CMPE 297, INSTRUCTOR: JORJETA JETCHEVA

N-GRAM LANGUAGE MODELS

LANGUAGE MODELS

- Models that assign probabilities to sequences of words are called Language Models (LM)

More formally:

- A language model is the probability distribution describing the likelihood of any combination of words into a sequence/string
  - E.g., “Do I dare disturb the universe?” has a high likelihood in English whereas “Universe dare the I disturb do” has a low likelihood
- A language model captures the full joint probability distribution of sequences composed of words
  - I.e., you can compute a probability for any sequence of words

PROBABILITY BASICS REVIEW

- Joint probability distribution, e.g., P(Weather, Cavity), denotes the probabilities of all combinations of values of the variables
  - A probability model is determined by the joint distribution for all of the random variables for the domain – this is called the *full joint probability distribution*
- A set of variables are independent if they do not influence each other, e.g., P(Weather, Cavity)
  - For variables that are independent, the product rule is:

    $P(A, B) = P(A)P(B)$
  - For variables that are not independent, the product rule is:

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

Weather	Cavity	Probability
Sun	Small	P(sun)P(small)
Sun	Large	P(sun)P(large)
Rain	Small	P(rain)P(small)
Rain	Large	P(rain)P(large)
Cloud	Small	P(cloud)P(small)
Cloud	Large	P(cloud)P(large)
Snow	Small	P(snow)P(small)
Snow	Large	P(snow)P(large)

APPLICATIONS OF LANGUAGE MODELS

- Predict what word would come next: can be used for auto-complete of email and text in general
- Predict what alterations would make a sentence more probable: can be used for spelling and grammar corrections
- Use of a pair of models to find the most probable translation from one language to another
- When training with example question/answer pairs, can help find the most likely answer among a set of choices

N-GRAMS

- An n-gram is a sequence of n words
  - Unigrams (or 1-grams) contain 1 word, e.g., “the”, “cat” , “is”, “hungry”
  - Bigrams (or 2-grams) are sequences of 2-words, e.g., “the cat”, “cat is”, “is hungry”
  - Trigrams (or 3-grams) are sequences of 3-words, e.g., “the cat is”, “cat is hungry”
- An n-gram model predicts the likelihood of n-grams (often the word “model” is dropped)
  - The model looks at the n-1 words that precede the word we are trying to predict
    - E.g., a trigram model may try to predict, the most likely next word, given the first two words “the cat”

NOTATION & BASIC CONCEPTS

- The probability that a random variable X_i takes on a particular value, e.g., the value "the" is denoted as $P(X_i = \text{"the"})$ or $P(\text{the})$
- Probability of the sequence of n words: $w_1, w_2, \ldots, w_n$ is:

  $P(X_1 = w_1, X_2 = w_2, \ldots, X_n = w_n)$ or $P(w_1, w_2, \ldots, w_n)$
- The way to decompose the probability $P(X_1, X_2, \ldots, X_n)$ is based on using the chain rule of probability as follows:

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

In terms of "words", $P(w_{1:n}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1})$, where $w_{1:k-1}$ denotes the words from index 1 to index k-1

The above states that the probability of a word in a sequence depends on the probability of the preceding sequence of words.

N-GRAM MODEL APPROACH

- The n-gram model simplifies this computation by using the last few words to approximate the word history (sequence)

Doubt :
2-gram models: Auto-correct and next word
3-gram models: Email completion suggestion

- Assuming n in n-gram is denoted by N, we can approximate the probability of a word:

$$P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-N+1:n-1})$$, e.g., for 4th word (n = 4) in a 3-gram model (N=3), we get n-N+1 = 4-3+1 = 2, thus $P(w_n \mid w_{n-N+1:n-1}) = P(w_4 \mid w_{2:3})$

- When using bigrams, we get the following probability for a word in a sequence:

$P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-1})$ (Assumption: prob. of w_n only depends on the previous word, w_{n-1})

The probability of a ***sequence*** of words is $P(w_{1:n}) \approx \prod_{j=1}^{n} P(w_k \mid w_{k-1})$ (chain rule from prev. slide)

COMPUTING N-GRAM MODEL PROBABILITIES

But how do we compute this probability for a ***sequence*** of words $P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$?

We use Maximum Likelihood Estimation (MLE) for the bigram scenario, where we count the number of times w_{n-1} is followed by w_n: $C(w_{n-1}w_n)$, relative to all the bigrams that start with w_{n-1}

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

C = Count of number words occurring together

We can simplify this by observing that the unigram counts for each word w_{n-1} are the same as the count of that same word (w_{n-1}) at the start of a bigram, $C(w_{n-1})$:

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

Note that to make up a bigram for the first word, and the last word, we need to create special start and end symbols: <s> denotes the beginning of a sentence, <\s> denotes the end of a sentence

In a trigram setting, we would use <s> <s> and <\s> <\s> to complete the first and last trigram.

EXAMPLE FROM YOUR BOOK

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

$$P(w_{1:n}) \approx \prod_{j=1}^{n} P(w_k \mid w_{k-1})$$

- Sample document corpus

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

- Bigram probability examples based on the corpus:

$$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67 \qquad P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33 \qquad P(\text{am}|\text{I}) = \frac{2}{3} = .67$$
$$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5 \qquad P(\text{Sam}|\text{am}) = \frac{1}{2} = .5 \qquad P(\text{do}|\text{I}) = \frac{1}{3} = .33$$

- Probability of the sentence “I am Sam” using a bigram model:

P(“I am Sam”) = 0.67*0.67*0.5*0.5 (includes start and end sentence tokens)

P(I / <S>) * P(am/I) * P(Sam/am) * P(</s>/Sam)

LOG PROBABILITY-BASED COMPUTATION

- Multiplying many small (probability) values together results in underflow
  - Underflow occurs when a number is too small to be represented on a particular computer architecture
  - Probabilities are always in the range 0-1 and often very close to 0 in n-gram computations
- Typically, we compute the log of the probability and convert back to probability by applying exponentiation at the end:

$$P(\text{“I am Sam”}) = \exp(\log(0.67*0.67*0.5)) = \exp(\log(0.67) + \log(0.67)+\log(0.5))$$

Recall that log of a product is equivalent to the sum of the logs of the individual components of the product.

TRADE-OFFS OF DIFFERENT VALUES OF N IN N-GRAM

- Words can be part of different phrases, each with a different meaning, which short n-grams cannot represent
  - E.g., it takes a 4-gram to capture phrases like “first quarter earnings report” which common only in business, and “fourth quarter touchdown passes” which is common only in sports
- Performance challenges of using large values of n
  - In the absence of the n-gram model approximation, with a vocabulary of 100,000 (10^5) words and a sentences of length 40, we’d need to estimate $(10^5)^{40} = 10^{200}$ parameters if we made each word dependent on all other words in the vocabulary
  - Estimating priors for trigrams given a vocabulary of 100,000 words results in $(10^5)^3 = 10^{15}$ trigrams which is better though still quite a bit

MODEL EVALUATION

GENERAL APPROACH TO ML MODEL EVALUATION

Usually we split the data into training, dev(elopment/validation) and test sets

- Training set is used to train the model
- Dev set is used to evaluate candidate models & model parameters, and choose the best one
- Test set is only used to test the model once the algorithm has been fine-tuned
  - Using a test set that the model has not been trained on, ensures that the model can generalize to unseen data

Dataset, e.g., document corpus

Note that generally, the different sets need to be from the same genre.

EVALUATING LANGUAGE MODELS

- Intrinsic evaluation
  - You evaluate your model based on various performance metrics, and independent of any particular application
  - E.g., measure how well two different language models fit the test set (better fit is indicated by the model assigning a higher probability to the test set)
- Extrinsic or end-to-end evaluation
  - Evaluate how well the overall application in which your model is embedded performs
  - E.g., compare a speech transcriber using two different language models

PERPLEXITY

$$P(w_{1:n}) \approx \prod_{j=1}^{n} P(w_k \mid w_{k-1})$$

- Perplexity (or PP) is a probability-based metric typically used for evaluating language models (rather than using raw probabilities)
- It is the inverse probability of the test set, normalized by the number of words

PP for test set $W = w_1, w_2 \ldots, w_N$:

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

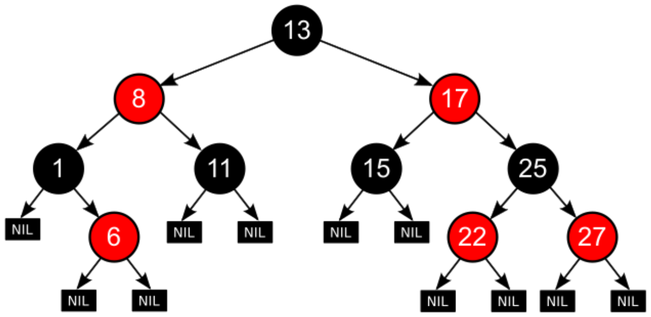
We apply the chain rule to get:

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

PERPLEXITY (CONT.)

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

- Low perplexity value is better – higher probability of a test set, means lower perplexity
- Perplexity can also be thought of as the weighted average branching factor of a language
  - The number of possible words that can follow a word
  - Higher branching factor makes the language less predictable/there is more uncertainty around what the next word should be
  - Trigram models have more information/less uncertainty than bigram models (e.g., "bird is" has more information than just "is" when predicting the next word)
- Note that the perplexity of two language models is comparable only if they use identical vocabularies



By Cburnett - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1508398

	Unigram	Bigram	Trigram
Perplexity	962	170	109

WSJ Dataset

SAMPLING FROM A LANGUAGE MODEL

- To figure out if what your model has learned is reasonable, you would sample from it randomly and inspect the results
- More likely phrases will be sampled more frequently and generate sentences the model considers to be more likely

LANGUAGE MODEL CHALLENGES

- Some sequences of a words may be missing from a particular corpus, or may be very infrequent (sparsity)
  - This leads to reasonable sequences in the test set that our model assigns very low or 0 probability to
- Some words in our vocabulary may even be missing from the training set (unknown words or out of vocabulary (OOV) words)

UNKNOWN WORDS

- We can model unknown words by assigning them a special token, e.g., <UNK>
- The <UNK> token is treated the same way as any other token in terms of probability computation
- <UNK> is also used to mask out words in the dataset in order to limit the size of the vocabulary we want to use (to reduce computational requirements)
  - Trade-off between accuracy and compute requirements

SMOOTHING & DISCOUNTING

- How do we avoid assigning 0 probability to unseen strings/events in the test set?
  - Smoothing (or discounting) re-allocates some of the probability mass from more frequent events to unseen events
- Basic approach: Laplace Smoothing (also knows as add 1 smoothing)

For example, the smoothing is applied to each unigram (to each word in the vocabulary) to ensure each word has a count of at least 1:

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

The minimum count of unseen evens will be adjusted from 0 to 1

c_i - number occurrences of the word w_i, N - number of tokens/words,
V = extra observations (1 per vocabulary word since we have "+ 1" in the count for each vocabulary word)

IMPACT OF LAPLACE SMOOTHING

	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	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

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

	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

Figure 3.6 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

	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

Figure 3.7 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

- Laplace smoothing is too aggressive about moving probability to unseen cases, and is not typically used in practice any more

BASIC BACKOFF

- **Backoff:** Use lower order n-grams when you don't have enough examples to accurately compute probability of a particular value of n
  - E.g., use bigram counts to estimate trigram probability

INTERPOLATION

- Basic Interpolation: Combine all lower order n-gram probabilities into a weighted average, along with the n-gram for the n you are targeting
  - E.g., for the following trigram model, we use weighted factors $\lambda_1, \lambda_2, \lambda_3$ that sum up to 1

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

The λ parameters can be estimated based on a dev set.

- Context-Based Interpolation: Compute λ parameters for each specific n-gram

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

ABSOLUTE DISCOUNTING

- Subtracts a fixed (absolute) quantity (discount d) from each count
  - How is this discount d computed?
    - Empirically (i.e., through experiments)
    - d=0.75 has been found to work well for bigrams
- Adds discounted lower order n-grams (as in context-based interpolation)

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Fig 3.9. Experiment reported by Church and Gale in 1991.

Original Bigram Model: $$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_v C(w_{n-1}v)}$$

Bigram Model with Absolute Discounting:

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$$

KNESER-NEY SMOOTHING

- Uses absolute discounting
- Also uses the continuation novelty of the word – how likely is a word to appear in a new context
  - E.g., the word "glasses" is more likely to appear in a new context than the word "Kong"

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|}$$

→ Number of times w appears in bigrams

→ Total number of bigram types

Overall formulation for bigrams:

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i)$$

MODEL CHALLENGES FOR LARGE LANGUAGE MODELS

- Computationally infeasible to apply the detailed smoothing we discussed so far
- Instead, apply an algorithm called “Stupid Backoff” (S in the formula below)
  - Use counts for n-grams that are present in the dataset
  - For ones not present, go to a lower order n-gram (lower value of n) and use a discount weight of λ = 0.4
  - Note: S is not a probability (does not range from 0 - 1)

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

ENTROPY

- Entropy is a measure of information
- In information theory, entropy is the smallest number of bits required to encode a piece of information, or a message
- The entropy (H) of a random variable X which takes on a range of values χ is defined as follows:

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x)$$

- The range of values in χ is the set of words in the language for example, or the set of letters, etc.

ENTROPY IN LANGUAGE MODELING

- The entropy of all finite sequences of n words can be computed as follows:

$$H(w_1, w_2, \ldots, w_n) = -\sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

- The per word entropy is expressed as:

$$\frac{1}{n} H(w_{1:n}) = -\frac{1}{n} \sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

- The entropy of a language assumes infinite sequences are possible:

$$\begin{aligned} H(L) &= \lim_{n \to \infty} \frac{1}{n} H(w_1, w_2, \ldots, w_n) \\ &= -\lim_{n \to \infty} \frac{1}{n} \sum_{W \in L} p(w_1, \ldots, w_n) \log p(w_1, \ldots, w_n) \end{aligned}$$

- And with some assumptions can be simplified to:

$$H(L) = \lim_{n \to \infty} -\frac{1}{n} \log p(w_1 w_2 \ldots w_n)$$

CROSS-ENTROPY

- Used when we don't know the probability distribution p used to generate the data
- Instead we use probability distribution m which is a model (approximation) of p

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

- Using the same simplifying assumptions as in the previous slide, we get:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n)$$

Implication: We can estimate the cross-entropy of a model m on distribution p by computing entropy over a single long (enough) sequence instead of over all possible sequences.

CROSS-ENTROPY (CONT.)

- Moreover, for any model m, the following holds true: $H(p) \leq H(p,m)$
  - The lower the cross-entropy H(p,m), the closer it is to H(p) and therefore the more accurately m represents p
  - When we have 2 models, m_1 and m_2, the more accurate one will have a lower cross-entropy

- There is also a relationship between perplexity and entropy computed for a sequence of words W:

$$\text{Perplexity}(W) = 2^{H(W)}$$