Natural Language Processing

WBAI059-05

university of groningen / faculty of science and engineering

Tsegaye Misikir Tashu

Lecture 2: n-gram Language Models

(Some slides are adapted from Dan Jurafsky and Danqi Chen)

Today’s lecture plan

- What is N-gram Language model
- **Text Generating** from a language model
- **Evaluating** a language model (perplexity)

Recommended reading:
JM3 3.1-3.5

CHAPTER
3
N-gram Language Models

What is an n-gram language model?

Language models and n-gram

- Models that assign probabilities to sequences of words (n-gram) are called language models or LMs.
  - An n-gram is a sequence of n words

Example : “Please turn your homework”: a 2-gram (bigram) will be

”please turn”, “turn your”, ”your homework”

- Language can be defined as a model that assigns probabilities to sentences

(Probabilistic) Language Modeling

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

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$ -> **Joint probability**

- Related task: probability of an upcoming word:

  $P(w_5|w_1,w_2,w_3,w_4)$ -> **Conditional probability**

- 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**.

What is a language model?

- **Language Modeling** is the task of predicting the probability of a sentence or what word comes next.
- More formally: given a sequence of words w_1, w_2...,w_t compute the probability distribution of the next word w_{t+1}.

$$P(w_{(t+1)} | w_1, w_2, w_3, \ldots, w_t)$$

- Where $w_{(t+1)}$ can be any word in the vocabulary
- You can also think of a Language Model as a system that assigns probability to a piece of text.

How to compute P(W)

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)
- Intuition: let’s rely on the Chain Rule of Probability

Reminder: The Chain Rule

- Recall the definition of conditional probabilities

  $P(A \mid B) = P(A,B)/P(B)$ Rewriting: $P(A,B) = P(B)P(A \mid B)$

- More variables:

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

- The Chain Rule in General

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

Using Chain rule

- For example, if we have some text $w_1, \ldots\ldots, w_n$ then the probability of this text (according to the Language Model) is:

Conditional probability:
$p(w_2 \mid w_1), \forall w \in V$

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

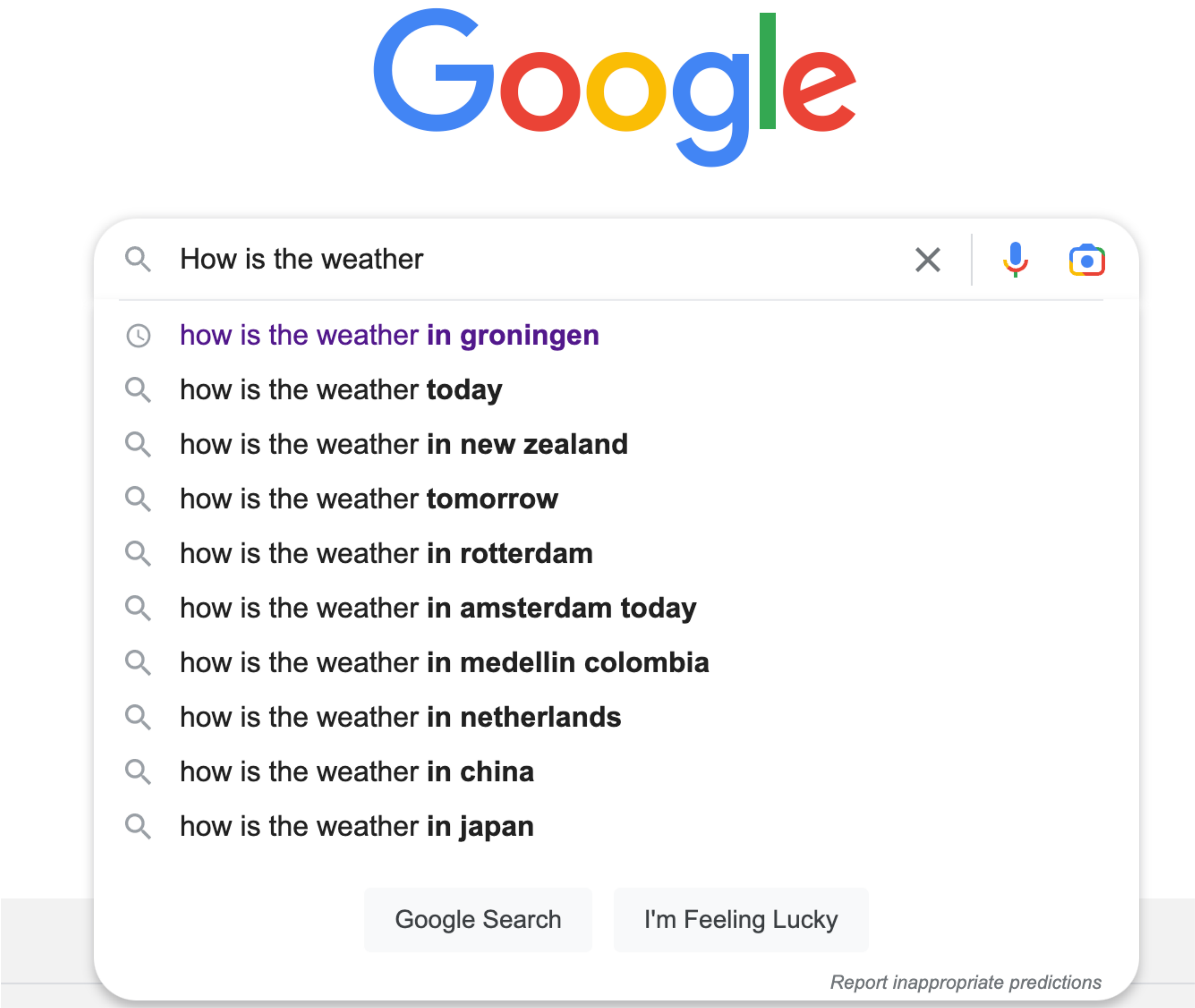
$$P(w_1 w_2 \ldots w_n) = \underbrace{\prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})}_{\text{This is what our LM provides}}$$

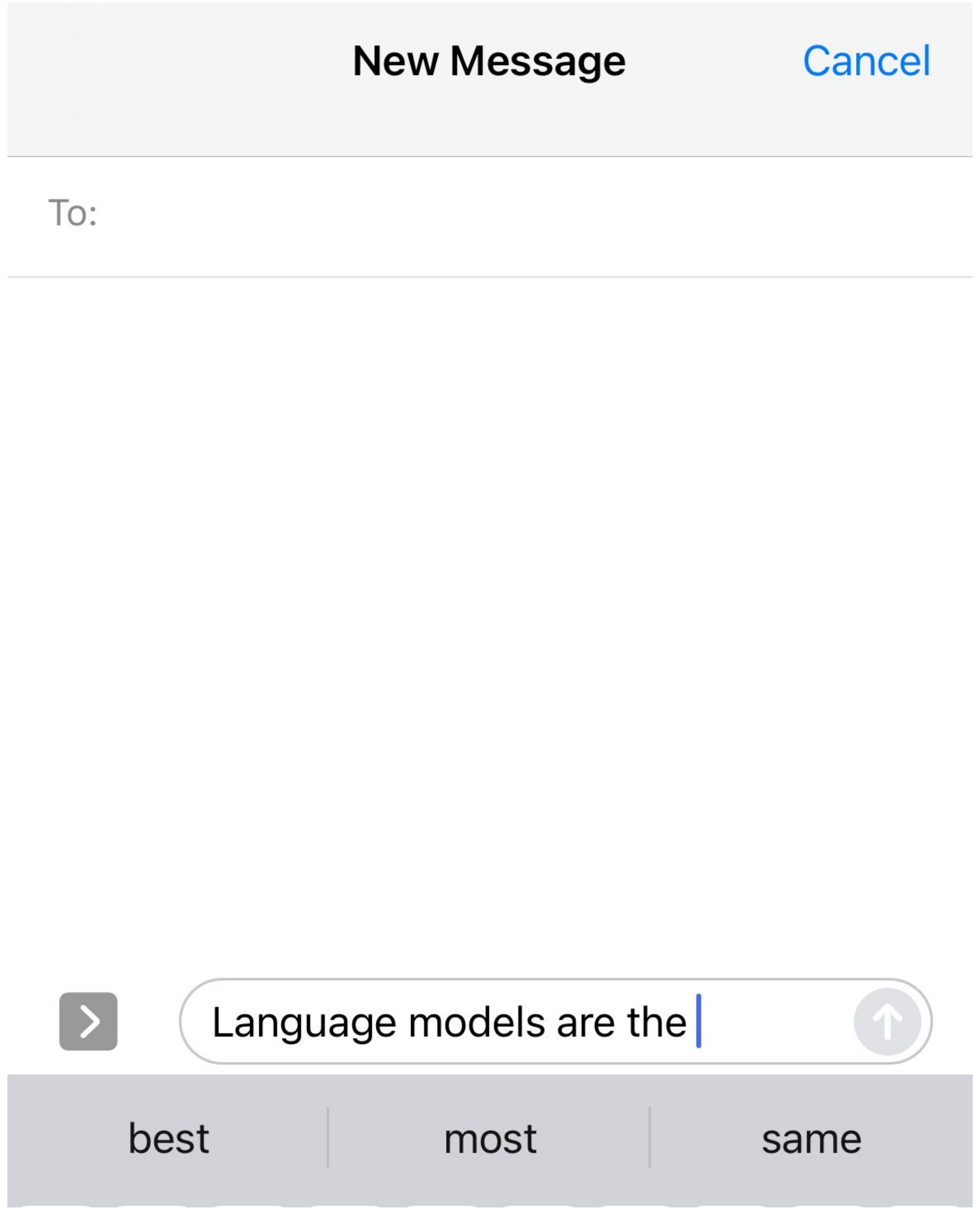
Example

Example Sentence: “the cat sat on the mat”

$P(\text{the cat sat on the mat}) = P\,(\text{the}) * P\,(\text{cat}\,|\,\text{the}) * P\,(\text{sat}\,|\,\text{the cat}) * P\,(\text{on}\,|\,\text{the cat sat}) * P\,(\text{the}\,|\,\text{the cat sat on}) * P(\text{mat}\,|\,\text{the cat sat on the})$

Language models are everywhere





Estimating probabilities

- One way to estimate this probability is from relative frequency counts

$$P(\text{sat} \mid \text{the cat}) = \frac{\text{count(the cat sat)}}{\text{count(the cat)}}$$

Trigram

Bigram

$$P(\text{on} \mid \text{the cat sat}) = \frac{\text{count(the cat sat on)}}{\text{count(the cat sat)}}$$

$$\vdots$$

- Assume we have a vocabulary of size V , how many sequences of length n do we have?
  - With a vocabulary of size V, # sequences of length n = V^n

- Typical English vocabulary ~ 40k words

Markov assumption

- Use only the recent past to predict the next word
- Reduces the number of estimated parameters in exchange for modeling capacity.
- 1st order

$$P(\text{mat}/\text{the cat sat on the}) \approx P(\text{mat}/\text{the})$$

- 2nd order

$$P(\text{mat}/\text{the cat sat on the}) \approx P(\text{mat}/\text{on the})$$

n-gram Language Models- Markov assumption

- First, we make a Markov assumption: w_i depends only on the preceding *n*-1 words: Consider only the last k words (or less) for context

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

- which implies the probability of a sequence is:

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

- **Question:** How do we get these *n*-gram and (*k*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

n-gram Language Models

- Definition: A n-gram is a chunk of n consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are and use these to predict next word.

n-gram language models

- Unigram : In the Unigram model, we estimate the probability of a whole sequence of words by the product of probabilities of individual words.

$$P(w_1, w_2, \ldots w_n) \approx \prod_{i=1}^{n} P(wi)$$ E.g. P(the) P(cat) P(sat)

- Bigram: we estimate the probability of a word, given the entire prefix from the beginning to the previous word.
- We condition on a single previous word

$$P(w_i \mid w_1, w_2, \ldots w_{i-1}) \approx P(w_i / w_{i-1}) \approx \prod_{i=1}^{n} P(w_i | w_{i-1})$$

E.g. P(the) P(cat | the) P(sat | cat)

Larger the n, more accurate and better the language model (but also higher costs)

N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:

    "The computer which I had just put into the machine room on the fifth floor crashed."

- But we can often get away with N-gram models

Estimating bigram probabilities

- How do we estimate these n-gram probabilities?

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

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

n-gram Language Models: Example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ $\underbrace{\textit{students opened their}}_{\text{condition on this}}$ ______

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - ➔ P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - ➔ P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Practical Issues

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

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

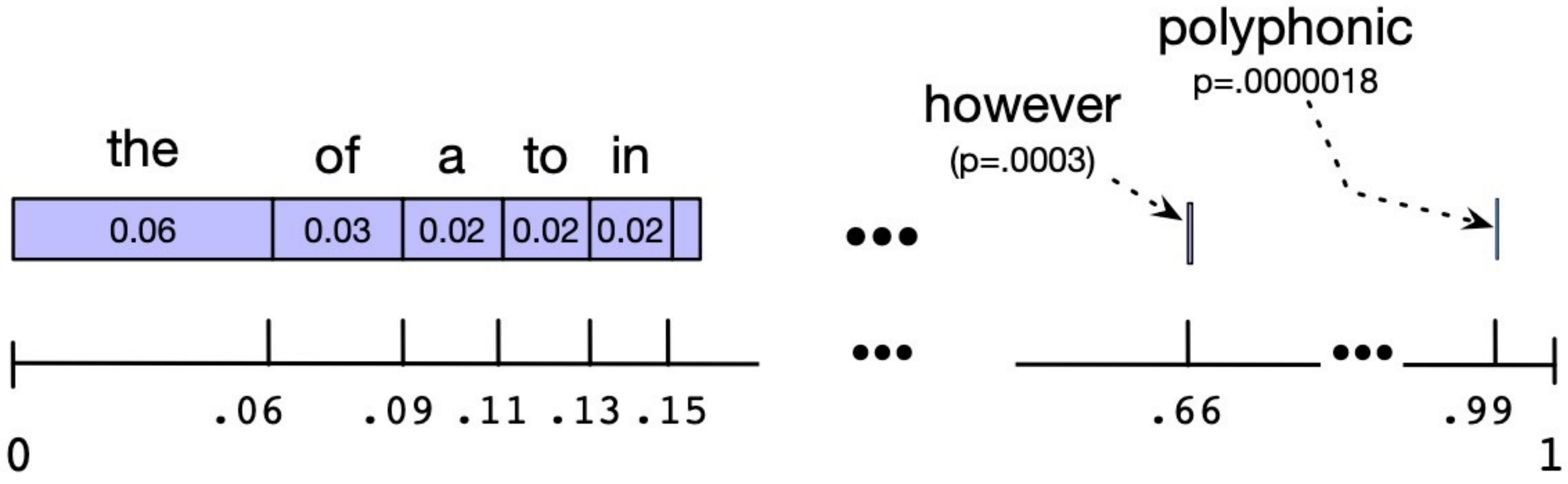
Generating from a language model

Generating from a language model

- Given a bigram language model, how to generate a sequence?

$$P(x^1, x^2, \ldots x^t) = \prod_{i=1}^{t} P(x^i | x^{i-1})$$

- Generate the first word $x_1 \sim P(x)$
- Generate the second word $x^2 \sim P(x \mid x^1)$
- Generate the third word $x^3 \sim P(x \mid x^2)$



Generating from a language model

- Given a bigram language model, how to generate a sequence?

$$P(x^1, x^2, \dots x^t) = \prod_{i=1}^{t} P(x^i | x^{i-2}, x^{i-1})$$

- Generate the first word $x_1 \sim P(x)$
- Generate the second word $x^2 \sim P(x \mid x^1)$
- Generate the third word $x^3 \sim P(x \mid x^1, x^2)$
- Generate the fourth word $x^4 \sim P(x \mid x^2, x^3)$

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _______

get probability distribution

company	0.153
bank	0.153
price	0.170
italian	0.039
emirate	0.039
	…

* **Try for yourself:** https://nlpforhackers.io/language-models/

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text

today the _

condition on this

get probability distribution

company	0.153
bank	0.153
price	0.170
italian	0.039
emirate	0.039
…	

sample

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price ______

condition on this

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
	…

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text

*today the price of*____

condition on this

get probability distribution

the	0.032
18	0.043
oil	0.043
its	0.036
gold	0.078
…	

sample

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

Evaluating a language model

Evaluation: How good is our model?

- 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.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, 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
- But Time-consuming; can take days or weeks

Intrinsic evaluation of language models

Research process:

- Train parameters on a suitable training corpus
  - Assumption: observed sentences ~ good sentences
- Test on *different, unseen* corpus
  - If a language model assigns a higher probability to the test set, it is better
- Evaluation metric - perplexity!
  - Measure of how well a LM **predicts** the next word

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Measure of model's uncertainty about next word : **Lower** perplexity is better!

RNNs have greatly improved perplexity

n-gram model

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language.
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
    - Predictive typing
    - Speech recognition
    - Handwriting recognition
    - Spelling/grammar correction
    - Authorship identification
    - Machine translation
    - Summarization
    - Dialogue
    - etc.

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

<u>Problem:</u> What if *"students opened their w"* never occurred in data? Then w has probability 0!

<u>(Partial) Solution:</u> Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count(students opened their } \boldsymbol{w})}}{\boxed{\text{count(students opened their)}}}$$

Sparsity Problem 2

<u>Problem:</u> What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

<u>(Partial) Solution:</u> Just condition on *"opened their"* instead. This is called *backoff*.

<u>Note:</u> Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

Questions