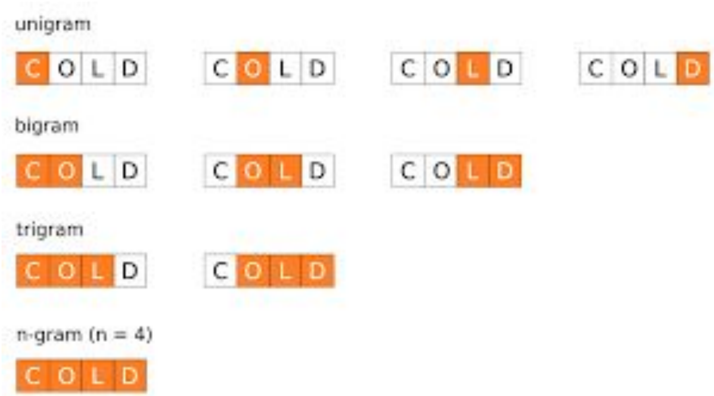
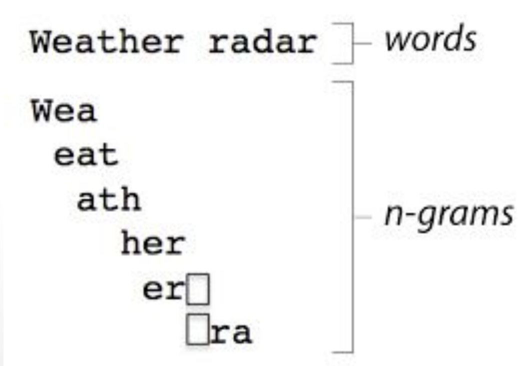
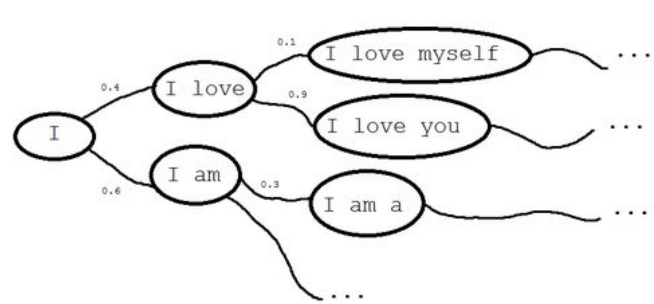
Topic 5 (Pt 1) :
Language Modeling







Language Models

- A language model is a statistical model that assigns probabilities over sequences of words
- Formal grammars (e.g, regular or context free language) give a hard “binary” model of valid sentences in natural language
- For NLP, a probabilistic model that gives a probability that a string is a member of a language is more useful
- Needs lots of data
- To specify the correct probability distribution, the probability of all sentences in a language must sum to 1

Uses of Language Models

- Speech recognition
  - “I ate a cherry “ is a more likely sentence than “Eye eight uh Jerry”
- OCR & hand-writing recognition
  - More probable sentences means more likely correct readings

Uses of Language Models (cont.)

- Machine translation
  - More likely sentences are probably better translations
- Spelling error detection
  - “Their are problems wit this sentence”
- Augmentative communication (for disabled),
- Text summarization, etc…

Completion Prediction

- A language model supports predicting the completion of a sentence
- Imagine listening to someone as they speak and trying to guess the next word that they are going to say…
  - I would like to make a …..
    - call?
    - drink?
    - visit?
    - guess?
  - Please turn off your …..
    - cell phone?
    - television?
    - brain cells?

N-gram Models

- WHAT?
  - *n*-gram Consecutive sequences of tokens
  - Language modeling assign probabilities to sequences of tokens
  - n-gram models can be seen as a probabilistic automata for generating sentences
- WHY?
  - Statistical machine translation, speech recognition, handwriting recognition, predictive text input
- HOW?
  - Based on previous token histories

Unigram

- *n* = 1 (*1*-gram)

This | is | a | Natural | Language | Processing | class

- This
- is
- a
- Natural
- Language
- Processing
- class

Bigram

- n = 2 (*2*-gram)

This is a Natural Language Processing class

- This is
- is a
- a Natural
- Natural Language
- Language Processing
- Processing class

Trigram

- *n* = 3 (*3*-gram)

This is a Natural Language Processing class

- This is a
- is a Natural
- a Natural Language
- Natural Language Processing
- Language Processing class

nth gram (n > 3)

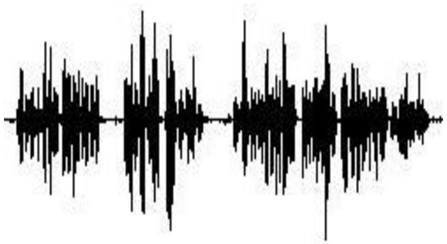
Contexts/histories
for the word “here”

Protein & DNA Sequencing

- **Protein**
  - Cys-Gly-Leu-Ser-Trp, ……,
    - Cys, Gly, Leu, Ser, Trp, ……, (Unigram)
    - Cys-Gly, Gly-Leu, Leu-Ser, Ser-Tr, ……, (Bigram)
    - Cys-Gly-Leu, Gly-Leu-Ser, Leu-Ser-Trp, ……, (Trigram)
- **DNA**
  - AGCTTCGA……,
    - A, G, C, T, T, C, G, A, ……, (Unigram)
    - AG, GC, CT, TT, TC, CG, GA, ……, (Bigram)
    - AGC, GCT, CTT, TTC, TCG, CGA, …..., (Trigram)

Why *n*-gram Language Modeling?

- If we have an English speech recognition system, which answer is better? (@https://talktyper.com/)



$\boxed{w_1 = \text{natural language processing}}$

w_2 = nature language processing

w_3 = natural english person

w_4 = natural eggs for sale

w_5 = watch remember school system

Why *n*-gram Language Modeling?

- or an English speech recognition system tested on Malay words? (@https://talktyper.com/)

w_1 = burglary

w_2 = library

$\boxed{w_3 = \text{berlari}}$

w_4 = the lolly

w_5 = benetti

Language models can give us an answer!

Word Prediction

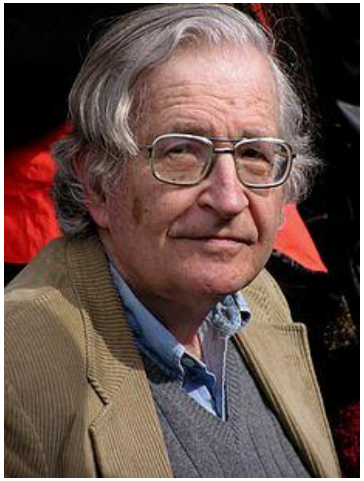
- Guessing the next word is an **essential subtask** of speech recognition, hand-writing recognition, machine translation, augmentative communication (for disabled), and spelling error detection.
- In such tasks, **word-identification is difficult** because the input is very noisy and ambiguous.
- Thus looking at previous words can give us an important cue about what the next words are going to be.

The History…

Statement from a linguist…

"But it must be recognized that the notion 'probability of a sentence' is an entirely useless one, under any known interpretation of this term"

~Noam Chomsky~



Statement from a computer scientist…

"Anytime a linguist leaves the group the recognition rate goes up"

~Fred Jelinek~

Counting Words in Corpora

- Probabilities are based on counting things. We need to decide what we are going to count and where we are going to find the things to count
- For counting probabilities, we count words in a training corpus
- Some things to consider…
  - Count types or token?
  - What kind of punctuations/stop words should be ignored?
  - Should lower case and upper case words be treated as the same word?

Probability Estimation

- We can use counts of words in a corpus (i.e., relative frequencies) to assign a probability distribution across words.
- Given a list of words in a sentence: w_1, w_2, ….., w_{n-1} , w_n, we can represent the probability of a sentence as :

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

Probability Estimation (cnt…)

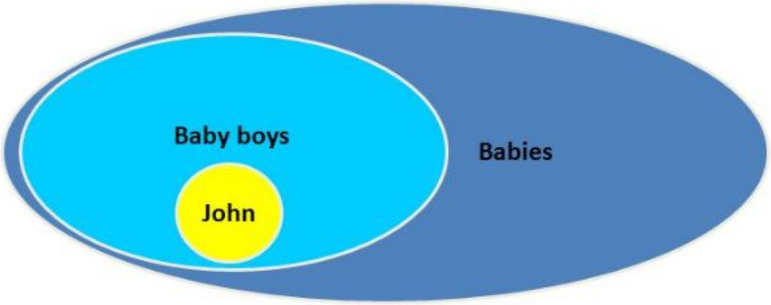
- This assumption that the probability of a word depends only on the previous word is known as the *Markov Chain*
- A Markov chain is a kind of *weighted finite-state* automaton
- Intuition : the next state of a weighted FSA is always dependent on a finite history (since the number of states in a finite
- The simple bigram model can be viewed as a simple kind of Markov chain which has one state for each word.

Sentence Probability

- P(X) means the probability that X is true.
  - P(baby is a boy) ≈ 0.5 (% of total that are boys)
  - P(baby is named John) ≈ 0.01 (% of total that are named John)

Bayes Rule

- Bayes rule: P(X|Y) = P(Y|X) × P(X) / P(Y)
- P(named John | boy) = P(boy | named John) × P(named John) / P(boy)



Basic Probability

- Assume ***a*** and ***b*** are events:
  - $P(not\ \boldsymbol{a}) = 1 - P(\boldsymbol{a})$
  - $P(\boldsymbol{a}\ or\ \boldsymbol{b}) = P(\boldsymbol{a}) + P(\boldsymbol{b}) - P(\boldsymbol{a}\ and\ \boldsymbol{b})$
  - $P(\boldsymbol{a}\ and\ \boldsymbol{b}) = P(\boldsymbol{a}, \boldsymbol{b})$
  - $P(\boldsymbol{a}, \boldsymbol{b}) = 0$
    - ***a*** and ***b*** are **mutually exclusive** (i.e., cannot occur at the same time)

Basic Probability (cnt…)

- **Conditional Probability:**
  - **P(a|b)**
  - The probability of ***a*** given that we know ***b***

- **Joint Probability:**
  - **P(a, b)**
  - The probability of ***a*** and ***b*** occurring together

The Bayes Theorem

-

$$P(a,b) = P(b,a)$$

$$P(a,b) = P(a|b)\,P(b)$$

$$P(a|b)\,P(b) = P(b|a)\,P(a)$$

$$or$$

$$P(a|b) = \frac{P(b|a)\,P(a)}{\cancel{P(b)}}$$

The Bayes Theorem (cnt…)

- Two events **a** and **b** are independent if :

$$P(a|b) = P(a) \quad OR$$

$$P(b|a) = P(b) \; AND$$

$$\left.\begin{aligned} P(a,b) &= P(a|b)\,P(a) \\ &= P(a)\,P(b) \end{aligned}\right\} **$$

- **The last equation is known as the definition of independence

Probabilistic Language Models

- Language models assign a probability to each word/sentence

w_1 = natural language processing $p(w_1) = 2.125 * 10^{-2}$

w_2 = nature language processing $p(w_2) = 4.513 * 10^{-4}$

w_3 = natural english person $p(w_3) = 3.348 * 10^{-7}$

w_4 = natural eggs for sale $p(w_4) = 7.562 * 10^{-15}$

w_5 = watch remember school system $p(w_5) = 5.612 * 10^{-24}$

- We expect to get $p(w_1) > p(w_2) > p(w_3) > p(w_4)$

Building N-gram Language Models

- Use existing sentences to compute ***n*-gram** probability estimates (training)
- Terminologies:
  - N = total number of words in training data (tokens)
  - V = vocabulary size or number of unique words (types)
  - $C(w_1,...,w_k)$ = frequency of n-gram $w_1, ..., w_k$ in training data
  - $P(w_1,...,w_k)$ = probability estimate for *n*-gram $w_1 ... w_k$
  - $P(w_1|w_1 ... w_{k-1})$ = conditional probability of producing w_k given the history $w_1 ... w_{k-1}$

Unigram Language Model

- Do not use histories
- A ***zeroth-order*** Markov model (simplest Markov)
- We compute the probability of a unigram model for a sentence/text using the following equation

$$P(w_i|w_1 \ldots w_{i-1}) \approx P(w_i) = \frac{c(w_i)}{\sum_{\tilde{w}} c(\tilde{w})}$$

- $c(w_i)$ is the counts of word w_i seen in training data
- $\sum_w c(\tilde{w})$ is the counts of all words(tokens) in the training data

Unigram Model Example

- <s> i live in cheras. </s>
- <s> i am an undergraduate student . </s>
- <s> my campus is in gombak . </s>

***Note: <s> and </s> is the beginning and end markers**

P(<s>) = 3/20 = 0.15 P(in) = 2/20 = 0.1
P(i) = 2/20 = 0.1 P(gombak) = 1/20 = 0.05
P(</s>) = 3/20 = 0.15
P(live) = 1/20 = 0.05

P(S = <s> i live in gombak. </s>) = 0.15 * 0.1 * 0.05 * 0.1 * 0.05 * 0.15 = **5.625 * 10^{-7}**

Bigram Language Model

- Looks one word into the past
- A ***first-order*** Markov model
- We compute the probability of a bigram model for a sentence/text using the following equation :

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

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

- $C(w_{n-1}w_n)$ is the counts of two word sequences seen in training data
- $C(w_{n-1})$ is the counts of previous word in training data

Bigram Model Example

- <s> i live in cheras. </s>
- <s> i am an undergraduate student . </s>
- <s> my campus is in gombak . </s>

P(i | <s>) = 2/3 = 0.666

P(live | i) = 1/2 = 0.5

P(in | live) = 1/1 = 1

P(gombak | in) = 1/2 = 0.5

P(</s> | gombak) = 1/1 = 1

P(am | i) = 1/2 = 0.5

P(an | am) = 1/1 = 1

P(S = <s> i live in gombak. </s>) = P(i | <s>) * P(live | i) * P(in | live) * P(gombak | in) * P(</s> | gombak)

= 0.666 * 0.5 * 1 * 0.5 * 1 = **0.1665**

Trigram Language Model

- Looks two words into the past
- A ***second-order*** Markov model
- We compute the probability of a trigram model for a sentence/text using the equation:

$$P(w_n | w_{n-1}) = \frac{C(w_{n-1} w_{n-2} w_n)}{C(wn_{-2} w_{n-1})}$$

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1} w_{k-2})$$

- $C(w_{n-2}w_{n-1}w_n)$ is the counts of three word sequences seen in training data
- $C(w_{n-2}w_{n-1})$ is the counts of previous two words in training data

Trigram Model Exercise

- <s> i live in cheras. </s>
- <s> i am an undergraduate student . </s>
- <s> my campus is in gombak . </s>

P(live|<s> i) = ?

P(in|i live) = ?

P(gombak|live in) = ?

p(</s>|in gombak) = ?

P(S = <s> i live in gombak. </s>) = ?

Python n-gram (Due 1/4)

- Write a Python program (modify the file ngram_test.py) that calculates the probability of **the following sentence** using a bigram model (ignore the start <s> and end <s/> marker for now) :

  "KICT also aspires to enhance the quality of learning and teaching"
- Use the training data set in the file "text-gram.dat" provided in italeem.
- Steps:
  1. Create a python dictionary to find and store the counts of all bigrams of the sentences in the training file.
  2. Create a python dictionary to find and store the counts of all unigrams in the training file.
  3. Create bigrams and unigrams for the test sentence above
  4. Calculate probability of each bigram in test sentence using the formula :

     $$\frac{\textbf{no. of times the bigram in test is found in training}}{\textbf{no. of times the unigram in test is found in training}}$$

     Example: $\frac{C(\text{to enhance})}{C(\text{to})} = 1/6 = 0.1667$
  5. Read Topic 5 Pt 2 on Laplace Smoothing (Slides 9-11) and apply smoothing on bigrams or unigrams with zero counts (not found at all in training data set).
  6. Calculate the probability of the whole sentence by multiplying all prob. of bigrams together.
  7. **Submit** your solution though italeem (individual) **by 11.20 am next Tuesday (14 March)**.