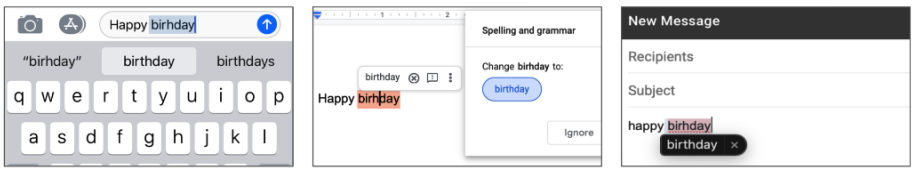
Week 1

Autocorrect and Minimum Edit distance

• Autocorrect



To implement autocorrect in this week's assignment, you have to follow these steps:

- Identify a misspelled word
- Find strings **n** edit distance away: (these could be random strings)
- Filter candidates: (keep only the real words from the previous steps)
- Calculate word probabilities: (choose the word that is most likely to occur in that context)

• Building the model

① Identify the misspelled word

- 사전에 없다면 typo일 수 있음

② Find Strings n edit distance away

- Edit: an operation performed on a string to change it

- Insert (add a letter) 'to': 'top', 'two' ...
- Delete (remove a letter) 'hat': 'ha', 'at', 'ht'
- Switch (swap 2 adjacent letters) 'eta': 'eat', 'tea'
- Replace (change 1 letter to another) 'jaw': 'jar', 'paw', ...

③ Filter candidates

- ②로 생성된 단어중 실제 단어만 취함

④ Calculate word probabilities

- language model을 이용할수도 있지만 이번장은 word frequency 이용

Calculate word probabilities

Example: "I am happy because I am learning"

$$P(w) = \frac{C(w)}{V} \qquad P(\text{am}) = \frac{C(\text{am})}{V} = \frac{2}{7}$$

$P(w)$ Probability of a word

$C(w)$ Number of times the word appears

V Total size of the corpus

Word	Count
I	2
am	2
happy	1
because	1
learning	1

Total : 7

• Minimum edit distance

Minimum edit distance allows you to:

- Evaluate similarity between two strings
- Find the minimum number of edits between two strings
- Implement spelling correction, document similarity, machine translation, DNA sequencing, and more

Example:

Source: p l a y

Target: s t a y

p → s : replace
l → t : replace
} edits = 2

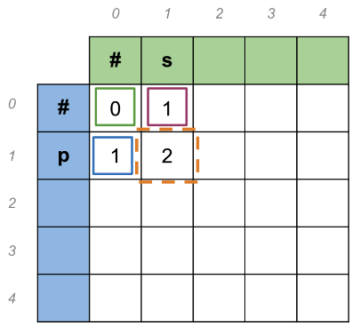
Edit cost:
Insert 1
Delete 1
Replace 2

edit distance = 2 * 2 = 4

Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

p → s

insert + *delete*: p → ps → s: 2
delete + *insert*: p → # → s: 2
replace: p → s: 2

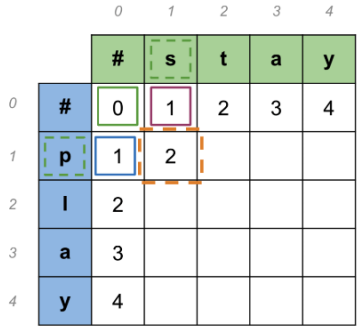


Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

p → s

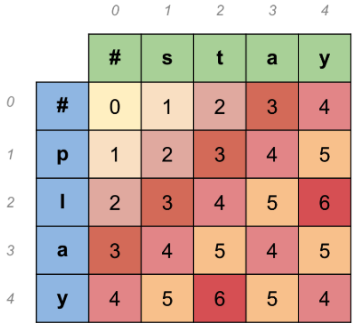
$D[i, j] =$

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



Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance (specifies cost per operation)
- Backtrace (어떤 op으로 same → 가능한이 타당한지 추적)
- Dynamic programming



Week 2

Part of Speech Tagging

- POS Tagging: process of assigning a part of speech to a word.

Subject

- Markov chain
- Hidden Markov Models
- Viterbi algorithm

Part of speech tags:

lexical term	tag	example
noun	NN	something, nothing
verb	VB	learn, study
determiner	DT	the, a
w-adverb	WRB	why, where
...	...	

Why not learn something ?

WRB RB VB NN .

Usage

- Identifying named entities
- Speech recognition
- Conference Resolution

POS 확률 co-occurance를 추론시 사용가능 (LM과 비슷하게)

Markov Chains

- 다음 단어의 확률을 나타낼때 사용

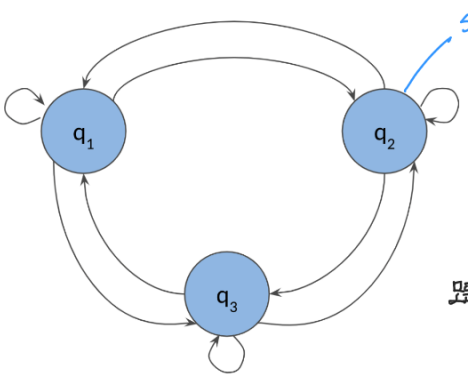
Why not learn swimming?

verb noun 확률↑

Why not learn swim?

verb verb

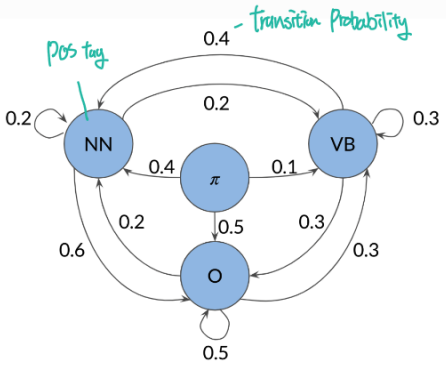
- 확률 모델을 만드려면 POS tag와 word 확률이 필요함



State : 형태소간의 특정한 조건을 나타냄
Ex. 형태소 단어의 POS tag.

모든 상태집합

$$Q = \{q_1, q_2, q_3\}$$



Transition matrix

$A =$

	NN	VB	O
π (initial)	0.4	0.1	0.5
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

← initial distribution. Ex. 40% 확률로 문장의 시작으로 시작

Formal Notation

States

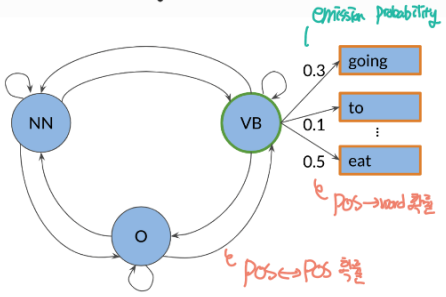
$$Q = \{q_1, \ldots, q_N\}$$

Transition matrix

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \cdots & a_{N+1,N} \end{pmatrix}$$

Hidden Markov Models

• emission probability를 통해 State(POS)에서 specific word로의 확률을 나타냄



$B =$ <emission matrix>

	going	to	eat	...
NN (noun)	0.5	0.1	0.02	
VB (verb)	0.3	0.1	0.5	
O (other)	0.3	0.5	0.68	

labelled dataset에서 각 POS태그에 대해 확률 계산

States

$$Q = \{q_1, \ldots, q_N\}$$

Transition matrix

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \cdots & a_{N+1,N} \end{pmatrix}$$

Σ=1

Emission matrix

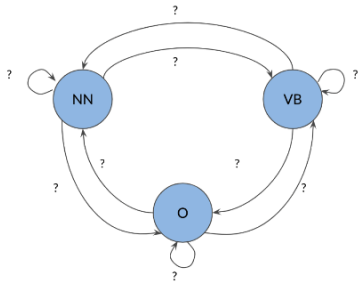
$$B = \begin{pmatrix} b_{11} & \cdots & b_{1V} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NV} \end{pmatrix}$$

Σ=1

$$\sum_{j=1}^{V} b_{ij} = 1$$

Calculating Probabilities

transition probability: ■ + ■ = ⅔



1. Count occurrences of tag pairs

$C(t_{i-1}, t_i)$: t_{i-1} 다음 t_i가 나올경우 count

2. Calculate probabilities using the counts

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

t_{i-1} 다음 anything count

Populating Transition Matrix

$A =$

	NN	VB	O
π	1	0	2
NN (noun)	0	0	6
VB (verb)	0	0	0
O (other)	6	0	8

① count (t_{i-1}, t_i)

<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

$A =$

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

② 확률 계산

$$P(\text{NN}|\text{O}) = \frac{C(\text{O}, \text{NN})}{\sum_{j=1}^{N} C(\text{O}, t_j)} = \frac{6}{14}$$

To generalize:

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

Smoothing

$A =$

	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

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

- 확률이 0이되지 않도록

Populating Emission Matrix

$B =$

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

<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

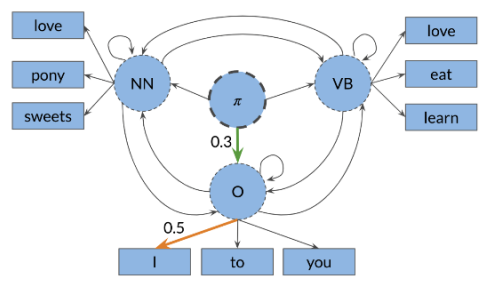
Smoothing

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

- $C(t_i, w_i)$: tag t_i와 w_i가 연관된 count

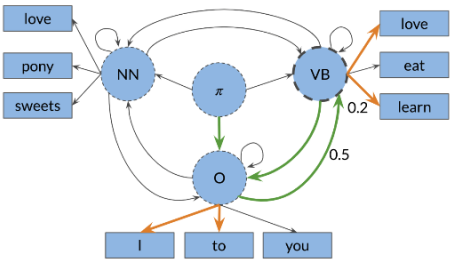
Viterbi : transition matrix + emission matrix로 확률계산

The Viterbi Algorithm



<s> I love to learn
π → O
0.15

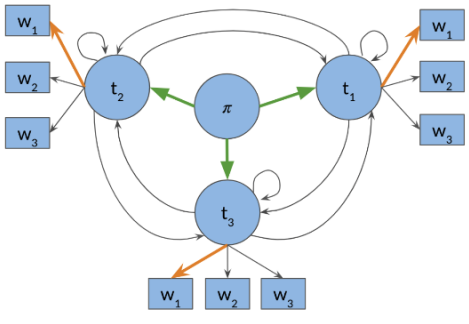
0.3 x 0.5
transition x emission



<s> I love to learn
π → O → VB → O → VB
0.15 * 0.25 * 0.08 * 0.1

Probability for this sequence of hidden states: 0.0003

Viterbi Initialization



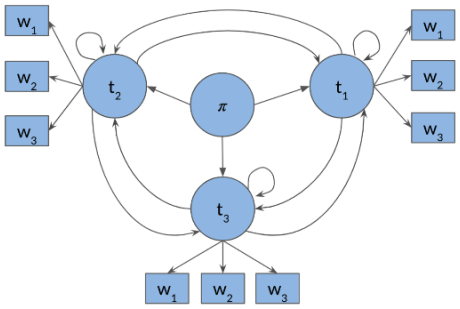
$C =$

	w_1	w_2	...	w_K
t_1	$c_{1,1}$			
...				
t_N	$c_{N,1}$			

※ of words
※ of tags
← what POS each word belongs to

$$c_{i,1} = \pi_i * b_{i,cindex(w_1)} = a_{1,i} * b_{i,cindex(w_1)}$$

tag
emission matrix에 word의 index
emission 값
transition matrix의 첫행



$D =$

	w_1	w_2	...	w_K
t_1	$d_{1,1}$			
...				
t_N	$d_{N,1}$			

- What POS coming from

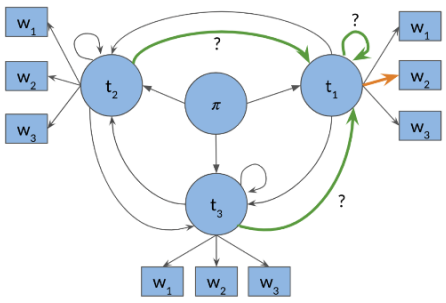
- $w_1, \cdots, w_K$의 sequence가 주어졌을때 가장 높은 확률의 POS sequence를 찾을때 사용

$d_{i,1} = 0$

(처음엔 다른 POS로부터 온 값이 0을 초기화

Viterbi :

Forward Pass



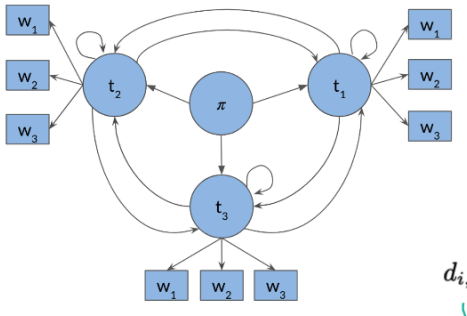
$C =$

	w_1	w_2	...	w_K
t_1	$c_{1,1}$	$c_{1,2}$		$c_{1,K}$
...				
t_N	$c_{N,1}$	$c_{N,2}$		$c_{N,K}$

$$c_{1,2} = \max_k c_{k,1} * a_{k,1} * b_{1,cindex(w_2)}$$

(transition, emission)

모든 k에 대해 확률의 최댓값



$D =$

	w_1	w_2	...	w_K
t_1	$d_{1,1}$	$d_{1,2}$		$d_{1,K}$
...				
t_N	$d_{N,1}$	$d_{N,2}$		$d_{N,K}$

- keep track of which was used to get max probability

(Probability)

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

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

(index)

Viterbi :

Backward Pass

$C =$

	w_1	w_2	w_3	w_4	w_5
t_1	0.25	0.125	0.025	0.0125	0.01
t_2	0.1	0.025	0.05	0.01	0.003
t_3	0.3	0.05	0.025	0.02	0.0000
t_4	0.2	0.1	0.000	0.0025	0.0003

$$s = \operatorname{argmax}_i c_{i,K} = 1$$

(마지막 열의 row index 시작!)

$D =$

	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

(직전 열의 index)

<s> w1 w2 w3 w4 w5

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

∴ reconstruct POS tags of sentence

Week 3

Autocomplete and Language Models

N-Grams Overview

- Calculate probabilities of certain words happening in a specific sequence.

<Application>

Speech recognition

P(I saw a van) > P(eyes awe of an)

Spelling correction

"He entered the ship to buy some groceries" - "ship" a dictionary word
- P(entered the shop to buy) > P(entered the ship to buy)

Augmentative communication

Predict most likely word from menu for people unable to physically talk or sign.
(Newell et al., 1998)

N-grams and Probabilities

An N-gram is a sequence of N words

Corpus: I am happy because I am learning

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

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

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

Corpus: This is great ... teacher drinks tea. $m = 500$

$w_1 \; w_2 \; w_3$ ... $w_{498} \; w_{499} \; w_{500}$

- 개념

- $w_1^m = w_1 w_2 w_3 .... w_m$
- $w_1^3 = w_1 w_2 w_3$
- $w_{m-2}^m = w_{m-2} w_{m-1} w_m$

Unigram

- Size of corpus m = 7.
- $P(I) = \frac{2}{7}$
- $P(happy) = \frac{1}{7}$

To generalize, the probability of a unigram is $P(w) = \frac{C(w)}{m}$

Bigram Probability:

Corpus: I am happy because I am learning

$P(am|I) = \frac{C(I\ am)}{C(I)} = \frac{2}{2} = 1$ $\quad P(happy|I) = \frac{C(I\ happy)}{C(I)} = \frac{0}{2} = 0$ ✖ I happy

$P(learning|am) = \frac{C(am\ learning)}{C(am)} = \frac{1}{2}$

Probability of a bigram: $P(y|x) = \frac{C(x\ y)}{\sum_w C(x\ w)} = \frac{C(x\ y)}{C(x)}$

Trigram Probability:

To compute the probability of a trigram:

- $P\left(w_3 \mid w_1^2\right) = \frac{C\left(w_1^2 w_3\right)}{C\left(w_1^2\right)}$
- $C\left(w_1^2 w_3\right) = C\left(w_1 w_2 w_3\right) = C\left(w_1^3\right)$

N-gram Probability:

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

Sequence Probabilities

- How to approximate N-gram probabilities?
(구하려면 sentence의 probability를 고사해야함

- 조건부 확률

- $P(B \mid A) = \frac{P(A,B)}{P(A)} \Longrightarrow P(A,B) = P(A)P(B \mid A)$
- $P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A,B)P(D \mid A,B,C)$

Ex.

$P(\text{ the teacher drinks tea }) = P(\text{the})P(\text{ teacher the})P(\text{ drinks the teacher})P(\text{tea} \mid \text{the teacher drinks })$

- Corpus에 내가 구하려는 문장이 매우 드물게 등장한다. (특히 문장이 길때) → 확률이 0에 가까움

- Markov Assumption: Only the last word matters

- Bigram $P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-1}\right)$ ← 직전 1개
- N-gram $P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-N+1}^{n-1}\right)$ ← 직전 N-1개

- 전체문장 확률 근사

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

Starting and Ending Sentences

- N-gram language model 일때 맨앞에 N-1개 <s> 추가, 마지막에는 </s> 하나 추가

Corpus
<s> Lyn drinks chocolate </s>
<s> John drinks tea </s>
<s> Lyn eats chocolate </s>

$P(sentence) = \frac{2}{3} * \frac{1}{2} * \frac{1}{2} * \frac{2}{2} = \frac{1}{6}$

$P(John|<s>) = \frac{1}{3}$

$P(</s>|tea) = \frac{1}{1}$

$P(chocolate|eats) = \frac{1}{2}$

$P(Lyn|<s>) =? = \frac{2}{3}$

The N-gram Language Model

- Count matrix

- Bigram count matrix

Corpus: <s>I study I learn</s>

	<s>	</s>	I	study	learn
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	1	1
study	0	0	1	0	0
learn	0	1	0	0	0

Unique corpus words

"study I" bigram

Unique corpus N-1 grams.
지금은 bigram이라 단어임.

- Probability matrix로 변환

- $P\left(w_n \mid w_{n-N+1}^{n-1}\right) = \frac{C\left(w_{n-N+1}^{n-1}, w_n\right)}{C\left(w_{n-N+1}^{n-1}\right)}$

- $\text{sum}(row) = \sum_{w \in V} C\left(w_{n-N+1}^{n-1}, w\right) = C\left(w_{n-N+1}^{n-1}\right)$

$\frac{\text{해당 row, col}}{\text{sum(row)}}$

- Probability matrix로 LM 생성 가능. 문장확률 / 단어예측 가능

To compute the probability of a sequence, you needed to compute:

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

To avoid underflow, you can multiply by the log.

$$\log\left(P\left(w_1^n\right)\right) \approx \sum_{i=1}^{n} \log\left(P\left(w_i \mid w_{i-1}\right)\right)$$

Finally here is a summary to create the generative model:

Corpus:
<s> Lyn drinks chocolate </s>
<s> John drinks tea </s>
<s> Lyn eats chocolate </s>

1. (<s>, Lyn) or (<s>, John)?
2. (Lyn,eats) or (Lyn,drinks) ?
3. (drinks,tea) or (drinks,chocolate)?
4. (tea,</s>) - always

Algorithm:

1. Choose sentence start
2. Choose next bigram starting with previous word
3. Continue until </s> is picked

Language Model Evaluation

① Splitting the Data

Train/Val/Test splits

Smaller Corpora:

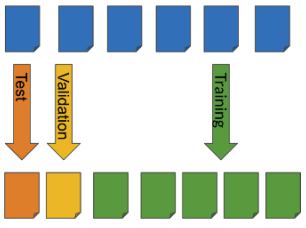
- 80% train
- 10% val
- 10% test

Larger Corpora:

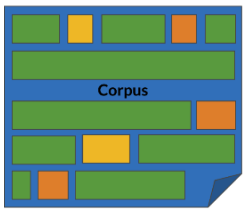
- 98% train
- 1% val
- 1% test

There are two main methods for splitting the data:

- Continuous text



- Random short sequences



② Perplexity

- 사람이 쓴 문장: ↓ , random: ↑

$$PP(W) = P(s_1, s_2, \ldots, s_m)^{-\frac{1}{m}}$$

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m} \prod_{j=1}^{|s_i|} \frac{1}{P\left(w_j^{(i)} \mid w_{j-1}^{(i)}\right)}}$$

$w_j^{(i)}$: i번째 문장의 j번째 단어

→ 모든 문장을 이으면 N_w로 표현가능

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i|w_{i-1})}}$$ 문장확률을 bigram으로 근사한건데, N-gram근사도 있을까?

To

$$\log PP(W) = -\frac{1}{m} \sum_{i=1}^{m} \log_2 \left(P\left(w_i \mid w_{i-1}\right)\right)$$

OOV words

- Vocabulary: set of unique words supported by language model
  - Closed: only fixed set of words (Ex. speech recognition, question answering)
  - open: encounter words outside the vocabulary

- Unknown word 다루기

- Create vocabulary V
- Replace any word in corpus and not in V by <UNK>
- Count the probabilities with <UNK> as with any other word

Corpus
<s> Lyn drinks chocolate </s>
<s> John drinks tea </s>
<s> Lyn eats chocolate </s>

Corpus
<s> Lyn drinks chocolate </s>
<s> <UNK> drinks <UNK> </s>
<s> Lyn <UNK> chocolate </s>

Min frequency f=2 ~이보다 적게 나타나면 <UNK>. <UNK>는 다른 관점과 동일하게 취급

Vocabulary
Lyn, drinks, chocolate

Input query
<s>Adam drinks chocolate</s>
<s><UNK> drinks chocolate</s>

Criteria to create the vocabulary

- Min word frequency f
- Max |V|, include words by frequency
- Use <UNK> sparingly (Why?)
- Perplexity - only compare LMs with the same V

Smoothing

$$P\left(w_n \mid w_{n-N+1}^{n-1}\right) = \frac{C\left(w_{n-N+1}^{n-1}, w_n\right)}{C\left(w_{n-N+1}^{n-1}\right)} \text{ can be } 0$$

Hence we can add-1 smoothing as follows to fix that problem:

$$P\left(w_n \mid w_{n-1}\right) = \frac{C\left(w_{n-1}, w_n\right)+1}{\sum_{w \in V}\left(C\left(w_{n-1}, w\right)+1\right)} = \frac{C\left(w_{n-1}, w_n\right)+1}{C\left(w_{n-1}\right)+V}$$

Add-k smoothing is very similar:

$$P\left(w_n \mid w_{n-1}\right) = \frac{C\left(w_{n-1}, w_n\right)+k}{\sum_{w \in V}\left(C\left(w_{n-1}, w\right)+k\right)} = \frac{C\left(w_{n-1}, w_n\right)+k}{C\left(w_{n-1}\right)+k*V}$$

back off

- If N-gram missing => use (N-1)-gram, …: Using the lower level N-grams (i.e. (N-1)-gram, (N-2)-gram, down to unigram) distorts the probability distribution. Especially for smaller corpora, some probability needs to be discounted from higher level N-grams to use it for lower level N-grams.
- Probability discounting e.g. Katz backoff: makes use of discounting.
- "Stupid" backoff: If the higher order N-gram probability is missing, the lower order N-gram probability is used, just multiplied by a constant. A constant of about 0.4 was experimentally shown to work well.

back off example

Corpus
<s> Lyn drinks chocolate </s>
<s> John drinks tea </s>
<s> Lyn eats chocolate </s>

$$P(chocolate|John\ drinks) = ?$$

$$0.4 \times P(chocolate|drinks)$$

- Interpolation

$$\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 \times P(w_n \mid w_{n-2}w_{n-1}) + \lambda_2 \times P(w_n \mid w_{n-1}) + \lambda_3 \times P(w_n)$$

Where

$$\sum_i \lambda_i = 1$$

Week 4

Word Embeddings

- Basic Word Representations

- Integers
- One-hot vectors
- Word embeddings

No reason for 어떤 단어는 큰 수, 다른 단어는 작은 수

장점: 간단함, 순서없음

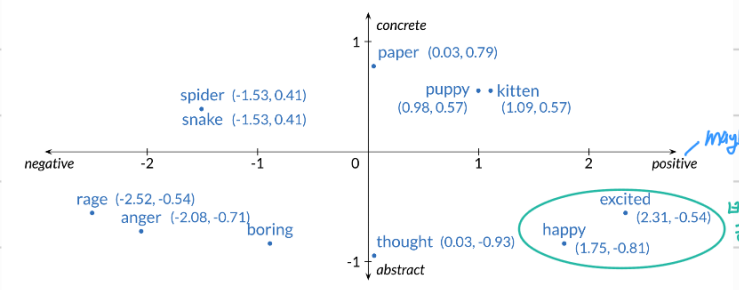
단점: huge, encode no meaning

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

"happy"

1	0	a
2	0	able
3	0	about
...	⋮	...
615	0	hand
...	⋮	...
621	1	happy
...	⋮	...
1000	0	zebra

- Word Embeddings



장점: Low dimensions (less than V)

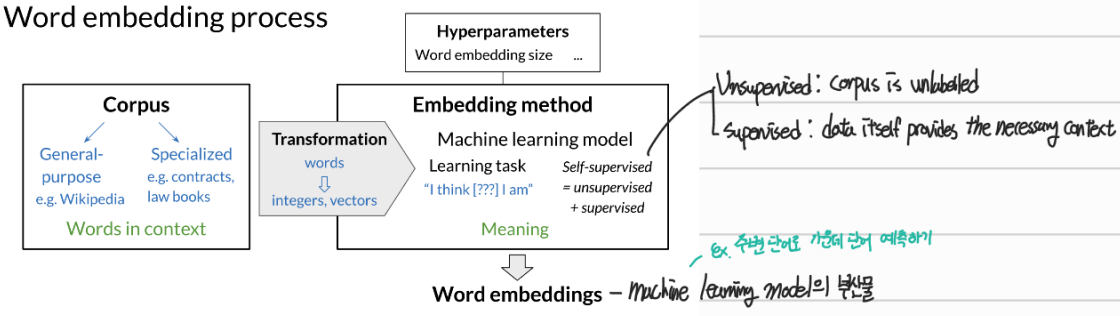
encode meaning

- Coordinates이 word에 대해 어떤 정보를 포착한다고 생각

How to create Word Embeddings

- need: ① Corpus ② embedding method
- Context가 각 word embedding에 의미를 부여함

Word embedding process



Word Embedding Methods

Classical Methods

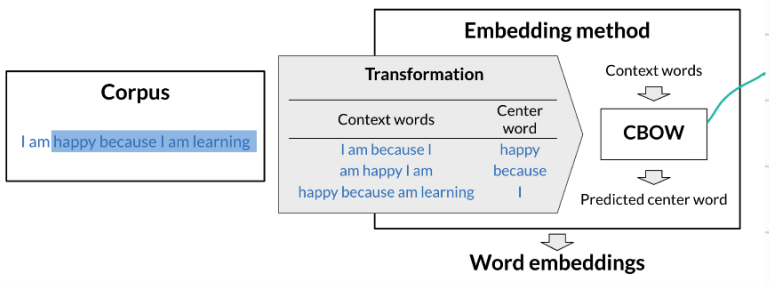
- word2vec (Google, 2013)
- *Continuous bag-of-words (CBOW):* the model learns to predict the center word given some context words.
- *Continuous skip-gram / Skip-gram with negative sampling (SGNS):* the model learns to predict the words surrounding a given input word.
- *Global Vectors (GloVe) (Stanford, 2014):* factorizes the logarithm of the corpus's word co-occurrence matrix, similar to the count matrix you've used before.
- *fastText (Facebook, 2016):* based on the skip-gram model and takes into account the structure of words by representing words as an n-gram of characters. It supports out-of-vocabulary (OOV) words.

Deep learning, contextual embeddings

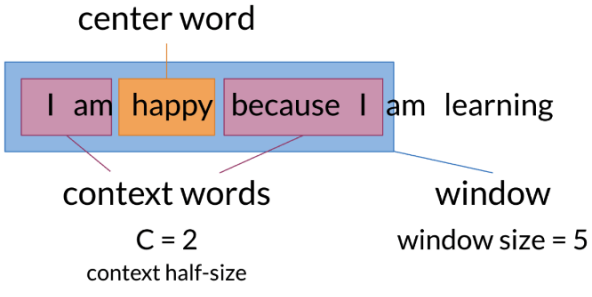
In these more advanced models, words have different embeddings depending on their context. You can download pre-trained embeddings for the following models.

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

CBOW model



predict a missing word based on the surrounding words



Cleaning and Tokenization

• NLP algorithm implement 전에 처리 절차

- Letter case — "The" == "the" == "THE" → *lowercase / upper case*
- Punctuation — , ! . ? → . " ' « » ' " → ∅ ... !! ??? → .
- Numbers — 1 2 3 5 8 → ∅ 3.14159 90210 → *as is/<NUMBER>*
- Special characters — ∇ $ € § ¶ ** → ∅
- Special words — 😊 #nlp → :happy: #nlp

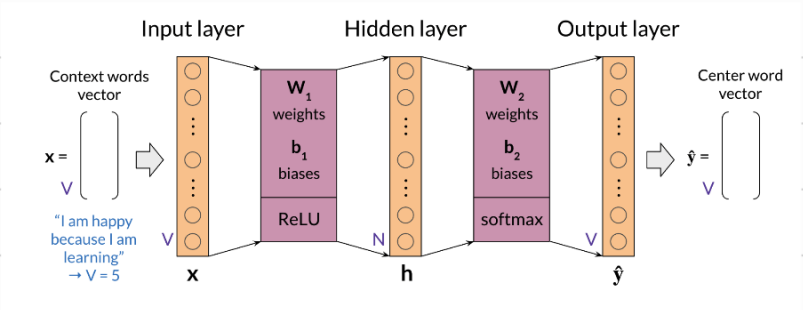
Transforming Words into Vectors

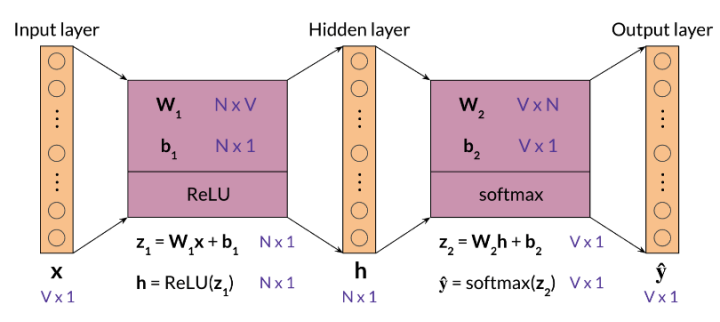
$$\begin{bmatrix} & \text{I} & & \text{am} & & \text{because} & & \text{I} \\ \text{am} & 0 & & 1 & & 0 & & 0 \\ \text{because} & 0 & & 0 & & 1 & & 0 \\ \text{happy} & 0 & + & 0 & + & 0 & + & 0 \\ \text{I} & 1 & & 0 & & 0 & & 1 \\ \text{learning} & 0 & & 0 & & 0 & & 0 \end{bmatrix} / 4 = \begin{matrix} \text{I am because I} \\ \begin{bmatrix} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} \end{matrix}$$

• one-hot vector 사용

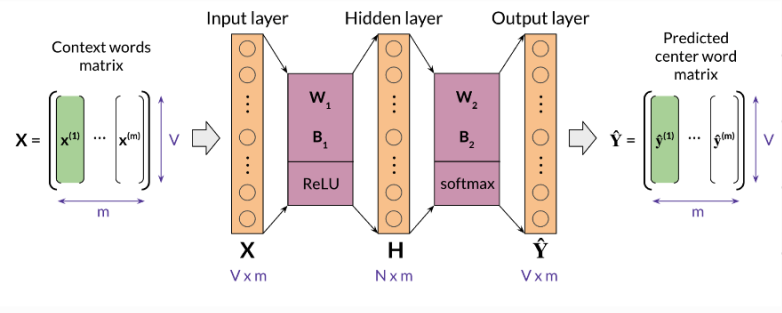
Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]

Architecture for the CBOW Model



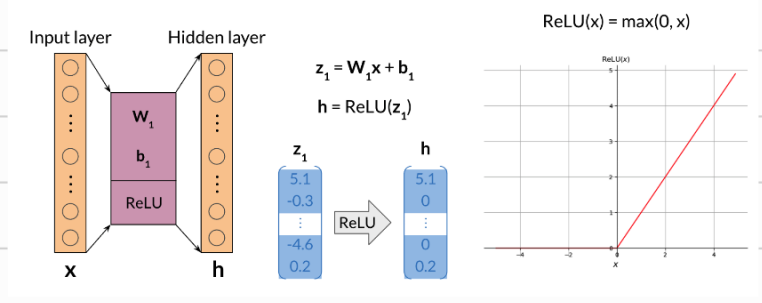


- Batch Input dimension

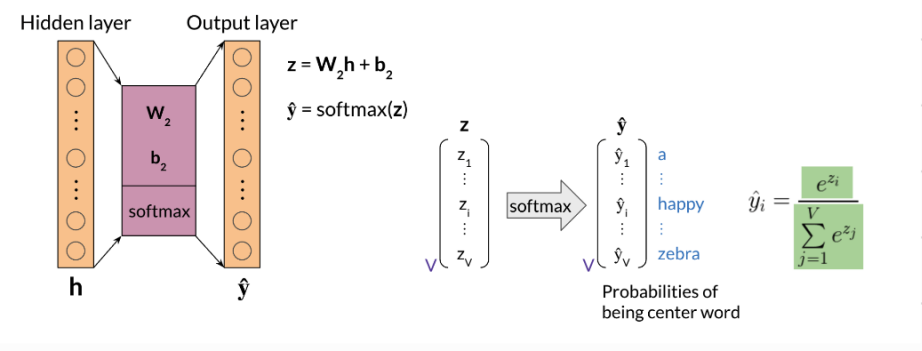


Activation Functions

- ReLU



- Softmax : vector → probability distribution



Training a CBOW model

- Cost function

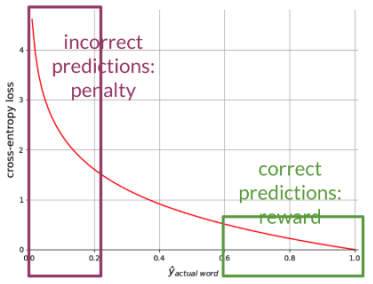
• Cross-entropy error 사용

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$

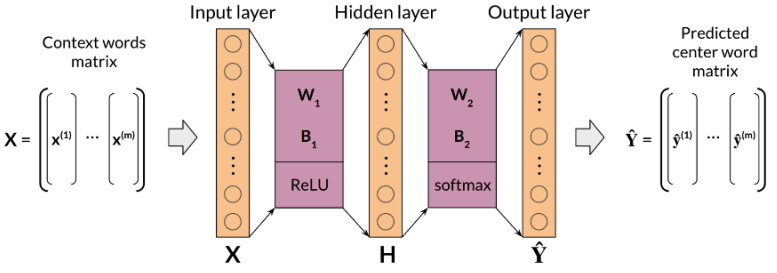
$J = -\log \hat{y}_{\text{actual word}}$

y		$\hat{y}$
0	am	0.96
0	because	0.01
1	happy	0.01
0	I	0.01
0	learning	0.01

→ J = 4.61



Forward Propagation



$$Z_1 = W_1 X + B_1$$

$$H = ReLU(Z_1)$$

$$Z_2 = W_2 H + B_2$$

$$\hat{Y} = softmax(Z_2)$$

$$J_{batch} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{V} y_j^{(i)} \log \hat{y}_j^{(i)}$$

Backpropagation and Gradient descent

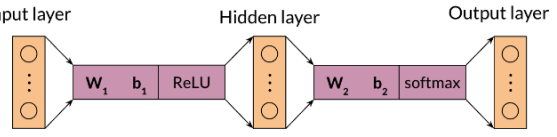
learning rate

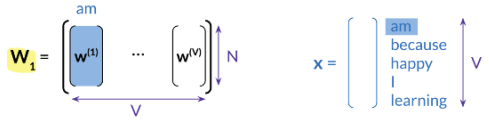
$$\mathbf{W_1} := \mathbf{W_1} - \alpha \frac{\partial J_{\text{batch}}}{\partial \mathbf{W_1}}$$
$$\mathbf{W_2} := \mathbf{W_2} - \alpha \frac{\partial J_{\text{batch}}}{\partial \mathbf{W_2}}$$
$$\mathbf{b_1} := \mathbf{b_1} - \alpha \frac{\partial J_{\text{batch}}}{\partial \mathbf{b_1}}$$
$$\mathbf{b_2} := \mathbf{b_2} - \alpha \frac{\partial J_{\text{batch}}}{\partial \mathbf{b_2}}$$

} Calculate partial derivatives w.r.t weights & biases.

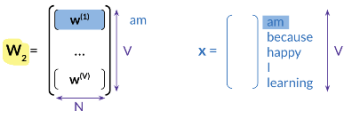
Extracting Word Embedding Vectors

① W_1 or W_2





If you were to use w_1, each column will correspond to the embeddings of a specific word. You can also use w_2 as follows:



② W_1과 W_2의 평균 취하기

$$\mathbf{W_3} = 0.5\,(\mathbf{W_1} + \mathbf{W_2}^T) = \begin{bmatrix} \mathbf{w_3}^{(1)} & \cdots & \mathbf{w_3}^{(V)} \end{bmatrix}$$

N, V

Evaluating Word Embeddings: Intrinsic Evaluation

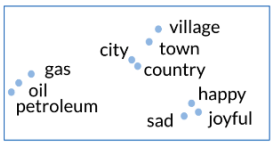
- Test relationships between words

Ex. Semantic analogies: France : Paris = Italy : ?

Syntactic analogies: Seen : saw = been : ?

Test relationships between words

- Analogies
- Clustering
- Visualization



Extrinsic Evaluation

- tests embeddings on external tasks like named entity recognition, parts-of-speech tagging, etc..

- + Evaluates actual usefulness of embeddings
- - Time Consuming
- - More difficult to trouble shoot