19ECS775 NATURAL LANGUAGE PROCESSING

M.Tech. II Semester (CSE, DS)

Module-II

Dr. K. Srinivasa Rao

Associate Professor

Department of CSE

GITAM School of CSE

Visakhapatnam – 530045

Email: **skonda@gitam.edu**

Mobile: 98486 73655

TEXT BOOKS

TEXT BOOKS:

1. Daniel Jurafsky, James H Martin, “Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition”, 2/e, Prentice Hall, 2008.

2. C. Manning, H. Schutze, “Foundations of Statistical Natural Language Processing”, MIT Press. Cambridge, MA, 1999.

3. Jacob Eisenstein, Introduction to Natural Language Processing, MIT Press, 2019.

4. EBook: Le Deng, Yang Liu, Deep Learning in Natural Language Processing, Springer, 2018.

5. Jalaj Thanaki, Python Natural Language Processing: Explore NLP with machine Learning and deep learning Techniques, Packt, 2017.

1. N-gram Language Models

- We require *models that assign a probability to each possible next word*.
- The same models will also serve to assign a probability to an entire sentence.

- Such a model, for example, could predict that the following sequence has a much higher probability of appearing in a text:

all of a sudden I notice three guys standing on the sidewalk

than does this same set of words in a different order:

on guys all I of notice sidewalk three a sudden standing the

Why would you want to predict upcoming words, or assign probabilities to sentences?

(1) Speech Recognition: Probabilities are essential to identify words in noisy, ambiguous input

- For a speech recognizer, consider:

I will be back soonish.

I will be bassoon dish.

back soonish is a much more probable sequence than bassoon dish.

Continued...

(2) For writing tools like spelling correction or grammatical error correction:

Their are two midterms. (There was mistyped as Their), or
Everything has improve. (improve should have been improved).

- There are (is more probable than Their are), and
- has improved (is more probable than has improve), allowing us to help users by detecting and correcting these errors.

❖ **Language models (LMs):** Models that assign probabilities to sequences of words.

❖ **N-gram model:** The simplest language model n-gram is a sequence of n words:

a 2-gram (called bigram) is a two-word sequence of words like
"please turn", "turn your", or "your homework", and
a 3-gram (a trigram) is a three-word sequence of words like
"please turn your", or "turn your homework".

Continued...

(1) N-grams:

- P(w|h) -- the probability of a word **w** given some history **h**
- Suppose **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). (3.1)

- One way to estimate this probability is from *relative frequency counts*:

  take a very large corpus, count the number of times we see *its water is so transparent that*,

  and count the number of times this is followed by *the*.

➢ This would be answering the question "Out of the times we saw the history **h**, how many times was it followed by the word **w**", as follows:

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)} \quad (3.2)$$

Continued...

- With a large enough corpus, such as the web, we can compute these **counts** and estimate the probability from Eq. 3.2.
- This method works fine in many cases, but *in most cases, we may not get good estimates*.
- This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences.

- "Walden Pond's water is so transparent that the".

  (a simple extension of the above sentence may have counts of zero).

- To know the joint probability of "*its water is so transparent*",

  we can ask "out of all possible sequences of five words, how many of them are *its water is so transparent*?"

- It is the count of *its water is so transparent* and divide by

  the sum of the counts of all possible five word sequences

Continued...

Represent

$P(X_i$ = "the") as -- P(the)

a sequence of N words -- either as $w_1 \ldots w_n$ or $w_{1:n}$

- For the joint probability of each word in a sequence having a particular value

  $P(X=w_1, Y=w_2, Z=w_3, \ldots\ldots, W=w_n)$ we will use $P(w_1, w_2, \ldots., w_n)$.

To compute $P(w_1, w_2, \ldots., w_n)$:

We decompose this probability using the **chain rule of probability**:

$$P(X_1 \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1})$$
$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \quad (3.3)$$

Applying the chain rule to words, we get

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \quad (3.4)$$

Continued...

- 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, $P(w_n \mid w_1^{n-1})$.

- We *can't just estimate* by counting the number of times every word occurs *following every long string*, because language is creative and any particular context might have never occurred before!

n-gram model:

The idea is that instead of computing the probability of a word given its entire history, we can *approximate the history by just the last few words*.

- The **bigram model**, for example, approximates the probability of a word given all the previous words $P(w_n \mid w_{1:n-1})$ by using only the conditional probability of the preceding word $P(w_n \mid w_{n-1})$.

Continued...

- In other words, instead of computing the probability

  P(the | Walden Pond's water is so transparent that) (3.5)

  we approximate it with the probability

  P(the | that) (3.6)

- When we use a bigram model to predict the conditional probability of the next word,

  we are thus making the following approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1}) \quad (3.7)$$

Markov assumption:

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

- ❑ Markov models are the class of probabilistic models that assume :

  "we can predict the probability of some future unit *without looking too far into the past*".

Continued...

- We can generalize the bigram (which looks one word into the past)
  to the trigram (which looks two words into the past) and thus
  to the n-gram (which looks n-1 words into the past).

- Thus, the *general equation for this n-gram approximation* to the conditional probability of the next word in a sequence is:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}) \qquad (3.8)$$

Ex: n= 10, and we use a trigram (N=3). RHS becomes $P(W_{10} \mid w_{10-3+1:10-1})$
i.e., $P(W_{10} \mid w_{8:9})$

- Given the bigram assumption for the probability of an individual word,
  we can compute the probability of a complete word sequence by substituting Eq. 3.7 into Eq. 3.4:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \qquad (3.9)$$

Continued...

Maximum Likelihood Estimation (MLE):

- It is a *method to estimate the probabilities of bigram or n-gram*.
- This is done by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.

Ex: To compute a particular *bigram probability of a word* **y** given a previous word **x**,
we'll compute the *count of the bigram* **C(xy)** and normalize by the *sum of all the bigrams*
that share the same first word **x**:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \qquad (3.10)$$

- Since the sum of all bigram counts that start with a given word $\mathbf{w_{n-1}}$ must be equal to the *unigram count for that word* $\boldsymbol{w_{n-1}}$, equation 3.10 becomes:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \qquad (3.11)$$

Continued...

The denominator in eqn. 3.10 is the same as the denominator in eqn. 3.11.

From equation 3.10:

bigram: w_{n-1} may be followed by w (any word)

Ex: bigrams:

that the	5	
that of	6	
that how	3	Total = 20
that where	6	

unigram: only one word (w_{n-1})

unigrams:

that	5	
that	6	
that	3	Total =20
that	6	

Hence, the denominators are the same.

Continued...

Ex: A mini-corpus of three sentences

- Augment each sentence with a special symbol <s> at the beginning of the sentence
  (to give us the *bigram context of the first word*).

- Use a special end-symbol. </s>

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

➢ Calculations for some of the bigram probabilities from this corpus:

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

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

Continued...

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

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\ w_n)}{C(w_{n-N+1:n-1})} \qquad (3.12)$$

- Equation 3.12 (like Eq. 3.11) estimates the n-gram probability
  by dividing the *observed frequency of a particular sequence* by the *observed frequency of a prefix*.
- This ratio is called a ***relative frequency.***

An Example Corpus: **Berkeley Restaurant Project**

-- a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California.

Some text-normalized sample user queries:

can you tell me about any good cantonese restaurants close by

mid priced thai food is what i'm looking for

tell me about chez panisse

can you give me a listing of the kinds of food that are available

i'm looking for a good place to eat breakfast

when is caffe venezia open during the day

Continued...

- Figure 3.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project.
- Note that the *majority of the values are zero*.
- In fact, we have chosen the sample words to cohere with each other;

  a matrix selected from a random set of seven words would be even more sparse.

	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.

Continued...

- Figure 3.2 shows the bigram probabilities after normalization

  (dividing each cell in Fig. 3.1 by the appropriate unigram for its row, taken from the following set of unigram probabilities):

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

	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.

Continued...

Here are a few other useful probabilities:

- P(I|<s>) = 0.25 P(English|want) = 0.0011
- P(food|English) = 0.5 P(</s>|food) = 0.68

Ex: Computing the probability of "I want English food"

Simply multiply the appropriate bigram probabilities together as follows:

P(<s> I want English food </s>)

= P(I|<s>) P(want|I) P(English|want) P(food|English) P(</s>|food)

= 0.25 x 0.33 x 0.0011 x 0.5 x 0.68

= 0.000031

Exercise: Compute the probability of "I want chinese food".

(2) Smoothing:

- Smoothing in NLP is a technique used in probabilistic language models to address the zero-probability problem for unseen or rare N-grams in training data.
- By redistributing probability mass from frequent events to unseen ones, it ensures models (e.g., in speech recognition, machine translation) can handle new data, enhance robustness and reduce overfitting.

2.1 Laplace Smoothing (or Add-one Smoothing):

- The simplest way to do smoothing is *to add one to all the bigram counts*, before we normalize them into probabilities.
- All the *counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on*.
- This algorithm is called **Laplace Smoothing**.

Application to unigram probabilities:

- The *unsmoothed maximum likelihood estimate* of the unigram probability of the word $\mathbf{w_i}$

  is its count $\mathbf{c_i}$ normalized by the total number of word tokens N:

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

Continued...

- Laplace smoothing merely adds one to each count (hence its alternate name add-one smoothing).

- Since there are V words in the vocabulary and each one was incremented,
  we also need to adjust the denominator to take into account the extra V observations.

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V} \qquad (3.20)$$

Example: If "I study" appears 0 times in training, but “I” appears 10 times and vocabulary size (V) is 100, the smoothed probability becomes (0+1) / (10+100) rather than 0/10.

- Instead of changing both the numerator and denominator, it is convenient to describe
  how a smoothing algorithm affects the numerator, by defining an adjusted count **c***.

- This adjusted count is easier to compare directly with the MLE counts and can be turned into a probability like an MLE count by normalizing by N.

Continued...

- *To define this count*, since we are only changing the numerator in addition to adding 1,
  we'll also need to multiply by a normalization factor N / (N+V) :

$$c_i^* = (c_i + 1)\frac{N}{N+V} \qquad (3.21)$$

Discounting (lowering):

- *A related way to view smoothing* is as
  discounting (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts.
- Thus, instead of referring to the discounted counts **c**,
  we might describe a smoothing algorithm in terms of a relative discount $\mathbf{d_c}$,
  the ratio of the discounted counts to the original counts:

$$d_c = \frac{c^*}{c}$$

Continued...

Smoothing Berkeley Restaurant Project bigrams:

Figure 3.5 shows the add-one smoothed counts for the bigrams in Fig. 3.1.

	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 3.5 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.

Continued...

- The normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \qquad (3.22)$$

- *For add-one smoothed bigram counts*, we need to augment the unigram count by the number of total word types in the vocabulary V:

$$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \qquad (3.23)$$

Continued...

- Thus, each of the unigram counts given in the previous section will need to be augmented by V =1446.

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

- i.e., i=2533+1446=3979 want=927+1446=2373 to=2417+1446=3863 and so on.

- The first three rows of Fig. 3.5:

	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

- Row-1 values: 6/3979 = 0.0015 828/3979=0.21 1/3979= 0.00025 etc.
- Row-2 values: 3/2373=0.0013 1/2373=0.00042 609/2373=0.26 etc.
- Row-3 values: 3/3863=0.00078 1/3863=0.00026 5/3863=0.0013 etc.

  and so on.

Continued...

- Figure 3.6 shows the add-one smoothed probabilities for the bigrams in Fig. 3.2

  (smoothed bigram 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

Figure 3.6 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.

Continued...

- It is often convenient to reconstruct the count matrix so we can see how much a smoothing algorithm has changed the original counts.
- These adjusted counts can be computed by Eq. 3.24.
- Figure 3.7 shows the reconstructed counts.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V} \quad (3.24)$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

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

Continued...

Add-one smoothing has made a very big change to the counts.

C(want to) changed from 609 to 238.

P(to | want) decreases from 0.66 in the unsmoothed case to 0.26 in the smoothed case.

- The discount **d** (the ratio between new and old counts) shows how strikingly the counts for each prefix word have been reduced;

  -- the discount for the bigram want to is 0.39, while

  -- the discount for Chinese food is 0.10, a factor of 10!

- The sharp change in counts and probabilities occurs because *too much probability mass is moved to all the zeros*.

Continued...

3.4.2 Add-k Smoothing:

- This is *an alternative to add-one smoothing*.
- It *moves 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* (0.5? 0.05? 0.01?).
- This algorithm is therefore called *add-k smoothing*.

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV} \qquad (3.25)$$

- Add-k smoothing requires that we have a method for choosing k.
- Add-k is useful for some tasks (including text classification).
- It still doesn't work well for language modeling, generating counts with poor variances and often inappropriate discounts.

Continued...

3.4.3 Backoff and Interpolation:

- The *discounting* discussed earlier *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 $P(w_n \mid w_{n-2}\, w_{n-1})$ but we have no examples of a particular *trigram* $w_{n-2}\, w_{n-1} w_n$, we can instead estimate its probability by using the *bigram* probability $P(w_n \mid w_{n-1})$.
- Similarly, if we don't have counts to compute $P(w_n \mid w_{n-1})$, we can look to the *unigram* $P(w_n)$.
- Sometimes using less context is a good thing, helping to generalize more for contexts that the model hasn't learned much about.

Continued...

- There are two ways to use this n-gram "hierarchy".

  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 n-gram.)

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

- In simple linear interpolation, *we combine different order n-grams by linearly interpolating all the models*.
- Thus, we estimate the trigram probability $P(w_n \mid w_{n-2}\, w_{n-1})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ :

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n)\end{aligned} \tag{3.26}$$

such that the λs sum to 1:

$$\sum_i \lambda_i = 1 \tag{3.27}$$

Continued...

3. Katz Backoff:

- In order for a backoff model to give a correct probability distribution, *we have to discount the higher-order n-grams* to save some probability mass for the lower order n-grams.

- This kind of backoff with discounting is also called **Katz backoff**.

- In Katz backoff, we rely on a discounted probability P* if we've seen this n-gram before (i.e., if we have non-zero counts).

- Otherwise, we recursively back off to the Katz probability for the shorter-history (N-1)-gram.

- In addition to this explicit discount factor, we'll need a function **α** to distribute this probability mass to the lower order n-grams.

Continued...

The probability for a backoff n-gram $\mathbf{P_{BO}}$ is:

$$P_{\mathrm{BO}}(w_n|w_{n-N+1:n-1}) = \begin{cases} P^*(w_n|w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2:n-1}), & \text{otherwise.} \end{cases} \quad (3.29)$$

4. Good-Turing Backoff (Discounting): Katz backoff is often combined with a smoothing method called Good-Turing.

- Good-Turing smoothing is a technique in NLP that reallocates probability mass from frequent n-grams to unseen or rare ones, effectively handling data sparsity.

- It calculates adjusted counts (c*) for n-grams based on the frequency of items that appear r+1 times, typically setting the count of unknown items to the number of items seen once (N_1). This method improves model accuracy and reduces overfitting by assigning non-zero probabilities to unseen events.

Adjusted Count Formula:

$c^* = (c + 1)\, N_{c+1} / N_c$

c is the original frequency and N_c is the count of items that appear c times

Continued...

5. Kneser-Ney Smoothing:

- The equation for interpolated absolute discounting applied to bigrams:

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

- The first term is the discounted bigram, and the second term is the unigram with an interpolation weight l. We could just set all the d values to .75, or we could keep a separate discount value of 0.5 for the bigrams with counts of 1.

2. Vector Semantics and Embeddings

(1) Lexical Semantics

- Lexical semantics is the branch of linguistics that studies the meaning of words, phrases, and lexical units within a language.
- It focuses on how individual words convey meaning, how those meanings relate to one another, and how they contribute to the overall meaning of sentences and discourse.

(Lexical unit: A fundamental building block of language, representing a single unit of meaning, which can be a single word (like "cat"), a phrase (like "traffic light"), or even an idiom (like "kick the bucket").)

1.1 Lemmas and Senses:

Meaning of the word ‘mouse’ from a dictionary:

mouse (N)

1. any of numerous small rodents...

2. a hand-operated device that controls a cursor...

Continued…

- Here mouse is the **lemma** (also called the citation form.).
- The form mouse would also be the lemma for the word mice.
  (dictionaries don't have separate definitions for inflected forms like mice)
- Similarly sing is the lemma for sing, sang, sung.
- The specific forms sung or carpets or sing are called wordforms.
- Each lemma can have multiple meanings; the lemma mouse can refer to the rodent or the cursor control device.
- We call each of these aspects of the meaning of mouse a word sense.
- Lemmas can be polysemous (have multiple senses).
  Hence, they can make interpretation difficult
  (is someone who types "mouse info" into a search engine looking for a pet or a tool?).

1.2 Synonymy:

- When one word has a sense whose meaning is identical to a sense of another word, or nearly identical,
  we say the two senses of those two words are synonyms.

Examples of Synonyms: couch/sofa car/automobile

Continued…

- Propositional meaning -- two words are synonymous if they are substitutable for one another in any sentence without changing the truth conditions of the sentence
- Principle of contrast -- no two words are absolutely identical in meaning

  H_2O (scientific context), Water (general context)

1.3 Word Similarity:

- Most words do have lots of similar words.

  Cat is not a synonym of dog, but cats and dogs are certainly similar words.
- Moving from synonymy to similarity means --

  talking about relations between word senses (like synonymy) to relations between words (like similarity).

- The notion of word similarity is very useful in larger semantic tasks.
- Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of natural language understanding tasks like question answering, paraphrasing, and summarization.

Continued…

From the SimLex-999 data set:

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

1.4 Word Relatedness:

- The meaning of two words can be related in ways other than relatedness similarity.

Ex: Coffee and Cup.

Coffee is not similar to cup; they share practically no features

(coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape).

But coffee and cup are clearly related.

They are associated by co-participating in an everyday event (the event of drinking coffee out of a cup).

Continued…

1.5 Semantic Field:

- A semantic field is a set of words which cover a particular semantic domain and bear structured relations with each other.

  Ex: The semantic field of :

  hospitals (surgeon, scalpel, nurse, anesthetic, hospital),

  restaurants (waiter, menu, plate, food, chef), or houses (door, roof, kitchen, family, bed).

1.6 Semantic Frames and Roles:

- A semantic frame is a set of words that denote perspectives or participants in a particular type of event.

  Ex: A commercial transaction is a kind of event in which one entity trades money to another entity in return for some good or service, after which the good changes hands or perhaps the service is performed.

- This event uses verbs like buy (the event from the perspective of the buyer), sell (from the perspective of the seller), pay (focusing on the monetary aspect), or nouns like buyer.

- Frames have semantic roles (like buyer, seller, goods, money), and words in a sentence can take on these roles.

Continued…

- Knowing that buy and sell have this relation makes it possible for a system to know that a sentence like

  'Sam bought the book from Ling' could be paraphrased as

  'Ling sold the book to Sam', and that

  Sam has the role of the buyer in the frame and Ling the seller.
- Being able to recognize such paraphrases is important for question answering, and can help in shifting perspective for machine translation.

1.7 Connotation:

- It means the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations.

  Ex: Some words have positive connotations (happy) while others have negative connotations (sad).

  Even words whose meanings are similar in other ways can vary in connotation.

  Consider the difference in connotations between fake, knockoff, forgery, on the one hand,

  and copy, replica, reproduction on the other, or

  innocent (positive connotation) and naive (negative connotation).

(2) Vector Semantics

- Vectors semantics is the standard way to represent word meaning in NLP.
- The idea was that two words that occur in very similar distributions (whose neighbouring words are similar) have similar meanings.

Ex: Suppose, you didn't know the meaning of the word **ongchoi** but you see it in the following contexts:

1. Ongchoi is delicious sauteed with garlic.
2. Ongchoi is superb over rice.
3. ....ongchoi leaves with salty sauces...

And suppose that you had seen many of these context words in other contexts:

4. ...spinach sauteed with garlic over rice...
5. ...chard stems and leaves are delicious...
6. ...collard greens and other salty leafy greens

Continued…

- The fact that ongchoi occurs with words like -- rice and garlic and delicious and salty,
  as do words like -- spinach, chard, and collard greens might suggest that
  ongchoi is a leafy green similar to these other leafy greens.

- We can do the same thing computationally by just counting words in the context of ongchoi.

Vector Semantics:

It represents a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbours.

Embeddings:

Vectors for representing words are called embeddings.

- The word “embedding” derives from its mathematical sense as a mapping from one space or structure to another.

Continued…

- Fig. 6.1 shows a visualization of embeddings learned for sentiment analysis, showing the location of selected words projected down from 60-dimensional space into a 2-D space.

- Notice the distinct regions containing positive words, negative words, and neutral function words.

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

3. Words and Vectors

- Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur.
- There are two popular matrices:

  the term-document matrix and the term-term matrix.

3.1 Vectors and Documents:

Term-document matrix:

In this, each row represents a word in the vocabulary and each column represents a document from some collection of documents.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Continued…

- The term-document matrix of Fig. 6.2 was first defined as part of the vector space model of information retrieval.
- In this model, a document is represented as a count vector, a column in Fig. 6.3.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Vector -- a list or array of numbers.

As You Like It -- is represented as the list [1,114,36,20] (the first column vector in Fig. 6.3) and

Julius Caesar -- is represented as the list [7,62,1,2] (the third vector space column vector).

Vector space -- is a collection of vectors, characterized by its dimension.

In the example in Fig. 6.3, the document vectors are of dimension 4.

In real term-document matrices, the vectors representing each document would have dimensionality |V|, the vocabulary size.

Continued…

- The ordering of the numbers in a vector space indicates different meaningful dimensions on which documents vary.
- Thus, the first dimension for both these vectors corresponds to the number of times the word **battle** occurs.

- We can compare each dimension, noting for example that the vectors for As You Like It and Twelfth Night have similar values (1 and 0, respectively) for the first dimension.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Continued…

- We can think of the vector for a document as a point in |V|-dimensional space.
- Thus, the documents in Fig. 6.3 are points in 4-dimensional space.
- 4-dimensional spaces are hard to visualize. So, Fig. 6.4 shows a visualization in two dimensions.
- We have arbitrarily chosen the dimensions corresponding to the words battle and fool.

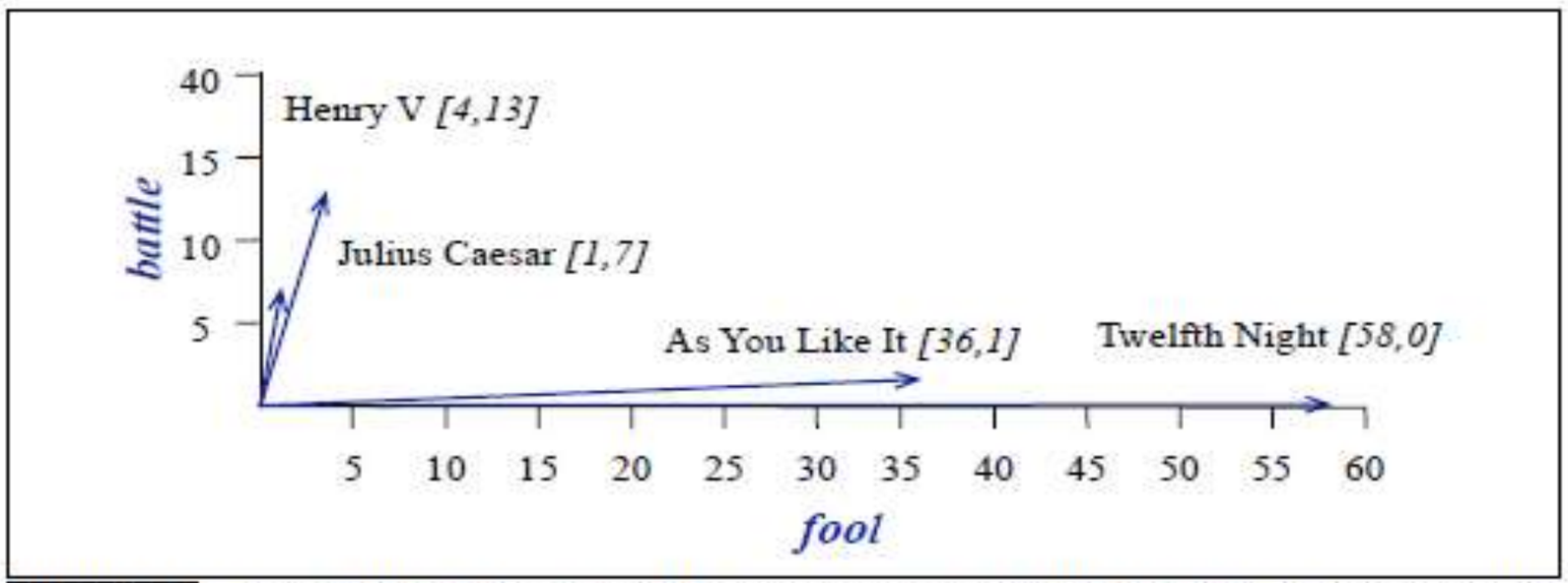

Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

3.2 Words as Vectors: Document Dimensions

- Documents can be represented as vectors in a vector space.

- *Vector semantics can also be used to represent the meaning of words.*

- We do this by associating each word with a word vector— a row vector rather than a column vector, hence with different dimensions (Fig. 6.5).

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

Continued…

- The four dimensions of the vector for fool, [36,58,1,4], correspond to the four Shakespeare plays.

- Word counts in the same four dimensions are used to form the vectors for the other 3 words:
  wit, [20,15,2,3]; battle, [1,0,7,13]; and good [114,80,62,89].

- *For documents*,
  similar documents have similar vectors, because similar documents tend to have similar words.

- *This same principle applies to words*:
  similar words have similar vectors because they tend to occur in similar documents.

- The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

3.3 TF-IDF: Weighing terms in the vector

- TF-IDF (Term Frequency-Inverse Document Frequency) is used to evaluate a word's importance in a document within a collection, helping to identify keywords and rank relevance in information retrieval, search engines, and text analysis by highlighting words frequent in a specific text but rare overall.

- It's crucial for converting text into numerical vectors for machine learning models, powering tasks like text summarization, topic modeling, recommendation systems, and customer sentiment analysis.

Key Uses:

(1) Information Retrieval & Search:

Scores documents based on query relevance, helping search engines rank results (e.g., Google).

(2) Text Mining & Analysis:

Extracts significant keywords, identifies trending topics, and analyzes customer feedback for insights.

Continued…

(3) Text Classification:

Helps machine learning models classify documents by giving more weight to distinguishing terms.

(4) Recommendation Systems:

Surfaces relevant content in platforms like digital libraries and e-commerce by understanding document themes.

(5) Machine Learning Feature Engineering:

Converts text into numerical features (vectors) that algorithms can process, improving model performance.

How it works (Simplified):

- **Term Frequency (TF):** Measures how often a word appears in one document.
- **Inverse Document Frequency (IDF):** Measures how rare a word is across all documents; common words get lower scores.
- **TF-IDF Score:** Multiplies TF and IDF; words frequent in a document but rare in the corpus get high scores, indicating high relevance.

Continued…

TF-IDF: Term Frequency-Inverse Document Frequency

- The co-occurrence matrices (above) represent each cell by frequencies (words with documents (Fig. 6.5)).

- But raw frequency is not the best measure of association between words.
- Raw frequency is much skewed (unbalanced) and not very discriminative.

- Words that occur nearby frequently are more important than words that only appear once or twice.
  Yet words that are too frequent like the or good— are unimportant.

- Two solutions to balance these two conflicting constraints:

    1. tf-idf algorithm (usually used when the dimensions are documents)
    2. PPMI algorithm (usually used when the dimensions are words)

Continued…

- The tf-idf algorithm is the product of two terms.

➢ The first is the term frequency: the frequency of the word **t** in the document **d**.

- We can just use the raw count as the term frequency:

$$tf_{t,d} = count(t, d) \qquad (6.11)$$

- We squash (flatten or compress) the raw frequency a bit, by using the $\mathbf{log_{10}}$ of the frequency instead.

- That is, a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document.

- Because we can't take the log of 0, we normally add 1 to the count:

$$tf_{t,d} = \log_{10}(count(t, d)+1) \qquad (6.12)$$

Continued…

- Then term occurrence in a document:

  0 times -- $tf = \log_{10}(1) = 0$

  10 times -- $tf = \log_{10}(11) = 1.4$

  100 times -- $tf = \log_{10}(101) = 2.004$

  1000 times -- $tf = 3.00044$, and so on.

- The second factor in tf-idf is document frequency df_t :

  df_t of a term **t** is the number of documents it occurs in.

- Idea is to give a higher weight to words that occur only in a few documents.

- Terms that are *limited to a few documents are useful* for discriminating those documents from the rest of the collection; terms that *occur frequently across the entire collection aren't as helpful*.

- Document frequency is *not the same as the collection frequency* of a term.

  (collection frequency: total number of times the word appears in the whole collection in any document)

Continued…

- Consider the two words Romeo and action in the collection of Shakespeare's 37 plays.

- They have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies (since Romeo only occurs in a single play).

	Collection frequency	Document frequency
Romeo	113	1
action	113	31

- If our goal is to find documents about the romantic tribulations of Romeo, the *word Romeo should be highly weighted*, but not action.

- We emphasize discriminative words like Romeo via the inverse document frequency (or idf term weight).

  idf = $\mathbf{N/df_t}$

  where **N** is the total number of documents in the collection, and

  $\mathbf{df_t}$ is the number of documents in which term **t** occurs.

Continued…

- The lowest weight of 1 is assigned to terms that occur in all the documents.

- It's usually clear what counts as a document:

  -- in Shakespeare we would use a play;
  -- when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page;
  -- in processing newspaper articles, the document is a single article.

- Because of the large number of documents in many collections, this measure too is usually squashed with a log function.

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right) \qquad (6.13)$$

Continued…

From Shakespeare corpus:

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

The words 'good' and 'sweet' occur in all the 37 documents.
Hence the respective **idf** values are zero.

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

- The tf-idf weighted value $\mathbf{w_{t,d}}$ for word **t** in document **d** thus combines term frequency $\mathbf{tf_{t,d}}$ (from 6.11 or 6.12) with **idf** from 6.13:

$$w_{t,d} = tf_{t,d} \times idf_t \qquad (6.14)$$

Continued…

- In Fig. 6.9, Eq. 6.12 is used.
- The tf-idf values for the dimension corresponding to the word **good** have now all become **0**;
  since *this word appears in every document*, the tf-idf algorithm leads it to be ignored.

- Similarly, the word **fool**, which appears in 36 out of the 37 plays, *has a much lower weight*.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20+1) = 1.322$ and idf = .037. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

Continued…

References for Creating TF-IDF model for a Sample Text (a Collection of Documents):

1. https://www.askpython.com/python/examples/tf-idf-model-from-scratch

2. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

3. https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/

***

4. Word2vec

Embeddings:

These are short dense vectors with number of dimensions **d** ranging from 50-1000, rather than the much larger vocabulary size |V| or number of documents D.

Skip-gram with Negative Sampling (SGNS):

This algorithm is one of two algorithms in a software package called **word2vec**

(sometimes the algorithm is referred to as word2vec).

The intuition (insight) of skip-gram:

1. Treat the target word and a neighbouring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the learned weights as the embeddings.

4.1 The classifier:

- Consider the following sentence with a target word apricot, and assume we're using a window of 2 context words:

  ... lemon, a [tablespoon of **apricot** jam, a] pinch ...

  c1 c2 w c3 c4

- Our goal is to train a classifier such that, given a tuple **(w, c)**

  of a target word **w** paired with a candidate context word **c** (for example (apricot, jam), or perhaps (apricot, aardvark)) it will return the probability that **c** is a real context word (true for jam, false for aardvark):

  $$P(+|w, c) \quad (6.24)$$

- The probability that word **c** is not a real context word for **w** is just 1 minus Eq. 6.24:

  $$P(-|w, c) = 1 - P(+|w, c) \quad (6.25)$$

Continued…

- Computing the probability P by Classifier:

  This is based on embedding similarity:

  a word is likely to occur near the target if its embedding is similar to the target embedding.

- We rely on the intuition that two vectors are similar if they have a high dot product.

  Similarity (w, c) ≈ c . w (6.26)

- The dot product c . w is not a probability; it's just a number ranging from - infinity to infinity.

To turn the dot product into a probability, we'll use the logistic or sigmoid function σ (x), the fundamental core of logistic regression:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \quad (6.27)$$

Now probability that word c is a real context word for target word w as:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)} \quad (6.28)$$

Continued…

- The sigmoid function returns a number between 0 and 1,
- To make it a probability we'll also need the total probability of the two possible events (c is a context word, and c isn't a context word) to sum to 1.

- We thus estimate the probability that word c is not a real context word for w as:

$$P(-|w,c) = 1 - P(+|w,c) = \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \quad (6.29)$$

- Equation 6.28 gives us the probability for one word, but there are many context words in the window.
- Skip-gram makes the simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(-c_i \cdot w) \quad (6.30)$$

$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(-c_i \cdot w) \quad (6.31)$$

Continued…

- In summary, skip-gram trains a probabilistic classifier that, given a test target word **w** and its context window of **L** words $\mathbf{c}_{1:L}$, assigns a probability based on how similar this context window is to the target word.
- The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word.
- To compute this probability, we just need embeddings for each target word and context word in the vocabulary.

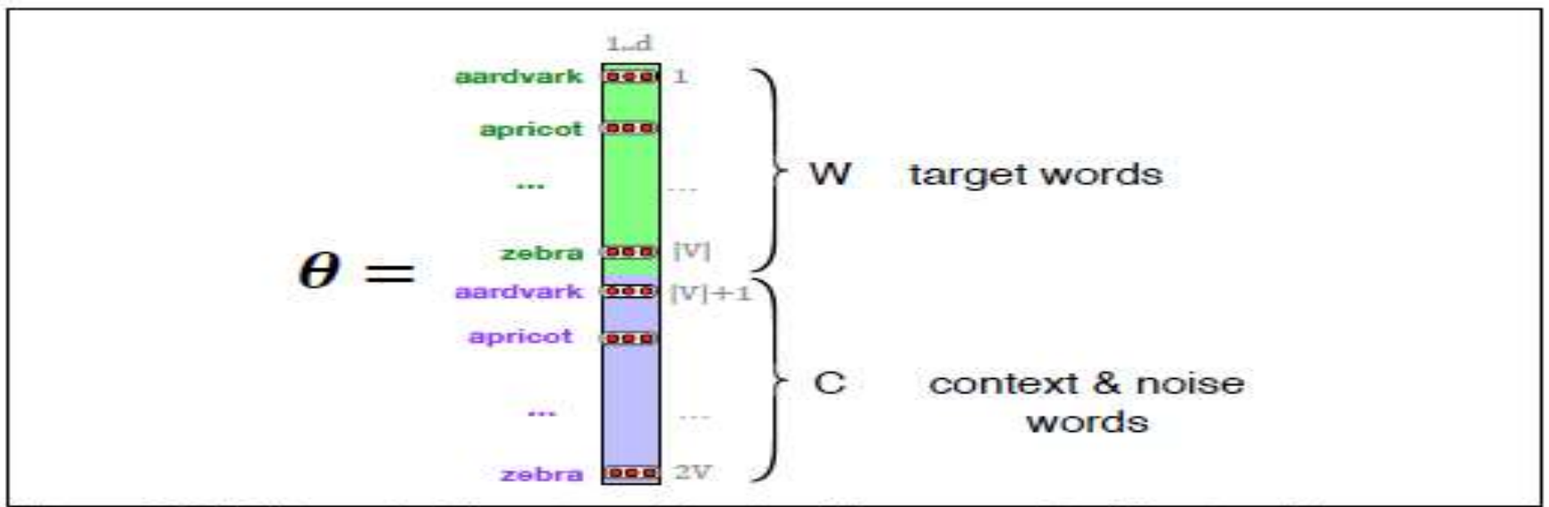

Figure 6.13 The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d, formed by concatenating two matrices, the target embeddings W and the context+noise embeddings C.

Continued…

- Skip-gram actually stores two embeddings for each word,

  one for the word as a target, and *one for the word considered as context*.

- The parameters need to learn are two matrices W and C,

  each containing an embedding for every one of the |V| words in the vocabulary V.

****