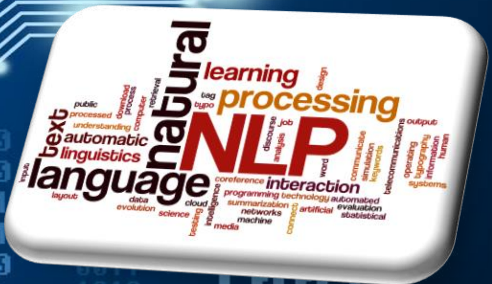
Cairo University

Cairo University
Faculty of Engineering
Computer Engineering Department

Natural Language Processing

Dr. Sandra Wahid



Prediction

- Predict next few words someone is going to say? What word, for example, is likely to follow

"Please turn your homework ..."

in/over but not refrigerator/the

- We will introduce models that assign a **probability** to each possible **next word**.

- The same models will also serve to assign a probability to an **entire sentence**.

Prediction

Why predict upcoming words, or assign probabilities to sentences?

- Speech Recognition: identify words in noisy, ambiguous input
  - Ex: *"I will be back soon"* is more probable than *"I will be back tone"*
- Spelling Correction/Grammatical Error Correction:
  - Ex: *"Their are two midterms" There* was mistyped as *Their.* The phrase *There are* will be much more probable than *Their are*
- Machine Translation:
  - Ex: Suppose we are translating a Chinese source sentence to English:
  - The following set consists of potential rough English translations:

他	向	记者	介绍了	主要	内容
He	to	reporters	introduced	main	content

de msh gomla gmb b3dha, kol wahda bt3br 3n 7aga.

he introduced reporters to the main contents of the statement
he briefed to reporters the main contents of the statement
he briefed reporters on the main contents of the statement

  - A probabilistic model of word sequences could suggest that *"briefed reporters on"* is a more probable English phrase than *"briefed to reporters"* (which has an awkward *to* after *briefed*) or *"introduced reporters to"* (which uses a verb that is less fluent English in this context), allowing us to correctly select the boldfaced sentence above.

Language Models (LMs)

- Models that assign probabilities to sequences of words are called language models or LMs, examples: **N-Gram**, **RNN** LMs.

- An **n-gram** is a sequence of **n words**:
  - a 2-gram (**bigram**) is a *two-word* sequence of words like “please turn”, “turn your”, or “your homework”
  - a 3-gram (**trigram**) is a *three-word* sequence of words like “please turn your”, or “turn your homework”.

Goals:
- estimate the **probability of the last word** of an n-gram given the previous words
- assign **probabilities to entire sequences**

N-Gram

- Computing *P(w|h)*, the probability of a word *w* given some history *h*.
  - Example: the history h is *"its water is so transparent that"* and we want to know the probability that the next word is *the*: $P(the|its\ water\ is\ so\ transparent\ that)$.
- One way to estimate this probability is from **relative frequency counts** using very large corpus.
  - "Out of the times we saw the history *h*, how many times was it followed by the word *w*"

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

C -> Count # of

Problems with this scheme:

- Even the web **isn't big enough** to give us good estimates in most cases.
  - Some cases can have **zero counts**.

→This is because language is creative.

feh gomal ktera msh htl2eha mwgoda.

El mushkela eny 3auz corpus kber gedan w feh gomal keter gedan metkrra, fa el estimates bt3tk msh htb2a dakeka.

N-Gram

- Computing probabilities of entire sequences like *P(w1;w2;…..;wn)* → use **chain rule of probability**

P(A,B) = P(A|B) P(B)

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

$$P(X_1 \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1})$$
$$= \underbrace{\prod_{k=1}^{n} P(X_k|X_{1:k-1})}$$

Applying the chain rule to words, we get

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

But using the chain rule doesn't really seem to help us!
We don't know any way to compute the exact probability of a word given a long sequence of preceding words
→we can **approximate** the history by just the last few words.

Bigram Model

- Approximates the probability of a word given all the previous words by using only the conditional probability of the **preceding word.**

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

- The assumption that the probability of a word depends only on the previous word is called a Markov assumption.

- Generalization of the bigram:

: means -> to

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

  - where N=2→bigram, N=3→trigram,

N here is the number of the N used in the N-Gram
so in bigram N = 2, triGram, so N = 3
and so on.

Maximum Likelihood Estimation (MLE)

Chain Rule mohema gedan. + **Markov Assumption** = $P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$

- How do we estimate these bigram or n-gram probabilities?
  - An intuitive way to estimate probabilities is MLE by getting counts from a corpus, and **normalizing** the counts so that they lie between 0 and 1.
  - Count of the bigram *C(xy)* and normalize by the sum of all the bigrams that share the same first word *x*:

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

  - We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}

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

special case: if the word was at the end, we may just insert any special word to make it correct!

For the general case of MLE n-gram parameter estimation:

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

Some Practical Issues

- In practice it's more common to use trigram models, which condition on the previous two words rather than the previous word, or 4-gram or even 5-gram models, when there is sufficient training data.

- Represent and compute language model probabilities in log format as **log probabilities**.
  - Since probabilities are less than or equal to 1, the more probabilities we multiply together, the smaller the product becomes→ may result in **numerical underflow**.
  - Adding in log space is equivalent to multiplying in linear space.
  - To convert back into probabilities if we need to report them at the end; then we can just take the exp of the logprob.

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

Evaluating Language Models

- Does our language model prefer good sentences to bad ones?

→Assign higher probability to "real" or "frequently observed" sentences than "ungrammatical" or "rarely observed" sentences.

Performance evaluation of a language model

Extrinsic Evaluation:

embed it in an application and measure how much the **application** improves →running big NLP systems end-to-end is often very expensive

Intrinsic Evaluation:

measures the quality of a model **independent** of any application →need a **test set** (**held out** corpora) that are not part of the training set

Evaluating Language Models: Perplexity

- In practice, we don't use raw probability as our metric for evaluating language models but a variant called *perplexity* (sometimes called *PP* for short).

- PP of a language model on a test set is the **inverse probability** of the test set, normalized by its number of words.

- For a test set $W = w_1 w_2 \ldots\ldots w_N$:

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

- Using chain rule:

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

- For bigram model:

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

Evaluating Language Models: Perplexity

Minimizing perplexity is equivalent to **maximizing** the test set probability. If a model assigns a high probability to the test set, it means that it is **not surprised** to see it (it’s **not perplexed** by it).

- The entire sequence of words in some test set will cross many sentence boundaries

→we need to include the begin- and end-sentence markers <s> and </s> in the probability computation.

- Example of how perplexity can be used to compare different n-gram models.

	Unigram	Bigram	Trigram
Perplexity	962	170	$\boxed{109}$

- An (intrinsic) improvement in perplexity **does not guarantee** an (extrinsic) improvement in the performance of a language processing task. Perplexity is commonly used as a quick check on an algorithm. But a model’s improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.

Note

- If there is **no overlap** in the generated sentences between the training set and the test set, then the model is **useless**.

Use a training corpus that has a similar genre to whatever task we are trying to accomplish,

e.g., to build a language model for a question-answering system, we need a training corpus of questions.

It is equally important to get training data in the appropriate dialect.

Zeros

- If any corpus is limited, some perfectly acceptable English **word sequences** are subject to be missing from it

→we'll have many cases of **"zero probability n-grams"** that should really have some **non-zero probability**.

- Example: the words that follow the bigram <*denied the*> with their counts:

denied the allegations:	5
denied the speculation:	2
denied the rumors:	1
denied the report:	1

But suppose our test set has phrases like:

denied the offer
denied the loan

Our model will incorrectly estimate that the *P(offer|denied the)* is **0**

Problems:

1. underestimating the probability of all sorts of words that might occur which will hurt the **performance** of any application.
2. if the probability of any word in the test set is 0, the entire **probability of the test set is 0** and we can't compute perplexity at all, since we can't divide by 0.

Unknown Words

The previous slide discussed the problem of words whose bigram probability is zero. But what about words we simply **have never seen before**?

- We can have a **closed vocabulary** system: where the test set can only contain words from a certain lexicon, and there will be no unknown words.

- Unknown words, or out of vocabulary (OOV) words:
  - The percentage of OOV words that appear in the test set is called the **OOV rate**.
  - An **open vocabulary** system is one in which we model these potential unknown words in the test set by adding a pseudo-word called <UNK>.

Unknown Words

- <u>Possible Solutions:</u>

1. Turn the problem back into a closed vocabulary one:

   1. **Choose a vocabulary** (word list) that is fixed in advance.
   2. **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token <UNK> in a text normalization step.
   3. **Estimate** the probabilities for <UNK> from its counts just like any other regular word in the training set.

2. Create such a vocabulary implicitly:

   by replacing words in the training data by <UNK> based on their frequency. For example, we can replace by <UNK> all words that occur fewer than n times in the training set, where n is some small number.

Smoothing

- What do we do with words that are in our vocabulary (they are **not unknown words**) but appear in a test set in an **unseen context** (for example they appear after a word they never appeared after in training)?

- Zeros problem solution: to keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen.

This modification is called smoothing or discounting

Laplace Smoothing

- Alternate name **add-one** smoothing: since it adds one to each count.
  - All the counts that used to be zero will now have a count of 1

- Unigram probabilities: $$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

N: total number of word tokens
Need to adjust the denominator to take into account the extra V observations.
V: number of unique words

- Bigram probabilities: $$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

Add-k Smoothing

- One alternative to add-one smoothing is to move a **bit less** of the probability mass from the seen to the unseen events.
  - Instead of adding 1 to each count, we add a **fractional count k** (.5? .05? .01?).

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

- Requires a method for **choosing k**; this can be done, for example, by optimizing on a development set (devset).

Backoff and Interpolation

- The **discounting** we have been discussing so far can help solve the problem of **zero frequency n-grams**. But there is an additional source of knowledge we can draw on.

- If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_nw_{n-2}w_{n-1}$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$ we can look to the unigram $P(w_n)$.

- In other words, sometimes using **less context** is a good thing, helping to **generalize** more for contexts that the model hasn't learned much about.

Backoff and Interpolation

1. **Backoff:** we use the *trigram* if the evidence is sufficient, otherwise we use the *bigram*, otherwise the *unigram*. →**we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order**.

2. **Interpolation:** we always **mix the probability estimates from all the n-gram** estimators, weighing and combining the *trigram*, *bigram*, and *unigram* counts.

   Example: simple linear interpolation:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+ \lambda_2 P(w_n|w_{n-1}) \\ &+ \lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$


Thank You