.There is a whole spectrum of opportunities.

Types of language models

It is important to note the difference between

a) probabilistic methods

b) neural network based modern language models.

A simple probabilistic language model

is constructed by calculating n-gram probabilities (an n-gram being an n word sequence, n being an integer greater than 0).
An n-gram's probability is the conditional probability that the n-gram's last word follows the a particular n-1 gram (leaving out the last word).
Practically, it is the proportion of occurences of the last word following the n-1 gram leaving the last word out. This concept is a Markov assumption — given the n-1 gram (*the present*), the n-gram probabilities (*future*) does not depend on the n-2, n-3, etc grams (*past*) .

Statistical Language Processing

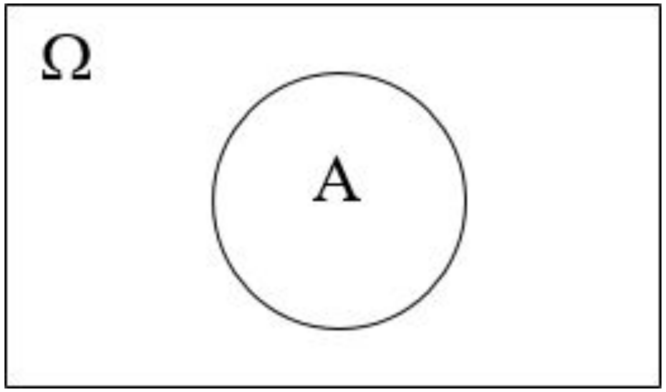
- In the solution of many problems in the natural language processing, **statistical language processing** techniques can be also used.
  - optical character recognition
  - spelling correction
  - speech recognition
  - machine translation
  - part of speech tagging
  - parsing
- Statistical techniques can be used to disambiguate the input.
- They can be used to select the most probable solution.
- Statistical techniques depend on the probability theory.
- To be able to use statistical techniques, we will need corpora to collect statistics.
- Corpora should be big enough to capture the required knowledge.

Basic Probability

- **Probability Theory**: predicting how likely it is that something will happen.
- **Probabilities**: numbers between 0 and 1.
- **Probability Function**:
  - P(A) means that how likely the event A happens.
  - P(A) is a number between 0 and 1
  - P(A)=1 => a certain event
  - P(A)=0 => an impossible event
- **Example**: a coin is tossed three times. What is the probability of 3 heads?
  - 1/8
  - uniform distribution

Probability Spaces

- There is a sample space and the subsets of this sample space describe the events.
- Ω is a sample space.
  - Ω is the certain event
  - the empty set is the impossible event.



P(A) is between 0 and 1

P(Ω) = 1

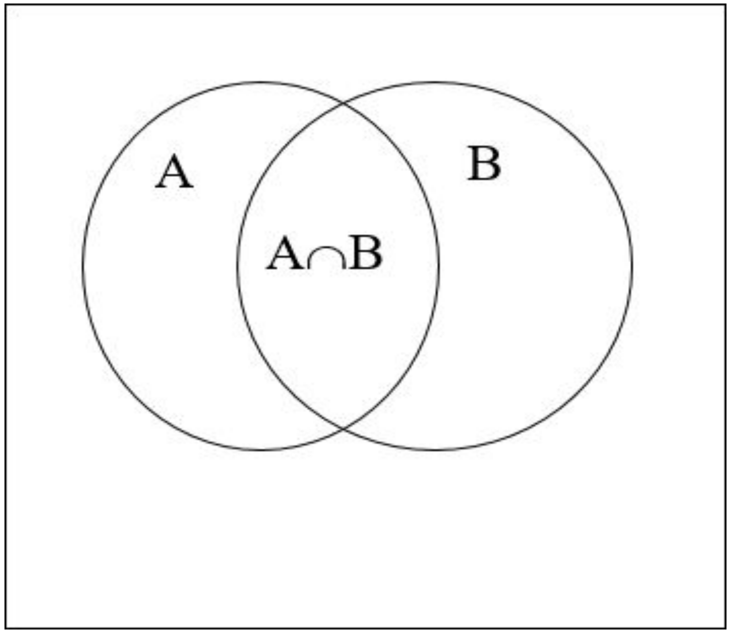
Unconditional and Conditional Probability

- **Unconditional Probability** or **Prior Probability**
  - P(A)
  - the probability of the event A does not depend on other events.

- **Conditional Probability -- Posterior Probability -- Likelihood**
  - P(A|B)
  - this is read as the probability of A given that we know B.

- Example:
  - P(put) is the probability to see the word *put* in a text
  - P(on|put) is the probability to see the word *on* after seeing the word *put*.

Unconditional and Conditional Probability



$P(A|B) = P(A \cap B) / P(B)$

$P(B|A) = P(A \cap B) / P(A)$

Bayes' Theorem

- Bayes' theorem is used to calculate P(A|B) from given P(B|A).
- We know that:

$P(A \cap B) = P(A|B)\, P(B)$

$P(A \cap B) = P(B|A)\, P(A)$

- So, we will have:

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

Language Model

- Models that assign probabilities to sequences of words are called **language models (LMs).**
- The simplest language model that assigns probabilities to sentences and sequences of words is the **n-gram**.
- An **n-gram** is a sequence of N words:
  - A 1-gram (unigram) is a single word sequence of words like “please” or “ turn”.
  - A 2-gram (bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”.
  - A 3-gram (trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.
- We can use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire word sequences.

Probabilistic Language Models

- Probabilistic language models can be used to assign a probability to a sentence in many NLP tasks.
- Machine Translation:
  - P(high winds tonight) > P(large winds tonight)
- Spell Correction:
  - Thek office is about ten minutes from here
  - P(The Office is) > P(Then office is)
- Speech Recognition:
  - P(I saw a van) >> P(eyes awe of an)
- Summarization, question-answering, …

Probabilistic Language Models

- Our goal is to compute the probability of a sentence or sequence of words W ($=w_1,w_2,\ldots w_n$):
  - $P(W) = P(w_1,w_2,w_3,w_4,w_5\ldots w_n)$
- What is the probability of an upcoming word?:
  - $P(w_5|w_1,w_2,w_3,w_4)$
- A model that computes either of these:
  - $P(W)$ or $P(w_n|w_1,w_2\ldots w_{n-1})$ is called a **language model**.

Chain Rule of Probability

- How can we compute probabilities of entire word sequences like $w_1,w_2,\ldots w_n$?
  - The probability of the word sequence $w_1,w_2,\ldots w_n$ is $P(w_1,w_2,\ldots w_n)$.
- We can use the **chain rule of the probability** to decompose this probability:

$$\mathbf{P(w_1^n) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1^2) \ldots P(w_n|w_1^{n-1})}$$

$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

Example:

P(the man from jupiter) =

P(the) P(man|the) P(from|the man) P(jupiter|the man from)

Chain Rule of Probability and Conditional Probabilities

- The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

- Definition of Conditional Probabilities:

$$\mathbf{P(B|A) = P(A,B) / P(A) \quad \Rightarrow \quad P(A,B) = P(A)\, P(B|A)}$$

- Conditional Probabilities with More Variables:

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

- **Chain Rule:**

$$\mathbf{P(w_1 \ldots w_n) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1 w_2) \ldots P(w_n|w_1 \ldots w_{n-1})}$$

Computing Conditional Probabilities

- To compute the exact probability of a word given a long sequence of preceding words is difficult (sometimes impossible).
- We are trying to compute **$P(w_n|w_1 \ldots w_{n-1})$** which is the **probability of seeing w_n after seeing w_1^{n-1}**.
- We may try to compute **$P(w_n|w_1 \ldots w_{n-1})$** exactly as follows:

$$P(w_n|w_1 \ldots w_{n-1}) = count(w_1 \ldots w_{n-1} w_n) \,/\, count(w_1 \ldots w_{n-1})$$

- Too many possible sentences and we may never see enough data for estimating these probability values.
- So, we need to compute **$P(w_n|w_1 \ldots w_{n-1})$** approximately.

N-Grams

- The intuition of the n-gram model (simplifying assumption):
  - instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

$P(w_n \mid w_1 \ldots w_{n-1}) \approx P(w_n)$	**unigram**
$P(w_n \mid w_1 \ldots w_{n-1}) \approx P(w_n \mid w_{n-1})$	**bigram**
$P(w_n \mid w_1 \ldots w_{n-1}) \approx P(w_n \mid w_{n-1} w_{n-2})$	**trigram**
$P(w_n \mid w_1 \ldots w_{n-1}) \approx$ $P(w_n \mid w_{n-1} w_{n-2} w_{n-3})$	**4-gram**
$P(w_n \mid w_1 \ldots w_{n-1}) \approx P(w_n \mid w_{n-1} w_{n-2} w_{n-3} w_{n-4})$	**5-gram**

- In general, **N-Gram** is

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

N-Grams

computing probabilities of word sequences

Unigrams -- $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k)$

Bigrams -- $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$

Trigrams -- $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1} w_{k-2})$

4-grams -- $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1} w_{k-2} w_{k-3})$

N-Grams

computing probabilities of word sequences (Sentences)

Unigram

P(<s> the man from jupiter came </s>) ≈
P(the) P(man) P(from) P(jupiter) P(came)

Bigram

P(<s> the man from jupiter came </s>) ≈
P(the|<*s*>) P(man|the) P(from|man) P(jupiter|from) P(came|jupiter)
P(</s>|came)

Trigram

P(<s> the man from jupiter came </s>) ≈
P(the|<*s*> <*s*>) P(man|<*s*> the) P(from|the man) P(jupiter|man from)
P(came|from jupiter) P(</s>|jupiter came) P(</s>|came </s>)

N-gram models

- In general, a n-gram model is an insufficient model of a language because languages have **long-distance dependencies**.
  - “The **computer(s)** which I had just put into the machine room **is (are)** crashing.”
  - But we can still effectively use N-Gram models to represent languages.
- Which N-Gram should be used as a language model?
  - Bigger N, the model will be more accurate.
    - But we may not get good estimates for N-Gram probabilities.
    - The N-Gram tables will be more sparse.
  - Smaller N, the model will be less accurate.
    - But we may get better estimates for N-Gram probabilities.
    - The N-Gram table will be less sparse.
  - In reality, we do not use higher than Trigram (not more than Bigram).
  - How big are N-Gram tables with 10,000 words?
    - Unigram -- 10,000
    - Bigram – 10,000*10,000 = 100,000,000
    - Trigram – 10,000*10,000*10,000 = 1,000,000,000,000

N-Grams and Markov Models

- The assumption that the probability of a word depends only on the previous word(s) is called **Markov assumption**.
- **Markov models** are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past.
- A **bigram** is called a **first-order** Markov model (because it looks one token into the past);
- A **trigram** is called a **second-order** Markov model;
- In general a **N-Gram** is called a **N-1 order** Markov model.

Estimating N-Gram Probabilities

- Estimating n-gram probabilities is called **maximum likelihood estimation (**or **MLE**).
- We get the MLE estimate for the parameters of an n-gram model by ***getting counts from a corpus***, and **normalizing** the counts so that they lie between 0 and 1.
- Estimating bigram probabilities:

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

where C is the count of that pattern in the corpus

- Estimating N-Gram probabilities

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

Estimating N-Gram Probabilities

A Bigram Example

- ***A mini-corpus:*** We augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word, and special end-symbol </s>.

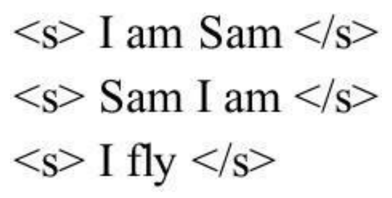
  <s> I am Sam </s>
  <s> Sam I am </s>
  <s> I fly </s>

- ***Unique words:*** I, am, Sam, fly
- ***Bigrams:*** <s> and </s> are also tokens. There are 6(4+2) tokens and 6*6=36 bigrams

P(I|<s>)=2/3	P(Sam|<s>)=1/3	P(am|<s>)=0	P(fly|<s>)=0	P(<s>|<s>)=0	P(</s>|<s>)=0
P(I|I)=0	P(Sam|I)=0	P(am|I)=2/3	P(fly|I)=1/3	P(<s>|I)=0	P(</s>|I)=0
P(I|am)=0	P(Sam|am)=1/2	P(am|am)=0	P(fly|am)=0	P(<s>|am)=0	P(</s>|am)=1/2
P(I|Sam)=1/2	P(Sam|Sam)=0	P(am|Sam)=0	P(fly|Sam)=0	P(<s>|Sam)=0	P(</s>|Sam)=1/2
P(I|fly)=0	P(Sam|fly)=0	P(am|fly)=0	P(fly|fly)=0	P(<s>|fly)=0	P(</s>|fly)=1
P(I|</s>)=0	P(Sam|</s>)=1/3	P(am|</s>)=1/3	P(fly|</s>)=1/3	P(<s>|</s>)=0	P(</s>|</s>)=0

Estimating N-Gram Probabilities

Example


<s> I am Sam </s>
<s> Sam I am </s>
<s> I fly </s>

- **Unigrams:** I, am, Sam, fly

  P(I)=3/8   P(am)=2/8   P(Sam)=2/8   P(fly)=1/8

- **Trigrams:** There are 6*6*6=216 trigrams.
  - Assume there are two tokens <s> <s> at the begining, and two tokens </s> </s> at the end.

P(I|<s> <s>)=2/3

P(Sam|<s> <s>)=1/3

P(am|<s> I)=1/2

P(fly|<s> I)=1/3

P(I|<s> Sam)=1

P(Sam|I am)=1/2

P(</s>|I am)=1/2

P(</s>|am Sam)=1

P(</s>|Sam </s>)=1

Estimating N-Gram Probabilities

Corpus: Berkeley Restaurant Project Sentences

- There are 9222 sentences in the corpus.
- Raw biagram counts of 8 words (out of 1446 words)

	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

Estimating N-Gram Probabilities

Corpus: Berkeley Restaurant Project 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

- Unigram counts:

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

- Normalize bigrams by unigram counts:

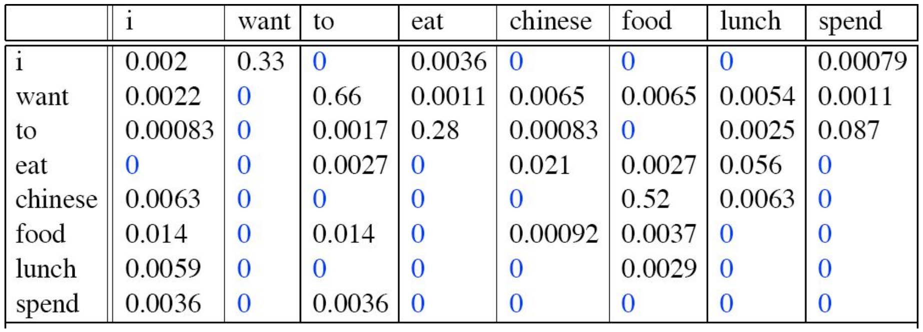
	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059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

Bigram Estimates of Sentence Probabilities

- Some other bigrams:

  P(i|<s>)=0.25 P(english|want)=0.0011

  P(food|english)=0.5 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

- Compute the probability of sentence **I want English food**

P(<s> i want english food </s>)

= P(i|<s>) P(want|i) P(english|want) P(food|english) P(</s>|food)

= 0.25*0.33*0.0011*0.5*0.68

= 0.000031

Log Probabilities

- Since probabilities are less than or equal to 1, the more probabilities we multiply together, the ***smaller the product becomes***.
  - Multiplying enough n-grams together would result in **numerical underflow**.
- By using **log probabilities** instead of ***raw probabilities***, we get numbers that are not as small.
  - Adding in log space is equivalent to multiplying in linear space, so we combine log probabilities by adding them.
    - adding is faster than multiplying
  - The result of doing all computation and storage in log space is that we only need to convert back into probabilities if we need to report them at the end

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

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

Evaluating Language Models

- 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*.

Evaluating Language Models

Extrinsic Evaluation

- **Extrinsic Evaluation** of a N-gram *language model* is to use it in an application and measure how much the application improves.
- To compare two language models A and B:
  - Use each of language model in a task such as spelling corrector, MT system.
  - Get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B
    - The model produces the better accuracy is the better model.
- Extrinsic evaluation can be time-consuming.

Evaluating Language Models

Intrinsic Evaluation

- An **intrinsic evaluation** metric is one that measures the quality of a model independent of any application.
- When a corpus of text is given and to compare two different n-gram models,
  - Divide the data into ***training*** and ***test sets***,
  - Train the parameters of both models on the *training set*, and
  - Compare how well the two trained models fit the test set.
    - Whichever model assigns a higher probability to the test set
- In practice, *probability-based metric* called **perplexity** is used instead of raw probability as our metric for evaluating language models.

Evaluating Language Models

Perplexity

- The ***best language model*** is one that best predicts an unseen test set
  - Gives the highest P(testset)
- The **perplexity** of a language model on a test set is the *inverse probability of the test set*, normalized by the number of words.
- **Minimizing perplexity is the same as maximizing probability**
- The **perpelexity PP** for a test set W=$w_1w_2 \ldots w_n$ is

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1w_2\ldots w_N)}} = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1\ldots w_{i-1})}}$$

- The **perpelexity PP** for bigrams:

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

Evaluating Language Models

Perplexity as branching factor

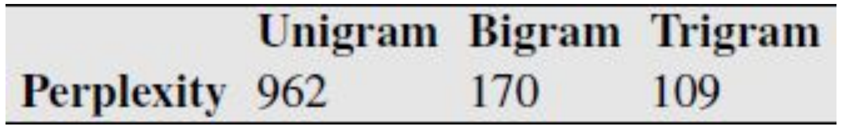
- Perplexity can be seen as the weighted average *branching factor of a language*.
  - The branching factor of a language is the number of possible next words that can follow any word.
- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

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

Evaluating Language Models

Perplexity

- Lower perplexity = better model
- Training 38 million words, test 1.5 million words, WSJ



	Unigram	Bigram	Trigram
Perplexity	962	170	109

- An intrinsic improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation.
  - Nonetheless, because perplexity often correlates with such improvements, it is commonly used as a quick check on an algorithm.
  - But a model's improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.