Week 1 - Autocorrect

Assumption for this week : Only spelling errors will be corrected and not contextual errors

Major Steps

➤ Identify misspelled word

➤ Find strings 'n' distance away

➤ Filter candidates

➤ Calculate word probabilities

Step 1:
If a word is not in the dictionary, then it is a misspelt word.

Step 2:
Edit is an operation performed on a string to change it. There are four types of edit operation

➤ insert (add a letter)

➤ delete (remove a letter)

➤ switch (swap two adjacent letter)

➤ replace (change 1 letter to another)

The edit distance between two strings is defined as the minimum number of operations required to transform from one string to another.

Conversely, two strings are n edit distance apart if it requires n edit operations to transform from one word to another.

'n' is usually between 1 to 3.

Step 3:
Not all strings resulting from edit operations are valid. Again, check the dictionary to filter out the incorrect strings.

Step 4:
From the possible candidates, find the most probable word (by using a probability dictionary)

Cost of Edit
Since a replace operation can be thought of an insert+delete operation (but the former is easy to visualize), we introduce the concept of cost.

Edit	Cost
Insert	1
Delete	1
Replace	2

There must be a way to find the minimum edit distance between two strings. This is done via 'dynamic programming'

Dynamic Programming
What is the edit distance between play (source) and stay (target)? The following table will help. Each

cell represents edit distance between partial strings of the source and target words. The last cell is the edit distance between the two words.

target

source	#	s	t	a	y
#					
p					
l					
a					
y					

Steps

Consider the first column. How many steps will it take to change a source string to # (i.e. null)? (This only requires delete operations!)

Edit	Cost
# → #	0
p → #	1
pl → #	2
⋮	⋮

Consider the first row. How many steps will it take to change a # string to the target string? (This only requires insert operations!)

Edit	Cost
# → #	0
# → s	1
# → st	2
⋮	⋮

So our table becomes:

target

source	#	s	t	a	y
#	0	1	2	3	4
p	1				
l	2				
a	3				
y	4				

Consider cell (1,1) (p → s). This requires either of the following three operations

➤ delete p + insert s ; p → # → s ; Cost = 1 + 1 = 2

➤ insert s + delete p ; p → ps → s ; Cost = 1 + 1 = 2

➤ replace p with s ; p → s ; Cost = 2

Formula to fill the table.

$$D[i,j] = \min \begin{cases} D[i-1,j] + del\_cost \\ D[i,j-1] + ins\_cost \\ D[i-1,j-1] + \begin{cases} rep\_cost; & \text{if } src[i] \neq tar[j] \\ 0; & \text{if } src[i] = tar[j] \end{cases} \end{cases}$$

Figure 1

When complete, our table looks like

target

source	#	s	t	a	y
#	0	1	2	3	4
p	1	2	3	4	5
l	2	3	4	5	6
a	3	4	5	4	5
y	4	5	6	5	4

Our minimum edit distance is 4.

Week 2 - Part of Speech Tagging and Hidden Markov Models

We will try to complete sentence using HMM by tagging parts of speech...

==========

Wrong?

In this week we will label words with their corresponding part of speech (noun, verb, etc.). This will be used to build a HMM which conveys the probabilities of a word to be followed by another word with the same or different part of speech.

==========

Such transition probabilities can be visualized using the Markov Chain. Each shaded circle is called a state, and represents a part of speech.

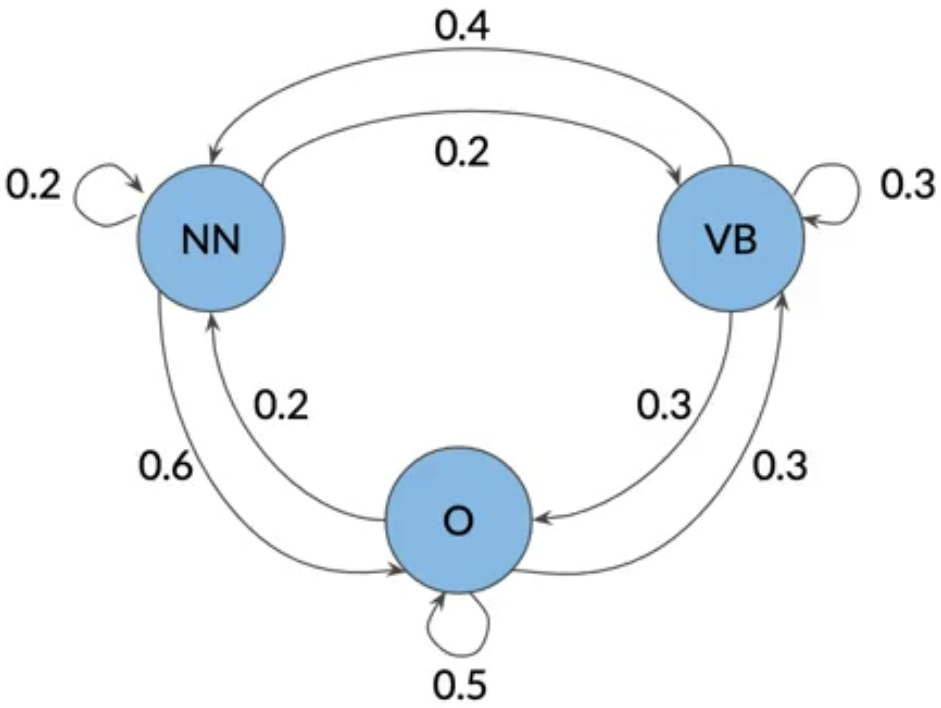

Figure 2

Markov Property: The next state only depends on the current state and not the previous state.

For the sake of programming, the same information is stored in the form of table.

next state

current state	NN	VB	O
NN	0.2	0.2	0.6
VB	0.4	0.3	0.3
O	0.2	0.3	0.5

However, we also need to model an initial state!

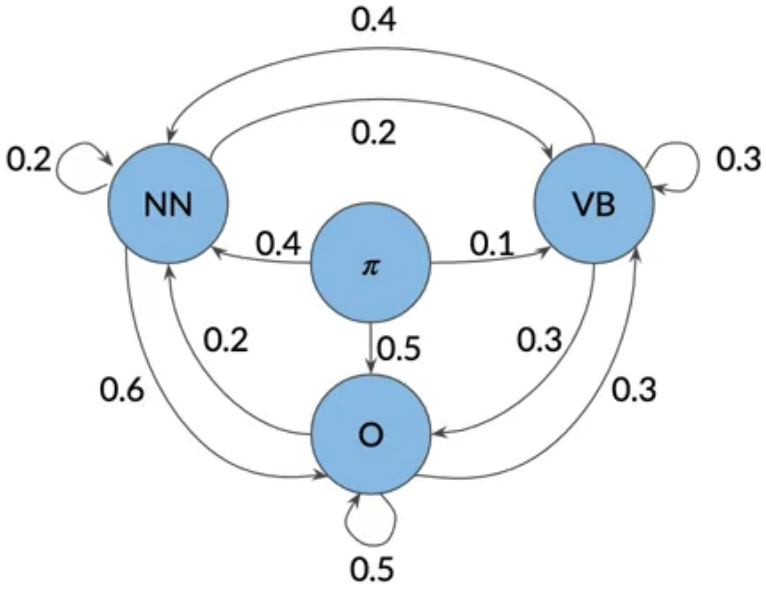

Figure 3

next state

current state	NN	VB	O
π	0.4	0.1	0.5
NN	0.2	0.2	0.6
VB	0.4	0.3	0.3
O	0.2	0.3	0.5

Emission Probabilities

In our corpus, we will not have just one verb or noun. We will have multiple ones. E.g. lets say we have three verbs going, to, and eat. And the probability that given a state is verb, the probability of the words are 0.3, 0.1, and 0.5 respectively. These probabilities are known as emission probabilities and can be shown as follows (in both forms)

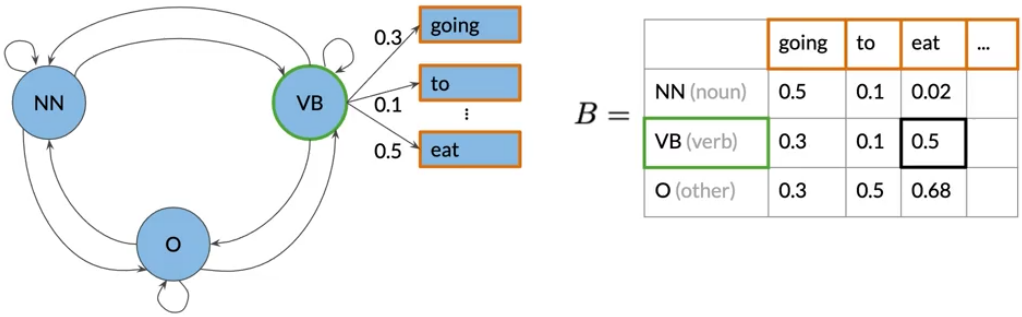

Figure 4

The table shows the possibility that a given word belongs to multiple parts of speech (just for illustration in this example!)

Now, let's see how to calculate these probabilities!

Calculating Transition Probabilities

Steps:

➤ Add a start token $< s >$ to the beginning of each sentence

➤ Label each

$A =$

	NN	VB	O
π	$C(\pi,\text{NN})$		
NN (noun)	$C(\text{NN,NN})$		
VB (verb)	$C(\text{VB,NN})$		
O (other)	$C(\text{O,NN})$		

<s> in a station of the metro
<s> the apparition of these faces in the crowd :
<s> petals on a wet , black bough .

Ezra Pound – 1913

Figure 5

➤ Count all occurrence of "tag pairs" (two given parts of speech - or tags - appear together)

$$C(t_{i-1}, t_i)$$

$A =$

	NN	VB	O
π	1	0	2
NN	0	0	6
VB	0	0	0
O	6	0	8

Figure 6

➤ Calculate the probabilities using the count

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

$A =$

	NN	VB	O	
π	1	0	2	3
NN	0	0	6	6
VB	0	0	0	0
O	6	0	8	14

Figure 7

➤ But non-zero rows (if any at all!) might give errors. So we add a small number epsilon (= 0.001) to the numerator and denominator

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \epsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N * \epsilon}$$

$A =$

	NN	VB	O	
π	1+ε	0+ε	2+ε	3+3*ε
NN	0+ε	0+ε	6+ε	6+3*ε
VB	0+ε	0+ε	0+ε	0+3*ε
O	6+ε	0+ε	8+ε	14+3*ε

Figure 8

(it might also make sense to 'not' add smoothing to the first row (since it will never be zero!) since there might be some parts of speech with which a sentence may never start, so we might assign it a non-zero probability)

➤ The final result will be as follows

$A =$

	NN	VB	O
π	0.3333	0.0003	0.6663
NN	0.0001	0.0001	0.9996
VB	0.3333	0.3333	0.3333
O	0.4285	0.0000	0.5713

Figure 9

Calculating Emission Probabilities

Steps:

➤ Build a table which shows the tags (parts of speech) as the rows and words as the columns.

$B =$

	in	a	...
NN (noun)	$C(\text{NN}, \text{in})$		
VB (verb)	$C(\text{VB}, \text{in})$		
O (other)	$C(\text{O}, \text{in})$		

<s> in a station of the metro
<s> the apparition of these faces in the crowd :
<s> petals on a wet , black bough .

Ezra Pound – 1913

Figure 10

➤ Populate the table and then calculate the probabilities

$B =$

	in	a	...
NN (noun)	0	...	...
VB (verb)	0	...	...
O (other)	2	...	...

$$P(w_i|t_i) = \frac{C(t_i, w_i) + \epsilon}{\sum_{j=1}^{V} C(t_i, w_j) + N * \epsilon}$$
$$= \frac{C(t_i, w_i) + \epsilon}{C(t_i) + N * \epsilon}$$

Figure 11

The Viterbi Algorithm

Now begins the test phase!.

This is a graph algorithm that computes the probabilities of words in a sentence with given parts of speech. As we will see later, the transition and emission matrix will be used to find the numerical values in a mathematical form.

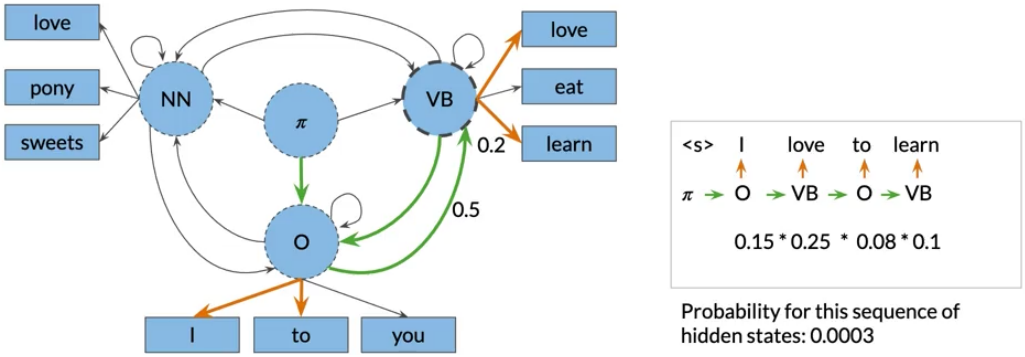

Figure 12

Steps for Viterbi Algorithm

- ➤ Initialization Step
- ➤ Forward pass
- ➤ Backward pass

(Recall that matrices A and B hold the transition and emission probabilities respectively)

This algorithm uses two auxiliary matrices: C and D. C holds the highest probabilities and D holds the corresponding indices.

Initialization Step

The first column represents the probability that the after the initial tag, the first tag is t_x (x can be any row) and the first word is w_1. This can be found by using the following

$$c(i,1) = \pi_i * b_{i,cindex(w1)} = a(1,i) * b_{i,cindex(w1)}$$

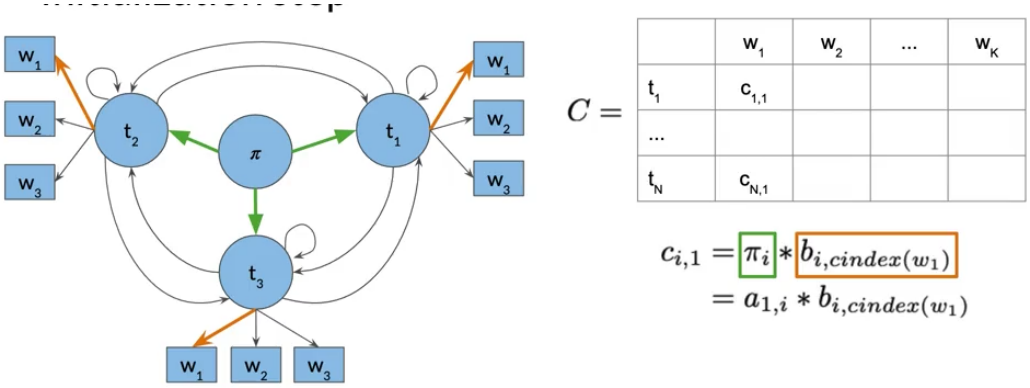

Figure 13

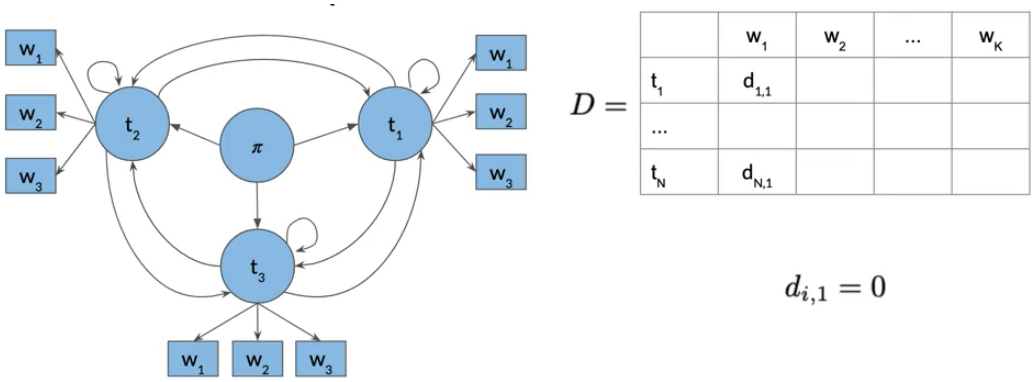

Figure 14

Forward Pass

$$C = \begin{array}{|c|c|c|c|c|} \hline & w_1 & w_2 & \dots & w_K \\ \hline t_1 & c_{1,1} & c_{1,2} & & c_{1,K} \\ \hline \dots & & & & \\ \hline t_N & c_{N,1} & c_{N,2} & & c_{N,K} \\ \hline \end{array}$$

Figure 15

$$c_{i,j} = \max\{c_{k,j-1} * a_{k,i} * b_{i,cindex(w_j)}\}$$

e.g. for $c_{1,2}$

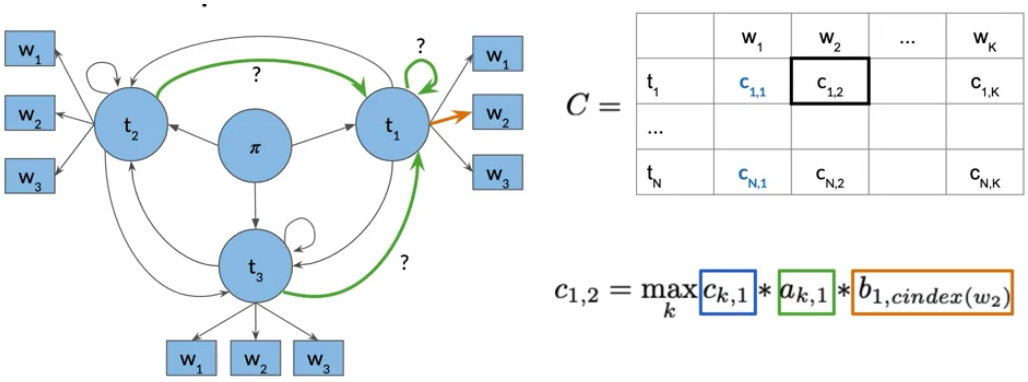

Figure 16

D will simply store the value of k that maximizes $c_{i,j}$

$$d_{i,j} = \text{argmax}\{c_{k,j-1} * a_{k,i} * b_{i,cindex(w_j)}\}$$

Backward Pass

➤ Find the row number of the cell with the highest probability in the last column.

$$s = \text{argmax} c_{i,K}$$

➤ This will be the starting point while we traverse D. (we start from the end)

$D =$

	w_1	w_2	w_3	w_4	w_5
t_1	0	1	3	2	3
t_2	0	2	4	1	3
t_3	0	2	4	1	4
t_4	0	4	4	3	1

<s> w1 w2 w3 w4 w5

$\pi \leftarrow t_2 \leftarrow t_3 \leftarrow t_1 \leftarrow t_3 \leftarrow t_1$

Figure 17

Implementation hint: Use logs!

$$c_{i,j} = \max_k c_{k,j-1} * a_{k,i} * b_{i,cindex(w_j)}$$

$$\downarrow$$

$$log(c_{i,j}) = \max_k log(c_{k,j-1}) + log(a_{k,i}) + log(b_{i,cindex(w_j)})$$

Figure 18

Week 3 - N-gram Models

An N-gram is a sequence of N words with a given order! It can be considered as a stack of conditional probabilities.

Corpus: I am happy because I am learning

Unigrams: { I , am , happy , because , learning }

Bigrams: { I am , am happy , happy because ... }

Trigrams: { I am happy , am happy because, ... }

Figure 19

Notation

➤ m is the length of a text corpus

➤ W_i^j refers to the sequence of words from index i to j from the text corpus

Probability of Unigram

$$P(w) = \frac{C(w)}{m}$$

Probability of Bigram

$$P(y|x) = \frac{C(x,y)}{\sum_w C(x,w)} = \frac{C(x,y)}{C(x)}$$

Probability of N-gram

$$P(w_N|w_1^{N-1}) = \frac{C(w_1^{N-1}w_N)}{C(w_1^{N-1})} = \frac{C(w_1^N)}{C(w_1^{N-1})}$$

Chain Rule and Probability

$$P(B|A) = \frac{P(A,B)}{P(A)} \implies P(A,B) = P(A)P(B|A)$$

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

Figure 20

$$P(\textit{the teacher drinks tea}) =$$
$$P(\textit{the})P(\textit{teacher}|\textit{the})P(\textit{drinks}|\textit{the teacher})$$
$$P(\textit{tea}|\textit{the teacher drinks})$$

Figure 21

Assumption of Sequence Probability

But there is a problem! The probabilities of some length n-grams might be zero.

Input: the teacher drinks tea

$P(\textit{the teacher drinks tea}) = P(\textit{the})P(\textit{teacher}|\textit{the})P(\textit{drinks}|\textit{the teacher})P(\textit{tea}|\textit{the teacher drinks})$

$$P(\textit{tea}|\textit{the teacher drinks}) = \frac{C(\textit{the teacher drinks tea})}{C(\textit{the teacher drinks})}$$

← Both likely 0

Figure 22

So instead, we replace all n-gram probabilities with probabilities of bigram.

$$P(tea|the\ teacher\ drinks) \approx P(tea|drinks)$$

$$P(the\ teacher\ drinks\ tea) =$$
$$P(the)P(teacher|the)P(drinks|the\ teacher)P(tea|the\ teacher\ drinks)$$
$$\downarrow$$
$$P(the)P(teacher|the)P(drinks|teacher)P(tea|drinks)$$

Figure 23

Here, we have employed the Markov assumption that only the last N (=2 above) words matter.

Bigrams:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

N-grams:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

So the entire sentence can be modelled with bigram

$$P(w_1^n) \approx \prod_{i=1}^{n} P(w_i|w_{i-1})$$

$$P(w_1^n) \approx P(w_1)P(w_2|w_1)\dots P(w_n|w_{n-1})$$

However, we won't be able to build N-grams for the first word!

Solution: Append (N-1) 'start of sentence tokens' ($< s >$) at the start.

Also, append a single 'end of sentence tokens' ($< /s >$) at the end.

Lec 05 onwards missing

Week 4 - Word Embeddings

Given a dictionary of words, how to represent words in mathematical form

- As integers
  - simple ☺
  - ordering doesn't make sense ☹
- One-hot vectors
  - simple ☺
  - no ordering implies ☺
  - huge vectos ☹
  - relationship between words cannot be modelled. ☹

➤ Embeddings
  – Low dimensions ☺
  – Incorporate meaning ☺

Word Embedding Methods

➤ word2vec (Google, 2013)
➤ GloVe (Stanford, 2014)
➤ fastText (Facebook, 2016)

Advanced Word Embedding Methods

➤ BERT (Google, 2018)
➤ ELMo (Allen Institute for AI, 2018)
➤ GPT-2 (OpenAI, 2018)

The Continuous Bag-of-Words Model

The premise is that two words that appear together more often are relates semantically.

How to create a training example? Consider the sentence "I am happy because I am learning"

But before this, recall that we have to clean and tokenize the corpus!

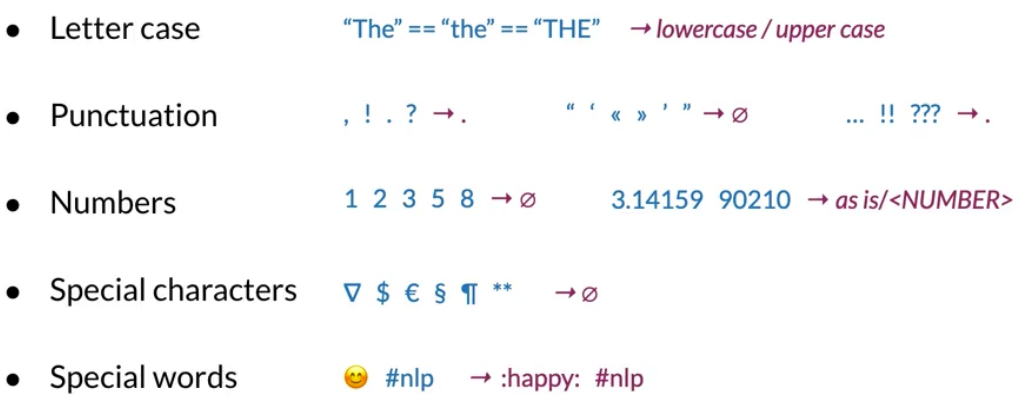

Figure 24

Concept of center word and context word...

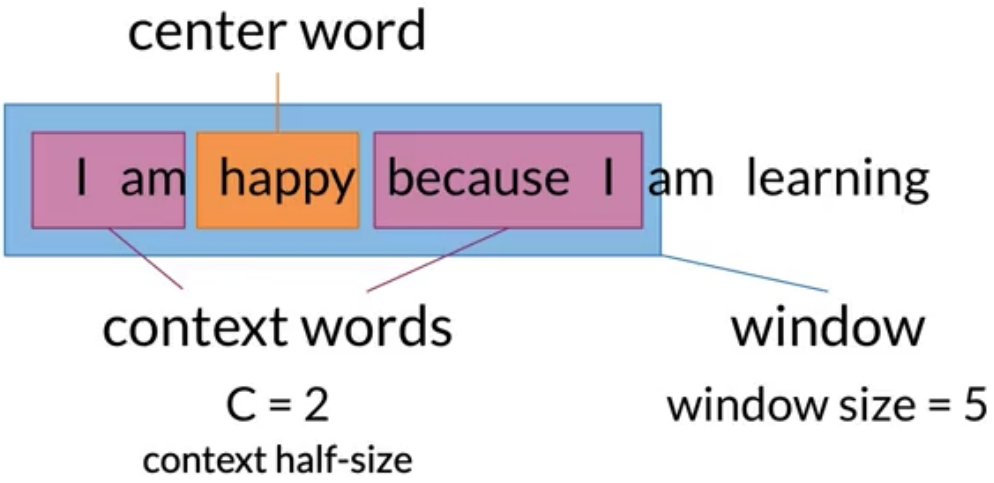

Figure 25

Context Words	Center Words
I am because I	happy
am happy I am	because
happy because am learning	I

Sliding windows for the generation of context and center word

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

Figure 26

```
for x, y in get_windows(
            ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
            2
        ):
    print(f'{x}\t{y}')
```

Figure 27

Generating Vectors

We will be using one-hot vectors to train NN, and then use the trained NN to extract word embeddings.

Steps

- ➤ Given the corpus, generate the dictionary - list of unique words.
- ➤ Encode them using one-hot vectors
- ➤ For the context words, take the average of the one-hot vectors. This will be your **X**
- ➤ Your center word will be represented by the original one-hot vector.

CBOW Model

Architecture

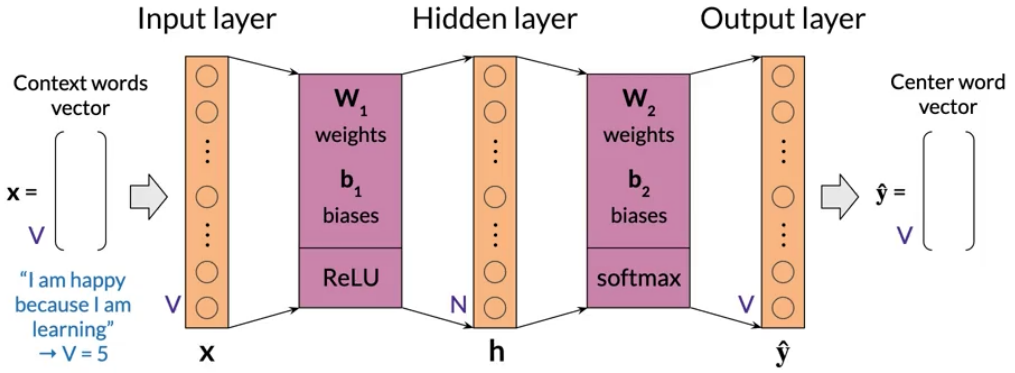

Figure 28

Dimensions of CBOW - Single Input

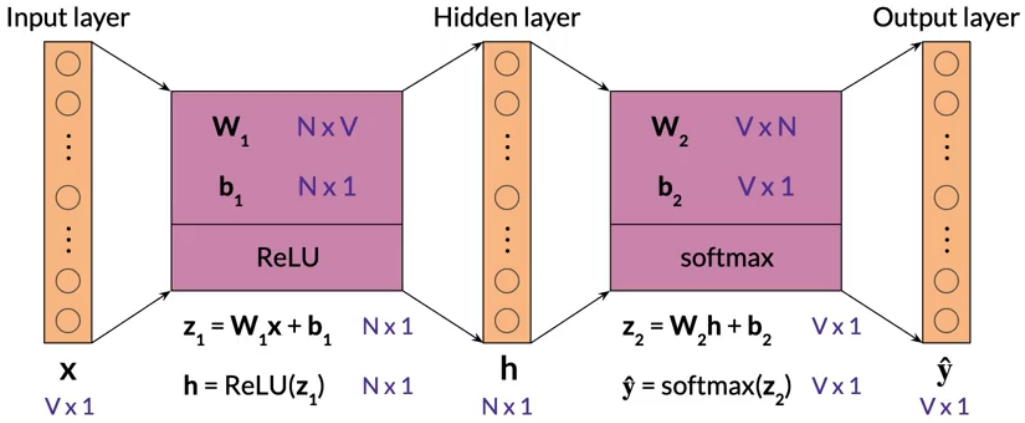

Figure 29

Dimensions of CBOW - Batch Input

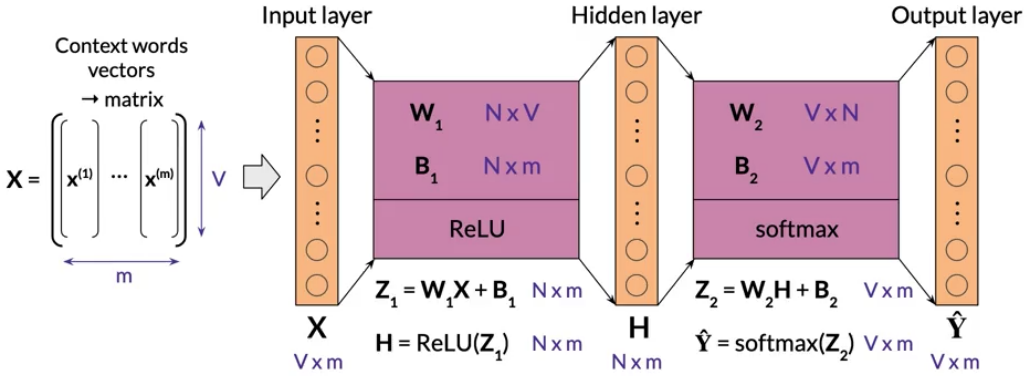

Figure 30

Output of the CBOW Model

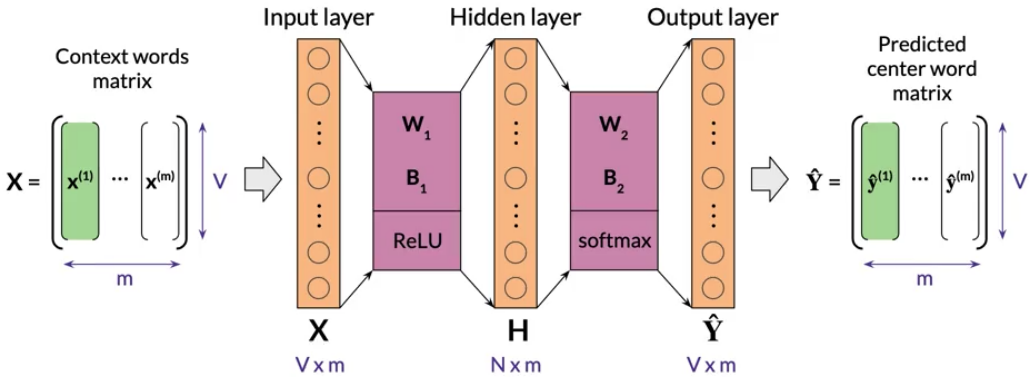

Figure 31

Ways of Extracting Embedding Vectors

- Use columns of W_1
- Use rows of W_2
- Use average of the first two options.

How to Evaluate Language Model

There are two metrics

- Intrinsic Evaluation
- Extrinsic Evaluation

Intrinsic Evaluation

It uses the corpus to 'verify' relations. Examples of such relations could be

- Analogies
- Clustering
- Visualization

Extrinsic Evaluation

It tests the embedding on external tasks

Pros and Cons

- Evaluates actual usefulness of embedding
- time consuming
- difficult to troubleshoot