Language Models

A famous quote

It must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.

Noam Chomsky, 1969

A famous quote

It must be recognized that the notion “probability of a sentence” is an entirely useless one, under any known interpretation of this term.

Noam Chomsky, 1969

- “useless”: To everyone? To linguists?
- “known interpretation”: What are possible interpretations?

Intuitive interpretation

- "Probability of a sentence" = how likely is it to occur in natural language
  - Consider only a specific language (English)
  - Not including meta-language (e.g. linguistic discussion)

$$\mathrm{P}(\text{She studies morphosyntax}) > \mathrm{P}(\text{She studies more faux syntax})$$

Automatic speech recognition

Sentence probabilities (**language model**) help decide between similar-sounding options.

speech input		
↓	(Acoustic model)	
possible outputs		She studies morphosyntax She studies more faux syntax She's studies morph or syntax ...
↓	(Language model)	
best-guess output		She studies morphosyntax

Machine translation

Sentence probabilities help decide word choice and word order.

non-English input

$\downarrow$ (Translation model)

possible outputs

She is going home
She is going house
She is traveling to home
To home she is going
...

$\downarrow$ (Language model)

best-guess output

She is going home

But, what about zero probability sentences?

the Archaeopteryx winged jaggedly amidst foliage

vs

jaggedly trees the on flew

- Neither has ever occurred before.
  $\Rightarrow$ both have zero probability.

- But one is grammatical (and meaningful), the other not.
  $\Rightarrow$ "Sentence probability" is useless as a measure of grammaticality.

The logical flaw

- "Probability of a sentence" = how likely is it to occur in natural language.
- Is the following statement true?

  $$\text{Sentence has never occurred} \Rightarrow \text{sentence has zero probability}$$

- More generally, is this one?

  $$\text{Event has never occurred} \Rightarrow \text{event has zero probability}$$

Events that have never occurred

- Each of these events has never occurred:

  My hair turns blue
  I injure myself in a skiing accident
  I travel to Finland

- Yet, they clearly have different (and non-zero!) probabilities.

Events that have never occurred

- Each of these events has never occurred:

  My hair turns blue
  I injure myself in a skiing accident
  I travel to Finland

- Yet, they clearly have differing (and non-zero!) probabilities.

- Most sentences (and events) have never occurred.
  - This doesn't make their probabilities zero (or meaningless), but
  - it does make **estimating** their probabilities trickier.

Probability theory vs estimation

- Probability theory can solve problems like:
  - I have a jar with 6 blue marbles and 4 red ones.
  - If I choose a marble uniformly at random, what's the probability it's red?
- But what about:
  - I have a jar of marbles.
  - I repeatedly choose a marble uniformly at random and then replace it before choosing again.
  - In ten draws, I get 6 blue marbles and 4 red ones.
  - On the next draw, what's the probability I get a red marble?
- The latter also requires estimation theory.

Example: weather forecasting

What is the probability that it will rain tomorrow?

- To answer this question, we need
  - data: measurements of relevant info (e.g., humidity, wind speed/direction, temperature).
  - model: equations/procedures to estimate the probability using the data.

- In fact, to build the model, we will need data (including *outcomes*) from previous situations as well.

Example: weather forecasting

What is the probability that it will rain tomorrow?

- To answer this question, we need
  - data: measurements of relevant info (e.g., humidity, wind speed/direction, temperature).
  - model: equations/procedures to estimate the probability using the data.

- In fact, to build the model, we will need data (including *outcomes*) from previous situations as well.

- Note that we will never know the "true" probability of rain $P(\text{rain})$, only our estimated probability $\hat{P}(\text{rain})$.

Example: language model

What is the probability of sentence $\vec{w} = w_1 \dots w_n$?

- To answer this question, we need
  - data: words $w_1 \dots w_n$, plus a large corpus of sentences ("previous situations", or **training data**).
  - model: equations to estimate the probability using the data.
- Different models will yield different estimates, even with the same data.
- Deep question: what model/estimation method do humans use?

How to get better probability estimates

Better estimates definitely help in language technology. How to improve them?

- **More training data.** Limited by time, money. (Varies a lot!)
- **Better model.** Limited by scientific and mathematical knowledge, computational resources
- **Better estimation method.** Limited by mathematical knowledge, computational resources

We will return to the question of how to know if estimates are "better".

Example: estimation for coins

I flip a coin 10 times, getting 7T, 3H. What is $\hat{P}(\mathrm{T})$?

Example: estimation for coins

I flip a coin 10 times, getting 7T, 3H. What is $\hat{P}(\mathrm{T})$?

- **Model 1:** Coin is fair. Then, $\hat{P}(T) = 0.5$

Example: estimation for coins

I flip a coin 10 times, getting 7T, 3H. What is $\hat{P}(\mathrm{T})$?

- **Model 1:** Coin is fair. Then, $\hat{P}(T) = 0.5$
- **Model 2:** Coin is not fair. Then, $\hat{P}(T) = 0.7$ (why?)

Example: estimation for coins

I flip a coin 10 times, getting 7T, 3H. What is $\hat{P}(\mathrm{T})$?

- **Model 1:** Coin is fair. Then, $\hat{P}(T) = 0.5$
- **Model 2:** Coin is not fair. Then, $\hat{P}(T) = 0.7$ (why?)
- **Model 3:** Two coins, one fair and one not; choose one at random to flip 10 times. Then, $0.5 < \hat{P}(T) < 0.7$.

Example: estimation for coins

I flip a coin 10 times, getting 7T, 3H. What is $\hat{P}(\mathrm{T})$?

- **Model 1:** Coin is fair. Then, $\hat{P}(T) = 0.5$
- **Model 2:** Coin is not fair. Then, $\hat{P}(T) = 0.7$ (why?)
- **Model 3:** Two coins, one fair and one not; choose one at random to flip 10 times. Then, $0.5 < \hat{P}(T) < 0.7$.

Each is a **generative model**: a probabilistic process that describes how the data were generated.

Defining a model

Usually, two choices in defining a model:

- **Structure** (or **form**) of the model: the form of the equations, usually determined by knowledge about the problem.
- **Parameters** of the model: specific values in the equations that are usually determined using the training data.

Example: M&M colors

What is the proportion of each color of M&M?

- Assume a **discrete distribution** with parameters θ.
  - θ is a vector! That is, $\theta = (\theta_{\mathrm{R}}, \theta_{\mathrm{O}}, \theta_{\mathrm{Y}}, \theta_{\mathrm{G}}, \theta_{\mathrm{BL}}, \theta_{\mathrm{BR}})$.
  - For discrete distribution, the parameters ARE the probabilities, e.g., $P(\text{red}) = \theta_{\mathrm{R}}$.

Example: M&M colors

What is the proportion of each color of M&M?

- Assume a **discrete distribution** with parameters θ.
  - θ is a vector! That is, $\theta = (\theta_{\mathrm{R}}, \theta_{\mathrm{O}}, \theta_{\mathrm{Y}}, \theta_{\mathrm{G}}, \theta_{\mathrm{BL}}, \theta_{\mathrm{BR}})$.
  - For discrete distribution, params ARE the probabilities, e.g., $P(\text{red}) = \theta_{\mathrm{R}}$.
- In 48 packages, I find[1] 2620 M&Ms, as follows:

Red	Orange	Yellow	Green	Blue	Brown
372	544	369	483	481	371

- How to estimate θ from this data?

[1] Actually, data from: https://joshmadison.com/2007/12/02/mms-color-distribution-analysis/

Relative frequency estimation

- Intuitive way to estimate discrete probabilities:

$$P_{\mathrm{RF}}(x) = \frac{C(x)}{N}$$

  where $C(x)$ is the count of x in a large dataset, and $N = \sum_{x'} C(x')$ is the total number of items in the dataset.

- M&M example: $P_{\mathrm{RF}}(\text{red}) = \hat{\theta}_{\mathrm{R}} = \frac{372}{2620} = .142$

- Or, could estimate probability of word w from a large corpus.

- Can we justify this mathematically?

Formalizing the estimation problem

- What is the best choice of θ given the data d that we saw?

- Formalize using Bayes' Rule, try to maximize $P(\theta|d)$.

$$P(\theta|d) = \frac{P(d|\theta)P(\theta)}{P(d)}$$

  - $P(\theta)$: **prior** probability of θ
  - $P(d|\theta)$: **likelihood**
  - $P(\theta|d)$: **posterior** probability of θ given d

Formalizing the estimation problem

- Notation: choose $\hat{\theta}$ to be the θ that maximizes $P(\theta|d)$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(\theta|d)$$

Formalizing the estimation problem

- Notation: choose $\hat{\theta}$ to be the θ that maximizes $P(\theta|d)$:

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(\theta|d)$$

- Rewrite with Bayes' Rule, and $P(d)$ doesn't depend on θ, so:

$$\hat{\theta} = \operatorname*{argmax}_{\theta} \frac{P(d|\theta)P(\theta)}{P(d)}$$

$$= \operatorname*{argmax}_{\theta} P(d|\theta)P(\theta)$$

Maximum-likelihood estimation

- Not obvious how to choose prior... so maybe don't use one?

$$\underset{\theta}{\operatorname{argmax}} P(d|\theta)P(\theta) = \underset{\theta}{\operatorname{argmax}} P(d|\theta)$$

- Just choose θ to maximize the likelihood.
  - the parameters that make the observed data most probable

- This turns out to be just the relative frequency estimator, i.e.,

$$P_{\mathrm{ML}}(x) = P_{\mathrm{RF}}(x) = \frac{C(x)}{N}$$

Likelihood example

- For a fixed dataset, the likelihood depends on the model we use.
- Our coin example: $\theta = (\theta_{\mathrm{H}}, \theta_{\mathrm{T}})$. Suppose $d =$ HTTTHTHTTT.
- **Model 1:** Assume coin is fair, so $\hat{\theta} = (0.5, 0.5)$.
  - Likelihood of this model:
    $P(\text{HTTTHTHTTT}|\hat{\theta}) = (0.5)^3 \cdot (0.5)^7 = 0.00097$

Likelihood example

- For a fixed dataset, the likelihood depends on the model we use.
- Our coin example: $\theta = (\theta_{\mathrm{H}}, \theta_{\mathrm{T}})$. Suppose $d =$ HTTTHTHTTT.
- **Model 1:** Assume coin is fair, so $\hat{\theta} = (0.5, 0.5)$.
  - Likelihood of this model:
    $P(\text{HTTTHTHTTT}|\hat{\theta}) = (0.5)^3 \cdot (0.5)^7 = 0.00097$
- **Model 2:** Use ML estimation, so $\hat{\theta} = (0.3, 0.7)$.
  - Likelihood of this model: $(0.3)^3 \cdot (0.7)^7 = 0.00222$
- Maximum-likelihood estimate does have higher likelihood!

Recap: Language models

- **Language models** tell us $P(\vec{w}) = P(w_1 \ldots w_n)$: *How likely to occur is this sequence of words?*

  Roughly: *Is this sequence of words a "good" one in my language?*

Example uses of language models

- Machine translation: reordering, word choice.

$$\begin{aligned} P_{\text{LM}}(\text{the house is small}) &> P_{\text{LM}}(\text{small the is house}) \\ P_{\text{LM}}(\text{I am going home}) &> P_{\text{LM}}(\text{I am going house}) \\ P_{\text{LM}}(\text{We'll start eating}) &> P_{\text{LM}}(\text{We shall commence consuming}) \end{aligned}$$

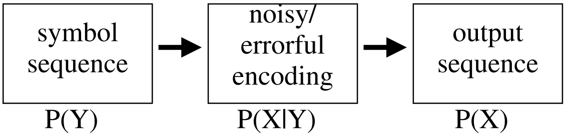
- Speech recognition: word choice:

$$\begin{aligned} P_{\text{LM}}(\text{morphosyntactic analyses}) &> P_{\text{LM}}(\text{more faux syntactic analyses}) \\ P_{\text{LM}}(\text{I put it on today}) &> P_{\text{LM}}(\text{I putted onto day}) \end{aligned}$$

But: How do systems use this information?

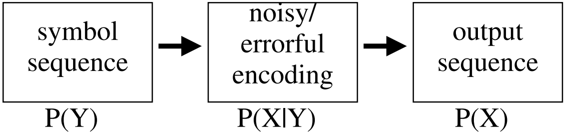
Noisy channel model

- Concept from Information Theory, used widely in NLP
- We imagine that the observed data (output sequence) was generated as:



Noisy channel model

- Concept from Information Theory, used widely in NLP
- We imagine that the observed data (output sequence) was generated as:



Application	Y	X
Speech recognition	true words	acoustic signal
Machine translation	words in L_1	words in L_2
Spelling correction	true words	typed words

Example: spelling correction

- $P(Y)$: Distribution over the words the user intended to type. A language model!
- $P(X|Y)$: Distribution describing what user is **likely** to type, given what they **meant**. Could incorporate information about common spelling errors, key positions, etc. Call it a **noise model**.
- $P(X)$: Resulting distribution over what we actually see.
- Given some particular observation x (say, effert), we want to recover the most probable y that was intended.

Noisy channel as probabilistic inference

- Mathematically, what we want is $\mathrm{argmax}_y P(y|x)$.
  - Read as "the y that maximizes $P(y|x)$"
- Rewrite using Bayes' Rule:

$$\begin{aligned} \operatorname*{argmax}_y P(y|x) &= \operatorname*{argmax}_y \frac{P(x|y)P(y)}{P(x)} \\ &= \operatorname*{argmax}_y P(x|y)P(y) \end{aligned}$$

Noisy channel as probabilistic inference

So to recover the best y, we will need

- a **language model** $P(Y)$: relatively task-independent.
- a **noise model** $P(X|Y)$, which depends on the task.
  - acoustic model, translation model, misspelling model, etc.

Both are normally trained on corpus data.

Estimating a language model

- Y is really a sequence of words $\vec{w} = w_1 \dots w_n$.
- So we want to know $P(w_1 \dots w_n)$ for big n (e.g., sentence).
- What will not work: try to directly estimate probability of each full sentence.
  - Say, using MLE (relative frequencies): $C(\vec{w})/(\text{tot } \# \text{ sentences})$.
  - For nearly all $\vec{w}$ (grammatical or not), $C(\vec{w}) = 0$.
  - A **sparse data** problem: not enough observations to estimate probabilities well.

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $$\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$$
- So, $\mathrm{P(the\ cat\ slept\ quietly)} = \mathrm{P(the\ quietly\ cat\ slept)}$

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $$\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$$
- So, $\mathrm{P(the\ cat\ slept\ quietly)} = \mathrm{P(the\ quietly\ cat\ slept)}$
  - Not a *good* model, but still a model.
- Of course, $P(w_i)$ also needs to be estimated!

MLE for unigrams

- How to estimate $P(w)$, e.g., $P(\text{the})$?
- MLE is just relative frequencies:

$$P_{\mathrm{ML}}(w) = \frac{C(w)}{W}$$

  - $C(w)$ is the token count of w in a large corpus
  - $W = \sum_{x'} C(x')$ is the total number of word tokens in the corpus.

Unigram models in practice

- Seems like a pretty bad model of language: probability of word obviously *does* depend on context.
- Yet unigram (or **bag-of-words**) models are surprisingly useful for some applications.
  - Can model "aboutness": topic of a document, semantic usage of a word
  - Applications: lexical semantics (disambiguation), information retrieval, text classification. (See later in this course)
  - But, we will focus on models that capture at least some syntactic information.

General N-gram language models

Step 1: rewrite using chain rule:

$$\begin{aligned} P(\vec{w}) &= P(w_1 \ldots w_n) \\ &= P(w_n|w_1, w_2, \ldots, w_{n-1})P(w_{n-1}|w_1, w_2, \ldots, w_{n-2}) \ldots P(w_1) \end{aligned}$$

- Example: $\vec{w} = \text{the cat slept quietly yesterday}$.

  $P(\text{the, cat, slept, quietly, yesterday}) =$
  $P(\text{yesterday}|\text{the, cat, slept, quietly}) \cdot P(\text{quietly}|\text{the, cat, slept}) \cdot$
  $P(\text{slept}|\text{the, cat}) \cdot P(\text{cat}|\text{the}) \cdot P(\text{the})$

- But for long sequences, many of the conditional probs are also too sparse!

General N-gram language models

Step 2: make an independence assumption:

$$\begin{aligned} P(\vec{w}) &= P(w_1 \dots w_n) \\ &= P(w_n|w_1, w_2, \dots, w_{n-1})P(w_{n-1}|w_1, w_2, \dots, w_{n-2}) \dots P(w_1) \\ &\approx P(w_n|w_{n-2}, w_{n-1})P(w_{n-1}|w_{n-3}, w_{n-2}) \dots P(w_1) \end{aligned}$$

- **Markov** assumption: only a finite history matters.
- Here, two word history (**trigram** model):
  w_i is cond. indep. of $w_1 \dots w_{i-3}$ given w_{i-1}, w_{i-2}.
  $P(\text{the, cat, slept, quietly, yesterday}) \approx$
  $P(\text{yesterday}|\text{slept, quietly}) \cdot P(\text{quietly}|\text{cat, slept}) \cdot$
  $P(\text{slept}|\text{the, cat}) \cdot P(\text{cat}|\text{the}) \cdot P(\text{the})$

Trigram independence assumption

- Put another way, a trigram model assumes these are all equal:
  - $P(\text{slept}|\text{the cat})$
  - $P(\text{slept}|\text{after lunch the cat})$
  - $P(\text{slept}|\text{the dog chased the cat})$
  - $P(\text{slept}|\text{except for the cat})$

  because all are estimated as $P(\text{slept}|\text{the cat})$

- Not always a good assumption! But it does reduce the sparse data problem.

Another example: bigram model

- Bigram model assumes one word history:

$$P(\vec{w}) = P(w_1) \prod_{i=2}^{n} P(w_i | w_{i-1})$$

- But consider these sentences:

	w_1	w_2	w_3	w_4
(1)	the	cats	slept	quietly
(2)	feeds	cats	slept	quietly
(3)	the	cats	slept	on

- What's wrong with (2) and (3)? Does the model capture these problems?

Example: bigram model

- To capture behaviour at beginning/end of sentence, we need to augment the input:

	w_0	w_1	w_2	w_3	w_4	w_5
(1)	<s>	the	cats	slept	quietly	</s>
(2)	<s>	feeds	cats	slept	quietly	</s>
(3)	<s>	the	cats	slept	on	</s>

- That is, assume $w_0 =$ `<s>` and $w_{n+1} =$ `</s>` so we have:

$$P(\vec{w}) = P(w_0) \prod_{i=1}^{n+1} P(w_i|w_{i-1}) = \prod_{i=1}^{n+1} P(w_i|w_{i-1})$$

Estimating N-Gram Probabilities

- Maximum likelihood (relative frequency) estimation for bigrams:
  - How many times we saw w_2 following w_1,

    out of all the times we saw *anything* following w_1:

$$\begin{aligned} P_{\mathrm{ML}}(w_2|w_1) &= \frac{C(w_1, w_2)}{C(w_1, \cdot)} \\ &= \frac{C(w_1, w_2)}{C(w_1)} \end{aligned}$$

Estimating N-Gram Probabilities

- Similarly for trigrams:

$$P_{\mathrm{ML}}(w_3|w_1, w_2) = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)}$$

- Collect counts over a large text corpus
  - Millions to billions of words are usually easy to get
  - (trillions of English words available on the web)

Evaluating a language model

- Intuitively, trigram model captures more context than bigram model, so should be a “better” model.
- That is, more accurately predict the probabilities of sentences.
- But how can we measure this?

Entropy

- Definition of the **entropy** of a random variable X:

  $H(X) = \sum_x -P(x)\ \log_2 P(x)$

- Intuitively: a measure of uncertainty/disorder

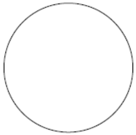
- Also: the expected value of $-\log_2 P(X)$

Entropy Example

One event (outcome)

$P(a) = 1$

$$\begin{aligned} H(X) &= -1\log_2 1 \\ &= 0 \end{aligned}$$

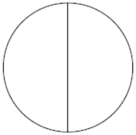


Entropy Example

2 equally likely events:

$$P(a) = 0.5$$
$$P(b) = 0.5$$

$$\begin{aligned} H(X) &= -0.5\log_2 0.5 - 0.5\log_2 0.5 \\ &= -\log_2 0.5 \\ &= 1 \end{aligned}$$

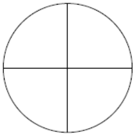


Entropy Example

4 equally likely events:

$$P(a) = 0.25$$
$$P(b) = 0.25$$
$$P(c) = 0.25$$
$$P(d) = 0.25$$

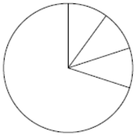
$$\begin{aligned} H(X) = & -0.25 \log_2 0.25 - 0.25 \log_2 0.25 \\ & -0.25 \log_2 0.25 - 0.25 \log_2 0.25 \\ = & -\log_2 0.25 \\ = & \, 2 \end{aligned}$$



Entropy Example

3 equally likely events and one more likely than the others:

$P(a) = 0.7$
$P(b) = 0.1$
$P(c) = 0.1$
$P(d) = 0.1$

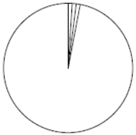


$$\begin{aligned} H(X) = & -0.7\log_2 0.7 - 0.1\log_2 0.1 \\ & -0.1\log_2 0.1 - 0.1\log_2 0.1 \\ = & -0.7\log_2 0.7 - 0.3\log_2 0.1 \\ = & -(0.7)(-0.5146) - (0.3)(-3.3219) \\ = & \, 0.36020 + 0.99658 \\ = & \, 1.35678 \end{aligned}$$

Entropy Example

3 equally likely events and one much more likely than the others:

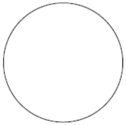
$P(a) = 0.97$
$P(b) = 0.01$
$P(c) = 0.01$
$P(d) = 0.01$



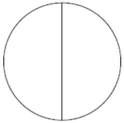
$$\begin{aligned} H(X) = & -0.97\log_2 0.97 - 0.01\log_2 0.01 \\ & -0.01\log_2 0.01 - 0.01\log_2 0.01 \\ = & -0.97\log_2 0.97 - 0.03\log_2 0.01 \\ = & -(0.97)(-0.04394) - (0.03)(-6.64 \\ = & \, 0.04262 + 0.19932 \\ = & \, 0.24194 \end{aligned}$$

Entropy take-aways

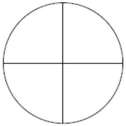
Entropy of a **uniform distribution** over N outcomes is $\log_2 N$:



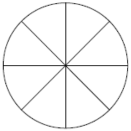
$H(X) = 0$



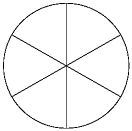
$H(X) = 1$



$H(X) = 2$



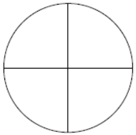
$H(X) = 3$



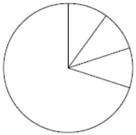
$H(X) = 2.585$

Entropy take-aways

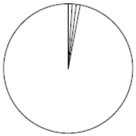
Any **non-uniform** distribution over N outcomes has **lower** entropy than the corresponding uniform distribution:



$H(X) = 2$



$H(X) = 1.35678$



$H(X) = 0.24194$

The Entropy of English

- Given the start of a text, can we guess the next word?
- Humans do pretty well: the entropy is only about 1.3.
- But what about N-gram models?
  - Ideal language model would match the true entropy of English.
  - The closer we get to that, the better the model.
  - Put another way, a good model assigns high prob to real sentences (and low prob to everything else).

How good is the LM?

- **Cross entropy** measures how well model M predicts the data.
- For data $w_1 \dots w_n$ with large n, well approximated by:

$$H_M(w_1 \dots w_n) = -\frac{1}{n} \log_2 P_M(w_1 \dots w_n)$$

  - Avg neg log prob our model assigns to each word we saw
- Or, **perplexity**:

$$\mathsf{PP}_M(\vec{w}) = 2^{H_M(\vec{w})}$$

Example: trigram (Europarl)

prediction	P_{ML}	$-\log_2 P_{\text{ML}}$
$P_{\text{ML}}(\text{i}\mid\text{</s><s>})$	0.109	3.197
$P_{\text{ML}}(\text{would}\mid\text{<s>i})$	0.144	2.791
$P_{\text{ML}}(\text{like}\mid\text{i would})$	0.489	1.031
$P_{\text{ML}}(\text{to}\mid\text{would like})$	0.905	0.144
$P_{\text{ML}}(\text{commend}\mid\text{like to})$	0.002	8.794
$P_{\text{ML}}(\text{the}\mid\text{to commend})$	0.472	1.084
$P_{\text{ML}}(\text{rapporteur}\mid\text{commend the})$	0.147	2.763
$P_{\text{ML}}(\text{on}\mid\text{the rapporteur})$	0.056	4.150
$P_{\text{ML}}(\text{his}\mid\text{rapporteur on})$	0.194	2.367
$P_{\text{ML}}(\text{work}\mid\text{on his})$	0.089	3.498
$P_{\text{ML}}(.\mid\text{his work})$	0.290	1.785
$P_{\text{ML}}(\text{</s>}\mid\text{work .})$	0.99999	0.000014
	average	2.634

Comparison: 1–4-Gram

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Unseen N-Grams

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus

Unseen N-Grams

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus
  - $\rightarrow$ $P(\text{consuming}|\text{shall commence}) = 0$
- Any sentence with shall commence consuming will be assigned probability 0

The guests shall commence consuming supper
Green inked shall commence consuming garden the

The problem with MLE

- MLE estimates probabilities that make the observed data maximally probable
- by assuming anything unseen cannot happen (and also assigning too *much* probability to low-frequency observed events).
- It **over-fits** the training data.
- We tried to avoid zero-probability sentences by modelling with smaller chunks (n-grams).
- But even these will sometimes have zero prob under MLE.

Recap: N-gram models

- We can model sentence probs by conditioning each word on $N-1$ previous words.
- For example, a bigram model:

$$P(\vec{w}) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

- Or trigram model:

$$P(\vec{w}) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1})$$

Recap: MLE estimates for N-grams

- To estimate each word prob, we could use MLE...

$$P_{\mathrm{ML}}(w_2|w_1) \;=\; \frac{C(w_1, w_2)}{C(w_1)}$$

- But even with a large training corpus, not all N-grams will be observed (we'll still have some zeros).

- **Smoothing** methods reassign some probability mass from observed to unobserved events.

Add-One Smoothing

- For all possible bigrams, add one more count.

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

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

Add-One Smoothing

- For all possible bigrams, add one more count.

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

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

- NO! Sum over possible w_i (in vocabulary V) must equal 1:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = 1$$

- True for P_{ML} but we increased the numerator; must change denominator too.

Add-One Smoothing: normalization

- We want: $$\sum_{w_i \in V} \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + x} = 1$$
- Solve for x: $$\begin{aligned} \sum_{w_i \in V} (C(w_{i-1}, w_i) + 1) &= C(w_{i-1}) + x \\ \sum_{w_i \in V} C(w_{i-1}, w_i) + \sum_{w_i \in V} 1 &= C(w_{i-1}) + x \\ C(w_{i-1}) + v &= C(w_{i-1}) + x \end{aligned}$$
- So, $P_{+1}(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + v}$ where $v =$ vocabulary size.

Add-One Smoothing: effects

- Add-one smoothing:

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

- Large vobulary size means v is often much larger than $C(w_{i-1})$, overpowers actual counts.

- Example: in Europarl, $v = 86,700$ word types (30m tokens, max $C(w_{i-1})$ = 2m).

Add-One Smoothing: effects

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

Using $v = 86,700$ compute some example probabilities:

$C(w_{i-1}) = 10,000$

$C(w_{i-1}, w_i)$	$P_{\mathrm{ML}} =$	$P_{+1} \approx$
100	1/100	1/970
10	1/1k	1/10k
1	1/10k	1/48k
0	0	1/97k

$C(w_{i-1}) = 100$

$C(w_{i-1}, w_i)$	$P_{\mathrm{ML}} =$	$P_{+1} \approx$
100	1	1/870
10	1/10	1/9k
1	1/100	1/43k
0	0	1/87k

The problem with Add-One smoothing

- All smoothing methods “steal from the rich to give to the poor”
- Add-one smoothing steals way too much
- ML estimates for frequent events are quite accurate, don’t want smoothing to change these much.

Add-α Smoothing

- Add $\alpha < 1$ to each count

$$P_{+\alpha}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha v}$$

- Simplifying notation: c is n-gram count, n is history count

$$P_{+\alpha} = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?

Optimizing α

- Divide corpus into **training** set (80-90%), **held-out** (or **development** or **validation**) set (5-10%), and **test** set (5-10%)
- Train model (estimate probabilities) on training set with different values of α
- Choose the value of α that minimizes perplexity on dev set
- Report final results on test set

A general methodology

- Training/dev/test split is used across machine learning
- Development set used for evaluating different models, debugging, optimizing parameters (like α)
- Test set simulates deployment; only used once final model and parameters are chosen. (Ideally: once per paper)
- Avoids overfitting to the training set and even to the test set

Adjusted Counts

- Previously, we estimated probabilities based on actual counts

$$P_{\mathrm{ML}} = \frac{c}{n}$$

- Then, we changed the formula to estimate smoothed probabilities

$$P_{+\alpha} = \frac{c+\alpha}{n+\alpha v}$$

- Another view: we adjusted the counts c

$$P_{+\alpha} = \frac{c^*}{n} \quad \Rightarrow \quad c^* = n\, P_{+\alpha} = (c+\alpha)\,\frac{n}{n+\alpha v}$$

Good-Turing Smoothing

- Adjust actual counts c to expected counts c^* with formula

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

- Assuming we are building an N-gram model:
  - N_c number of N-grams that occur exactly c times in corpus (e.g. bigram model: N_1= how many bigrams occurred once?)
  - N_0 total number of unseen N-grams

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
c	N_c	c^*	t_c
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

t_c are average counts of bigrams in test set that occurred c times in corpus: fairly close to estimate c^*.

Good-Turing justification: 0-count items

- Estimate the probability that the next observation is previously unseen (i.e., will have count 1 once we see it)

$$P(\text{unseen}) = \frac{N_1}{n} = \frac{\text{number of things we saw once}}{\text{number of things we saw}}$$

  This part uses MLE!

- Divide that probability equally amongst all unseen events (N_0 of them)

$$P_{\mathrm{GT}} = \frac{1}{N_0}\frac{N_1}{n} \quad \Rightarrow \quad c^* = \frac{N_1}{N_0}$$

Good-Turing justification: 1-count items

- Estimate the probability that the next observation was seen once before (i.e., will have count 2 once we see it)

$$P(\text{once before}) = \frac{2N_2}{n}$$

- Divide that probability equally amongst all 1-count events

$$P_{\mathrm{GT}} = \frac{1}{N_1}\frac{2N_2}{n} \quad \Rightarrow \quad c^* = \frac{2N_2}{N_1}$$

- Same thing for higher count items

Remaining problem

- In given corpus, suppose we never observe
  - Scottish beer drinkers
  - Scottish beer eaters
- If we build a trigram model smoothed with Add-α or G-T, which example has higher probability?

Remaining problem

- Previous smoothing methods assign equal probability to all unseen events.
- Better: use information from lower order N-grams (shorter histories).
  - beer drinkers
  - beer eaters
- Two ways: **interpolation** and **backoff**.

Interpolation

- Higher and lower order N-gram models have different strengths and weaknesses
  - high-order N-grams are sensitive to more context, but have sparse counts
  - low-order N-grams consider only very limited context, but have robust counts

- So, combine them:

$$\begin{aligned} P_I(w_3|w_1,w_2) = \ & \lambda_1 \ P_1(w_3) && \mathrm{P}_1(\mathrm{drinkers}) \\ & + \lambda_2 \ P_2(w_3|w_2) && \mathrm{P}_2(\mathrm{drinkers}|\mathrm{beer}) \\ & + \lambda_3 \ P_3(w_3|w_1,w_2) && \mathrm{P}_3(\mathrm{drinkers}|\mathrm{Scottish},\mathrm{beer}) \end{aligned}$$

Interpolation

- Note that λ_is must sum to 1:

$$\begin{aligned}
1 &= \sum_{w_3} P_I(w_3|w_1, w_2) \\
&= \sum_{w_3} [\lambda_1\ P_1(w_3) + \lambda_2\ P_2(w_3|w_2) + \lambda_3\ P_3(w_3|w_1, w_2)] \\
&= \lambda_1 \sum_{w_3} P_1(w_3) + \lambda_2 \sum_{w_3} P_2(w_3|w_2) + \lambda_3 \sum_{w_3} P_3(w_3|w_1, w_2) \\
&= \lambda_1 + \lambda_2 + \lambda_3
\end{aligned}$$

Fitting the interpolation parameters

- In general, any weighted combination of distributions is called a **mixture model**.
- So λ_is are **interpolation parameters** or **mixture weights**.
- The values of the λ_is are chosen to optimize perplexity on a held-out data set.

Back-Off

- Trust the highest order language model that contains N-gram, otherwise "back off" to a lower order model.
- Basic idea:
  - discount the probabilities slightly in higher order model
  - spread the extra mass between lower order N-grams

Discounting

- Say we've seen the following counts:

x	Count(x)	$P_{ML}(w_i \mid w_{i-1})$
the	48	
the, dog	15	15/48
the, woman	11	11/48
the, man	10	10/48
the, park	5	5/48
the, job	2	2/48
the, telescope	1	1/48
the, manual	1	1/48
the, afternoon	1	1/48
the, country	1	1/48
the, street	1	1/48

- The maximum-likelihood estimates are high (particularly for low count items)

Discounting

- Now define "discounted" counts, for example (a first, simple definition):

$$\text{Count}^*(x) = \text{Count}(x) - 0.5$$

- New estimates:

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Discounting

- We now have some "missing probability mass":

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

  e.g., in our example, $\alpha(the) = 10 \times 0.5/48 = 5/48$

- Divide the remaining probability mass between words w for which $\text{Count}(w_{i-1}, w) = 0$.

Back-Off Equation

$$P_{BO}(w_i|w_{i-N+1},...,w_{i-1}) =$$

$$= \begin{cases} P^*(w_i|w_{i-N+1},...,w_{i-1}) \\ \qquad\qquad \text{if count}(w_{i-N+1},...,w_i) > 0 \\ \alpha(w_{i-N+1},...,w_{i-1})\; P_{BO}(w_i|w_{i-N+2},...,w_{i-1}) \\ \qquad\qquad \text{else} \end{cases}$$

- Requires
  - adjusted prediction model $P^*(w_i|w_{i-N+1},...,w_{i-1})$
  - backoff weights $\alpha(w_1,...,w_{N-1})$

Katz Back-off Models (Bigrams)

- For a bigram model, define two sets

$$\begin{aligned}\mathcal{A}(w_{i-1}) &= \{w : \text{Count}(w_{i-1}, w) > 0\} \\ \mathcal{B}(w_{i-1}) &= \{w : \text{Count}(w_{i-1}, w) = 0\}\end{aligned}$$

- A bigram model

$$P_{KATZ}(w_i \mid w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-1},w_i)}{\text{Count}(w_{i-1})} & \text{If } w_i \in \mathcal{A}(w_{i-1}) \\ \alpha(w_{i-1}) \frac{P_{ML}(w_i)}{\sum_{w \in \mathcal{B}(w_{i-1})} P_{ML}(w)} & \text{If } w_i \in \mathcal{B}(w_{i-1}) \end{cases}$$

where

$$\alpha(w_{i-1}) = 1 - \sum_{w \in \mathcal{A}(w_{i-1})} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

Katz Back-off Models (Trigrams)

- For a trigram model, first define two sets

$$\begin{aligned}\mathcal{A}(w_{i-2}, w_{i-1}) &= \{w : \text{Count}(w_{i-2}, w_{i-1}, w) > 0\} \\ \mathcal{B}(w_{i-2}, w_{i-1}) &= \{w : \text{Count}(w_{i-2}, w_{i-1}, w) = 0\}\end{aligned}$$

- A trigram model is defined in terms of the bigram model:

$$P_{KATZ}(w_i \mid w_{i-2}, w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})} & \text{If } w_i \in \mathcal{A}(w_{i-2}, w_{i-1}) \\ \frac{\alpha(w_{i-2}, w_{i-1}) P_{KATZ}(w_i \mid w_{i-1})}{\sum_{w \in \mathcal{B}(w_{i-2}, w_{i-1})} P_{KATZ}(w \mid w_{i-1})} & \text{If } w_i \in \mathcal{B}(w_{i-2}, w_{i-1}) \end{cases}$$

where

$$\alpha(w_{i-2}, w_{i-1}) = 1 - \sum_{w \in \mathcal{A}(w_{i-2}, w_{i-1})} \frac{\text{Count}^*(w_{i-2}, w_{i-1}, w)}{\text{Count}(w_{i-2}, w_{i-1})}$$

Do our smoothing methods work here?

Example from MacKay and Bauman Peto (1994):

> Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet, 'you see', you see, in which the second word of the couplet, 'see', follows the first word, 'you', with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words 'you' and 'see', with equal frequency, you see.

- $P(\text{see})$ and $P(\text{you})$ both high, but *see* nearly always follows *you*.
- So $P(\text{see}|novel)$ should be much lower than $P(\text{you}|novel)$.

Diversity of histories matters!

- A real example: the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers

  → in unigram language model: a respectable probability

- However, it almost always directly follows New (473 times)

- So, in unseen bigram contexts, York should have low probability
  - lower than predicted by unigram model used in interpolation or backoff.

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account
- Count of distinct histories for a word:

$$N_{1+}(\bullet w_i) \ = |\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|$$

- Recall: maximum likelihood est. of unigram language model:

$$P_{ML}(w) = \frac{C(w_i)}{\sum_{w_i} C(w_i)}$$

- In KN smoothing, replace raw counts with count of histories:

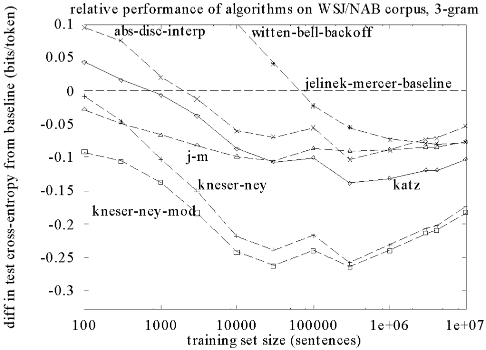
$$P_{KN}(w_i) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(\bullet w_i)}$$

Kneser-Ney in practice

- Original version used backoff, later “modified Kneser-Ney” introduced using interpolation (Chen and Goodman, 1998).
- Fairly complex equations, but until recently the best smoothing method for word n-grams.
- See Chen and Goodman for extensive comparisons of KN and other smoothing methods.
- KN (and other methods) implemented in language modelling toolkits like SRILM (classic), KenLM (good for really big models), OpenGrm Ngram library (uses finite state transducers), etc.

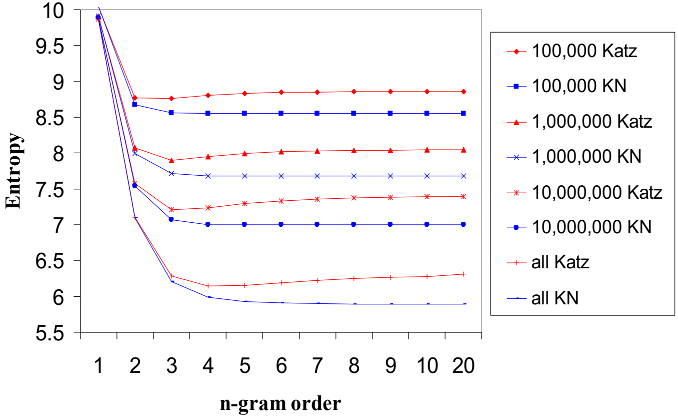
What works better in practice?

- Trigrams and beyond:
  - Unigrams, bigrams generally useless
  - Trigrams much better (when there's enough data)
  - 4-, 5-grams really useful in MT, but not so much for speech
- Discounting
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell, etc…
- See [Chen+Goodman] reading for tons of graphs…



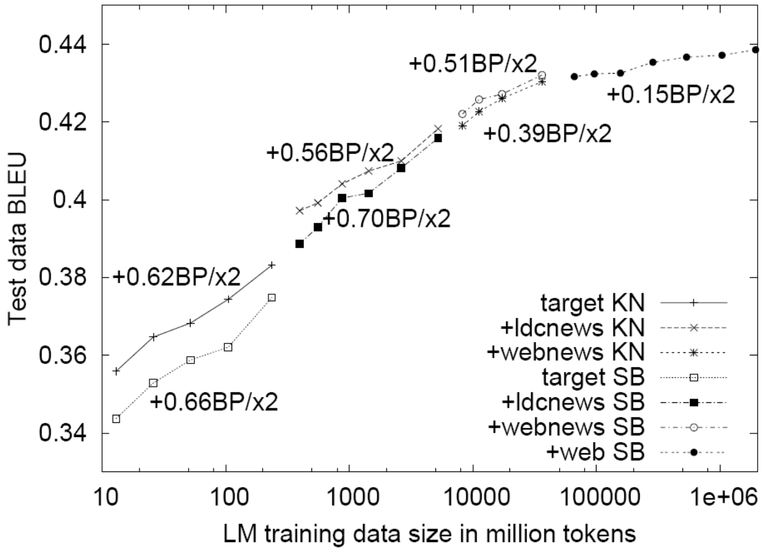
Data vs. Method ?

- Having more data is better…



- … but so is using a better estimator
- Another issue: N > 3 has huge costs in speech recognizers

Data vs. Method ?



- Tons of data closes gap, for extrinsic MT evaluation

Bayesian interpretations of smoothing

- We started by asking: What's the best θ given the data d that we saw?

$$P(\theta|d) \propto P(d|\theta)P(\theta)$$

- MLE ignored $P(\theta)$, and we had to introduce smoothing.

- It turns out that many smoothing methods are mathematically equivalent to forms of **Bayesian estimation**, i.e., the use of non-uniform priors!
  - Add-α smoothing: Dirichlet prior
  - Kneser-Ney smoothing: Pitman-Yor prior

See MacKay and Bauman Peto (1994); (Goldwater, 2006, pp. 13-17); Goldwater et al. (2006); Teh (2006).

Are we done with smoothing yet?

We've considered methods that predict rare/unseen words using

- Uniform probabilities (add-α, Good-Turing)
- Probabilities from lower-order n-grams (interpolation, backoff)
- Probability of appearing in new contexts (Kneser-Ney)

What's left?

Word similarity

- Two words with $C(w_1) \gg C(w_2)$
  - salmon
  - swordfish
- Can $P(\text{salmon}|\text{caught two})$ tell us something about $P(\text{swordfish}|\text{caught two})$?
- n-gram models: no.

Word similarity in language modeling

- Early version: class-based language models (J&M 4.9.2)
  - Define classes c of words, by hand or automatically
  - $P_{CL}(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$ (an HMM)
- Recent version: **distributed** language models
  - Current models have better perplexity than MKN.
  - Ongoing research to make them more efficient.
  - Examples: Log Bilinear LM (Mnih and Hinton, 2007), Recursive Neural Network LM (Mikolov et al., 2010), LSTM LMs, etc.

Distributed word representations

- Each word represented as high-dimensional vector (50-500 dims)

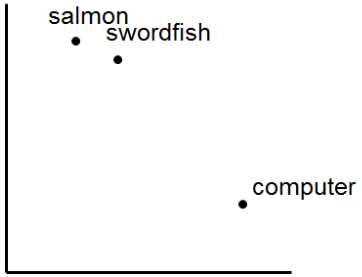
  E.g., salmon is $[0.1, 2.3, 0.6, -4.7, \ldots]$

- Similar words represented by similar vectors

  E.g., swordfish is $[0.3, 2.2, 1.2, -3.6, \ldots]$

Training the model

- Goal: learn word representations (**embeddings**) such that words that behave similarly are close together in high-dimensional space.
- 2-dimensional example:



Training the model

- N-gram LM: collect counts, maybe optimize some parameters
  - (Relatively) quick, especially these days (minutes-hours)
- distributed LM: learn the representation for each word
  - Solved with machine learning methods (e.g., neural networks)
  - Can be extremely time-consuming (hours-days)
  - Learned embeddings seem to encode both semantic and syntactic similarity (using different dimensions) (Mikolov et al., 2013).

Using the model

Want to compute $P(w_1 \dots w_n)$ for a new sequence.

- N-gram LM: again, relatively quick
- distributed LM: often prohibitively slow for real applications
- An active area of research for distributed LMs

Other Topics in Language Modeling

Many active research areas in language modeling:

- Factored/morpheme-based language models: back off to word stems, part-of-speech tags, and/or other morphemes in word
- Syntactic language models: using parse trees
- Domain adaptation: when only a small domain-specific corpus is available
- Time efficiency and space efficiency are both key issues (esp on mobile devices!)

What about *unknown* words ?

Q. How to handle words in the test corpus that did not occur in the training data, i.e. **out of vocabulary (OOV)** words ?

A. Simple solution : Train a model that includes an explicit symbol for an unknown word (`<UNK>`).

- Choose a vocabulary in advance and replace other words in the training corpus with `<UNK>`.
- Replace the first occurrence of each word in the training data with `<UNK>`.