Parts of Speech Taggings (POS)

Ex. POS tag : NN (Noun, singular), NNP (Noun, plural), VB (verb), ... In this case: state is a tag

I. Data Source

- Training data: WSJ_02-21.pos ; hmm_vocab.txt

Training corpus file (~1M (word/tag) pairs)

word	Tag
In	IN
an	DT
Oct.	NNP
⋮	

--unk--
--unk_adj--
Bello
Belo
⋮

~24K words

Note: in WSJ
- ≈ 87% token is unambiguous (one token has one tag)
- ≈ 13 % token is ambiguous (one token has more than one tag)

- Test data: WSJ_24.pos (same format as above)

test_word.text

Change to vocab = {'--unk--' : 0, '--unk_adj--' : 10, 'Bello' : 111, ...} (word : index of this word in a sorted list)

II. POS tagging

- Training
  - Transition count : Number of times (#) tag t_{i-1} happened next to tag t_i. Ex. DT → NNP
  - Emission count : # a word w_i given tag t_i. Ex. How many words given tag NNP
  - Tag count : # each tag appears

```
def create_dics ( training_corpus, vocab ):
    transition_count = defaultdict (int)   (access a key NOT exist, return value = 0)
```

Ex: transition_count = {('--s--', 'IN') : 5050, ('DT', 'NNP') : 9044, ...} (prev_tag, tag)

emission_count = {('DT', 'any') : 721, ('NN', 'decrease') : 7, ...} (tag, word)

tag_count = {'IN' : 90000, '--s--' : 34560, ...} ('--s--' : start state (a start of a sentence))

III. Hidden Markov Models (HMM) for POS

- state is a POS which is hidden.
- Markov Model utilizes a transition matrix A
- HMM adds an Emission matrix B which describes the probability of a visible observation when we are in a particular state.

- Transition Probability Matrix A. Ex. (built by transition count)

(prev state \ curr state)	NN	VB	O
NN	0.5	0.2	0.3
VB	0.4	0.2	0.4
O	0.1	0.1	0.8

(curr state: is a word in the corpus; each row sum = 1)

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

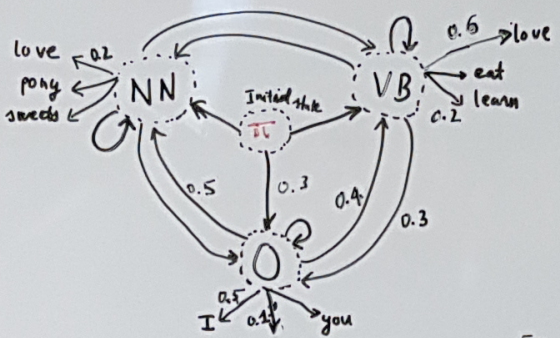
tag (state) $t_{i-1} \rightarrow t_i$

$C(t_{i-1}, t_i)$: transition count from t_{i-1}, t_i

N : number of tags (states)

α and $\alpha \neq N$ are used for smoothing

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



- Emission Probability Matrix $B_{N \times M}$

$$P(w_i \mid t_i) = \frac{C(t_i, word_i) + \alpha}{\sum_{j=1}^{M} C(t_i, word_j) + \alpha . M}$$

$C(t_i, word_i)$: count number of time of $word_i$ given by tag t_i

M : total number of words

α or αM : smoothing

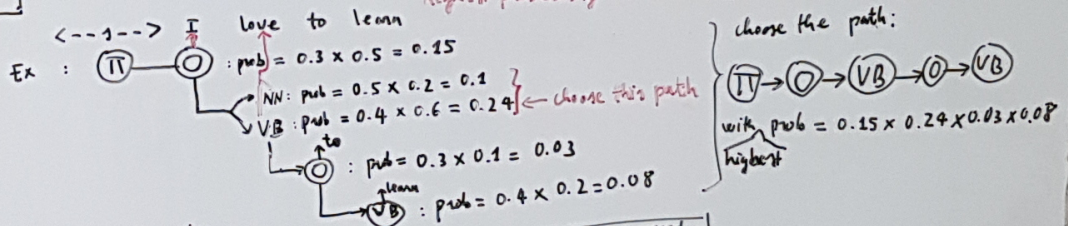
Ex.

	... 725	engineering	learn	...
NN	$\sim 10^{-8}$	$\sim 10^{-5}$	$\sim 10^{-8}$	...
VB	$\sim 10^{-8}$	$\sim 10^{-4}$	$\sim 10^{-3}$	
NNP				
⋮				

IV. Viterbi Algo

Ex. I love to learn → Find the sequence of hidden states (tags) that has the highest probability on this sequence.

(Note: love can be in states NN or VB)



Algo: Need 2 matrix

C = N x M (to store intermediate optimal probability)

	w_1	w_2	.....	word_M
t_1	c_{11}			
t_2				
⋮				
t_N	c_{1N}			

D = N x M (to store indices of visited state)

	w_1	w_2	...	word_M
t_1				
t_2				
⋮				
t_N				

i index: 1 to N states (tags)

j index: 1 to M words

t_i states = ['--s--', 'IN', 'DT', 'NN', 'NNP', ...] having 46 states in this case.

(1) Initialization step (First word in a sentence)

$c_{i,1}$: π → t_i → $word_1$: joint prob so we make log to become SUM

if A['--s--', t_i] = 0 : $C_{i,1}$ = float('-inf')

else $C_{i,1}$ = log A['--s--', t_i] + log B[t_i, first_word_of_corpus]

$d_{i,1} = 0$

(2) Forward step (populate C, D matrix: column by column ≡ word by word)

```
for i in range (1, M): # From second word
    for j in range (N):
        best_ci = float('-inf'); best_di = None
        for k in range (N):
            c = C_{k,i-1} + log A[k,j] + log B[j, word_i]
            if best_ci < c : best_ci = c ; best_di = k
    C_{j,i} = best_ci ; d_{j,i} = best_di
```

(3) Back word path

C =

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

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> w_1 w_2 w_3 w_4 w_5

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

- Step 1: $s = \arg\max(C_{i,M})$
- Step 2: look for last column of D with s = 3 ⇒ it is t_1.
- Step 3: at D move if back word.