# FNLP
# Lanuage Modelling

**Yansong Feng**
**fengyansong@pku.edu.cn**

Wangxuan Institute of Computer Technology
Peking University

March 6, 2025

# Outline

# Outline

1. **What are Language Models?**
   - Why do we need language models?
2. **N-gram Language Models**
   - Markov Assumptions
   - N-gram Language Models
   - Parameter Estimations
   - Evaluation
3. **Deal with Unseen Events**
   - Linear Interpolation
   - Smoothing
4. **Beyond N-gram Language Models**

- **How likely is a string of English words good English ?**
  - the dog barks
  - the dog laughs
  - dog the dog bark a laugh smile

- **How likely is a string of English words good English ?**
    - the dog barks
    - the dog laughs
    - dog the dog bark a laugh smile
- What about Chinese ? German ? ...

- **How likely is a string of English words good English ?**
  - the dog barks
  - the dog laughs
  - dog the dog bark a laugh smile
- What about Chinese ? German ? ...
- **How likely is a string of words in a certain language an expression in that language ?**

- **How likely is a string of English words good English ?**
  - the dog barks
  - the dog laughs
  - dog the dog bark a laugh smile
- What about Chinese ? German ? ...
- **How likely is a string of words in a certain language an expression in that language ?**
  - lexicons: lexical analysis?
  - syntax: grammars, syntactic analysis?
  - pragmatics?...

- **How likely is a string of English words good English ?**
  - the dog barks
  - the dog laughs
  - dog the dog bark a laugh smile
- What about Chinese ? German ? ...
- **How likely is a string of words in a certain language an expression in that language ?**
  - lexicons: lexical analysis?
  - syntax: grammars, syntactic analysis?
  - pragmatics?...
  - **how about relying on data solely?**

# Formally (sort of)

- Given a finite size of vocabulary:

$$\mathcal{V} = \{\text{a, an, dog, cat, bite, bark, .... zoo, ...}\}$$

- we can have an infinite set of strings of words, $\mathcal{S}$:
  - the dog barks .
  - the dog laughs .
  - dog the bark a laugh smile .
  - ...

# Formally (sort of)

- Given a finite size of vocabulary:

$$\mathcal{V} = \{\text{a, an, dog, cat, bite, bark, .... zoo, ...}\}$$

- we can have an infinite set of strings of words, $\mathcal{S}$:
  - the dog barks . $\rightarrow$ good
  - the dog laughs . $\rightarrow$ good
  - dog the bark a laugh smile . $\rightarrow$ bad
  - ...

# From a Probabilistic View

There is a probability distribution $p$ over $\mathcal{S}$ :

$$\sum_{s \in \mathcal{S}} p(s) = 1, p(s) \geq 0, \text{for all } s \in \mathcal{S}$$

that is :

- $p(\text{the dog barks . })$
- $p(\text{the dog laughs . })$
- $p(\text{dog the bark a laugh smile . })$
- ...

# From a Probabilistic View

There is a probability distribution $p$ over $\mathcal{S}$ :

$$\sum_{s \in \mathcal{S}} p(s) = 1, p(s) \geq 0, \text{for all } s \in \mathcal{S}$$

that is :

- $p$(the dog barks . ) $\rightarrow$ large
- $p$(the dog laughs . ) $\rightarrow$ large
- $p$(dog the bark a laugh smile . ) $\rightarrow$ very small
- ...

# From a Probabilistic View

There is a probability distribution $p$ over $\mathcal{S}$ :

$$\sum_{s \in \mathcal{S}} p(s) = 1, p(s) \geq 0, \text{for all } s \in \mathcal{S}$$

that is :

- $p$(the dog barks . ) $\rightarrow$ large $\rightarrow$ good
- $p$(the dog laughs . ) $\rightarrow$ large $\rightarrow$ good
- $p$(dog the bark a laugh smile . ) $\rightarrow$ very small $\rightarrow$ bad
- ...

# From a Probabilistic View

There is a probability distribution $p$ over $\mathcal{S}$ :

$$\sum_{s \in \mathcal{S}} p(s) = 1, p(s) \geq 0, \text{for all } s \in \mathcal{S}$$

that is :

- $p$(the dog barks . ) $\rightarrow$ large $\rightarrow$ good
- $p$(the dog laughs . ) $\rightarrow$ large $\rightarrow$ good
- $p$(dog the bark a laugh smile . ) $\rightarrow$ very small $\rightarrow$ bad
- ...

$p$: possibly what we want!

# Outline

# Why Language Model ?

- Speech Recognition
- Optical Character Recognition (OCR)
- Handwriting Recognition
- Word Segmentation (*whatdoesthismean*?)
- Machine Translation
- Language Generation
- Question Answering/Dialogue/Chat

# Why Language Model ?

- Speech Recognition
- Optical Character Recognition (OCR)
- Handwriting Recognition
- Word Segmentation (*whatdoesthismean*?)
- Machine Translation
- Language Generation
- Question Answering/Dialogue/Chat

**Example (Speech Recognition)**

- recognize speech
- wreck a nice beach

# Why Language Model ?

- Speech Recognition
- Optical Character Recognition (OCR)
- Handwriting Recognition
- Word Segmentation (*whatdoesthismean*?)
- Machine Translation
- Language Generation
- Question Answering/Dialogue/Chat

**Example (Speech Recognition)**

- recognize speech
- wreck a nice beach

$$\text{Faithfulness}(signals, words) + \text{Fluency}(words)$$

# Why Language Model ?

- Speech Recognition
- Optical Character Recognition (OCR)
- Handwriting Recognition
- Word Segmentation (*whatdoesthismean*?)
- Machine Translation
- Language Generation
- Question Answering/Dialogue/Chat

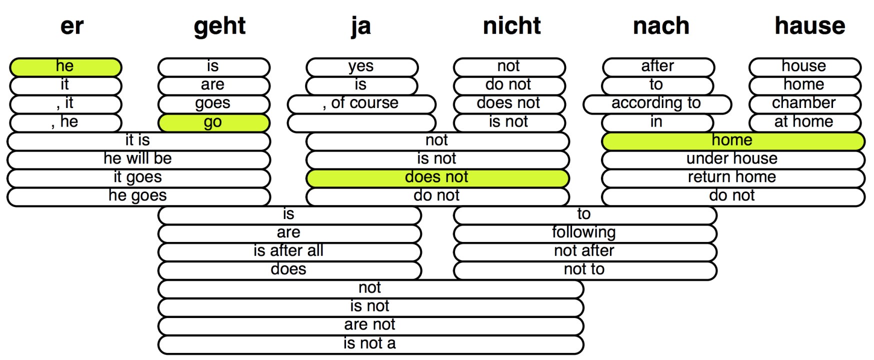
**Example (Speech Recognition)**

- recognize speech
- wreck a nice beach

$$words^* = \arg\max_{words} \text{Faithfulness}(signals, words) + \text{Fluency}(words)$$

# What can LM do?

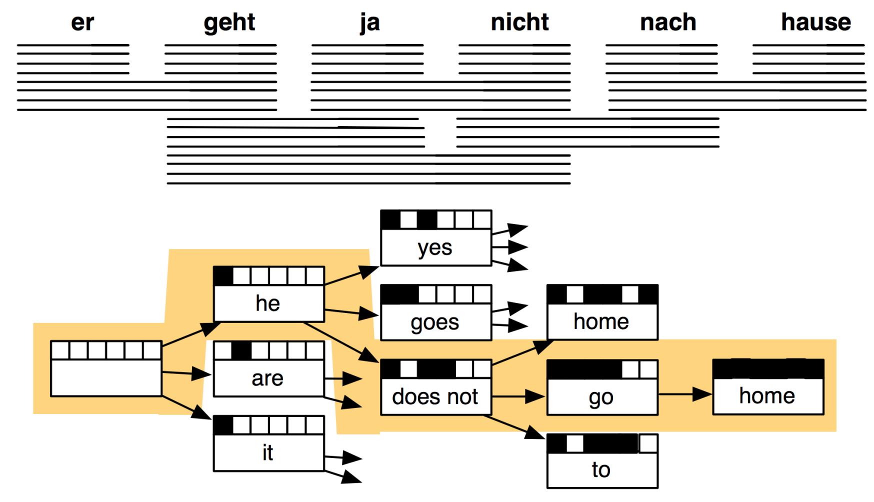
Statistical Machine Translation




[from Philipp Koehn]

# What can LM do?

Statistical Machine Translation




[from Philipp Koehn]

# What can LM do?

Text Generation

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 .

[from Chris Manning]

# What can LM do?

Text Generation

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 .

[from Chris Manning]

- Looks good?

# What can LM do?

Text Generation

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 .

[from Chris Manning]

- Looks good?
- Locally good?
- Globally not...

# What can LM do?

Text Generation

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 .

[from Chris Manning]
But, from ChatGPT *help me write a short passage that begins with "today the price of gold per ton"*

Today the price of gold per ton is on the rise, with experts predicting a surge in demand for the precious metal. As the global economy continues to recover from the pandemic, investors are turning to gold as a safe haven asset to protect their wealth from market fluctuations. This has led to a spike in gold prices, with many analysts projecting that the trend will continue in the coming months. Despite the volatility in the market, gold remains a reliable store of value and a popular choice among investors looking to diversify their portfolios.

# What can LM do?

Again, for all the above scenarios:

$$words^* = \arg\max_{words} \text{Faithfulness}(signals, words) + \text{Fluency}(words)$$

# What can LM do?

Sentence Completion [MS SCC]

- That is his _ fault, but on the whole he's a good worker.
  - generous
  - mother's
  - successful
  - favorite
  - main
- Was she his _, his friend, or his mistress?
  - client
  - musings
  - discomfiture
  - choice
  - opportunity
- All red-headed men who are above the age of _ years, are eligible.
  - seven
  - twenty-one
  - 1,200
  - 60,000
  - 800

# What can LM do?

How surprise you see the next/masked word given the context?

$$\text{Surprise}(context, answer_1) < \text{Surprise}(context, answer_2)$$

# What can LM do?

How surprise you see the next/masked word given the context?

$$\text{Surprise}(context, answer_1) < \text{Surprise}(context, answer_2)$$

- $\text{Surprisal}(w) = -\log_2 p(w|\text{history}_w)$
- *the difficulty of a word is proportional to its surprisal (its negative log-probability) in the context within which it appears.*

# What can LM do?

How surprise you see the next/masked word given the context?

$$\text{Surprise}(context, answer_1) < \text{Surprise}(context, answer_2)$$

- $\text{Surprisal}(w) = -\log_2 p(w|\text{history}_w)$
- *the difficulty of a word is proportional to its surprisal (its negative log-probability) in the context within which it appears.*
- the so-called **surprisal theory** (Hale, 2001; Levy, 2008)

# What can LM do?

How surprise you see the next/masked word given the context?

$$\text{Surprise}(context, answer_1) < \text{Surprise}(context, answer_2)$$

- $\text{Surprisal}(w) = -\log_2 p(w|\text{history}_w)$
- *the difficulty of a word is proportional to its surprisal (its negative log-probability) in the context within which it appears.*
- the so-called **surprisal theory** (Hale, 2001; Levy, 2008)
- *words are easier to comprehend in contexts where they are highly predictable than in unconstraining contexts*

# Outline

# Intuitively

If we treat every possible sentence $s$ as a single point in a huge space $\mathcal{S}$,

- there are $|\mathcal{S}|$ sentences in $\mathcal{S}$:
- $$p(s) = \frac{count(s)}{|\mathcal{S}|}$$

  where $count(s)$ is number of times we see sentence $s$ in $\mathcal{S}$.

# Intuitively

If we treat every possible sentence $s$ as a single point in a huge space $\mathcal{S}$,

- there are $|\mathcal{S}|$ sentences in $\mathcal{S}$:
- $$p(s) = \frac{count(s)}{|\mathcal{S}|}$$

  where $count(s)$ is number of times we see sentence $s$ in $\mathcal{S}$.
- Problematic!

# Intuitively

If we treat every possible sentence $s$ as a single point in a huge space $\mathcal{S}$,

- there are $|\mathcal{S}|$ sentences in $\mathcal{S}$:
- $$p(s) = \frac{count(s)}{|\mathcal{S}|}$$

  where $count(s)$ is number of times we see sentence $s$ in $\mathcal{S}$.
- Problematic!
  - how big is the $\mathcal{S}$?
  - $s$ can be in various forms
  - can we collect $count(s)$ as before?
  - can we properly estimate $p(s)$?

# Intuitively

If we treat every possible sentence $s$ as a single point in a huge space $\mathcal{S}$,

- there are $|\mathcal{S}|$ sentences in $\mathcal{S}$:
- $$p(s) = \frac{count(s)}{|\mathcal{S}|}$$

  where $count(s)$ is number of times we see sentence $s$ in $\mathcal{S}$.
- Problematic!
  - how big is the $\mathcal{S}$?
  - $s$ can be in various forms
  - can we collect $count(s)$ as before?
  - can we properly estimate $p(s)$?
- Break down?

# Another Point of View

- Sentence as a sequence of words $w_1, w_2, w_3, ..., w_n$
- Sentence as a sequence of random variables $X_1, X_2, X_3, ..., X_n$
- The model : $P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP)$
  - This is a **sequence**, not a set!
  - $P(X_1 = \text{I}, X_2 = \text{love})$ and $P(X_1 = \text{love}, X_2 = \text{I})$ are different
  - STOP and, sometimes, START

# A bit Probabilistic

$$P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP)$$
$$= P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})$$

- Chain Rule

# A bit Probabilistic

$$
\begin{aligned}
&P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=&P(X_1 = w_1)\prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})
\end{aligned} \tag{1}
$$

- Chain Rule
- when $n$ is big, ...

# Outline

# Make Assumptions

$$
\begin{aligned}
&P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=&P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})
\end{aligned}
$$

- Chain Rule

# Make Assumptions

- do we always need the big $n$ ?
- only previous history matters, in most cases
- the size of the history? maybe previous $k$ words

$$
\begin{aligned}
&P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=&P(X_1 = w_1)\prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})
\end{aligned}
$$

- Chain Rule
- when $n$ is big, ...

# Make Assumptions

- do we always need the big $n$ ?
- only previous history matters, in most cases
- the size of the history? maybe previous $k$ words

$$
\begin{aligned}
& P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=& P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})
\end{aligned}
$$

- Chain Rule
- when $n$ is big, ...
- Let's make ASSUMPTIONS!

# Make Assumptions

- do we always need the big $n$ ?
- only previous history matters, in most cases
- the size of the history? maybe previous $k$ words

$$
\begin{aligned}
& P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=& P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \\
\approx& P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_{i-1} = w_{i-1})
\end{aligned}
$$

- Chain Rule
- when $n$ is big, ...
- Let's make ASSUMPTIONS! ⇒ First-order Markov assumption:

$$P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \approx P(X_i = w_i | X_{i-1} = w_{i-1})$$

# Make Assumption

Let's make stronger ASSUMPTIONS! $\Rightarrow$ no history at all:

$$P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \approx P(X_i = w_i)$$

- Unigram Language Model
- under MLE estimations:

$$P(X_i = w_i) = \frac{count(w_i)}{\sum_{w_*} count(w_*)}$$

- seems weird, but rather simple, easy to understand
- **cheap**!

# Make Assumption

Let's make stronger ASSUMPTIONS! $\Rightarrow$ no history at all:

$$P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \approx P(X_i = w_i)$$

- Unigram Language Model
- under MLE estimations:

$$P(X_i = w_i) = \frac{count(w_i)}{\sum_{w_*} count(w_*)}$$

- seems weird, but rather simple, easy to understand
- **cheap**!
- This is actually bag-of-words (BoW), definitely not good enough

# Markov Assumption

$$P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP)$$
$$= P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1})$$

# Markov Assumption

$$
\begin{aligned}
&P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=&P(X_1 = w_1)\prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \\
\approx&P(X_1 = w_1)P(X_2 = w_2 | X_1 = w_1) \\
&\times \prod_{i=3}^{n} P(X_i = w_i | X_{i-2} = w_{i-2}, X_{i-1} = w_{i-1})
\end{aligned}
$$

- Second-order Markov assumption:

$$
\begin{aligned}
&P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \\
\approx&P(X_i = w_i | X_{i-2} = w_{i-2}, X_{i-1} = w_{i-1})
\end{aligned}
$$

# Markov Assumption

- Widely used in many applications!

$$
\begin{aligned}
&P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ..., X_n = STOP) \\
=&P(X_1 = w_1) \prod_{i=2}^{n} P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \\
\approx&P(X_1 = w_1) P(X_2 = w_2 | X_1 = w_1) \\
&\times \prod_{i=3}^{n} P(X_i = w_i | X_{i-2} = w_{i-2}, X_{i-1} = w_{i-1})
\end{aligned}
$$

- Second-order Markov assumption:

$$
\begin{aligned}
&P(X_i = w_i | X_1 = w_1, ..., X_{i-1} = w_{i-1}) \\
\approx&P(X_i = w_i | X_{i-2} = w_{i-2}, X_{i-1} = w_{i-1})
\end{aligned}
$$

# Outline

# Bigram and Trigram LMs

- `Widely used in many applications!`
- No history assumption $\rightarrow$ Unigram LM
- First-order Markov assumption $\rightarrow$ Bigram LM
- Second-order Markov assumption $\rightarrow$ Trigram LM
- 4-gram LM, 5-gram LM, ...

## Trigram LM

- A finite vocabulary $\mathcal{V}$
- Parameters $p(a|b, c)$, where $b$, $c$, $a$ is an arbitrary trigram $a \in \mathcal{V} \cup \{STOP\}$ and $b, c \in \mathcal{V}$
- For a new sentence $w_1, w_2, ...w_n$

$$p(w_1, w_2, ...w_n) = p(w_1)p(w_2|w_1)\prod_{i=3}^{n} p(w_i|w_{i-2}, w_{i-1})$$

# A Toy Example

For the sentence:

*the cat laughs . STOP*

given a trigram LM, we have:

$$
\begin{aligned}
p(\text{the cat laughs . STOP}) = & p(\text{the}) \\
& \times p(\text{cat}|\text{the}) \\
& \times p(\text{laughs}|\text{the, cat}) \\
& \times p(.|\text{cat, laughs}) \\
& \times p(\text{STOP}|\text{laughs, .})
\end{aligned}
$$

# Outline

# How to Estimate Parameters from Corpora

The **CORE** is:

- $p(w_i|w_{i-1})$ for bigram LM
- $p(w_i|w_{i-2}, w_{i-1})$ for trigram LM

# How to Estimate Parameters from Corpora

The **CORE** is:

- $p(w_i|w_{i-1})$ for bigram LM
- $p(w_i|w_{i-2}, w_{i-1})$ for trigram LM

Intuitively, given a large corpora:

$$p(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

and,

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$$

# How to Estimate Parameters from Corpora

The **CORE** is:

- $p(w_i|w_{i-1})$ for bigram LM
- $p(w_i|w_{i-2}, w_{i-1})$ for trigram LM

Intuitively, given a large corpora:

$$p(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

and,

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$$

- Maximum likelihood estimation
- More data lead to better model ?

# How Big is Our Model ?

- 10 million sentences from English newswire data (Gigaword)
- 275 million words

# How Big is Our Model ?

- 10 million sentences from English newswire data (Gigaword)
- 275 million words

- the unigram LM needs to store 716,706 probabilities (at most)
- bigram LM : 12,537,755
- trigram LM : 22,174,483
- ...

# How Big is Our Model ?

- 10 million sentences from English newswire data (Gigaword)
- 275 million words

- the unigram LM needs to store 716,706 probabilities (at most)
- bigram LM : 12,537,755
- trigram LM : 22,174,483
- ...

- 12,537,755 v.s. 51,505,109,010,436 ($\sim$ 1/4,000,000)
- Data sparsity: most are zeros
- Trigram LMs are often good enough

# Outline

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

For a test set $\mathcal{D}$, we can use a language model to

- compute the probability for a sentence: $p(s) = p(w_1, w_2, ..., w_n)$

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

For a test set $\mathcal{D}$, we can use a language model to

- compute the probability for a sentence: $p(s) = p(w_1, w_2, ..., w_n)$
- and for the whole dataset, $\prod_{s \in \mathcal{D}} p(s)$
- normalize over all $M$ words in $\mathcal{D}$, $\sqrt[M]{\prod_{s \in \mathcal{D}} p(s)}$

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

For a test set $\mathcal{D}$, we can use a language model to

- compute the probability for a sentence: $p(s) = p(w_1, w_2, ..., w_n)$
- and for the whole dataset, $\prod_{s \in \mathcal{D}} p(s)$
- normalize over all $M$ words in $\mathcal{D}$, $\sqrt[M]{\prod_{s \in \mathcal{D}} p(s)}$
- simplify with $\log$, $\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$
- also the cross entropy: $-\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

For a test set $\mathcal{D}$, we can use a language model to

- compute the probability for a sentence: $p(s) = p(w_1, w_2, ..., w_n)$
- and for the whole dataset, $\prod_{s \in \mathcal{D}} p(s)$
- normalize over all $M$ words in $\mathcal{D}$, $\sqrt[M]{\prod_{s \in \mathcal{D}} p(s)}$
- simplify with $\log$, $\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$
- also the cross entropy: $-\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$
- now **the perplexity of** $\mathcal{D}$ :

$$Perplexity(\mathcal{D}) = 2^{-\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)}$$

# How do We Evaluate a Language Model

Intuition: a good LM model should assign a real sentence of that language a high probability.

$\rightarrow$ the best LM should best predict an unseen test set.

For a test set $\mathcal{D}$, we can use a language model to

- compute the probability for a sentence: $p(s) = p(w_1, w_2, ..., w_n)$
- and for the whole dataset, $\prod_{s \in \mathcal{D}} p(s)$
- normalize over all $M$ words in $\mathcal{D}$, $\sqrt[M]{\prod_{s \in \mathcal{D}} p(s)}$
- simplify with $\log$, $\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$
- also the cross entropy: $-\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)$
- now **the perplexity of** $\mathcal{D}$ :

$$Perplexity(\mathcal{D}) = 2^{-\frac{1}{M} \sum_{s \in \mathcal{D}} \log p(s)}$$

- the lower perplexity, the better LM

# Perplexity

For a dataset $\mathcal{D}$, with vocabulary $\mathcal{V}$,

- if for any trigram, we have $p(a|b,c) = \frac{1}{|\mathcal{V}|+1}$ (a uniform distribution over $\mathcal{V} \cup STOP$) for all $b, c \in \mathcal{V} \cup \{*\}$
- what is the perplexity of $\mathcal{D}$ ??

# Perplexity

For a dataset $\mathcal{D}$, with vocabulary $\mathcal{V}$,

- if for any trigram, we have $p(a|b,c) = \frac{1}{|\mathcal{V}|+1}$ (a uniform distribution over $\mathcal{V} \cup STOP$) for all $b, c \in \mathcal{V} \cup \{*\}$
- what is the perplexity of $\mathcal{D}$ ??
- Perplexity $= |\mathcal{V}| + 1$

# Perplexity

For a dataset $\mathcal{D}$, with vocabulary $\mathcal{V}$,

- if for any trigram, we have $p(a|b,c) = \frac{1}{|\mathcal{V}|+1}$ (a uniform distribution over $\mathcal{V} \cup STOP$) for all $b, c \in \mathcal{V} \cup \{*\}$
- what is the perplexity of $\mathcal{D}$ ??
- Perplexity $= |\mathcal{V}| + 1$

- Perplexity is a measure of *branching factor*
- Considered as the *effective size of the vocabulary*
- Also, to evaluate how hard an NLP task is?

# Evaluating Language Models

- use the `Training dataset` to estimate the n-grams, possibly tuned
- compute perplexity on the `Test dataset`

**However,**

- Better to use test sets that are widely accepted by the community.
- **Perplexity** scores should be compared against the same $\mathcal{V}$.
- Perplexity is NOT a universal metric for downstream applications.

# Unigram v.s. Bigram v.s. Trigram ...

Intuitively, which one is better?

# Unigram v.s. Bigram v.s. Trigram ...

Intuitively, which one is better?

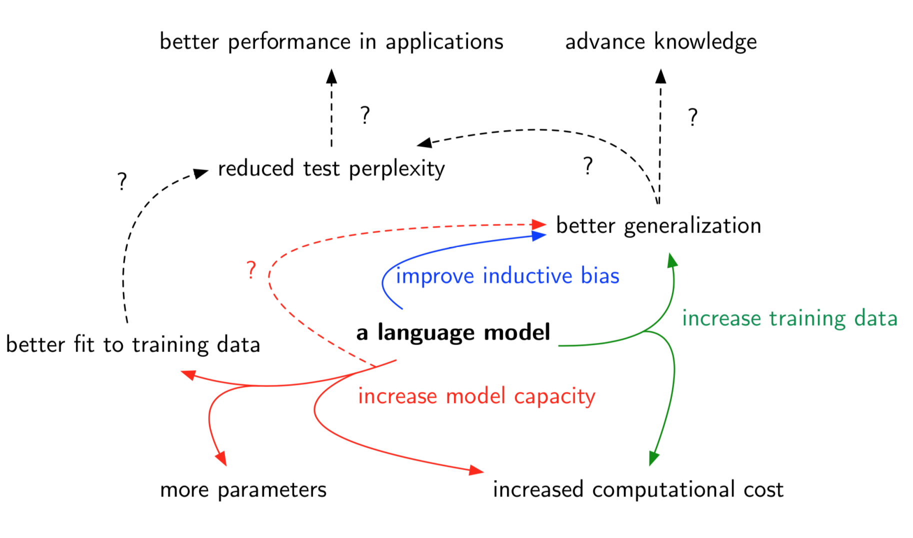
- if $n$ is smaller, ...
  - Does your model learn about the language?
- if $n$ is bigger, ...
  - The cost!
  - The engineering thing! (zeros)
  - ...

# Unigram v.s. Bigram v.s. Trigram ...

Intuitively, which one is better?

- if $n$ is smaller, ...
  - Does your model learn about the language?
- if $n$ is bigger, ...
  - The cost!
  - The engineering thing! (zeros)
  - ...
- if you have more training data, ...
  - perform better
  - generalize better
  - take a look at Google's N-gram

# Language Modelling Research




[from Noah Smith]

# An Example

## Training set

- i love pku .
- i like thu .
- you love pku .
- you do not like thu .

- Can you estimate a bigram LM from the training data?
  - $p(\text{thu}|\text{like})$
  - $p(\text{like}|\text{i})$

# An Example

## Training set

- i love pku .
- i like thu .
- you love pku .
- you do not like thu .

- Can you estimate a bigram LM from the training data?
  - $p(\text{thu}|\text{like})$
  - $p(\text{like}|\text{i})$

## Test set

- you like pku .
- i hate thu .

# An Example

## Training set

- i love pku .
- i like thu .
- you love pku .
- you do not like thu .

- Can you estimate a bigram LM from the training data?
  - $p(\text{thu}|\text{like})$
  - $p(\text{like}|\text{i})$

## Test set

- you like pku .
- i hate thu .

- Can you compute the perplexity using the bigram LM ?
  - $p(\text{you, like, pku, ..., STOP})$
  - $p(\text{i, hate, thu, ..., STOP})$

# Outline

# Unseen Events

- New words
  - *hate*
- New n-grams
  - *you like*, *like pku*

## Unseen Events

- New words
  - *hate*
- New n-grams
  - *you like*, *like pku*

Recall the `MLE` estimation ...

# Unseen Events

MLE yields bad estimations for Unseen Events

- many Zeros

## Unseen Events

MLE yields bad estimations for Unseen Events

- many Zeros

**Brain Storm**

- What can we do?

# Unseen Events

MLE yields bad estimations for Unseen Events

- many Zeros

**Brain Storm**

We should give some probability mass to those unseen events! This will inevitably reduce the mass for seen events.

# Unseen Events

MLE yields bad estimations for Unseen Events

- many Zeros

## Brain Storm

We should give some probability mass to those unseen events! This will inevitably reduce the mass for seen events.

- **back-off** to a lower order estimations
- produce a pseudo word UNKNOWN
- smooth our raw estimations

# Outline

# Linear Interpolation

## Intuition

If we do not have a higher-order ngram, can we use less context, i.e., related lower-order ngrams ?

# Linear Interpolation

## Intuition

If we do not have a higher-order ngram, can we use less context, i.e., related lower-order ngrams ?

- Back-off
  - use higher-order estimators if you have enough counts
  - otherwise lower-order, ...
- Interpolation
  - just combine estimators of different orders:

$$\begin{aligned} p_{il}(w_i|w_{i-2}, w_{i-1}) = \quad & \lambda_1 p_{ml}(w_i|w_{i-2}, w_{i-1}) \\ + \quad & \lambda_2 p_{ml}(w_i|w_{i-1}) \\ + \quad & \lambda_3 p_{ml}(w_i) \end{aligned}$$

    where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ and $\lambda_i \geq 0$ for all $i$
  - does it still yield a distribution?

# Linear Interpolation

Yes!

# Linear Interpolation

Yes! But why?

# Held-out Data and Parameter Estimations

## How to figure out those $\lambda s$ ?

- hold out part of training data as a validation set
- choose $\lambda$s to maximize the loglikelihood of the validation set, in the trigram case:

$$\sum_{w_1,w_2,w_3} count_{dev}(w_1, w_2, w_3) \log p_{il}(w_3|w_1, w_2)$$

where $\sum_i \lambda_i = 1$ and $\lambda_i \geq 0$ for all $i$

# Held-out Data and Parameter Estimations

## How to figure out those $\lambda s$ ?

- hold out part of training data as a validation set
- choose $\lambda$s to maximize the loglikelihood of the validation set, in the trigram case:

$$\sum_{w_1,w_2,w_3} count_{dev}(w_1, w_2, w_3) \log p_{il}(w_3|w_1, w_2)$$

where $\sum_i \lambda_i = 1$ and $\lambda_i \geq 0$ for all $i$

- using an iterative approach to estimate $\lambda$s
- may allow $\lambda$s to vary according to specific counts

# Estimating $\lambda$s Iteratively

Step 1: Initialization: randomly pick values for $\lambda_1$, $\lambda_2$, and $\lambda_3$

Step 2: Calculating

$$c_1 = \sum_{w_1,w_2,w_3} \frac{count_{dev}(w_1,w_2,w_3)\lambda_1 \cdot p_{ml}(w_3|w_1,w_2)}{\lambda_1 \cdot p_{ml}(w_3|w_1,w_2) + \lambda_2 \cdot p_{ml}(w_3|w_2) + \lambda_3 \cdot p_{ml}(w_3)}$$

$$c_2 = \sum_{w_1,w_2,w_3} \frac{count_{dev}(w_1,w_2,w_3)\lambda_2 \cdot p_{ml}(w_3|w_2)}{\lambda_1 \cdot p_{ml}(w_3|w_1,w_2) + \lambda_2 \cdot p_{ml}(w_3|w_2) + \lambda_3 \cdot p_{ml}(w_3)}$$

$$c_1 = \sum_{w_1,w_2,w_3} \frac{count_{dev}(w_1,w_2,w_3)\lambda_3 \cdot p_{ml}(w_3)}{\lambda_1 \cdot p_{ml}(w_3|w_1,w_2) + \lambda_2 \cdot p_{ml}(w_3|w_2) + \lambda_3 \cdot p_{ml}(w_3)}$$

Step 3: Re-calculate: $\lambda_i = \frac{c_i}{c_1+c_2+c_3}$

Step 4: if $\Delta\lambda_i > \theta$, go to Step 2

# Stupid Back-off

- for a fixed vocabulary at a web scale, we have trigrams :

$$s(w_i|w_{i-2}, w_{i-1}) = \begin{cases} \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} & count(w_{i-2}, w_{i-1}, w_i) > 0 \\ 0.4 s(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

- Brants et al. 2007 (Google)

# Stupid Back-off

- for a fixed vocabulary at a web scale, we have trigrams :

$$s(w_i|w_{i-2}, w_{i-1}) = \begin{cases} \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} & count(w_{i-2}, w_{i-1}, w_i) > 0 \\ 0.4 s(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

- Brants et al. 2007 (Google)
- is this model a probabilistic one?

# Stupid Back-off

- for a fixed vocabulary at a web scale, we have trigrams :

$$s(w_i|w_{i-2}, w_{i-1}) = \begin{cases} \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} & count(w_{i-2}, w_{i-1}, w_i) > 0 \\ 0.4 s(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

- Brants et al. 2007 (Google)
- is this model a probabilistic one?
- NO! $s()$s do not sum to 1

# Outline

# A Basic Smoothing Solution

Just add **one count** for all possible n-grams!

# A Basic Smoothing Solution

Just add **one count** for all possible n-grams!

- Does this work ?
- Any side effect you could imagine ?

# Add-One Smoothing

## Intuition

For all possible n-grams, just add an extra count.

$$p_{add1}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |\mathcal{V}|}$$

# Add-One Smoothing

## Intuition

For all possible n-grams, just add an extra count.

$$p_{add1}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |\mathcal{V}|}$$

- also called Laplace-smoothing

# Add-One Smoothing

## Intuition

For all possible n-grams, just add an extra count.

$$p_{add1}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |\mathcal{V}|}$$

- also called Laplace-smoothing

## Example

- $|\mathcal{V}| = 12$, $\mathcal{V} = \{\text{horse}, \text{cat}, \text{dog}, \text{dunkey}, \text{the}, ...\}$
- seen: $p(\text{horse}|\text{the}) = 0.5$, $p(\text{cat}|\text{the}) = 0.5$
- 0 for the unseen
- after Add-one
  - previously seen: $p_{add1}(\text{horse}|\text{the}) = 0.18$, $p_{add1}(\text{cat}|\text{the}) = 0.18$
  - previously unseen: $p_{add1}(\text{dog}|\text{the}) = 0.09$, $p_{add1}(\text{dunkey}|\text{the}) = 0.09$, ...

# Add-One Smoothing

- $p(\text{SEEN EVENTS}|\text{the})$ from 1 to 0.36
- $p(\text{UNSEEN EVENTS}|the)$ from 0 to 0.64

# Add-One Smoothing

- $p(\text{SEEN EVENTS}|\text{the})$ from 1 to 0.36
- $p(\text{UNSEEN EVENTS}|the)$ from 0 to 0.64
- are we putting too much probability mass on unseen events?

# Add-One Smoothing

- $p(\text{SEEN EVENTS}|\text{the})$ from 1 to 0.36
- $p(\text{UNSEEN EVENTS}|the)$ from 0 to 0.64
- are we putting too much probability mass on unseen events?
- too many zeros in our example
- suit cases where $\#(\text{UNSEEN})$ is not that huge

# Good-Turing Discounting

**Intuition**: Use the counts of n-grams that are **seen once** to help estimate the counts of **unseen events**

# Good-Turing Discounting

**Intuition**: Use the counts of n-grams that are **seen once** to help estimate the counts of **unseen events**

- count of count : $N_r$, number of words with frequency $r$
- Translate real counts $r$ into adjusted counts $r^*$:

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

- The probability mass reserved for unseen events is $N_1/N_{all}$
- For larger $r$ (where $N_{r-1}$ is often 0), various other methods can be applied (curve fitting or linear regression).

# Good-Turing Discounting

During testing, how possible we see a word that appears $r$ times in the training set? Imagine we can randomly delete a word from the training set to construct a **new but pseudo** training set:

- everytime, we delete a word $w$
  - we know $w$ appears $r+1$ times in the original set
  - now it is supposed to appear $r$ times in the new training set
- the total apearances of $r+1$ words in the original set is $N_{r+1}(r+1)$
- after many times of deletion, the number of those words in the new set could be considered as $N_r$
- the possible/average counts of those words should be $N_{r+1}(r+1)/N_r$

# Good-Turing Example

Trained on 22m words of AP News (Church and Gale, 1991)

| $r$ | $r^*$ | $r - r^*$ |
|---|---|---|
| 0 | 0.000027 | |
| 1 | 0.446 | 0.554 |
| 2 | 1.26 | 0.74 |
| 3 | 2.24 | 0.76 |
| 4 | 3.24 | 0.76 |
| 5 | 4.22 | 0.78 |
| 6 | 5.19 | 0.81 |
| 7 | 6.21 | 0.79 |
| 8 | 7.24 | 0.76 |

# Absolute Discounting

In Good-Turing discounting,

- It looks like $r^* = r - 0.75$, in most cases

So, we may

- Subtract a fixed number $d$ from each count

$$p_{abd}(w_2|w_1) = \frac{count(w_1, w_2) - d}{count(w_1)} + \lambda(w_1)p(w_2)$$

- Typical counts 1 and 2 are treated differently
- $\lambda(*)$: a weighting function

# Kneser-Ney Smoothing

- Popular in the traditional category
- Combines various smoothing method
  - absolute discounting
  - considers diversity of history, predicted words
  - interpolation

# Outline

# More in LM

- Language modeling is definitely an active field of research
- There are many back-off and interpolation methods
- Skip n-gram models: e.g., back-off to $p(w_n|w_{n-2})$, ...
- Factored language models: back-off to word stems, part-of-speech tags, ...
- Syntactic language models: using parse trees

# More in LM

- Language modeling is definitely an active field of research
- There are many back-off and interpolation methods
- Skip n-gram models: e.g., back-off to $p(w_n|w_{n-2})$, ...
- Factored language models: back-off to word stems, part-of-speech tags, ...
- Syntactic language models: using parse trees
- Language models trained on billions and trillions of words

# Pros and Cons

- Really easy to build, on billions of words
- Smoothing helps generalize to new data
- (Probablistic) scoring fits many downstream tasks

# Pros and Cons

- Really easy to build, on billions of words
- Smoothing helps generalize to new data
- (Probablistic) scoring fits many downstream tasks
- Synonyms: *car*, *van*, *vehicle*, ...
- Only capture short distance context
- Sparsity
- Storage
- Speed

# Pros and Cons

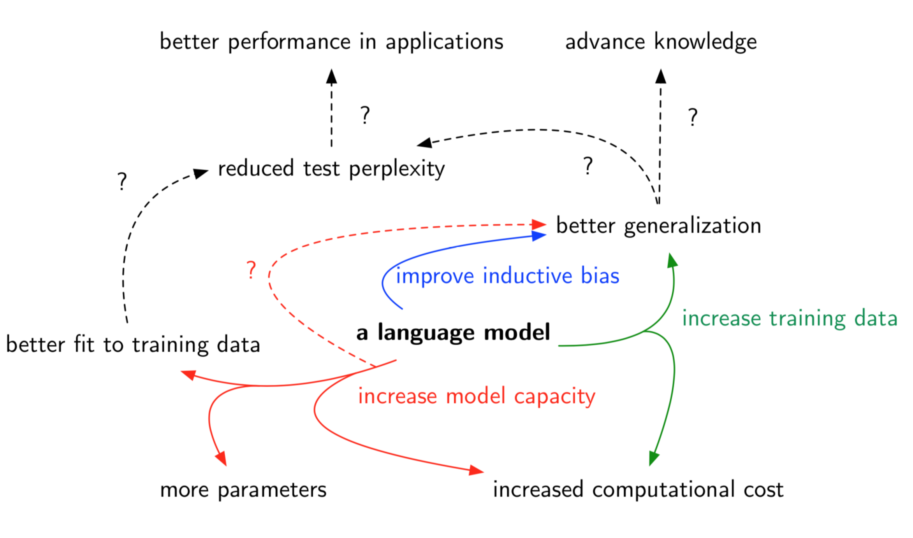
- Really easy to build, on billions of words
- Smoothing helps generalize to new data
- (Probablistic) scoring fits many downstream tasks
- Synonyms: *car*, *van*, *vehicle*, ...
- Only capture short distance context
- Sparsity
- Storage
- Speed

# Pros and Cons

- Really easy to build, on billions of words
- Smoothing helps generalize to new data
- (Probablistic) scoring fits many downstream tasks

- Synonyms: *car*, *van*, *vehicle*, ...
- Only capture short distance context
- Sparsity
- Storage
- Speed

# Language Modelling Research




[from Noah Smith]

# The Magic

modeling the context

# The Magic

## modeling the context

- history: 1-4 words
- history: 3-5-7 words window

# The Magic

modeling the context

- history: 1-4 words
- history: 3-5-7 words window

Seems that we are always counting words

# Now

Questions

# Now

## Questions

- can we go beyond *counting words*?
- can we use something else learned before?
- can we train on *much more words*?

# Now

## Questions

- can we go beyond *counting words*?
- can we use something else learned before? *classification*?
- can we train on *much more words*?

# Readings

- Chapter 3, Speech and Language Processing (3rd. **SLP**)
- SRI Language Modeling Toolkit, http://www.speech.sri.com/projects/srilm/manpages/
- Improved backing-off for m-gram language modeling, Reinhard Kneser and Hermann Ney, ICASSP 1995.
- A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams, Kenneth W.Church and William A.Gale, Computer Speech & Language, 1991, V5-1, pp19-54
- Large Language Models in Machine Translation, Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean, EMNLP 2007
- Google N-grams: https://blog.research.google/2006/08/all-our-n-gram-are-belong-to-you.html
- Google Book N-grams: https://books.google.com/ngrams/