CS 5034/6134 Natural Language Processing
Fall 2024

Lecture 4:
N-gram Language Models

University of CINCINNATI

Language Models

- How will a machine know what is a plausible language?

  我like自然XXXYUYAN$%^PROcessing^_^

- How to distinguish between word salad (random words) and normal sentences?

  "lamb apple water marry" vs. "mary had a little lamb"

- How to automatically correct spelling errors or grammatical mistakes?

  "fantastci" vs. "fantastic"

- How to generate human-like languages?

  I love natural language ___

Language Models

Language models define probability distributions over the sequences of words in a language.

Example: Imagine a spelling checker, if it knows
Prob("fantastci") << Prob("fantastic")
Then it will make a correction suggestion.

The question is, how to define these probabilities?

Probability Theory Recap

- Sample Space

  The sample space of a random experiment is defined as the set of all possible outcomes of an experiment.

- Sample point

  A sample point is an element of a sample space.

- Event

  An event is a subset of the sample space.

Probability Theory Recap

Think about tossing a coin, with two possible outcomes: heads (H) and tails (T).

- If I toss it once, what is the sample space?
  - {H, T}
- If I toss it twice, what is the sample space?
  - {HH, HT, TH, TT}
- If I toss it twice, what is the event that I have different outcomes?
  - {HT, TH}

Probability Theory Recap

Probability of an event is the sum of the probabilities of the individual sample point of which it is composed.

If I toss a "fair" coin twice, what is the probability that I have different outcomes?

Sample space: {HH, HT, TH, TT}

Event A: {HT, TH}

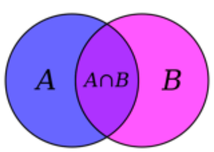
Prob(A) = Prob(HT) + Prob(TH) = 1/4 + 1/4 = 1/2

Probability Theory Recap

- Joint Probability

$P(A \cap B)$ sometimes also written as $P(A, B)$

- Combining events



$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- Conditional Probability

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Conditional Probability

- Conditional Probability

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Suppose I flip a "fair" coin twice. Given that at least one head was obtained, what is the probability of obtaining two heads?

Sample space: {HH, HT, TH, TT}

Event A - two heads: {HH}

Event B - at least one head: {HT, HH, TH}

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{1/4}{3/4} = \frac{1}{3}$$

Chain Rule

The joint probability $P(A, B)$ can also be expressed in terms of the conditional probability $P(A \mid B)$:

$$P(A, B) = P(A \mid B)P(B)$$

Generalizing this to N joint events in a general form is called the **chain rule**:

$$P(A_1, A_2, \dots, A_n) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1, A_2)\dots P(A_n \mid A_1, \dots A_{n-1})$$
$$= \prod_{i=1}^{n} P(A_i \mid A_1, \dots, A_{i-1})$$

Independence

Two events A and B are independent iff

$$P(A \mid B) = P(A)$$

By substituting it in the joint probability:

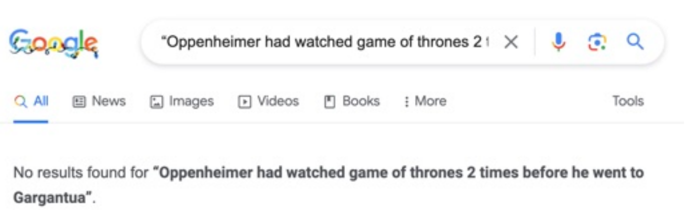
$$P(A, B) = P(A \mid B)P(B) = P(A)P(B)$$

Chain rule for n independent events:

$$P(A_1, A_2, \dots, A_n) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1, A_2)\dots P(A_n \mid A_1, \dots A_{n-1})$$
$$= \prod_{i=1}^{n} P(A_i)$$

Language Models in NLP

- In many scenarios, it will be difficult to know how likely a sentence is to occur in “natural language”, like English.



Language Models in NLP

- But we can define a language model that give us good approximations.
- N-gram models are the simplest and most common kind of language model.
- Let’s see how these models are defined, how to learn their parameters and what they can do.

N-grams

An n-gram is a word sequence of length n.

1-gram or **unigram**

2-gram or **bigram**

3-gram or **trigram**

Mary had a little lamb

unigrams: Mary, had, a, little, lamb

bigrams: Mary had, had a, a little, little lamb

trigrams: Mary had a, had a little, a little lamb

N-gram Language Model

Assume we have some **training data**: a large corpus of general English text.

The set of all words in this corpus is called its **vocabulary**.

We want to have a language model over the vocabulary V that estimate the probability of any given sequence.

How to compute the probability of a sequence of words "$w_1\ w_2\ \ldots\ w_n$"?

What is the most intuitive way?

Using Frequency Counts

We estimate the probability of an N-gram using the training data.

$$P(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{\#\ of\ all\ sentences}$$

where $C(x)$ is the count of x in our text corpus, the denominator is the total number of sentences in the text corpus.

In the previous example, we have

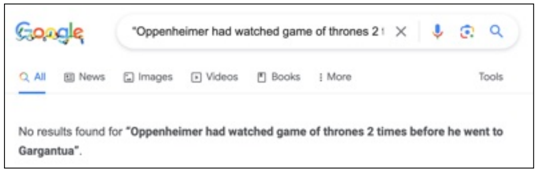
$$P(Mary\ had\ a\ little\ lamb) = \frac{C(Mary\ had\ a\ little\ lamb)}{\#\ of\ all\ sentences}$$

N-gram Language Model

VS.

N-gram Language Model

- What if the text corpus does not contain this sentence, i.e., $C(x) = 0$?



- Recall the **chain rule**:

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

- Yet still, many of these conditional probabilities can still be zero! (what is the last term?)

Independence Assumption

So we make an independence assumption: the probability of a word only depends on the most recent past words.

unigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i)$

bigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1})$

trigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1})$

This is also called Markov assumption.

Independence Assumption

So now we have:

unigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i)$

bigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$

trigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$

N-gram Language Model Example

Mary had a little lamb

unigram model: P = P(Mary)*P(had)*P(a)*P(little)*P(lamb)

bigram model: P = P(Mary)*P(had | Mary)*P(a | had)*P(little | a)*P(lamb|little)

trigram model: P = P(Mary)*P(had | Mary)*P(a | Mary, had)*P(little | had, a)*P(lamb | a, little)

How to get these conditional probabilities?

Computing N-gram Probabilities

For the unigram probability,

In our example, we estimate

$$P(lamb) = \frac{C(lamb)}{N}$$

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

More generally, for any unigram, we have

$$P(w_i) = \frac{C(w_i)}{\sum_{x'} C(x')}$$

Computing N-gram Probabilities

For the conditional probability,

In our example, we estimate

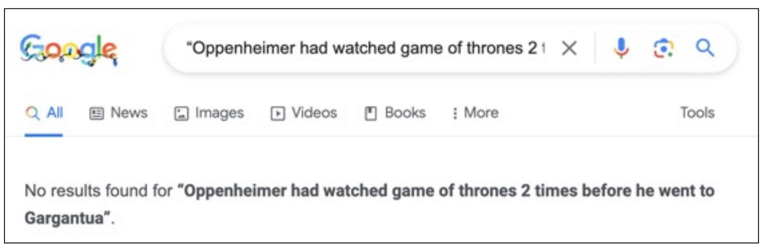
$$P(lamb \,|\, a, little) = \frac{C(a, little, lamb)}{C(a, little)}$$

where $C(x)$ is the count of x in our text corpus.

More generally, for any trigram, we have

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

Computing N-gram Probabilities



Now to estimate the probability of this sentence, we only need to find out the probabilities of a set of n-grams. If n=2, then we need:

P(Oppenheimer), P(had), P(watched), …

P(Oppenheimer, had), P(had, watched), P(watched, game), …

Practical Details 1: Beginning/End of Sequence

Trigram model assumes two word history:

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

But consider this pair:

	w_1	w_2	w_3	w_4
(1)	Mary	had	a	little
(2)	had	a	little	lamb

What is wrong?

Practical Details 1: Beginning/End of Sequence

To capture behavior at beginning/end of sequences, we manually define a beginning and end pseudo word:

	w_{-1}	w_0	w_1	w_2	w_3	w_4	w_5
(1)	<s>	<s>	Mary	had	a	little	</s>
(2)	<s>	<s>	had	a	little	lamb	</s>

Now P(had | <s>, <s>) and P(</s> | a, little) are low, so we know they are not good sentences.

Our equation becomes

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n+1} P(w_i \mid w_{i-2}, w_{i-1})$$

Practical Details 2: Log Probability

- Word probabilities are typically very small.
- The longer the sentence, the smaller the probability will become (more probabilities multiply together).
- Multiplying lots of small numbers will lead to numerical underflow, even using double precision floating point.
- So we use log space instead of linear space.

  $$p_1 * p_2 * p_3 * p_4 = e^{\log p_1 + \log p_2 + \log p_3 + \log p_4}$$

  just compute $\rightarrow$ $\log p_1 + \log p_2 + \log p_3 + \log p_4$

Practical Details 3: Unknown Words

- What about words we have never seen before (not in training set but pops up in test set)?
- Imagine we pick the corpus as the collection of Shakespeares's plays. Now we want to compute the probability for the sentence:

  "I play computer games"

  However, "computer" does not exist in Shakespeare's works!
- We call them **unknown words**, or out of vocabulary (**OOV**) words.
- To prevent zero probability, we can add a pseudo **<UNK>** word to represent any unknown words in the test set.

Practical Details 3: Unknown Words

- How to define the probabilities of <UNK> words?

  1. Select a prior vocabulary, anything in the training corpus but not in the vocabulary are considered as <UNK>
  2. Set a frequency threshold, anything in the training corpus below this threshold are considered as <UNK>

Practical Details 4: Smoothing

- Estimating probabilities works well when you have a lot of data. But no matter how "big" a corpus is, there will always be rare words.

  For example: the Brown corpus contains about 1 million words. But it contains only 49,000 unique words. And over 40,000 of those words occur <= 5 times!

- A particularly difficult problem is when word combinations **never** appears in the training data.

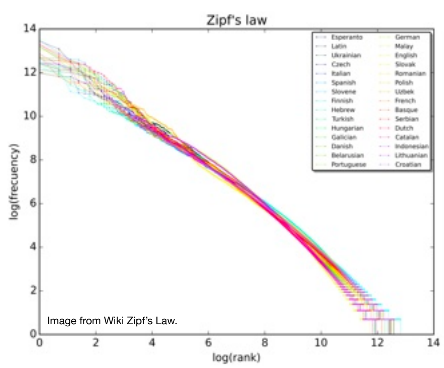
- How do we prevent zero probability in this case?

Zipf's Law

- How "common" are common words and how "rare" are rare words?

- For example, in the Brown Corpus,
  - the word "the" is the most common word, which accounts for nearly 7% of all word occurrences (69,971 out of 1 million).
  - the word "of" is the second most frequent word, about 3.5% of words (36,411)
  - third rank "and" (28,852)

Zipf's Law



$$\text{word frequency} \propto \frac{1}{\text{word rank}^{\alpha}}$$

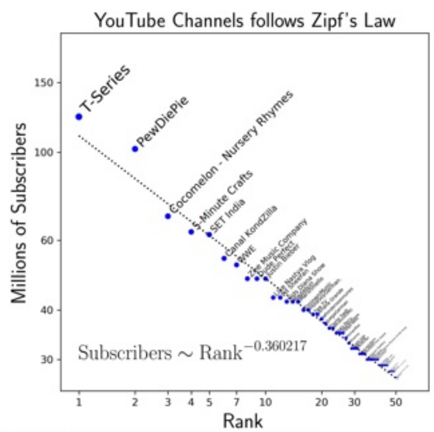
Zipf's Law

- A corpus will have a number of **common** words. We see them often enough to know (almost) anything about them.

- Regardless of how large our corpus is, there will be a lot of **rare** words and **unknown** words.

- This means we need to find clever ways to estimate probabilities for things we have rarely or never seen.

Zipf's Law in other Contexts



Source: Wikipedia pages "List of most-subscribed YouTube channels" and "Zipf's law"

Heaps' Law

- What is the relation between corpus size and vocabulary size?
- A corpus of N tokens has a vocabulary size:

  $$|V| \propto N^{\beta}$$

  where $0 < \beta < 1$.
- Both Zipf's law and Heaps' law are empirical.

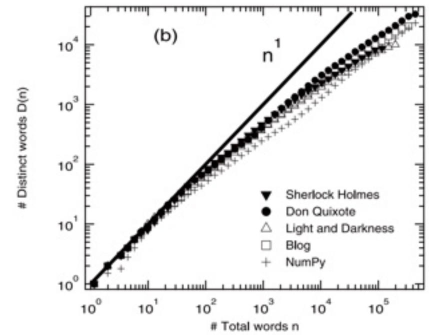


Image from: Sano, Yukie; Takayasu, Hideki; Takayasu, Misako (2012). "Zipf's Law and Heaps' Law Can Predict the Size of Potential Words". Progress of Theoretical Physics Supplement. 194: 202–209

Practical Details 4: Smoothing

- We apply a "modification" to the probabilities, which is called **smoothing**.
- To put it in an intuitive way, smoothing methods address the unseen combination (**NOT** unknown words) problem by stealing probability mass from seen events and reallocating it to unseen events.
- Different ways of smoothing.

Add-One (Laplace) Smoothing

- Assume we add 1 to the count of every possible n-gram
- Example, for unigram,

  $$P(w_i) = \frac{C(w_i)}{N}$$

  where $N = \sum_{x'} C(x')$
- After smoothing:

  $$P(w_i) = \frac{C(w_i) + 1}{N + |V|}$$

Add-One (Laplace) Smoothing

- Assume we add 1 to the count of every possible n-gram
- Example, for bigram,

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

After smoothing:

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

Why?

- We still need to make sure

$$\sum_{w'} P(w' \mid w_{i-1}) = 1$$

Add-One (Laplace) Smoothing

- However, add-one smoothing is not friendly to "common" n-grams.
- Suppose "little" occur 100 times, "little, lamb" occurs 10 times, and vocabulary size = 20,000.

Before smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} = \frac{C(little, lamb)}{C(little)} = \frac{10}{100} = 0.1$$

After add-one smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|} = \frac{10 + 1}{100 + 20{,}000} \approx 0.0005$$

Add-α Smoothing

- Assume we add α ($\alpha < 1$) to the count of every possible n-gram
- Example, for bigram:

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

After smoothing:

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

Linear Interpolation Smoothing

- Interpolate n-gram model with k-gram model (k<=n)
- bigram

$$\hat{P}(w_i \mid w_{i-1}) = \lambda P(w_i) + (1 - \lambda) P(w_i \mid w_{i-1})$$

- trigram

$$\begin{aligned} \hat{P}(w_i \mid w_{i-2}, w_{i-1}) = & \lambda_1 P(w_i) && \textbf{unigram} \\ & + \lambda_2 P(w_i \mid w_{i-1}) && \textbf{bigram} \\ & + (1 - \lambda_1 - \lambda_2) P(w_i \mid w_{i-2}, w_{i-1}) && \textbf{trigram} \end{aligned}$$

N-gram Model Summary

- To estimate the probability of a sequence of words $w_1, w_2, \ldots, w_n$, we need:
  - a training corpus
  - chain rule
  - independence assumption
- As a result, we get (here we use trigram):

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1}) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

- Then estimate each trigram probability from the training corpus:

$$P(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

- Beginning/end symbol, UNK symbol, using logarithms, smoothing

Generating text with Language Models

- How do we use language models: a random sentence generator.
- Suppose we already learned a bigram language model, how to generate a sequence?

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

  - Generate the 1st word $w_1 \sim P(w \mid \text{<s>})$
  - Generate the 2nd word $w_2 \sim P(w \mid w_1)$
  - Generate the 3nd word $w_3 \sim P(w \mid w_2)$
  - …
  - Until </s> is generated.

Generating text with Language Models

- Trigram language model:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

  - Generate the 1st word $w_1 \sim P(w \mid \text{<s>}, \text{<s>})$
  - Generate the 2nd word $w_2 \sim P(w \mid \text{<s>}, w_1)$
  - Generate the 3nd word $w_3 \sim P(w \mid w_1, w_2)$
  - …
  - Until </s> is generated.

Generating text with Language Models

- How do we select w_3 based on $P(w \mid w_1, w_2)$?
- Greedy

  Idea: choose the most likely word

$$w_3 = \arg\max_{w \in V} P(w \mid w_1, w_2)$$

…a major problem is a major problem is a major problem is a major problem is a major problem is a major problem…

Generating text with Language Models

- How do we select w_3 based on $P(w \mid w_1, w_2)$?
- Greedy

  Idea: choose the most likely word

$$w_3 = \arg\max_{w \in V} P(w \mid w_1, w_2)$$

- Sampling

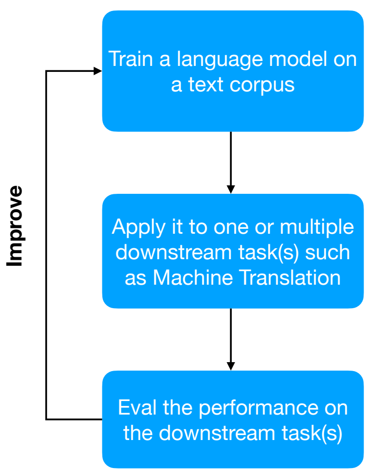
  Top-k: randomly select from top-k words with the highest probability

  Top-p: randomly select from the smallest set of tokens for which the cumulative probability reach a specified value p

Evaluating Language Models

- How do we know if one language model is better than another?
- Two types of evaluation in NLP
  1. **Intrinsic Evaluation**: design a measure that is inherent to the current task
  2. **Extrinsic Evaluation**: measure performance on a downstream application

Extrinsic Evaluation



- The most reliable evaluation.
- Can be more time consuming.
- Directly target downstream task.

Word Error Rate

- Originally developed for speech recognition.
- How much does the predicted sequence of words differ from the actual sequence of words in the correct transcript?

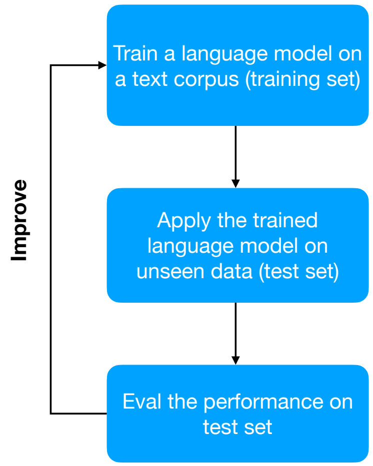
$$\text{WER} = \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\text{Actual words in transcript}}$$

Insertions: “had little lamb” → “had **a** little lamb”

Deletions: “go **to** home” → “go home”

Substitutions: “the **star** night” → “the **starry** night”

Intrinsic Evaluation



- Can be much quicker in the development cycle.
- Intrinsic improvement may not guarantee extrinsic improvement.
- Both intrinsic and extrinsic eval require an evaluation metric that allow us to compare the performance of different models.
- It is not always obvious how to design the evaluation metric.

Intrinsic Evaluation of Language Models: Perplexity

- Suppose we have 2 language model that can assign probabilities to sequence of words.
- How do we know which is better?
- We have a training set, and a test set.
- We trained the 2 models on the training set, and compute the probability of the test set. Whichever model has a high probability is better.

Intrinsic Evaluation of Language Models: Perplexity

- We have a language model that can assign probabilities to sequence of words.
- The **perplexity** of this model on a sequence of words $w_1, \ldots, w_n$ is defined as

$$\text{Perplexity}(w_1, \ldots, w_n) = P(w_1, \ldots, w_n)^{-\frac{1}{n}}$$
$$= \sqrt[n]{\frac{1}{P(w_1, \ldots, w_n)}}$$
$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}}$$

Intrinsic Evaluation of Language Models: Perplexity

$$\text{Perplexity}(w_1, \ldots, w_n) = P(w_1, \ldots, w_n)^{-\frac{1}{n}}$$

- unigram vs. bigram vs. trigram?
- Lower perplexity is better (higher probability)
- Perplexity (PPL) of unigram > PPL of bigram > PPL of trigram

	unigram	bigram	trigram
Perplexity	962	170	109

Train and test on the Wall Street Journal corpus

Intrinsic Evaluation of Language Models: Perplexity

- If my language model gets a perplexity on some dataset as 20. Is it good or not?
- Two language models' perplexity can only be compared if they use the same vocabularies.
  - Some language might be more predictable

Imagine a language with vocabulary size $|V|$. Suppose each word has equal probability, i.e., $P(w) = \frac{1}{|V|}$. What is the perplexity of the unigram model?

Intrinsic Evaluation of Language Models: Perplexity

Imagine a language with vocabulary size $|V|$. Suppose each word has equal probability, i.e., $P(w) = \frac{1}{|V|}$. What is the perplexity of the unigram model?

$$
\begin{aligned}
\text{Perplexity}(w_1, \ldots, w_n) &= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1, \ldots, w_{i-1})}} \\
&= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{1/|V|}} \\
&= |V|
\end{aligned}
$$

Intrinsic Evaluation of Language Models: Perplexity

- If my language model gets a perplexity on some dataset as 20. Is it good or not?
- Two language models' perplexity can only be compared if they use the same vocabularies.
  - Some languages might be more predictable ("easy")
- Test data must be disjoint from training data. Knowledge of the test set can cause the perplexity to be artificially low.

Intrinsic Evaluation of Language Models: Perplexity

- Use logarithms to prevent underflow:

$$
\begin{aligned}
\text{Perplexity}(w_1, \ldots, w_n) &= P(w_1, \ldots, w_n)^{-\frac{1}{n}} \\
&= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1, \ldots, w_{i-1})}} \\
&= \exp\Big(-\frac{1}{n}\sum_{i=1}^{n} \log P(w_i | w_1, \ldots, w_{i-1})\Big)
\end{aligned}
$$

Entropy

- In Information Theory, the entropy of a discrete random variable X is defined as:

$$H(X) = -\sum_{x} p(x) \log p(x)$$

- It is also referred to as Shannon entropy.
- It describes the level of “information” or “uncertainty”.
- Here we use base 2 for $\log$, which provides **unit of bits**.

Entropy Example

- Suppose we flip a fair coin. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_{x} p(x) \log p(x) \\ &= -\sum_{i=1}^{2} \frac{1}{2} \log_2 \frac{1}{2} \\ &= 1 \end{aligned}$$

Entropy Example

- Suppose we flip an unfair coin. P(heads)=0.9 and P(tails)=0.1. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_{x} p(x) \log p(x) \\ &= -0.9 \log_2 0.9 - 0.1 \log_2 0.1 \\ &= 0.469 < 1 \end{aligned}$$

Entropy Example

- Suppose we flip an unfair coin. P(heads)=1 and P(tails)=0. The entropy of the outcome:

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

Entropy Example

- Suppose we roll a fair 6-sided dice. The entropy of the outcome:

$$H(X) = -\sum_x p(x)\log p(x)$$
$$= -\sum_{i=1}^{6} \frac{1}{6}\log_2 \frac{1}{6}$$
$$= 2.585$$

- Uniform probability yields maximum uncertainty and therefore maximum entropy.

Entropy of a Language

- Entropy over a sequence $W = \{w_1, \ldots, w_n\}$ from a language L:

$$H(w_1, \ldots, w_n) = -\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

- This will depend on how long the sequence is. To have a more meaningful measure, we get the average, also called entropy rate:

$$\frac{1}{n}H(w_1, \ldots, w_n) = -\frac{1}{n}\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

Entropy of a Language

- Define entropy of the language L:

$$H(L) = -\frac{1}{n}\lim_{n\to\infty}\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

- This can be simplified (Shannon-McMillan-Breiman theorem) to:

$$H(L) = -\frac{1}{n}\lim_{n\to\infty}\log p(w_1, \ldots, w_n)$$

- But we don't know the true probability distribution p.

Cross-entropy

- In practice, we don't know the true probability distribution p for language L, only an estimated distribution $\hat{p}$ from a language model.

- Define cross-entropy as

$$H(p, \hat{p}) = -\sum_x p(x)\log \hat{p}(x)$$

- According to Gibb's inequality, entropy is less than or equal to its cross-entropy, i.e.,

$$-\sum_x p(x)\log p(x) \le -\sum_x p(x)\log \hat{p}(x)$$

Cross-entropy

- According to Gibb's inequality, entropy is less than or equal to its cross-entropy, i.e.,

$$-\sum_x p(x)\log p(x) \leq -\sum_x p(x)\log \hat{p}(x)$$

- This means that if we have two language models, the more accurate model will have a lower cross-entropy.

Cross-entropy

- Following Shannon-McMillan-Breiman theorem,

$$H(p,\hat{p}) = -\frac{1}{n}\lim_{n\to\infty}\log \hat{p}(w_1,\ldots,w_n)$$

- For a language model, lower $H(p,\hat{p})$ is better.
- Recall perplexity, perplexity is simply $2^{\text{cross-entropy}}$.

$$\text{Perplexity}(w_1,\ldots,w_n) = P(w_1,\ldots,w_n)^{-\frac{1}{n}}$$

$$2^{\text{cross-entropy}} = 2^{-\frac{1}{n}\log_2 P(w_1,\ldots,w_n)} = P(w_1,\ldots,w_n)^{-\frac{1}{n}}$$