Language modeling

IN4080
Natural Language Processing

Yves Scherrer

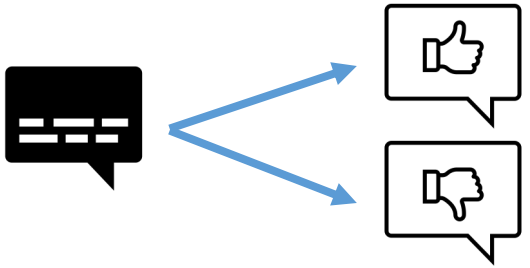
Main NLP tasks

Natural language generation



- Machine translation
- Question answering
- Grammatical error correction
- …

Annotation (natural language understanding)



- Hate speech detection
- Sentiment analysis
- Language identification
- Syntactic analysis
- …

Language models

Language models predict the continuation of a given text, one word at a time.

- The water of Walden Pond is …
  - Likely continuations: clear, dirty, cold, always, …
  - Unlikely continuations: angry, gives, refrigerator, …

- Prompt: It's my dad's birthday next week. Can you help me choose a gift for him? Answer: …

Why next word prediction?

It's how **large language models (LLMs)** work:

- LLMs are **trained** to predict words
- LLMs **generate** text by predicting words, one by one

But today, we'll look at small (and poorly performing) language models ☺

What can we do with language models?

- Predict the probability of the next word
  - LLM prompting, text completion, predictive keyboards
- Assign probabilities to sentences
  - Choose among different hypotheses
    - Disambiguation, reranking
      (e.g. for spellchecking, speech recognition)
    - P(*Their are two midterms*) < P(*There are two midterms*)
    - P(*I will be back soonish*) > P(*I will be bassoon dish*)
  - Score the same sentence with different models
    - Language identification, typology
- Generate entirely new sentences
  - Mostly for fun ☺

More formally

- Compute the probability of the next word w_n:

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

- Predict the most likely next word $\widehat{w_n}$:

$$\widehat{w_n} = \arg\max_{w_n \in V} P(w_n|w_1, w_2, \ldots, w_{n-1})$$

- Related task: compute the probability of an entire sentence W:

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

The chain rule of probabilities

In what way are the two computations on the previous slide related?

- Definition of conditional probabilities:

$$P(B|A) = \frac{P(A,B)}{P(A)} \quad \text{or} \quad P(A,B) = P(A) \cdot P(B|A)$$

- With more variables:

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

- The **chain rule**:

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

The chain rule of probabilities

The chain rule is applied to compute the joint probability of all words in a sentence:

$$P(w_{1..n}) = \prod_{k=1}^{n} P(w_k | w_{1..k-1})$$

- An example:

$P(\text{The water of Walden Pond})$
$= P(\text{The}) \cdot P(\text{water}|\text{The}) \cdot P(\text{of}|\text{The water})$
$\cdot P(\text{Walden}|\text{The water of}) \cdot P(\text{Pond}|\text{The water of Walden})$

  - Some of these probabilities are easy to compute, others not so much…

Estimating probabilities

Could we just count occurrences in a corpus?

- Example 1:

$$P(\text{water}|\text{The}) = \frac{C(\text{The water})}{C(\text{The})}$$

  - In a reasonably large corpus, we can expect to find the sequence *The water*.

- Example 2:

$$P(\text{Pond}|\text{The water of Walden}) = \frac{C(\text{The water of Walden Pond})}{C(\text{The water of Walden})}$$

  - Can we really expect to find the sequence *The water of Walden Pond* in any corpus?

N-gram language models and the Markov assumption

The Markov assumption

As the sentences get longer, it becomes increasingly hard to reliably estimate the probabilities.

The **Markov assumption** simplifies the problem by restricting the conditional probabilities to the k previous words.

- For $k = 1$: $P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-1})$
- For $k = 2$: $P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$
- For $k = 0$: $P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i)$

The Markov assumption

- Example 1 with $k = 1$:

$$P(\text{water}|\text{The}) = \frac{C(\text{The water})}{C(\text{The})}$$

  - No change



Andrei Markov, 1856-1922

- Example 2 with $k = 1$:

$$P(\text{Pond}|\text{The water of Walden}) \approx P(\text{Pond}|\text{Walden}) = \frac{C(\text{Walden Pond})}{C(\text{Walden})}$$

  - The sequence *Walden Pond* is much easier to find in a corpus than *The water of Walden Pond*

N-gram language models

Usually, one doesn't report the value of k, but of $n = k + 1$.

A sequence of n consecutive words is called an **n-gram**.

- **Bigram** model ($n = 2$):

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

Sequences of 2 words: $\langle w_{i-1}, w_i \rangle$

- **Trigram** model ($n = 3$):

$$P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$$

Sequences of 3 words: $\langle w_{i-2}, w_{i-1}, w_i \rangle$

- **Unigram** model ($n = 1$):

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

Sequences of 1 word: $\langle w_i \rangle$

N-gram language models

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

Models trained on Shakespeare texts (J&M Fig. 3.4)

Problems with n-gram language models

- N-gram models cannot handle long-distance dependencies
  - The soups that I made from that new cookbook I bought yesterday …
  - One can choose a higher n, but that makes parameter estimation more difficult
- N-gram models don't do well at modeling new sequences with similar meanings
  - Each word is estimated (and represented) independently

The solution…

Large language models

- can handle much longer contexts
  - because of the attention mechanism
- can model synonymy better, and generate better novel strings
  - because they use embedding spaces

We'll talk about LLMs later, but n-gram models are convenient to introduce many of the important issues and techniques for NLP in general.

An example

A corpus with 3 sentences:

```
I am Sam
Sam I am
I do not like green eggs and ham
```

- There are no punctuation signs, but if there were, they would have to be split off from the words first

We introduce start-of-sentence and end-of-sentence markers (why?):

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

An example Parameter estimation

Let's create a bigram model.

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

- $P(\mathrm{I}|\langle s\rangle) =$
- $P(\mathrm{Sam}|\mathrm{am}) =$
- $P(\mathrm{do}|\mathrm{I}) =$
- $P(\langle /s\rangle|\mathrm{Sam}) =$
- $P(\mathrm{eggs}|\mathrm{ham}) =$

An example Prediction

- What is the probability of *Sam*?
  - $P(\text{Sam}) = P(\text{Sam}|\langle s \rangle) \cdot P(\langle /s \rangle|\text{Sam})$
  - $P(\text{Sam}) = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$

- What is the probability of *I am ham*?
  - $P(\text{I am ham}) = P(\text{I}|\langle s \rangle) \cdot P(\text{am}|\text{I}) \cdot P(\text{ham}|\text{am}) \cdot P(\langle /s \rangle|\text{ham})$
  - $P(\text{ham}|\text{am}) = 0$
  - If one probability is 0, the probability of the entire sentence becomes 0.

Smoothing and interpolation techcniques

Dealing with unknown words and n-grams

Two situations in which the resulting probability is 0:

- A word does not occur in the training corpus, hence its probability cannot not estimated
  - blue, carrots, …
- All individual words of an n-gram occur in the training corpus, but the specific n-gram doesn't
  - am ham, like Sam, …

There are several techniques to prevent probabilities from becoming 0:

- **Smoothing**, **backoff**, **interpolation**

Add-one smoothing

- Parameter estimations for a bigram LM:

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

- **Add-one smoothing** (also called **Laplace smoothing**):
  - Pretend we saw each word one more time than we did

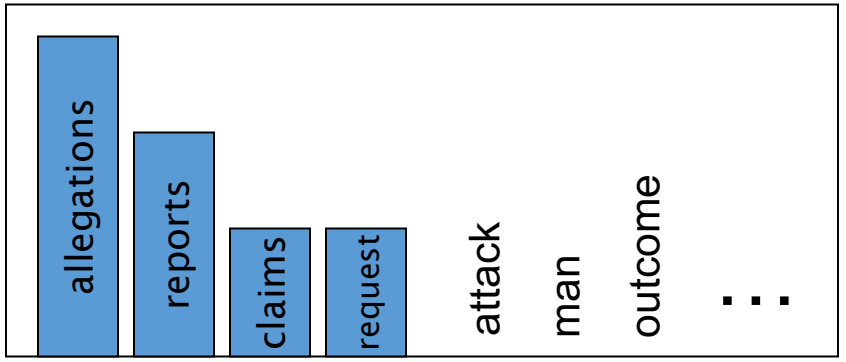
$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + 1 \cdot |V|}$$

  - For any word that we saw 0 times, pretend we saw it once

The intuition of smoothing (from Dan Klein)

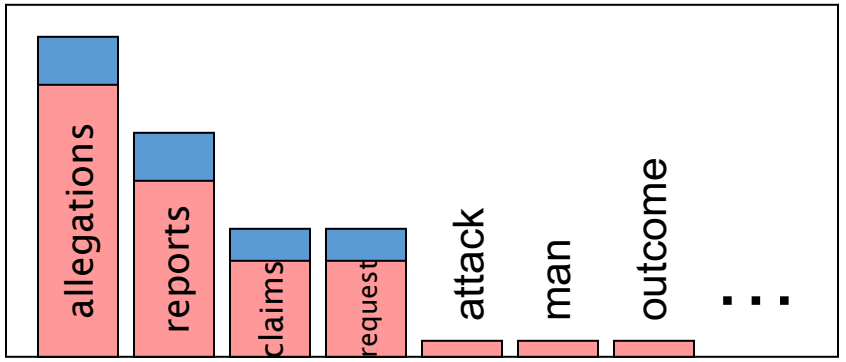
- When we have sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Steal probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Another example

Some bigram counts in a corpus of 9222 sentences:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Another example

Normalize by unigrams:

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

Divide to get probabilities (without smoothing):

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

Another example

Add 1 everywhere:

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Another example

Normalize again to get smoothed probabilities:

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Note: These numbers are really small, which can lead to problems. We'll come back to that issue.

Additive smoothing

- Generalization: **additive smoothing**

$$\hat{P}(x_i|y) = \frac{\text{Count}(x_i, y) + \alpha}{\sum_{x' \in V} (\text{Count}(x', y) + \alpha)}$$

  - Add-one smoothing: $\alpha = 1$

- Additive smoothing is useful for completely unknown words
- But there are better techniques for the second case (unknown n-grams of known words)
  - Note: This issue disappears if we only use a unigram model

Backoff and interpolation

We can train several models with different n-grams on the same data.

- Higher n-gram models are more accurate but contain more zeros.
- Lower n-gram models have fewer zeros but are less accurate.

Backoff:

- If the 4-gram model predicts non-zero score, use that
- Otherwise, if the trigram model predicts non-zero score, use that
- Otherwise, if the bigram model predicts non-zero score, use that
- Otherwise use the unigram model score

Backoff and interpolation

Interpolation: Always use a weighted combination of all models

$$
\begin{aligned}
&P(w_i|w_{i-3}, w_{i-2}, w_{i-1}) \\
&= \lambda_4 \cdot P_4(w_i|w_{i-3}, w_{i-2}, w_{i-1}) \\
&+ \lambda_3 \cdot P_3(w_i|w_{i-2}, w_{i-1}) \\
&+ \lambda_2 \cdot P_2(w_i|w_{i-1}) \\
&+ \lambda_1 \cdot P_1(w_i)
\end{aligned}
$$

- where $\lambda_4 + \lambda_3 + \lambda_2 + \lambda_1 = 1$

Note: We'll have to figure out how to set these λ values. We'll come back to that later.

- Interpolation works better than backoff

Evaluation of language models

Evaluation

How do we know if a LM is good? How do we know if one LM is better than another one?

A first attempt:

- Take a sentence from a corpus
- Pick the first word and repeatedly ask the LM to predict the next word
- Compute the similarity between the predicted sentence and the original sentence
- Repeat for other sentences and report average similarity

Evaluation

Problems:

- If the sentence starts with *The*, there are thousands of legitimate continuations and it is impossible for the LM to know which one is the desired one
- Computing a similarity value between two sentences of different lengths is not trivial

Possible solution:

- Feed the entire sentence to the LM and just ask it to produce the probability

Evaluation

- Intuition: A good LM prefers "real" sentences
  - Assigns a higher probability to “real” or “frequently observed” sentences
  - Assigns a lower probability to “word salad” or “rarely observed” sentences
- A good LM is one that assigns a higher probability to the actually occurring next word than to other words
  - Ideally, but not necessarily, the highest probability

Experimental setup

We need data:

1. To estimate the probabilities
   - This is called the **training set**
2. To evaluate the LM
   - This is called the **test set**

- A test set is an unseen dataset, different from the training set.
  - We want to measure how well the LM generalizes to unseen data.
  - Training and test set may not overlap!

Experimental setup

- Sometimes we need to set hyperparameters
  - E.g., the α for smoothing, or the λs for interpolation
- We could try different values and see which value gets the best scores on the test set.
  - But then, the test set isn't really "unseen". The hyperparameter values may not generalize to other data.

Three data splits:

1. To estimate the probabilities
   - This is called the **training set** (~80%)
2. To find the best hyperparameter values
   - This is called the **development set** (~10%)
3. To evaluate the LM
   - This is called the **test set** (~10%)

Evaluation of LMs

Suggested solution: Feed each sentence of the test set to the LM and ask it to produce the probability

- But the probability depends on the length of the sentence: It becomes smaller the longer the text
- It would be better to have a measure that is normalized by length

- **Perplexity** is the inverse probability of the sentence, normalized by the number of words:

$$PPL(w_1, w_2, \dots, w_n) = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_n)}}$$

Perplexity

Perplexity is the **inverse** probability of the sentence, normalized by the number of words:

$$PPL(w_1, w_2, \dots, w_n) = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_n)}}$$

- Minimizing perplexity is the same as maximizing probability
- Probability range is $[0,1]$, perplexity range is $[1, \infty]$

Perplexity with bigram Markov assumption:

$$PPL(w_1, w_2, \dots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_{i-1})}}$$

Evaluation of LMs

- LMs with different orders were trained and evaluated
  - Same training data: 38 million words from WSJ
  - Same test data: 1.5 million words from WSJ
- Resulting perplexities of the test set:

	Unigram	Bigram	Trigram
Perplexity	962	170	109

  - Note: The test set typically contains several sentences. Perplexity is simply computed over the concatenation of all sentences.

Extrinsic evaluation

LMs are sometimes used as components of larger NLP systems:

- Machine translation
  - Use a LM to choose between different options
  - a tall woman vs. a high woman
- Speech recognition
  - Use a LM to choose between different options
  - I will be back soonish vs. I will be bassoon dish

It is also possible to evaluate an LM on the impact it has on the final output of the system (**downstream task**). This is called **extrinsic evaluation**.

Subword segmentation

Unseen words

Unseen words = words that did not occur in the training corpus. Examples?

- Morphological processes (inflection, derivation, compounding) allow formation and understanding of unseen words
- No training corpus is large enough to contain all existing names, numbers, etc.

How does a LM deal with unseen words?

- Prediction: no, cannot predict unseen words
- Assign probabilities: yes, thanks to smoothing, but approximative

Can we do better?

Character-level LMs

Instead of splitting the data into words, split it into single characters

Bigram character LM:
Fif thad yourty
Fare sid on Che as al my he sheace ing.
Thy your thy ove dievest sord wit whand of sold iset?
Commet laund hant.
KINCESARGANT:
Out aboy tur Pome you musicell losts, blover.
How difte quainge to sh,
And usbas ey will Chor bacterea, and mens grou:

Four-gram character LM:
First Office, masters
To part at that she may direct my brance
I would he dead. Pleaseth profit,
Then we last awaked you to again,
Far that night I'll courteous Herneath,
Of circle off.
SPEED:
Not you.

Ten-gram character LM:
First Citizen:
Nay, then, that was hers,
It speaks against your other service:
But since the
youth of the circumstance be spoken:
Your uncle and one Baptista's daughter.
SEBASTIAN:
Do I stand till the break off.
BIRON:
Hide thy head.

https://nbviewer.org/gist/yoavg/d76121dfde2618422139

Character-level LMs

Character-level LMs can predict unseen words, but:

- Require much higher n-grams
  - Even a 10-gram model may cover only 1-2 words…
- Can also generate non-existing words

Is there an intermediate solution?

- **”Subwords”**: Can be entire words, or single characters, or anything in between
- Examples: BPE (byte-pair encoding), SentencePiece

Subword segmentation

Wishlist for a subword segmentation method

- Split words into pieces of variable length
  - In most cases, longer than single characters
  - In most cases, shorter than entire words
- Ideally in a morphologically sensible way
  - Such that any unseen word can be decomposed into subwords
- Without using any external resources
  - Shouldn’t rely on morphological analyzers, dictionaries, grammars, …

Subword segmentation schemes

- Byte pair encoding (BPE) ~2016-2019

  R. Sennrich, B. Haddow & A. Birch (2016). *Neural Machine Translation of Rare Words with Subword Units*. Proceedings of ACL 2016.

- SentencePiece unigram model Since 2019

  T. Kudo & J. Richardson (2018). *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. Proceedings of EMNLP 2018 System Demonstrations.

Byte pair encoding

- General algorithm:
  - Start with a vocabulary of single characters
  - Merge frequent character n-gram pairs into a new n-gram
  - Stop after k merging steps (this controls the final vocabulary size)
- Example:

Dictionary	
l o w	5
l o w e r	2
n e w e s t	6
w i d e s t	3

Vocabulary
l, o, w, e, r, n, s, t, i, d

Byte pair encoding

Dictionary	
l o w	5
l o w e r	2
n e w **es** t	6
w i d **es** t	3

Vocabulary
l, o, w, e, r, n, s, t, i, d, **es**

The pair e+s has a frequency of 9. Merge it.

Dictionary	
l o w	5
l o w e r	2
n e w **est**	6
w i d **est**	3

Vocabulary
l, o, w, e, r, n, s, t, i, d, es, **est**

The pair es+t has a frequency of 9. Merge it.

Dictionary	
lo w	5
lo w e r	2
n e w est	6
w i d est	3

Vocabulary
l, o, w, e, r, n, s, t, i, d, es, est, **lo**

The pair l+o has a frequency of 7. Merge it.

Byte pair encoding

Suppose we have the following new word: *lowest*

- l o w e s t (split into characters)
- l o w **es** t (apply merge *e s* → *es*)
- l o w **est** (apply merge *es t* → *est*)
- lo w **est** (apply merge *l o* → *lo*)
- Final segmentation: *lo w est*

Byte pair encoding

- Automatically creates a fixed-size vocabulary of subwords
- The operations learned on the training data can be applied to unseen words
- Compression of frequent character sequences improves efficiency (compared to a character-level model)
- Final vocabulary size is determined by a *hyperparameter*: the number of merging steps

Implementations

Name / Year	**subword-nmt** (2016)	**SentencePiece** (2019)
URL	https://github.com/rsennrich/subword-nmt	https://github.com/google/sentencepiece
Algorithms	BPE	Unigram LM BPE
Output format	Marks word continuation with @@: Hello wor@@ ld@@ .	Marks whitespace with _: Hello _wor ld .
Tokenization	Requires pre-tokenized text	Performs tokenization with the same method
Parameter	Number of merge operations	Size of resulting vocabulary

Subword n-gram language models

With training data:

- Determine the subword vocabulary (**token learner**)
- Segment the data into subwords (**token segmenter**)
- Train a n-gram LM on the segmented data

With test data:

- Segment the data into subwords (**token segmenter**)
- Compute perplexity on the segmented data

Note:

- Nowadays, *tokenization* is frequently used as a synonym for *subword segmentation*.

Readings

- Jurafsky & Martin, chapter 3:
  N-gram language models
- Jurafsky & Martin, chapter 2.5.2:
  Byte-pair encoding
- (Jurafsky & Martin, chapter 13.2.1:
  On the differences between subword segmentation techniques)