



deeplearning.ai

Overview

Learning objectives

- What is autocorrect?

Learning objectives

- What is autocorrect?
- Building the model

deah → dear 
yeah
dear
dean
... etc

Learning objectives

- What is autocorrect?
- Building the model
- Minimum edit distance

deah → dear ✓
yeah
dear
dean
... etc

#	#	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

Learning objectives

- What is autocorrect?
- Building the model
- Minimum edit distance
- Minimum edit distance algorithm

deah → dear ✓
yeah
dear
dean
... etc

	#	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

You use auto-correct everyday. When you send your friend a text message, or when you make a mistake in a query, there is an autocorrect behind the scenes that corrects the sentence for you. This week you are also going to learn about minimum edit distance, which tells you the minimum amount of edits to change one word into another. In doing that, you will learn about dynamic programming which is an important programming concept which frequently comes up in interviews and could be used to solve a lot of optimization problems.

- What is autocorrect?
- Building the model
- Minimum edit distance
- Minimum edit distance algorithm

deah → dear ✓

yeah
dear
dean
... etc

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



deeplearning.ai

Autocorrect

What is autocorrect?



What is autocorrect?

- Phones



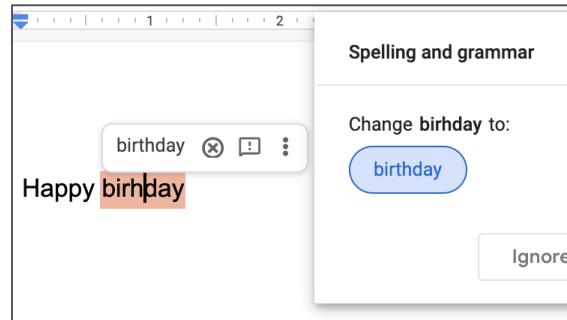
What is autocorrect?

- Phones
- Tablets



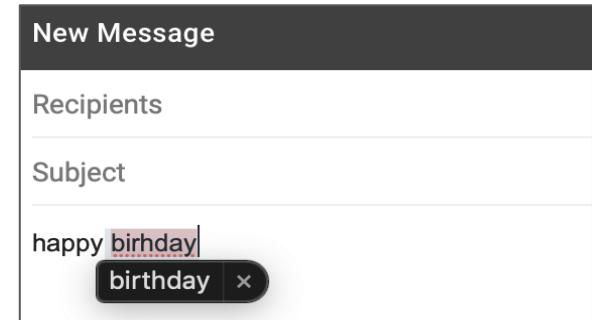
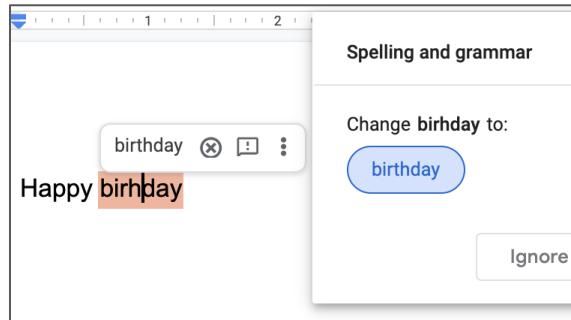
What is autocorrect?

- Phones
- Tablets
- Computers



What is autocorrect?

- Phones
- Tablets
- Computers



What is autocorrect?

- Example:

Happy birthday deah friend!



What is autocorrect ?

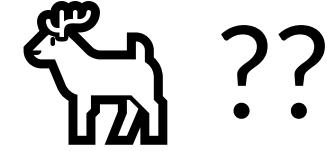
- Example:

Happy birthday   dear friend! 

What is autocorrect?

- Example:

Happy birthday deer friend!



How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah

How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah
_eah
d_ar
de_r
... etc

How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah
yeah
dear
dean
... etc

How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah
yeah
dear
dean
... etc

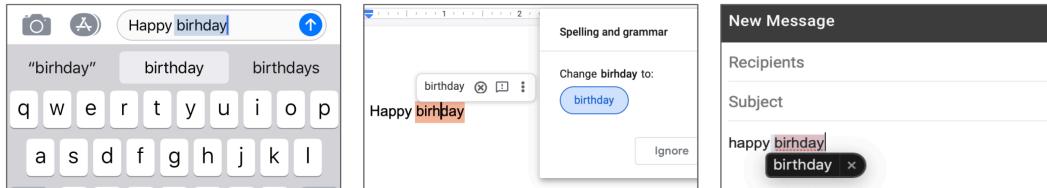
How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah → dear ✓
yeah
dear
dean
... etc

Autocorrects are used everywhere. You use them in your phones, tablets, and computers.

- Phones
- Tablets
- Computers



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)



deeplearning.ai

Building the model

Building the model

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

Building the model

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

Building the model

1. Identify a misspelled word

```
if word not in vocab:  
    misspelled = True
```

deah ??



Building the model

1. Identify a misspelled word

```
if word not in vocab:  
    misspelled = True
```

deah

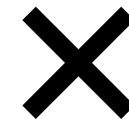


Building the model

1. Identify a misspelled word

```
if word not in vocab:  
    misspelled = True
```

deah



deer

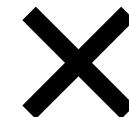


Building the model

1. Identify a misspelled word

```
if word not in vocab:  
    misspelled = True
```

deah



Happy birthday deer !



Building the model

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

Building the model

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' 'ate' X
• Replace (change 1 letter to another)	'jaw': 'jar', 'paw', ...

Building the model

2. Find strings n edit distance away

- Given a string find all possible strings that are n edit distance away using
 - Input
 - Delete
 - Switch
 - Replace

deah
 ~~eah~~
 d_ar
 de_r
 ... etc

Building the model

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

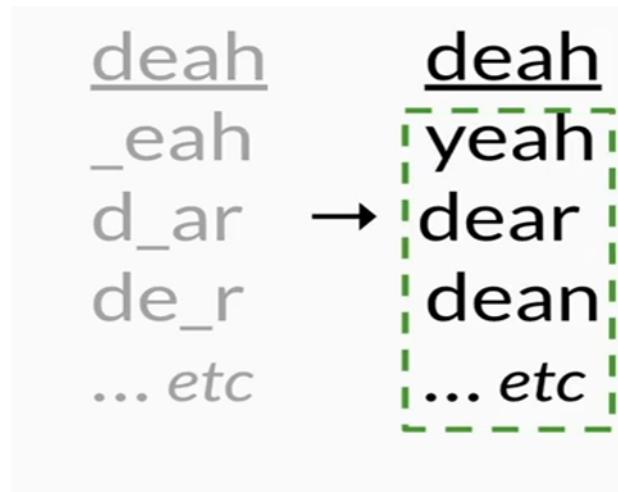
Building the model

3. Filter candidates

deah
_eah
d_ar
de_r
... *etc*

Building the model

3. Filter candidates



1. Identify the misspelled word

When identifying the misspelled word, you can check whether it is in the vocabulary. If you don't find it, then it is probably a typo.

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

In this step, you want to take all the words generated above and then only keep the actual words that make sense and that you can find in your vocabulary.

<u>deah</u>	<u>deah</u>
_eah	yeah
d_ar	→ dear
de_r	dean
... etc	... etc



deeplearning.ai

Building the model II

Building the model

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

Building the model

4. Calculate word probabilities

Example: “I am happy because I am learning”

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

Total : 7

Building the model

4. Calculate word probabilities

Example: "I am happy because I am learning"

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

Total : 7

Building the model

4. Calculate word probabilities

Example: "I **am** happy because **am** learning"

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

Total : 7

Building the model

4. Calculate word probabilities

Example: “I am happy because I am learning”

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

Total : 7

Building the model

4. Calculate word probabilities

Example: “I am happy because I am learning”

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

$P(w)$ 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

Building the model

4. Calculate word probabilities

Example: “I am happy because I am learning”

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

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

$P(w)$ 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

Building the model

4. Calculate word probabilities

deah
yeah
dear
dean
... etc

Building the model

4. Calculate word probabilities

deah → dear
yeah
dear
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear
yeah
dear
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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



Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear
yeah
dear
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert
Delete
Switch
Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear
_eah
d_ar
de_r
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear
yeah
dear
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear
yeah
dear
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear ✓
yeah
| dear |
dean
... etc

Summary

1. Identify a misspelled word
2. Find strings n edit distance away

Insert

Delete

Switch

Replace

1. Filter candidates
2. Calculate word probabilities

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

deah → dear 
yeah,
dear

dean
... etc

4.0 Calculating word probabilities

Calculate word probabilities

Example: "I am happy because I am learning"

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

$P(w)$ Probability of a word

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

V Total size of the corpus

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

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

Total : 7

Note that you are storing the count of words and then you can use that to generate the probabilities. For this week, you will be counting the probabilities of words occurring. If you want to build a slightly more sophisticated auto-correct you can keep track of two words occurring next to each other instead. You can then use the previous word to decide. For example which combo is more likely, *there friend* or *their friend*? For this week however you will be implementing the probabilities by just using the word frequencies. Here is a summary of everything you have seen before in the previous two videos.

1. Identify a misspelled word
2. Find strings n edit distance away
 - Insert
 - Delete
 - Switch
 - Replace
3. Filter candidates
4. Calculate word probabilities

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

deah → dear

yeah

dear

dean

... etc



deeplearning.ai

Minimum edit distance

Minimum edit distance

- How to evaluate similarity between 2 strings?
- Minimum number of edits needed to transform 1 string into the other
- Spelling correction, document similarity, machine translation, DNA sequencing, and more

Minimum edit distance

- Edits:
- Insert (add a letter) ‘to’: ‘top’, ‘two’ ...
- Delete (remove a letter) ‘hat’: ‘ha’, ‘at’, ‘ht’
- Replace (change 1 letter to another) ‘jaw’: ‘jar’, ‘paw’, ...

Minimum edit distance

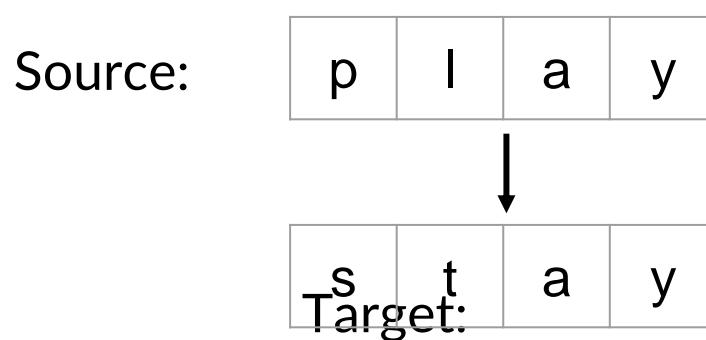
- Example:

Source:

p		a	y
---	--	---	---

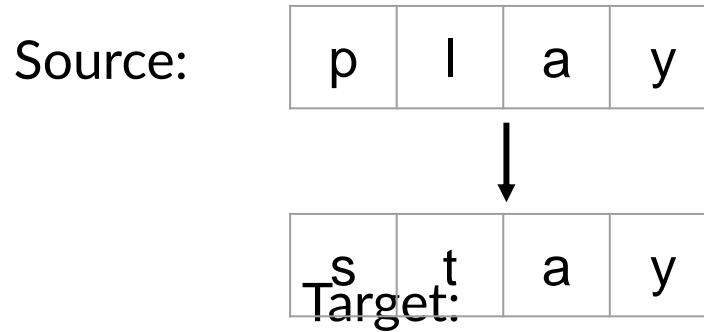
Minimum edit distance

- Example:



Minimum edit distance

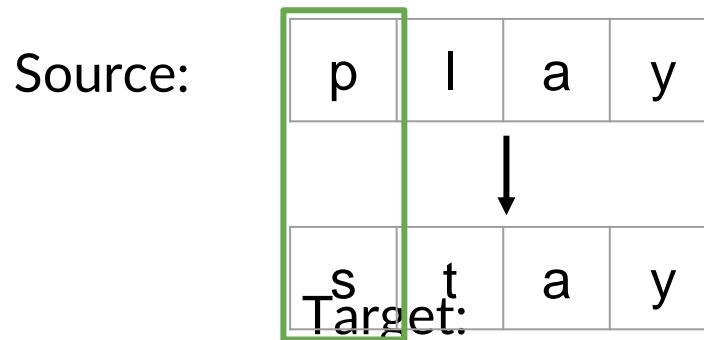
- Example:



What is the minimum number of edits to make this happen ?

Minimum edit distance

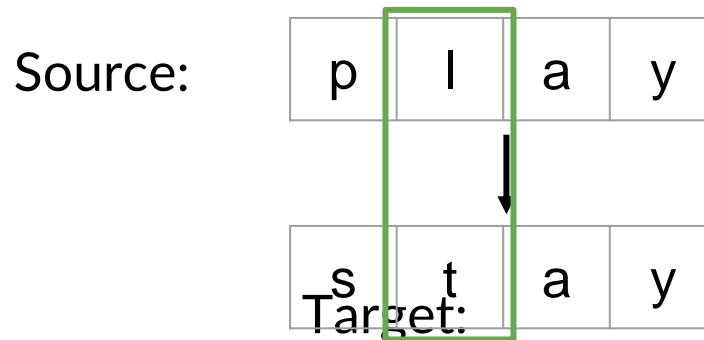
- Example:



$p \rightarrow s$: replace

Minimum edit distance

- Example:



$p \rightarrow s$: replace
 $i \rightarrow t$: replace

Minimum edit distance

- Example:

Source:

p		a	y
---	--	---	---



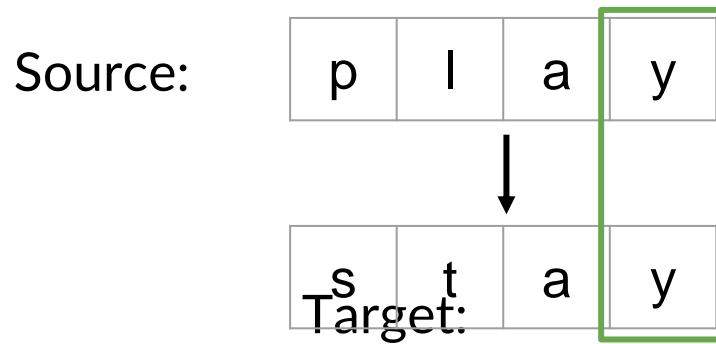
s		a	y
---	--	---	---

Target:

$p \rightarrow s$: replace
 $| \rightarrow t$: replace

Minimum edit distance

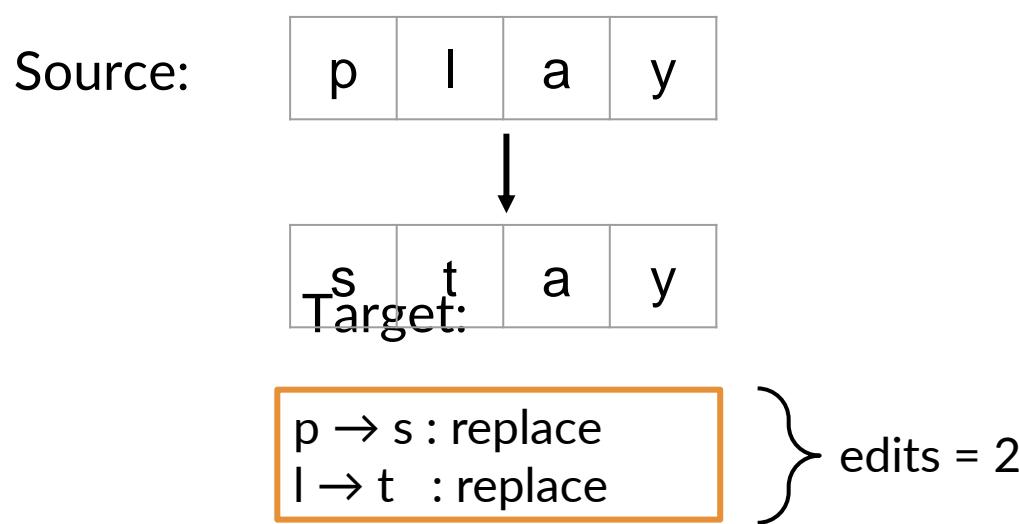
- Example:



$p \rightarrow s$: replace
 $i \rightarrow t$: replace

Minimum edit distance

- Example:



Minimum edit distance

- Example:

Source:

p		a	y
---	--	---	---



s		t	a	y
---	--	---	---	---

Target:

Edit cost:

Insert 1

Delete 1

Replace 2

$p \rightarrow s$: replace
 $| \rightarrow t$: replace

} edits = 2

Minimum edit distance

- Example:

Source:

p		a	y
---	--	---	---



s		a	y
---	--	---	---

Target:

$p \rightarrow s$: replace
 $| \rightarrow t$: replace

Edit cost:

Insert 1

Delete 1

Replace 2

$$\text{edit distance} = 2 * 2 = 4$$

} edits = 2

Minimum edit distance

- Example:

c	o	n	v	o	l	u	t	i	o	n	a	l	n	e	u	r	a	l	n	e	t	w	o	r	k
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Minimum edit distance

- Example:

c	o	n	v	o	l	u	t	i	o	n	a		n	e	u	r	a		n	e	t	w	o	r	k
---	---	---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---	---

CCAAGGGGTGACTCTAGTTAACATAACTGAGATCAAATTATGGGTGAT ? !!

Minimum edit distance allows you to:

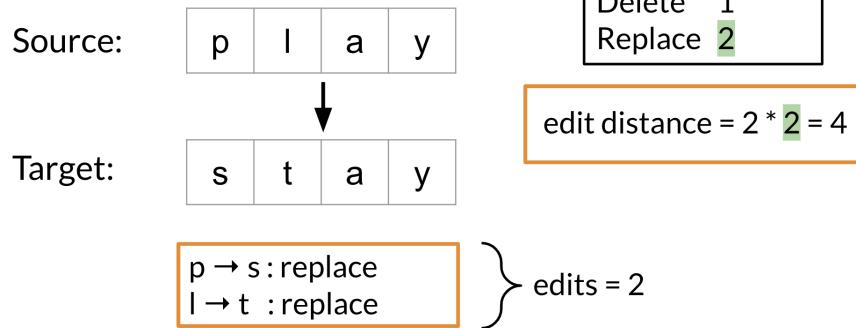
- Evaluate similarity between two strings
- Find the minimum number of edits between two strings
- Implement spelling correction, document similarity, machine translation, DNA sequencing, and more

Remember that the edits include :

- Insert (add a letter) ‘to’: ‘top’, ‘two’ ...
- Delete (remove a letter) ‘hat’: ‘ha’, ‘at’, ‘ht’
- Replace (change 1 letter to another) ‘jaw’: ‘jar’, ‘paw’, ...

Here is a concrete example where we calculate the cost (i.e. edit distance) between two strings.

Example:



Note that as your strings get larger it gets much harder to calculate the minimum edit distance. Hence you will now learn about the minimum edit distance algorithm!



deeplearning.ai

Minimum edit distance algorithm

Minimum edit distance

Source: play → Target: stay

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

Minimum edit distance

Source: play → Target: stay

	0	1	2	3	4
0	#				
1	p				
2	l				
3	a				
4	y				

Minimum edit distance

Source: play → Target: stay

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

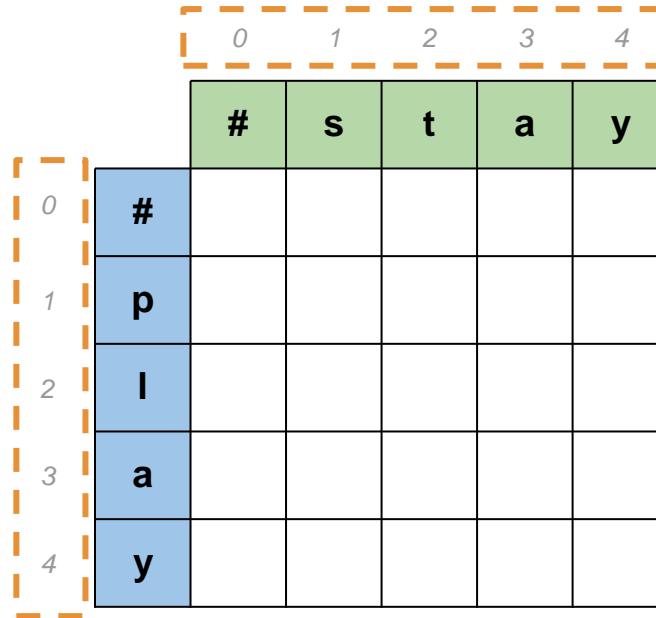
Minimum edit distance

Source: play → Target: stay

		0	1	2	3	4
		#	s	t	a	y
0	#					
	p					
	I					
	a					
	y					

Minimum edit distance

Source: play → Target: stay



Minimum edit distance

Source: play → Target: stay

$D[]$

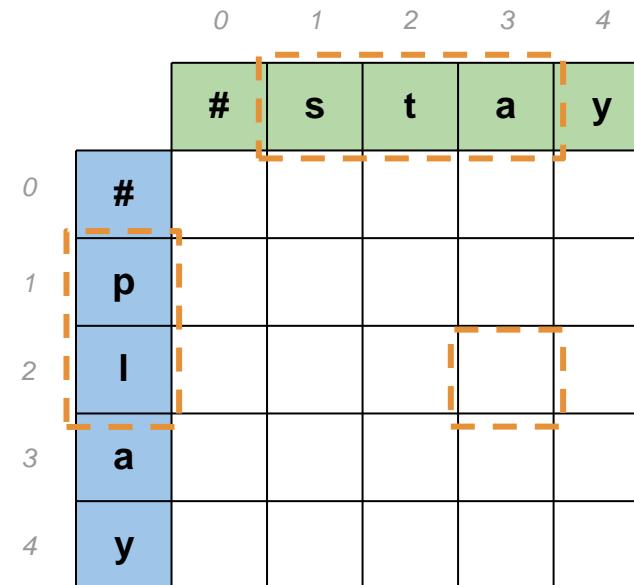
	0	1	2	3	4
0	#	s	t	a	y
1	p				
2	l				
3	a				
4	y				

Minimum edit distance

Source: play → Target: stay

$D[]$

$D[2,3] = \text{pl} \rightarrow \text{sta}$



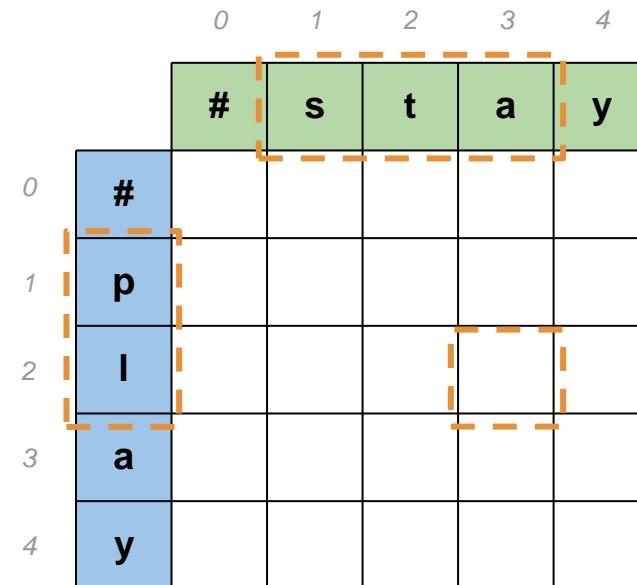
Minimum edit distance

Source: play → Target: stay

$D[]$

$D[2,3] = \text{pl} \rightarrow \text{sta}$

$D[2,3] = \text{source}[:2] \rightarrow \text{target}[3]$



Minimum edit distance

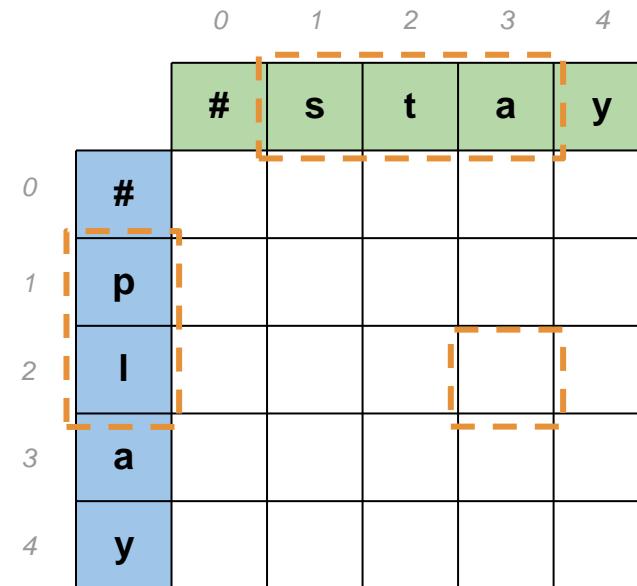
Source: play → Target: stay

$D[]$

$D[2,3] = \text{pl} \rightarrow \text{sta}$

$D[2,3] = \text{source}[2:] \rightarrow \text{target}[3:]$

$D[i, j] = \text{source}[: i] \rightarrow \text{target}[: j]$



Minimum edit distance

Source: play → Target: stay

$D[]$

$D[i, j] = \text{source}[:i] \rightarrow \text{target}[:j]$

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

Minimum edit distance

Source: play → Target: stay

$D[]$

$D[i, j] = \text{source}[:i] \rightarrow \text{target}[:j]$

$D[m, n] = \text{source} \rightarrow \text{target}$

		0	1	2	3	4
		#	s	t	a	y
0	#					
1	p					
2	i					
3	a					
4	y					

A grid diagram illustrating the minimum edit distance between the source string "play" and the target string "stay". The columns represent the target string "stay" with indices 0 to 4. The rows represent the source string "play" with indices 0 to 4. The first row contains the symbol "#". The first column contains the symbol "#". The grid cells are colored green for matches ('s' at index 1), blue for insertions ('#', 'p', 'i', 'a' in the first column), and white for deletions ('y' at index 4). A dashed orange rectangle highlights the path from the bottom-right cell to the top-left cell, representing the sequence of edits: insertion of 's', insertion of 't', insertion of 'a', and insertion of 'y'. The path starts at index (4,4) and ends at index (0,0).

Minimum edit distance

Source: play → Target: stay

$D[]$

$D[i, j] = \text{source}[:i] \rightarrow \text{target}[:j]$

$D[m, n] = \text{source} \rightarrow \text{target}$

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

An orange arrow starts at the cell (1, 0) and points to the cell (2, 1), indicating a substitution edit from 'p' to 's'.

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

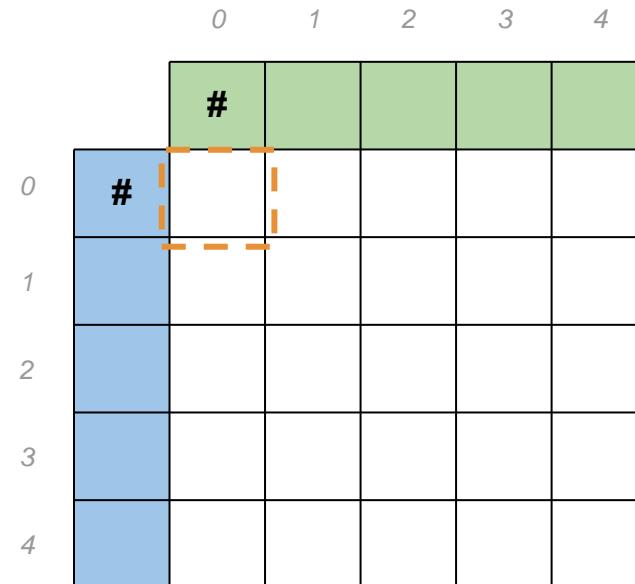
		0	1	2	3	4
		#	s	t	a	y
0	#					
1	p					
2	l					
3	a					
4	y					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→

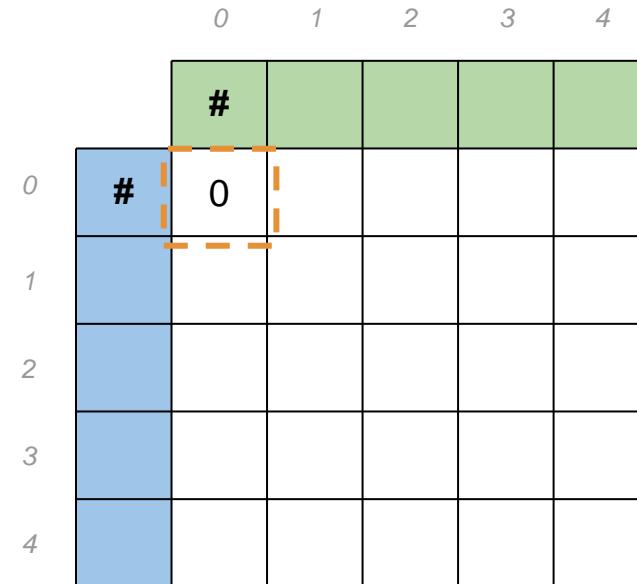


Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→

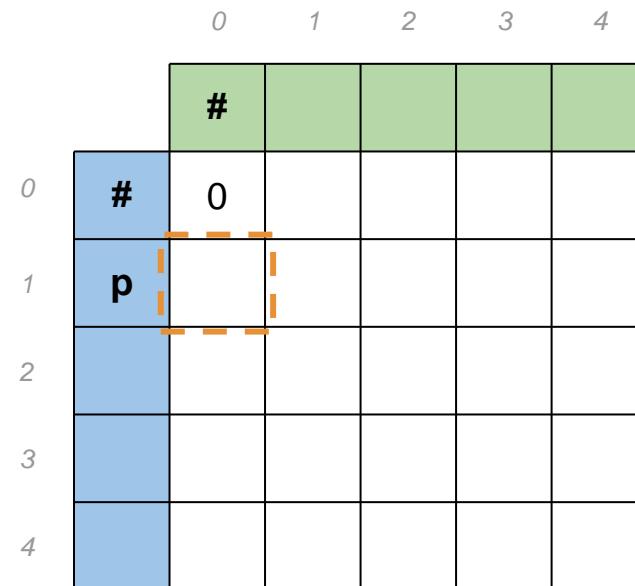


Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow \#$



Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow \#$

delete

		0	1	2	3	4
		#				
		0				
#		0				
p		1				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→ s

		0	1	2	3	4
		#	s			
		0				
#	play	0				
p	1					
2						
3						
4						

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→ s

insert

		0	1	2	3	4
		#	s			
		0				
#	play	0	1			
p	1					
2						
3						
4						

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

insert+**delete**: $p \rightarrow ps \rightarrow s$: 2

delete+**insert**: $p \rightarrow \# \rightarrow s$: 2

replace: $p \rightarrow s$: 2

		0	1	2	3	4
		#	s			
#	0	0	1			
	1	1	2			
2						
3						
4						

When computing the minimum edit distance, you would start with a *source word* and transform it into the *target word*. Let's look at the following example:

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

- insert+delete: $p \rightarrow ps \rightarrow s: 2$
- delete+insert: $p \rightarrow \# \rightarrow s: 2$
- replace: $p \rightarrow s: 2$

	0	1	2	3	4
0	#	0	1		
1	p	1	2		
2					
3					
4					

To go from $\# \rightarrow \#$ you need a cost of 0. From $p \rightarrow \#$ you get 1, because that is the cost of a delete. $p \rightarrow s$ is 2 because that is the minimum cost one could use to get from **p** to **s**. You can keep going this way by populating one element at a time, but it turns out there is a faster way to do this. You will learn about it next.



deeplearning.ai

Minimum edit distance algorithm II

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1			
1	p	1	2			
2	I					
3	a					
4	y					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → #

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1			
1	p	1	2			
2	I					
3	a					
4	y					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → #

$$D[i, j] = D[i-1, j] + \text{del_cost}$$

		0	1	2	3	4
		#	s	t	a	y
#	0	0	1			
	1	1	2			
	2					
	3					
	4					
	5					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → #

$$D[i, j] = D[i-1, j] + \text{del_cost}$$

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1			
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → #

$$D[i, j] = D[i-1, j] + \text{del_cost}$$

$$\begin{aligned} D[4,0] &= \text{play} \rightarrow \# \\ &= \text{source[:4]} \rightarrow \text{target}[0] \end{aligned}$$

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1			
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

A grid diagram illustrating the minimum edit distance between "play" and "stay". The columns represent the target string "stay" with indices 0 to 4. The rows represent the source string "play" with indices 0 to 4. The first row and column are labeled with their respective characters. The cost matrix shows the minimum cost to transform the source prefix up to index i into the target prefix up to index j. The diagonal from (0,0) to (4,4) represents the cost of transforming "play" into "stay". A dashed orange line highlights the path from (0,0) to (4,4), indicating the sequence of edits: insertion of 's' at index 1, insertion of 't' at index 2, insertion of 'a' at index 3, and insertion of 'y' at index 4.

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→ play

		0	1	2	3	4
		#	s	t	a	y
#	0	0	1			
	1	1	2			
	2	2				
	3	3				
	4	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→ play

$$D[i, j] = D[i, j-1] + \text{ins_cost}$$

		0	1	2	3	4
		#	s	t	a	y
#	#	0	1			
	p	1	2			
	I	2				
	a	3				
	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

→ play

$$D[i, j] = D[i, j-1] + \text{ins_cost}$$

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i, j] = \min \begin{cases} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{cases}$$

		0	1	2	3	4
		#	s	t	a	y
#	0	0	1	2	3	4
	p	1	2			
I	2					
a	3					
y	4					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$D[i, j] =$

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

	0	1	2	3	4
#	0	1	2	3	4
p	1	2			
I	2				
a	3				
y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

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

	0	1	2	3	4	
0	#	0	1	2	3	4
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i, j] = \min \left\{ \begin{array}{l} D[i - 1, j] + \text{del cost} \\ D[i, j - 1] + \text{ins cost} \\ D[i - 1, j - 1] + \left\{ \begin{array}{l} \text{rep cost; if } \text{src}[i] \neq \\ \text{tar}[j] \end{array} \right. \end{array} \right. \begin{array}{l} \\ \\ 0; \quad \quad \quad \text{if } \text{src}[i] \\ \quad \quad \quad = \text{tar}[j] \end{array}$$

	0	1	2	3	4
#	0	1	2	3	4
p	1	2			
I	2				
a	3				
y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$D[i, j] = \min_{\substack{\text{tar}[j] \\ D[i-1, j] + \text{del cost} \\ D[i, j-1] + \text{ins cost} \\ D[i-1, j-1] + \text{rep cost; if } \text{src}[i] \neq \text{tar}[j]}} 0; \quad \text{if } \text{src}[i] = \text{tar}[j]$

	0	1	2	3	4	
#	#	s	t	a	y	
0	#	0	1	2	3	4
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$D[i, j] =$

$$\min \left\{ \begin{array}{l} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{array} \right.$$

dev purposes only
image of how previous slide should be
appearing for everyone !

	#	s	t	a	y
#	0	1	2	3	4
p	1	2			
I	2				
a	3				
y	4				

Don't include text or images below this line. Delete this text and red line in the master template once you're finished with your slide creation

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

FORMULAS BUILDING ONLY
EQUATION USED IN NEXT SLIDES

$$D[i, j] = \min \begin{cases} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{cases}$$

$$D[i, j] = \min \begin{cases} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{cases}$$

	0	1	2	3	4	
0	#	0	1	2	3	4
1	p	1				
2	I	2				
3	a	3				
4	y	4				

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i-1, j] + 1 = 2$$

$$D[i, j-1] + 1 = 2$$

$$D[i-1, j-1] + 2 = 2$$

		0	1	2	3	4
		#	s	t	a	y
#	0	1				
	p	1	2			
I	2					
a	3					
y	4					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i-1, j] + 1 = 2$$

$$D[i, j-1] + 1 = 2$$

$$D[i-1, j-1] + 2 = 2$$

		0	1	2	3	4
		#	s	t	a	y
#	0	1				
	p	1	2			
I	2					
a	3					
y	4					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i-1, j] + 1 = 2$$

$$D[i, j-1] + 1 = 2$$

$$D[i-1, j-1] + 2 = 2$$

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

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$D[i-1, j] + 1 = 2$$

$$D[i, j-1] + 1 = 2$$

$$D[i-1, j-1] + 2 = 2$$

		0	1	2	3	4
		#	s	t	a	y
#	0	0	1	2	3	4
	p	1	2			
I	2					
a	3					
y	4					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

$p \rightarrow s$

$$\begin{aligned} D[i-1, j] + 1 &= 2 \\ D[i, j-1] + 1 &= 2 \\ = 2 & \\ D[i-1, j-1] + 2 &= 2 \end{aligned}$$

min

		0	1	2	3	4
		#	s	t	a	y
#	0	0	1	2	3	4
	p	1	2			
I	2					
a	3					
y	4					

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

		0	1	2	3	4
		#	s	t	a	y
#	0	1	2	3	4	
p	1	2				
I	2					
a	3					
y	4					

A dynamic programming table for minimum edit distance. The rows represent the source string "play" and the columns represent the target string "stay". The first row and column are headers. The cost of each edit operation is: insertion = 1, deletion = 1, replacement = 2. Dashed orange lines highlight the path from the start to the end of the target string.

Minimum edit distance

Source: to → Target: go

Cost: insert: 1, delete: 1, replace: 2

FOR QUIZ SETUP ONLY
... USED FOR IMAGES
ON QUIZ IN NEXT SLIDE

$$D[i, j] = \min \begin{cases} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{cases}$$

	0	1	2	
0	#	0	1	2
1	t	1	2	3
2	o	2	3	

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

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

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → stay

$$D[m, n] = 4$$

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

The diagram shows a 5x6 grid representing the edit distance between the source string "play" and the target string "stay". The rows are indexed from 0 to 4, and the columns are indexed from 0 to 5. The first column and row are labeled with the characters "#", "p", "I", "a", and "y" respectively. The grid contains numerical values representing the cost of edits. A dashed red line highlights a path from the bottom-right cell (4, 4) back towards the top-left cell (1, 0), illustrating a sequence of edits: a replace operation from 'y' to 's', followed by a series of insertions and replacements that lead to the final state.

Minimum edit distance

color coding or heat map

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → stay

$$D[m, n] = 4$$

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

color coding or heat map

 deeplearning.ai

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

play → stay

$$D[m, n] = 4$$

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

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

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

To populate the following table:

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

$p \rightarrow s$	
$D[i, j] =$	
$\min \left\{ \begin{array}{l} D[i - 1, j] + \text{del_cost} \\ D[i, j - 1] + \text{ins_cost} \\ D[i - 1, j - 1] + \begin{cases} \text{rep_cost}; & \text{if } \text{src}[i] \neq \text{tar}[j] \\ 0; & \text{if } \text{src}[i] = \text{tar}[j] \end{cases} \end{array} \right\}$	

		0	1	2	3	4
0	#	0	1	t	a	y
1	p	1	2			
2	I	2				
3	a	3				
4	y	4				

There are three equations:

- $D[i,j] = D[i-1, j] + \text{del_cost}$: this indicates you want to populate the current cell (i,j) by using the cost in the cell found directly above.
- $D[i,j] = D[i, j-1] + \text{ins_cost}$: this indicates you want to populate the current cell (i,j) by using the cost in the cell found directly to its left.
- $D[i,j] = D[i-1, j-1] + \text{rep_cost}$: the rep cost can be 2 or 0 depending if you are going to actually replace it or not.

At every time step you check the three possible paths where you can come from and you select the least expensive one. Once you are done, you get the following:

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

		0	1	2	3	4
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	I	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4



deeplearning.ai

Minimum edit distance algorithm III

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance

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

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance
- Backtrace

	0	1	2	3	4
#	0	1	2	3	4
p	1	2	3	4	5
I	2	3	4	5	6
a	3	4	5	4	5
y	4	5	6	5	4

Minimum edit distance

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance
- Backtrace
- Dynamic programming

	0	1	2	3	4
#	0	1	2	3	4
p	1	2	3	4	5
I	2	3	4	5	6
a	3	4	5	4	5
y	4	5	6	5	4

Source: play → Target: stay

Cost: insert: 1, delete: 1, replace: 2

- Levenshtein distance
- Backtrace
- Dynamic programming

	0	1	2	3	4	
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	I	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4

To summarize, you have seen the levenshtein distance which specifies the cost per operation. If you need to reconstruct the path of how you got from one string to the other, you can use a backtrace. You should keep a simple pointer in each cell letting you know where you came from to get there. So you know the path taken across the table from the top left corner, to the bottom right corner. You can then reconstruct it.

This method for computation instead of brute force is a technique known as dynamic programming. You first solve the smallest subproblem first and then reusing that result you solve the next biggest subproblem, saving that result, reusing it again, and so on. This is exactly what you did by populating each cell from the top right to the bottom left. It's a well-known technique in computer science!



deeplearning.ai

Summary

Summary - learning objectives

- What is autocorrect ?
- Building the model
- Minimum edit distance
- Minimum edit distance algorithm

deah → dear ✓
yeah
dear
dean
... etc

#	#	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



deeplearning.ai

Part of Speech Tagging

Outline

- What is part of speech tagging?
- Markov chains
- Hidden Markov models
- Viterbi algorithm
- Example
- Coding assignment!

What is part of speech?

Why not learn something ?

adverb adverb verb

noun

punctuation
mark,
sentence
closer

Part of speech (POS) tagging

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 .

Applications of POS tagging



Named entities

the eiffel tower located in
paris
eiffel tower, paris: named
entities



Co-reference resolution

the eiffel tower located in
paris. it is 324 meters high.
it: eiffel tower



Speech recognition

Part of Speech Tagging (POS) is the process of assigning a part of speech to a word. By doing so, you will learn the following:

- Markov Chains
- Hidden Markov Models
- Viterbi algorithm

Here is a concrete example:

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 .

You can use part of speech tagging for:

- Identifying named entities
- Speech recognition
- Coreference Resolution

You can use the probabilities of POS tags happening near one another to come up with the most reasonable output.



deeplearning.ai

Markov Chains

Example

Why not learn █ ...

verb verb?
noun?
... ?

Part of Speech Dependencies

Why not learn ...

verb verb?
 → noun?
 ... ?

The Most Likely Next Word

Why not learnswimming?

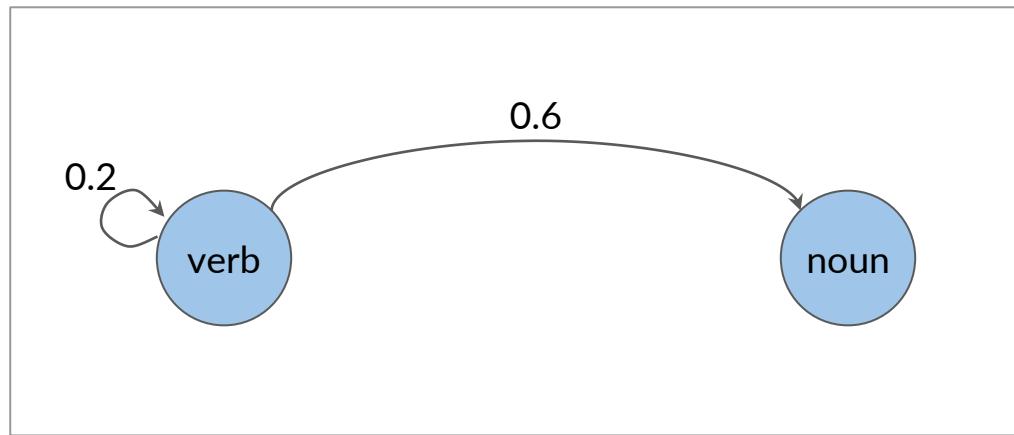
verb noun

Less Likely Words

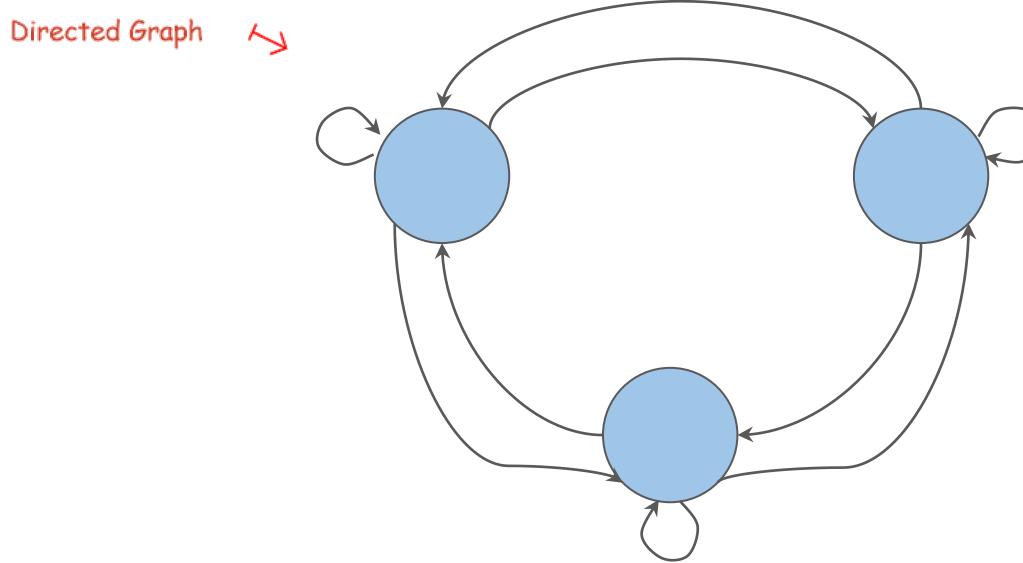
Why not learnswim?

verb verb

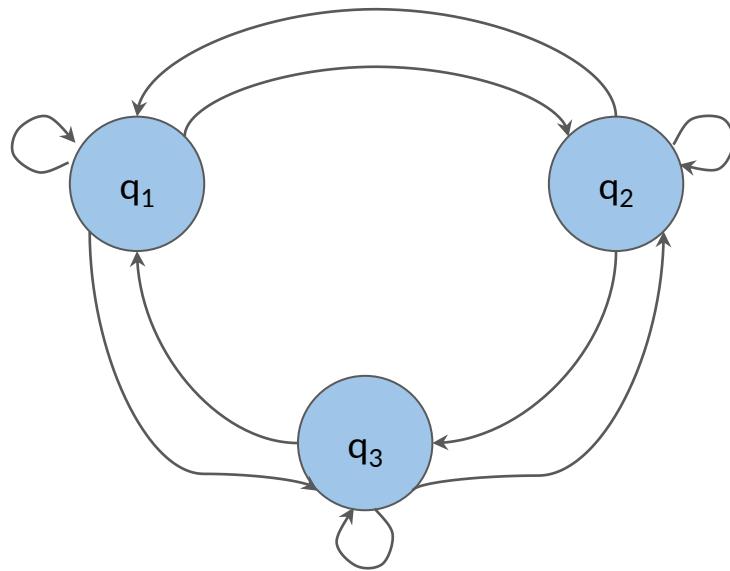
Visual Representation



What are Markov chains?



States



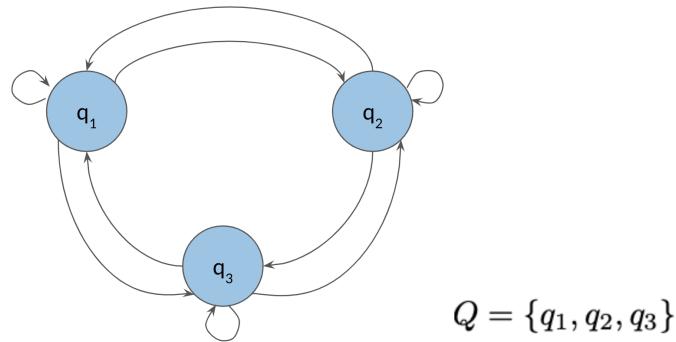
$$Q = \{q_1, q_2, q_3\}$$

You can use Markov chains to identify the probability of the next word. For example below, you can see that the most likely word after a verb is a noun.

Why not learn swimming?
verb noun

Why not learn swim?
verb verb

To properly model the probabilities we need to identify the probabilities of the POS tags and for the words.



The circles of the graph represent the states of your model. A **state** refers to a certain condition of the present moment. You can think of these as the POS tags of the current word.

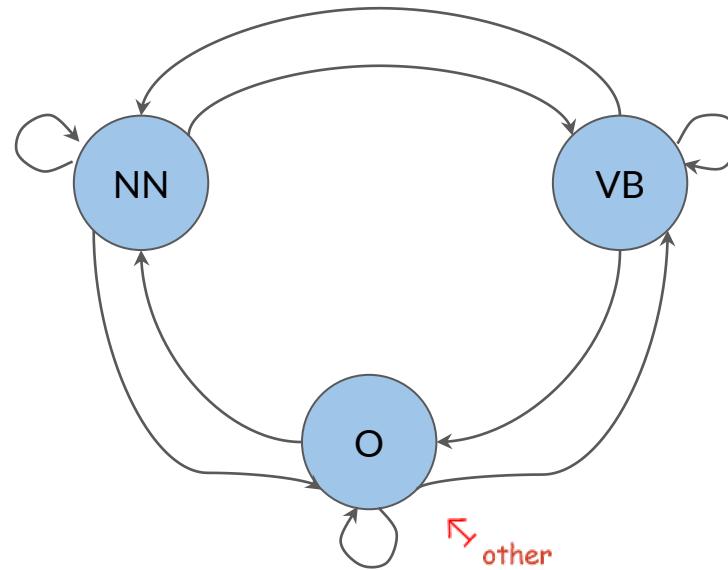
$Q = \{q_1, q_2, q_3\}$ is the set of all states in your model.



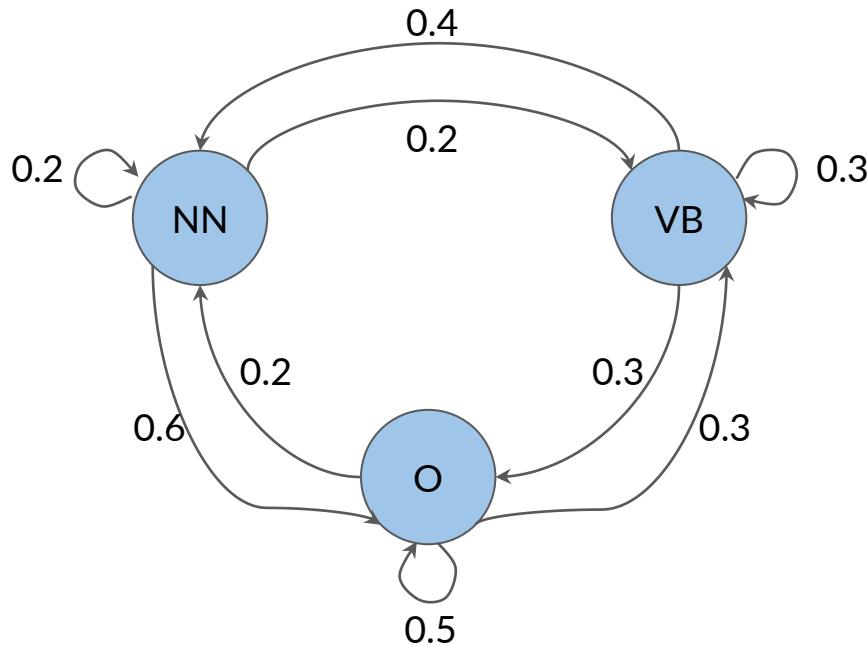
deeplearning.ai

Markov Chains and POS Tags

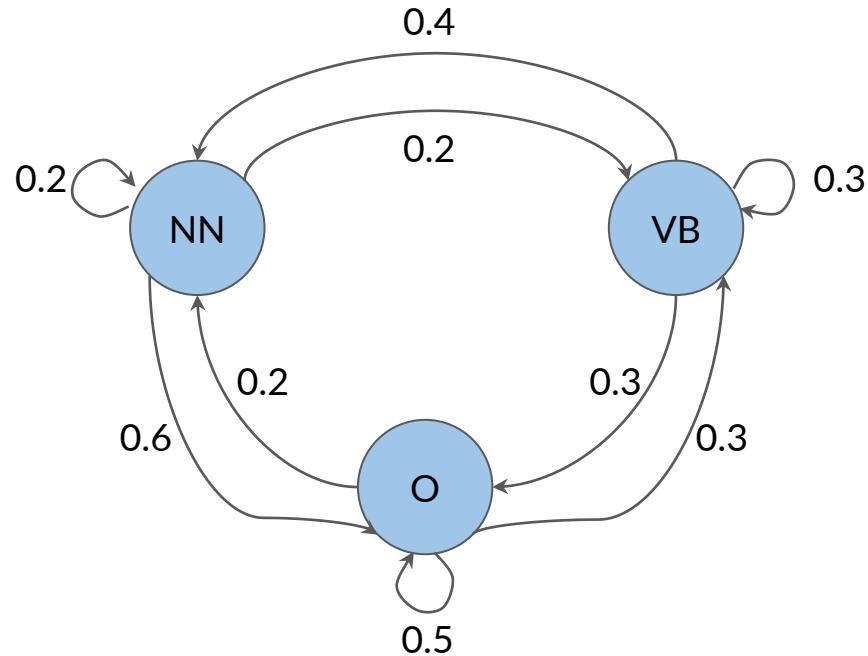
POS tags as States



Transition probabilities

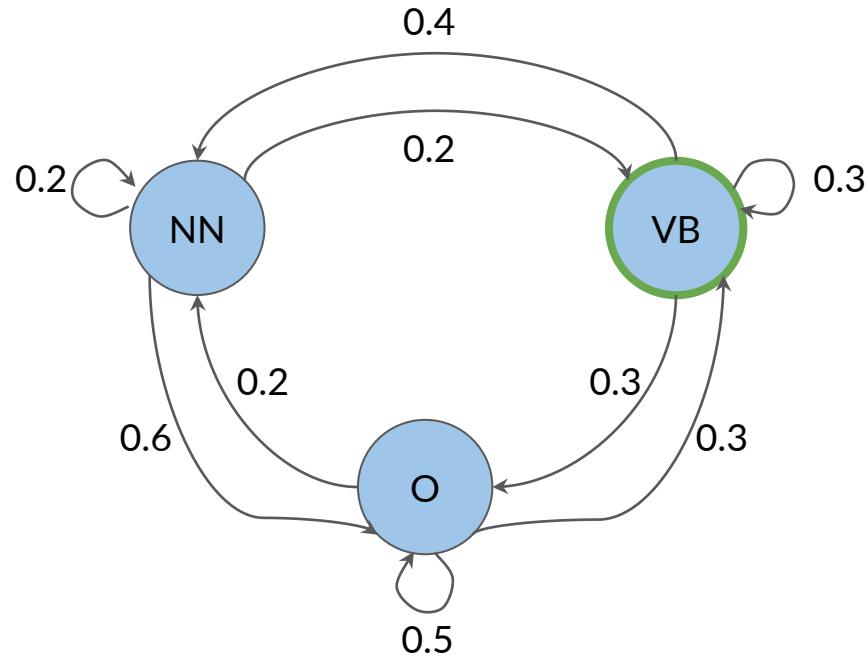


Transition probabilities



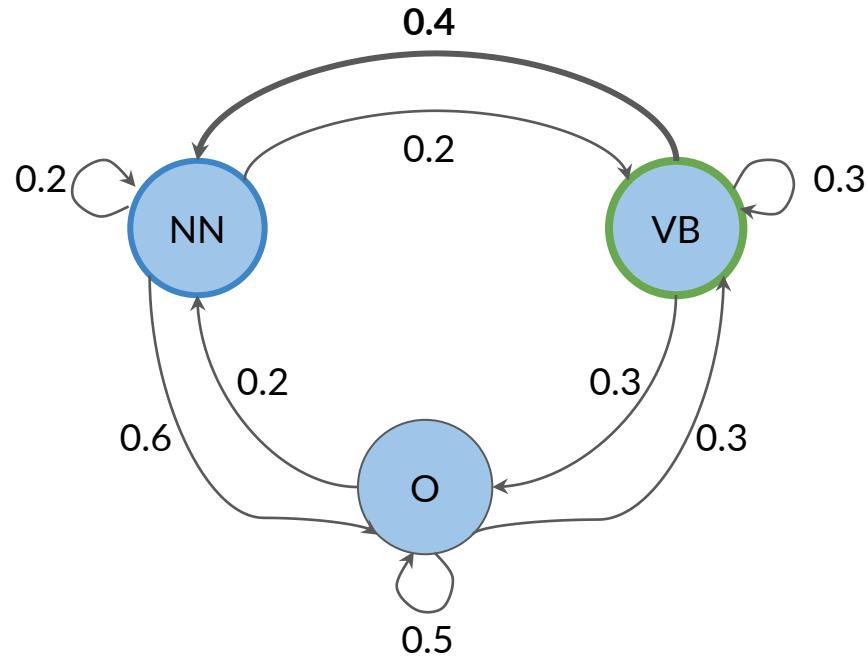
Why not **learn** something ?

Transition probabilities



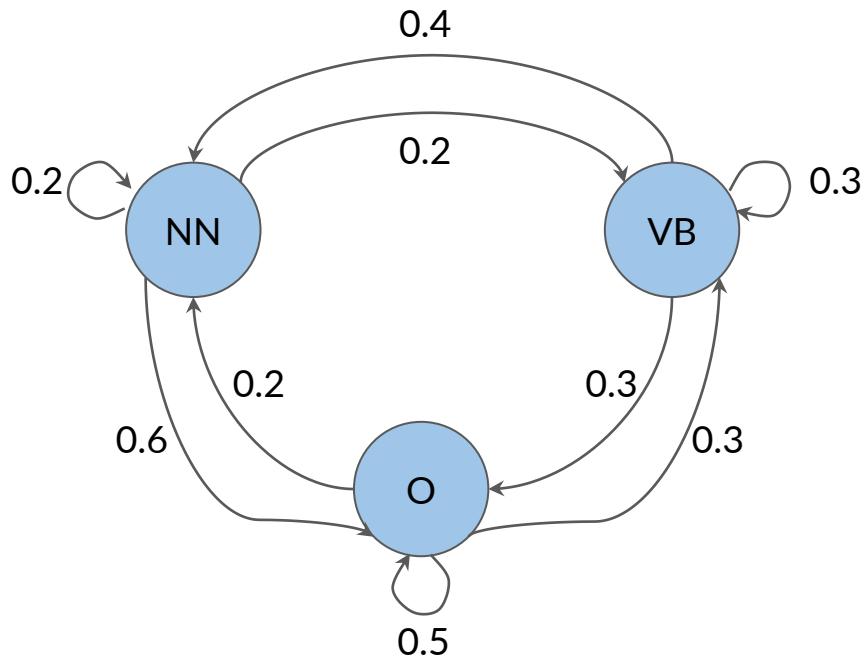
Why not **learn** something ?

Transition probabilities



Why not **learn** something ?

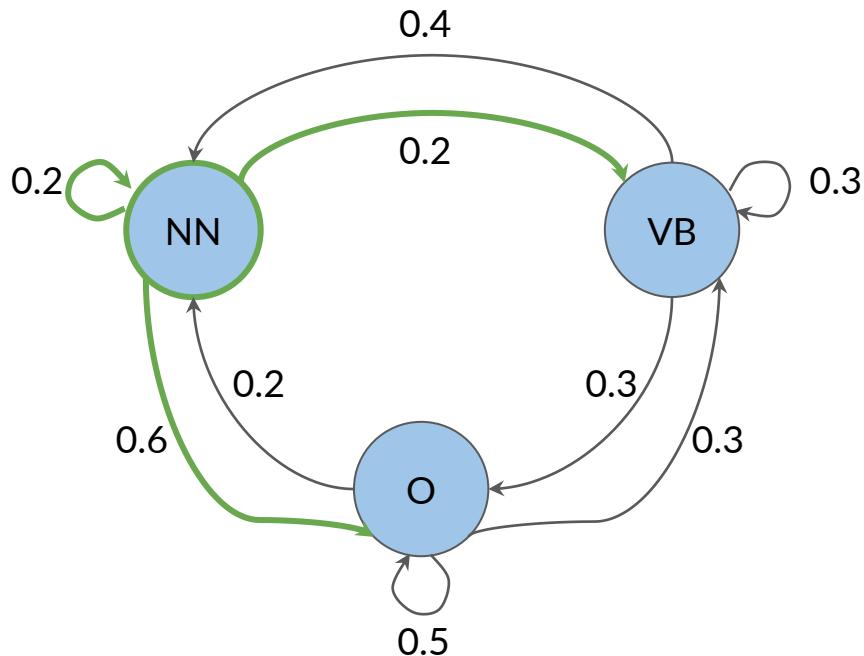
The transition matrix



$A =$

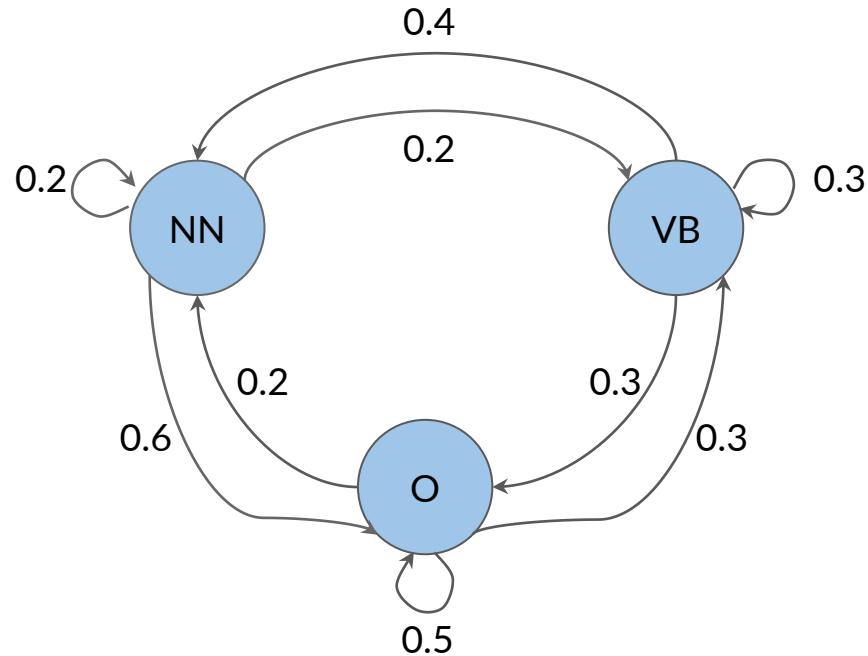
	NN	VB	O
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

The transition matrix


$$A = \begin{array}{c|ccc} & \text{NN} & \text{VB} & \text{O} \\ \hline \text{NN (noun)} & 0.2 & 0.2 & 0.6 \\ \text{VB (verb)} & 0.4 & 0.3 & 0.3 \\ \text{O (other)} & 0.2 & 0.3 & 0.5 \end{array}$$

$$\sum_{j=1}^N a_{ij} = 1$$

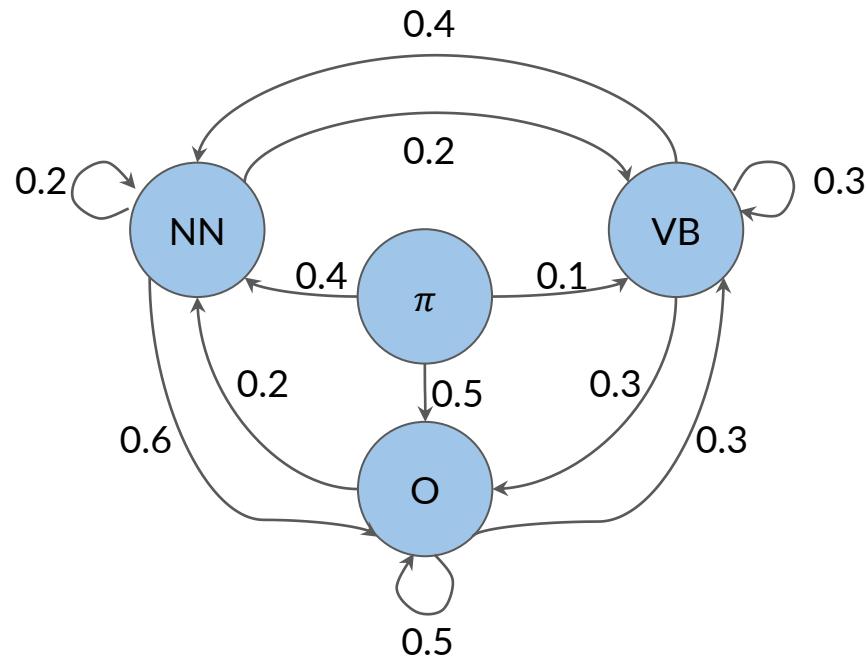
The first word



Why not learn something ?

NN?
VB?
O?

Initial probabilities



$A =$

	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

Transition table and 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

$$A = \begin{pmatrix} 0.4 & 0.1 & 0.5 \\ 0.2 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.3 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$

Summary

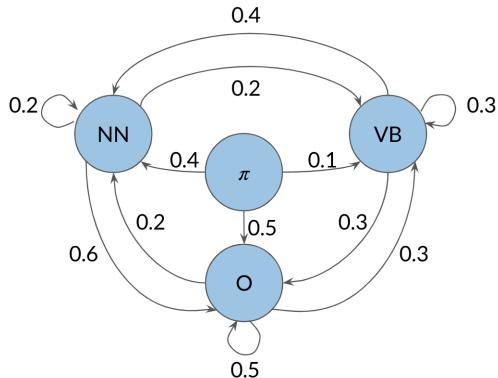
States

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

Transition matrix

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

To help identify the parts of speech for every word, you need to build a transition matrix that gives you the probabilities from one state to another.



	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

In the diagram above, the blue circles correspond to the part of speech tags, and the arrows correspond to the transition probabilities from one part of speech to another. You can populate the table on the right from the diagram on the left. The first row in your A matrix corresponds to the initial distribution among all the states. According to the table, the sentence has a 40% chance to start as a noun, 10% chance to start with a verb, and a 50% chance to start with another part of speech tag.

In more general notation, you can write the transition matrix A , given some states Q , as follows:

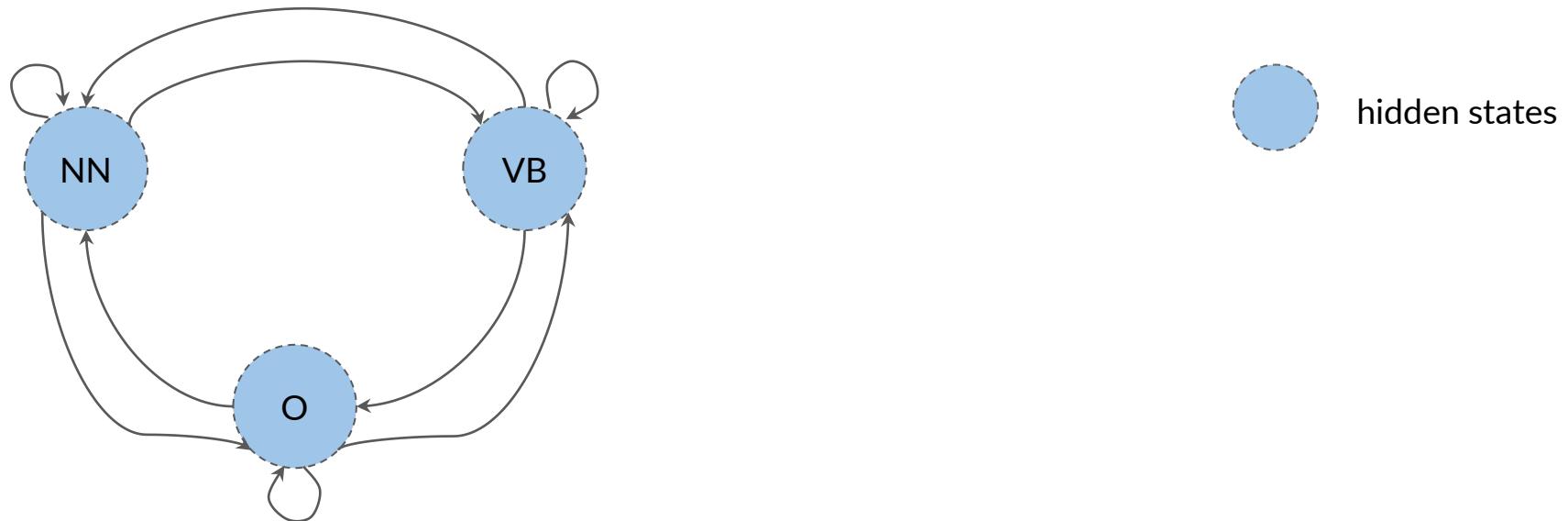
States	Transition matrix
$Q = \{q_1, \dots, q_N\}$	$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \dots & a_{N+1,N} \end{pmatrix}$



deeplearning.ai

Hidden Markov Models

Hidden Markov Model

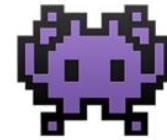


you



jump = verb

machine



jump = ?

you



**jump =
verb
run = verb
fly = verb**

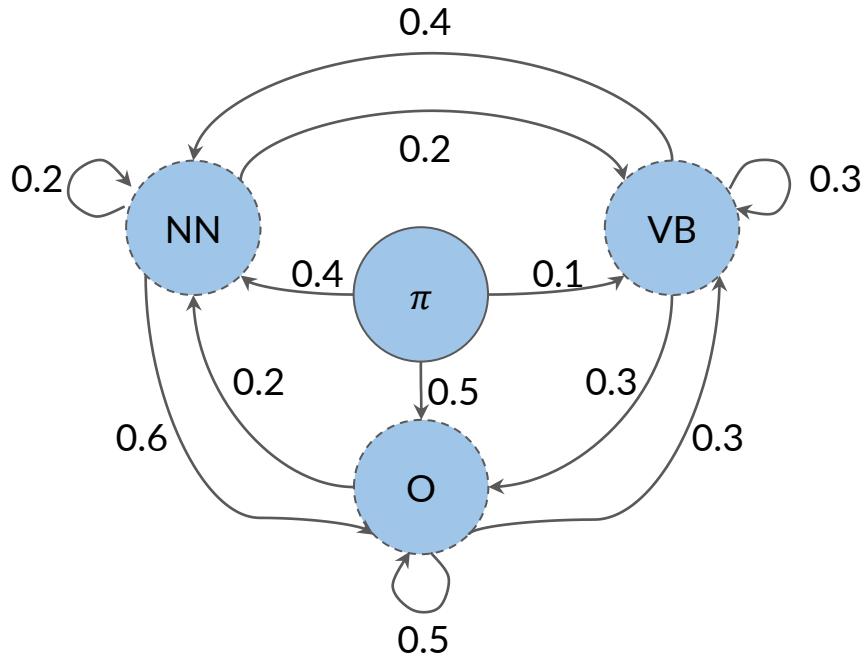
machine



**jump *
run
fly**

*observable

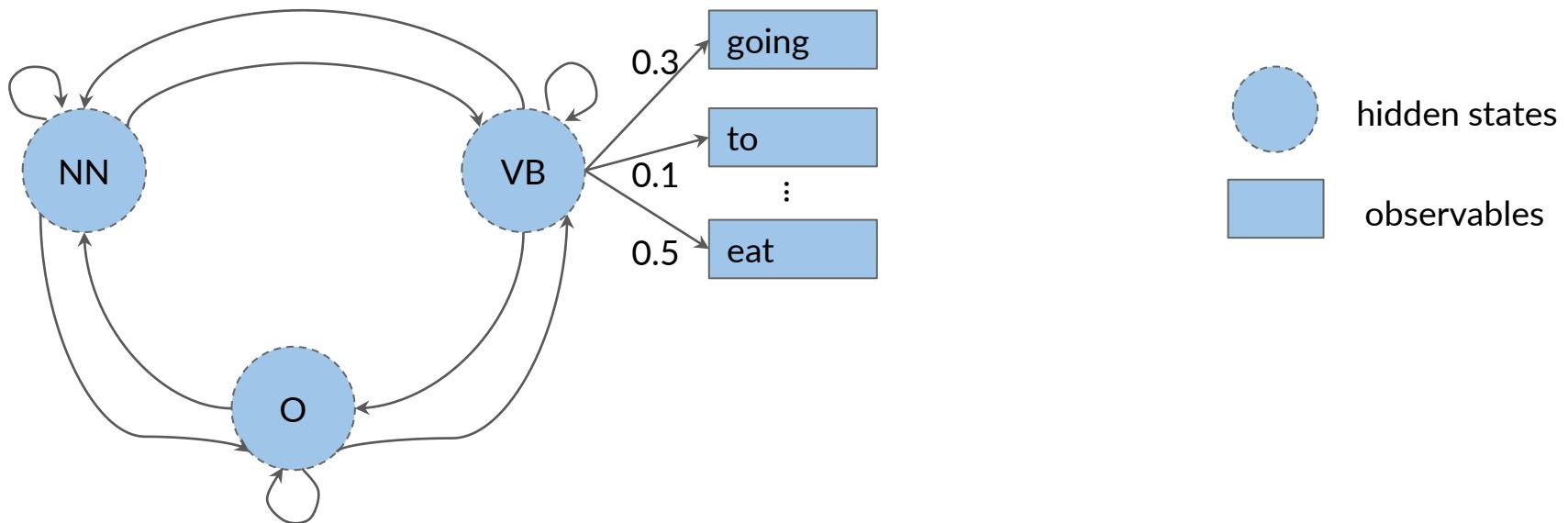
Transition probabilities



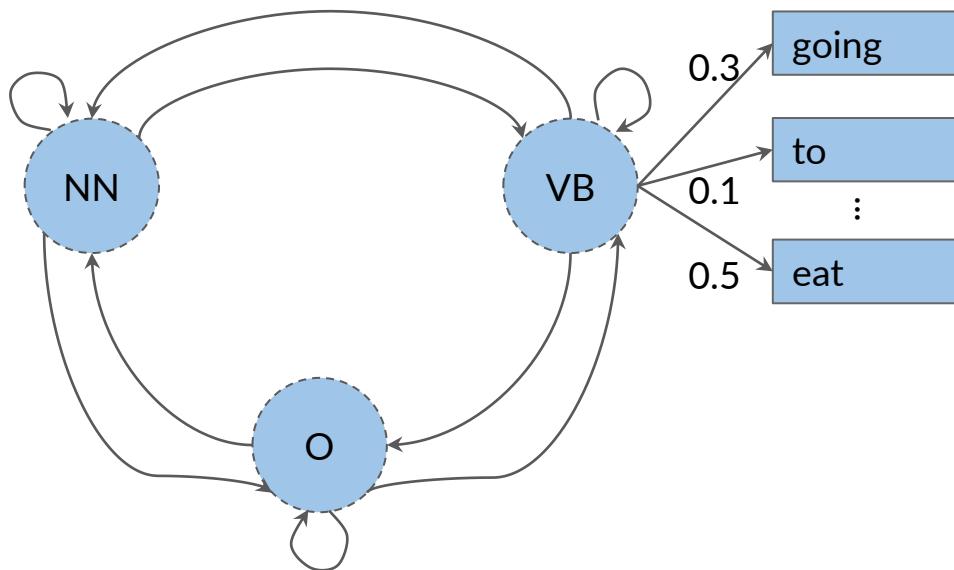
$A =$

	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

Emission probabilities



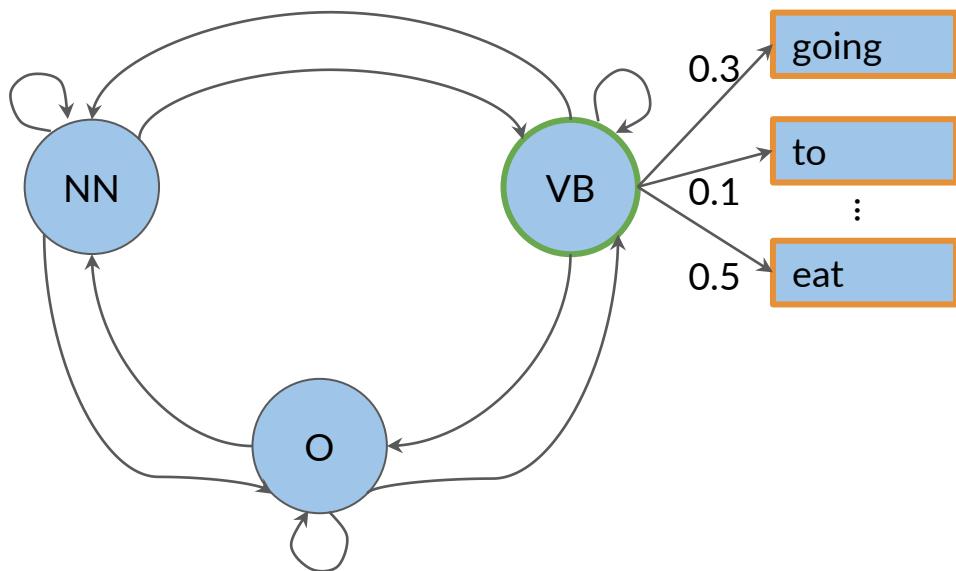
Emission probabilities



$B =$

	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	

Emission probabilities


$$B = \begin{array}{c|cccc} & \text{going} & \text{to} & \text{eat} & \dots \\ \hline \text{NN (noun)} & 0.5 & 0.1 & 0.02 & \\ \text{VB (verb)} & 0.3 & 0.1 & 0.5 & \\ \text{O (other)} & 0.3 & 0.5 & 0.68 & \end{array}$$

The emission matrix

$B =$

	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	

0.68

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

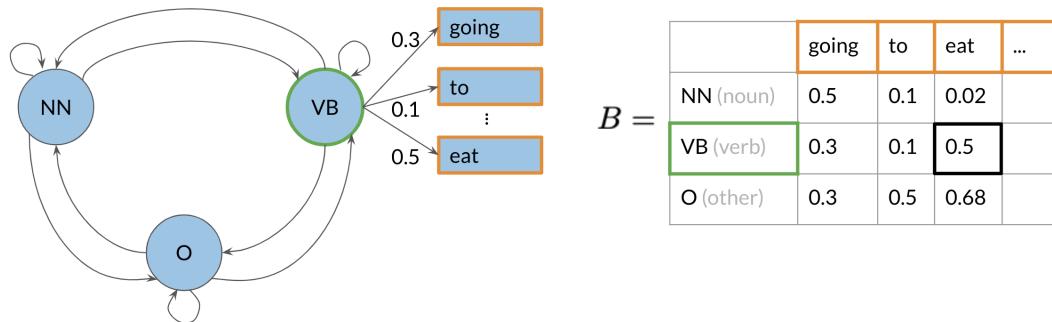
He lay on his back. noun

I'll be back. adverb

Summary

States	Transition matrix	Emission matrix
$Q = \{q_1, \dots, q_N\}$	$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \dots & a_{N+1,N} \end{pmatrix}$	$B = \begin{pmatrix} b_{11} & \dots & b_{1V} \\ \vdots & \ddots & \vdots \\ b_{N1} & \dots & b_{NV} \end{pmatrix}$

In the previous video, I showed you an example with a simple markov model. The **transition probabilities** allowed you to identify the transition probability from one POS to another. We will now explore hidden markov models. In hidden markov models you make use of **emission probabilities** that give you the probability to go from one state (POS tag) to a specific word.



For example, given that you are in a verb state, you can go to other words with certain probabilities. This emission matrix **B**, will be used with your transition matrix **A**, to help you identify the part of speech of a word in a sentence. To populate your matrix **B**, you can just have a labelled dataset and compute the probabilities of going from a POS to each word in your vocabulary. Here is a recap of what you have seen so far:

States $Q = \{q_1, \dots, q_N\}$	Transition matrix $A = \begin{pmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \dots & a_{N+1,N} \end{pmatrix}$	Emission matrix $B = \begin{pmatrix} b_{11} & \dots & b_{1V} \\ \vdots & \ddots & \vdots \\ b_{N1} & \dots & b_{NV} \end{pmatrix}$
-------------------------------------	---	---

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

Note that the sum of each row in your **A** and **B** matrix has to be 1. Next, I will show you how you can calculate the probabilities inside these matrices.



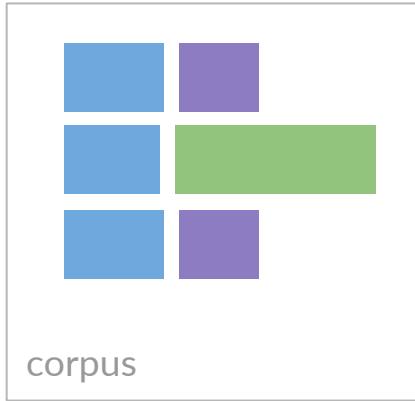
deeplearning.ai

Calculating Probabilities

Transition probabilities

You	eat
The	oatmeal
You	eat
corpus	

Transition probabilities

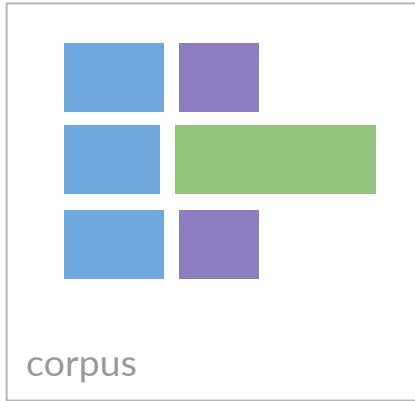


Count: 2



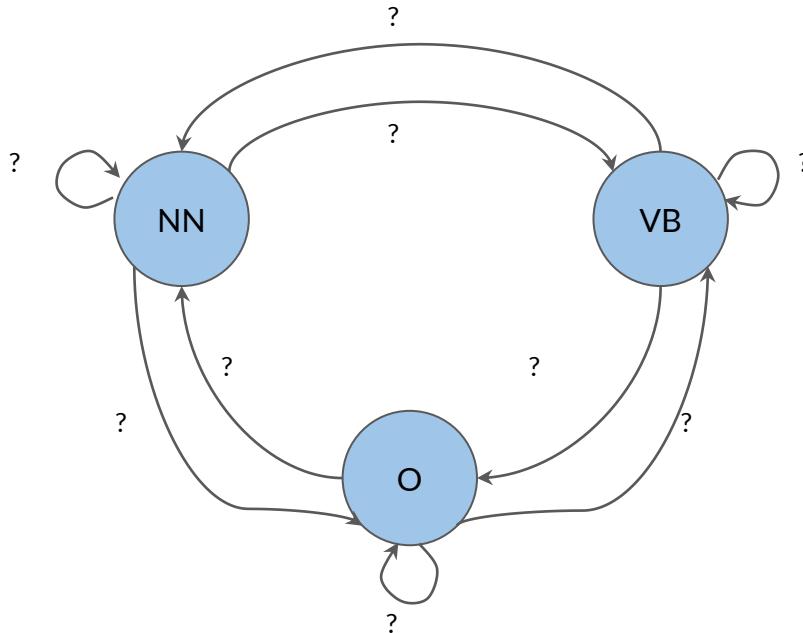
Count: 3

Transition probabilities



transition probability: + = $\frac{2}{3}$

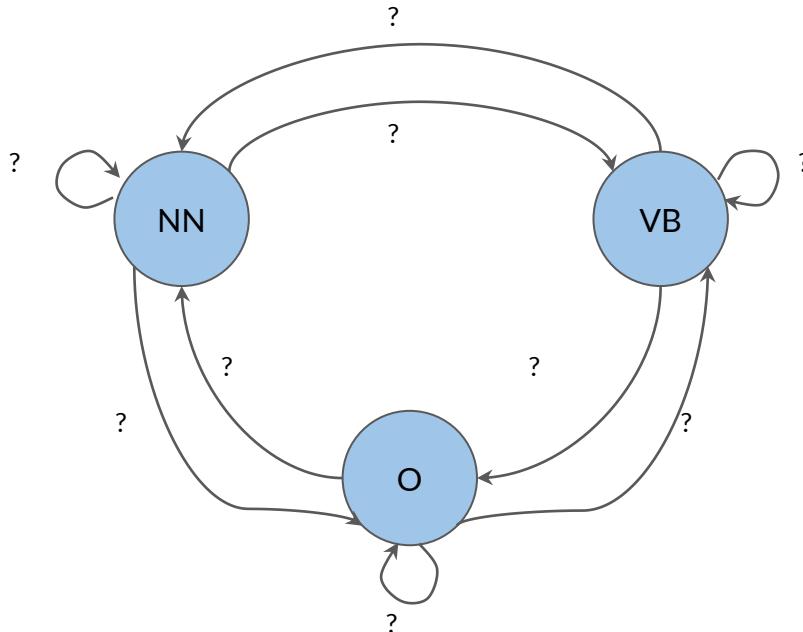
Transition probabilities



1. Count occurrences of tag pairs

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

Transition probabilities



1. Count occurrences of tag pairs

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

1. Calculate probabilities using the counts

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

The corpus

In a Station of the Metro
The apparition of these faces in the crowd :
Petals on a wet , black bough .

Ezra Pound –
1913

Preparation of the corpus

start token



<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

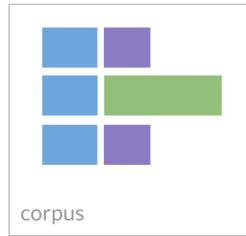
Preparation of the corpus

transfer all words to lower so that
model becomes case insensitive

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

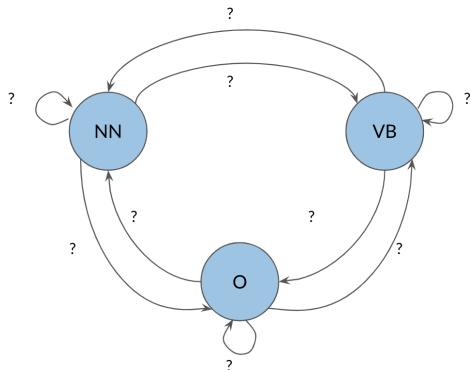
Ezra Pound –
1913

Here is a visual representation on how to calculate the probabilities:



transition probability: + = $\frac{2}{3}$

The number of times that blue is followed by purple is 2 out of 3. We will use the same logic to populate our transition and emission matrices. In the transition matrix we will count the number of times tag $t_{(i-1)}, t_{(i)}$ show up near each other and divide by the total number of times $t_{(i-1)}$ shows up (which is the same as the number of times it shows up followed by anything else).



1. Count occurrences of tag pairs
 $C(t_{i-1}, t_i)$
2. Calculate probabilities using the counts
$$P(t_i | t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^N C(t_{i-1}, t_j)}$$

$C(t_{(i-1)}, t_{(i)})$ is the count of times tag $(i-1)$ shows up before tag i . From this you can compute the probability that a tag shows up after another tag.



deeplearning.ai

Populating the Transition Matrix

Populating the transition matrix

$A =$

	NN	VB	O
π			
NN (noun)			
VB (verb)			
O (other)			

next state
↓

↑
current state

< 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

Populating the transition matrix

$A =$

	NN	VB	O
π			
NN (noun)			
VB (verb)			
O (other)			

*<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

Populating the transition matrix

$A =$

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

$\text{<}\$ \text{>} \text{ in a station of the metro}$

$\text{<}\$ \text{>} \text{ the apparition of these faces in the crowd}$

:

$\text{<}\$ \text{>} \text{ petals on a wet , black bough .}$

Ezra Pound –
1913

Populating the transition matrix

$A =$

	NN	VB	O
π	1		
NN (noun)	$C(\text{NN}, \text{NN})$		
VB (verb)	$C(\text{VB}, \text{NN})$		
O (other)	$C(\text{O}, \text{NN})$		

$\langle s \rangle$ in a station of the metro

$\langle s \rangle$ the apparition of these faces in the crowd

:

$\langle s \rangle$ petals on a wet , black bough .

Ezra Pound –
1913

Populating the transition matrix

$A =$

	NN	VB	O
π	1		
NN (noun)	0		
VB (verb)	$C(\text{VB}, \text{NN})$		
O (other)	$C(\text{O}, \text{NN})$		

<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

Populating the transition matrix

$A =$

	NN	VB	O
π	1		
NN (noun)	0		
VB (verb)	0		
O (other)	$C(O,NN)$		

<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

Populating the transition matrix

$A =$

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

`<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

Populating the transition matrix

$A =$

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

<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

Populating the transition matrix

$A =$

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

<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

Populating the transition matrix

$A =$

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

<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

Populating the transition matrix

$A =$

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

<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

Populating the 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

<S> in a station of the metro

<S> the apparition of these faces in the crowd

:

<S> petals on a wet , black bough .

on a
a wet
wet ,
, black

Ezra Pound –
1913

Populating the transition matrix

$A =$

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

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

Populating the transition matrix

$A =$

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

$$P(\text{NN}|\pi) = \frac{C(\pi, \text{NN})}{\sum_{j=1}^N C(\pi, t_j)} = \frac{1}{3}$$

Populating the transition matrix

$A =$

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

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

Populating the transition matrix

$A =$

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

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

Smoothing

$A =$

	NN	VB	O	
π	$1+\epsilon$	$0+\epsilon$	$2+\epsilon$	$3+3^*\epsilon$
NN	$0+\epsilon$	$0+\epsilon$	$6+\epsilon$	$6+3^*\epsilon$
VB	$0+\epsilon$	$0+\epsilon$	$0+\epsilon$	$0+3^*\epsilon$
O	$6+\epsilon$	$0+\epsilon$	$8+\epsilon$	$14+3^*\epsilon$

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

Smoothing

$A =$

	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

In a real world example, you might not want to apply smoothing to the initial probs in the first row. (cause we would allow a sentence to start with any part of speech tag, including punctuation)

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

To populate the transition matrix you have to keep track of the number of times each tag shows up before another tag.

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

< 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

In the table above, you can see that green corresponds to nouns (NN), purple corresponds to verbs (VB), and blue corresponds to other (O). Orange (π) corresponds to the initial state. The numbers inside the matrix correspond to the number of times a part of speech tag shows up right after another one.

To go from O to NN or in other words to calculate $P(O|NN)$ you have to compute the following:

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

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

To generalize:

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

Unfortunately, sometimes you might not see two POS tags in front each other. This will give you a probability of 0. To solve this issue, you will "smooth" it as follows:

	NN	VB	O	
π	$1+\epsilon$	$0+\epsilon$	$2+\epsilon$	$3+3^*\epsilon$
NN	$0+\epsilon$	$0+\epsilon$	$6+\epsilon$	$6+3^*\epsilon$
VB	$0+\epsilon$	$0+\epsilon$	$0+\epsilon$	$0+3^*\epsilon$
O	$6+\epsilon$	$0+\epsilon$	$8+\epsilon$	$14+3^*\epsilon$

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

The ϵ allows you to not have any two sequences showing up with 0 probability. Why is this important?



deeplearning.ai

Populating the Emission Matrix

Emission probabilities

You	eat
The	oatmeal
You	eat
corpus	

Transition probabilities

You	eat
The	oatmeal
You	eat
corpus	

You

Count: 2



Count: 3

Transition probabilities

You	eat
The	oatmeal
You	eat
corpus	

emission probability: You = $\frac{2}{3}$

The emission matrix

$B =$

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

< 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

The emission matrix

$B =$

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

$\text{<}\$ \text{>} \text{ in a station of the metro}$
 $\text{<}\$ \text{>} \text{ the apparition of these faces in the crowd}$
:
 $\text{<}\$ \text{>} \text{ petals on a wet , black bough .}$

Ezra Pound –
1913

The emission matrix

$B =$

	in	a	...
NN (noun)	0		
VB (verb)	$C(VB, \text{in})$		
O (other)	$C(O, \text{in})$		

$\langle s \rangle$ in a station of the metro
 $\langle s \rangle$ the apparition of these faces in the crowd
:
 $\langle s \rangle$ petals on a wet , black bough .

Ezra Pound –
1913

The emission matrix

$B =$

	in	a	...
NN (noun)	0		
VB (verb)	0		
O (other)	$C(O, \text{in})$		

$\langle s \rangle$ in a station of the metro

$\langle s \rangle$ the apparition of these faces in the crowd

:

$\langle s \rangle$ petals on a wet , black bough .

Ezra Pound –
1913

The 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

The emission matrix

$B =$

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

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

$V =$ size of vocab

$N =$ number of tags

Summary

1. Calculate transition and emission matrix
1. How to apply smoothing

To populate the emission matrix, you have to keep track of the words associated with their parts of speech tags.

	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

To populate the matrix, we will also use smoothing as we have previously used:

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

Where $C(t_i, w_i)$ is the count associated with how many times the tag t_i is associated with the word w_i . The epsilon above is the smoothing parameter. In the next video, we will talk about the Viterbi algorithm and discuss how you can use the transition and emission matrix to come up with probabilities.



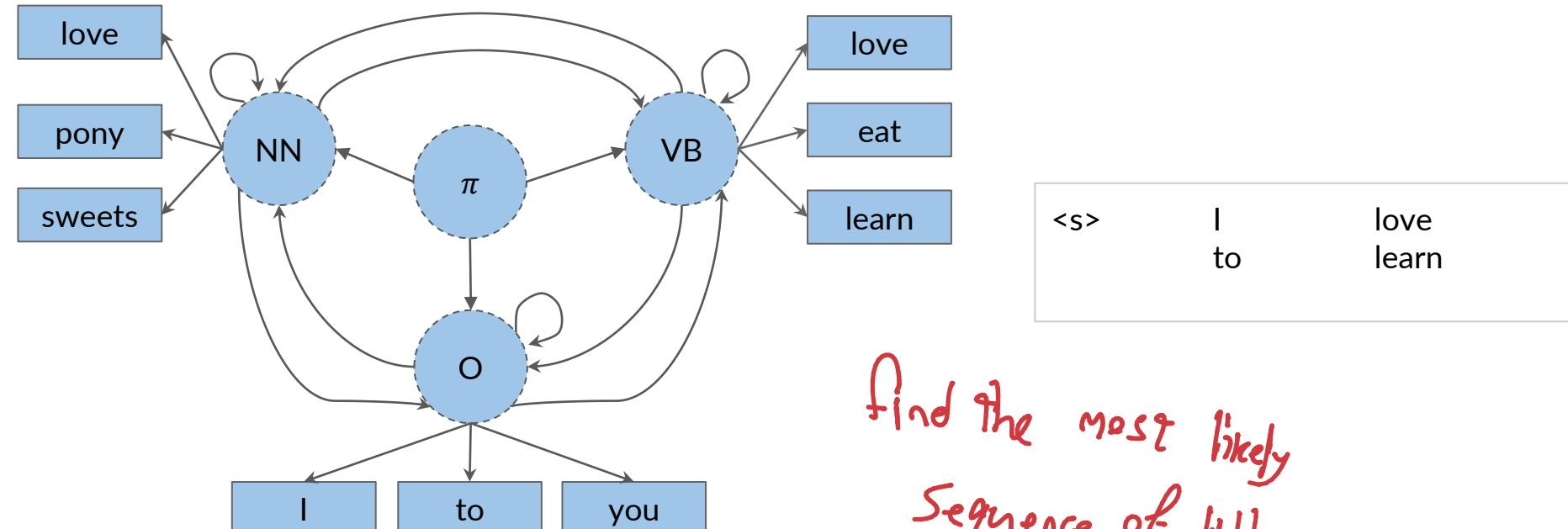
deeplearning.ai

The Viterbi Algorithm

Why not learn something ?

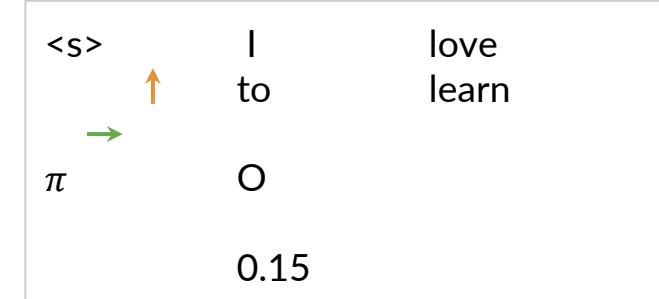
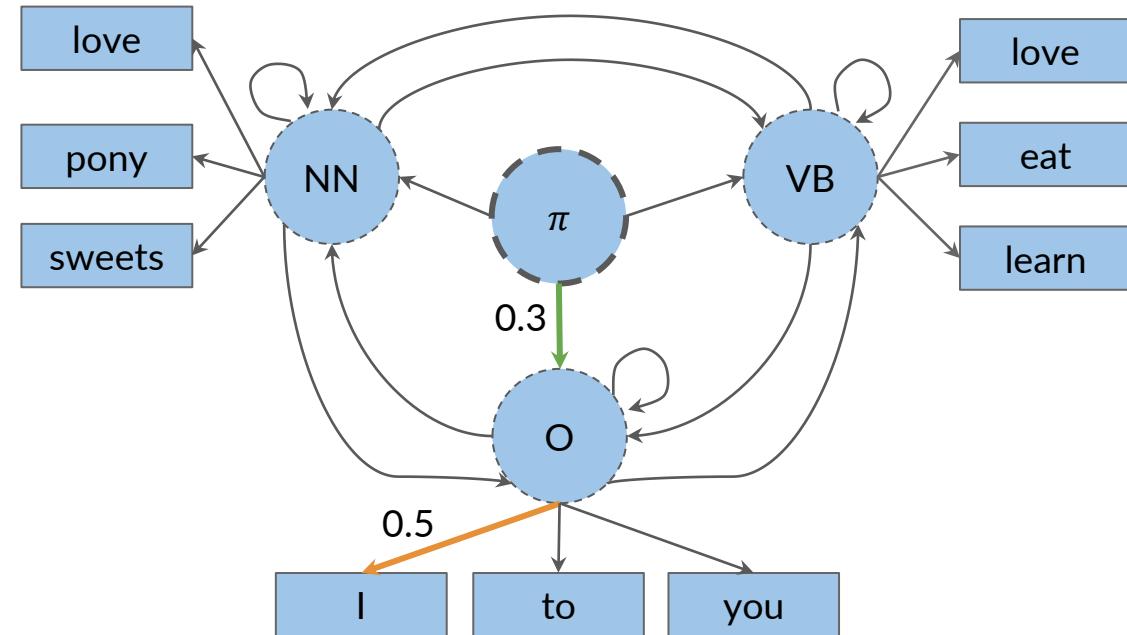
? ? ? ? ?

Viterbi algorithm – a graph algorithm

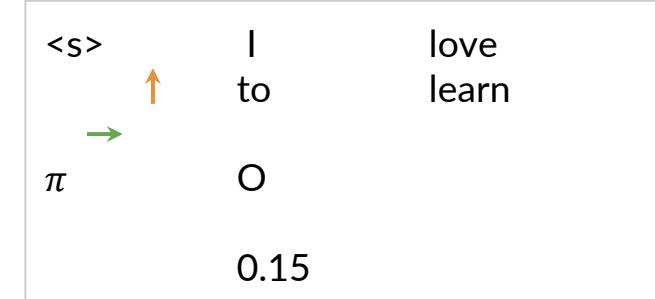
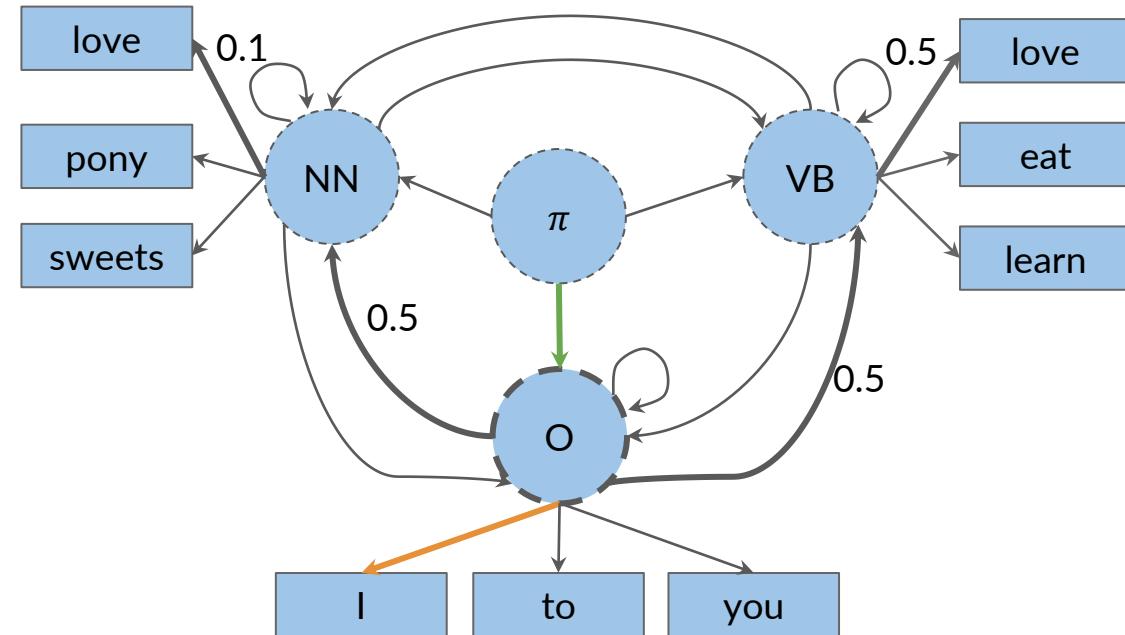


find the most likely
Sequence of hidden
states for given seq of word

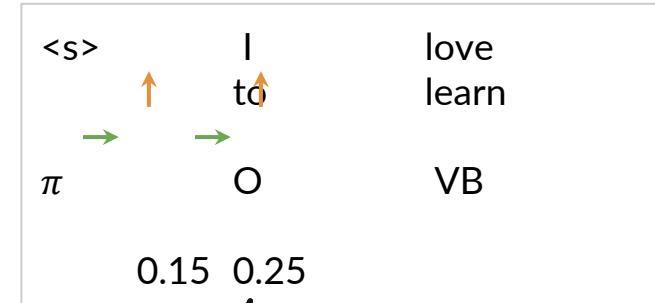
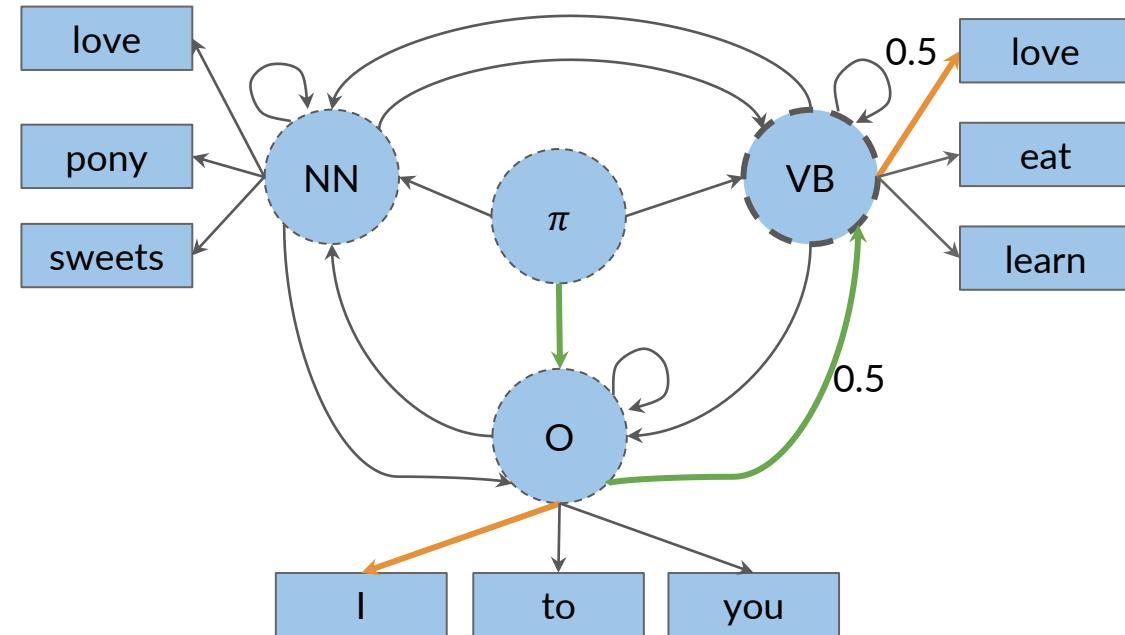
Viterbi algorithm – a graph algorithm



Viterbi algorithm – a graph algorithm

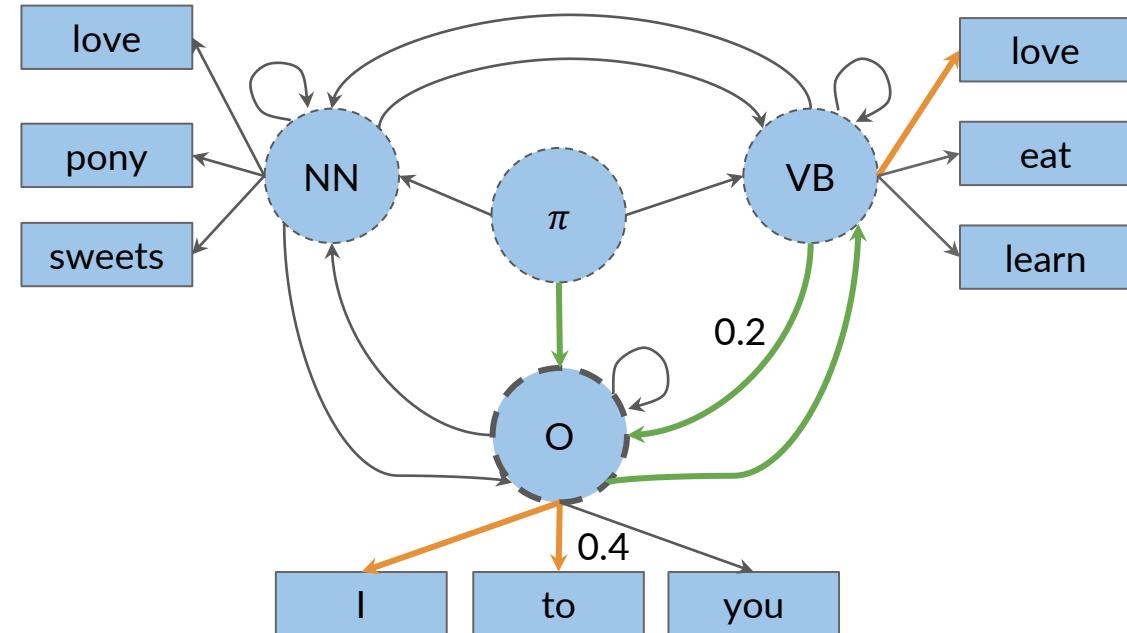


Viterbi algorithm – a graph algorithm



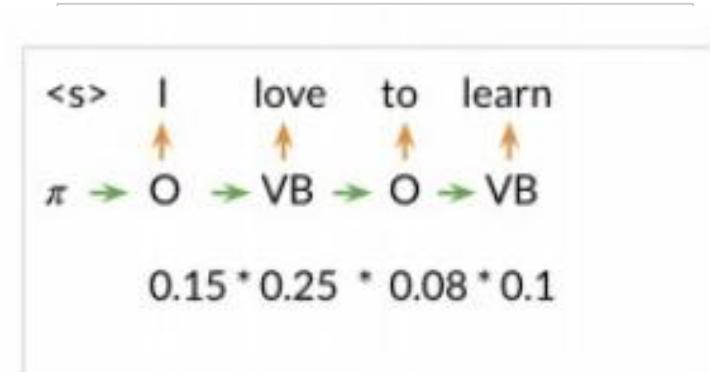
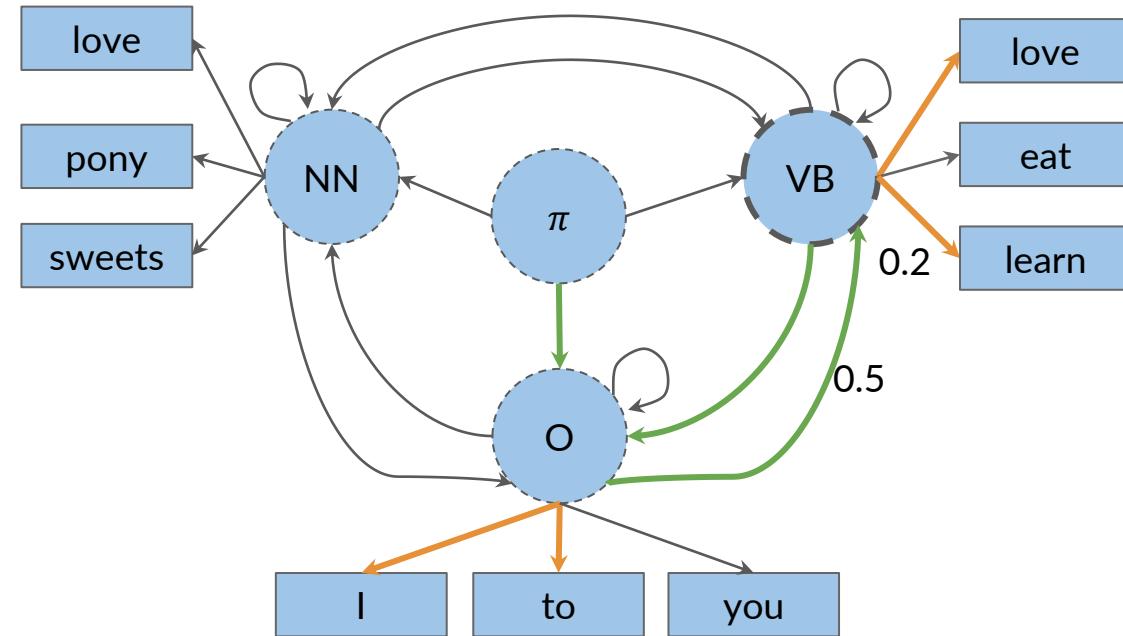
The emission Prob is higher so we choose love from VB

Viterbi algorithm – a graph algorithm

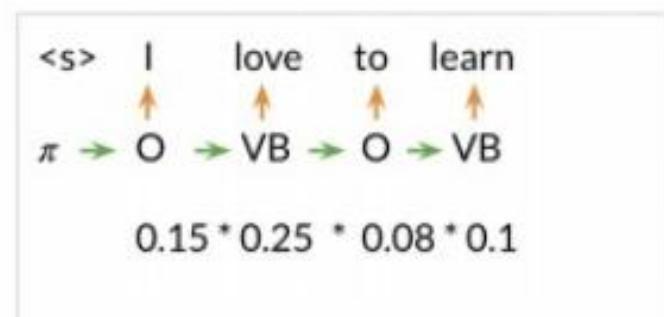
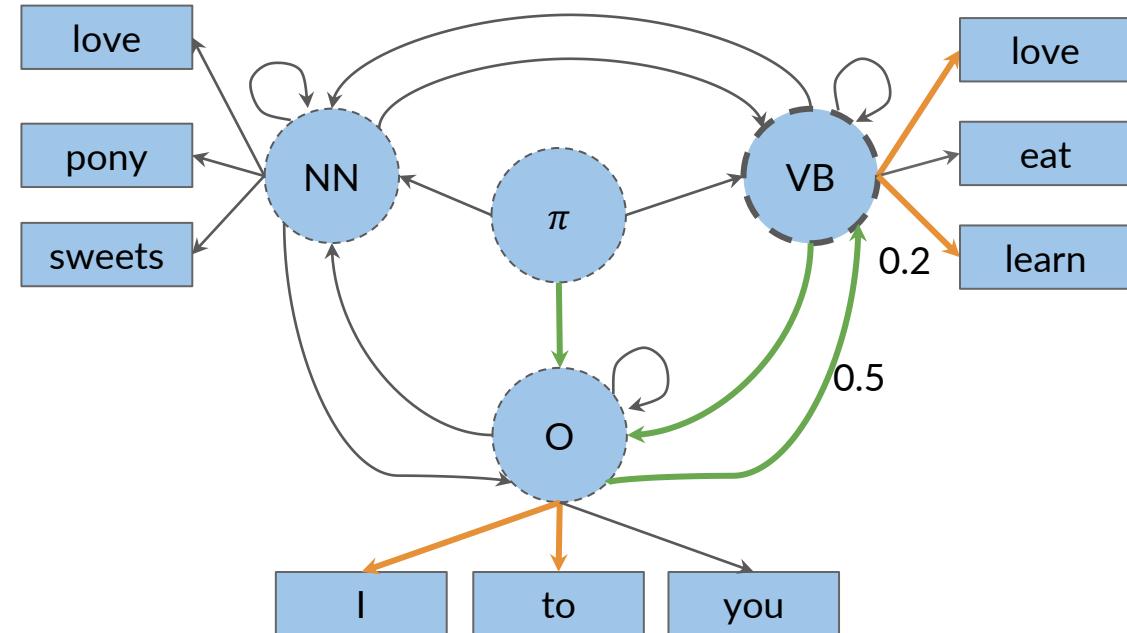


<S> I love to learn
 $\pi \rightarrow O \rightarrow VB \rightarrow O \rightarrow VB$
 $0.15 * 0.25 * 0.08 * 0.1$

Viterbi algorithm – a graph algorithm



Viterbi algorithm – a graph algorithm



Probability for this sequence of hidden states: 0.0003

Viterbi algorithm – Steps

1. Initialization step
 2. Forward pass
 3. Backward pass

N : # Postags

K: # words in the given seq

the indices
of the United
States

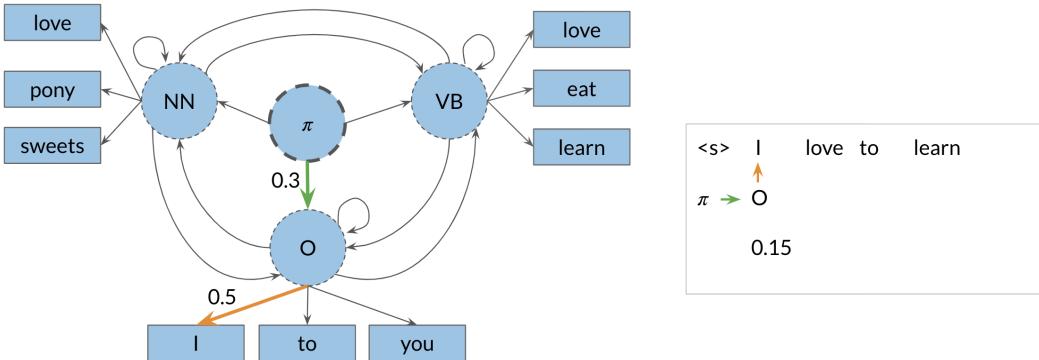
C =

	w_1	w_2	...	w_K
t_1				
...				
t_N				

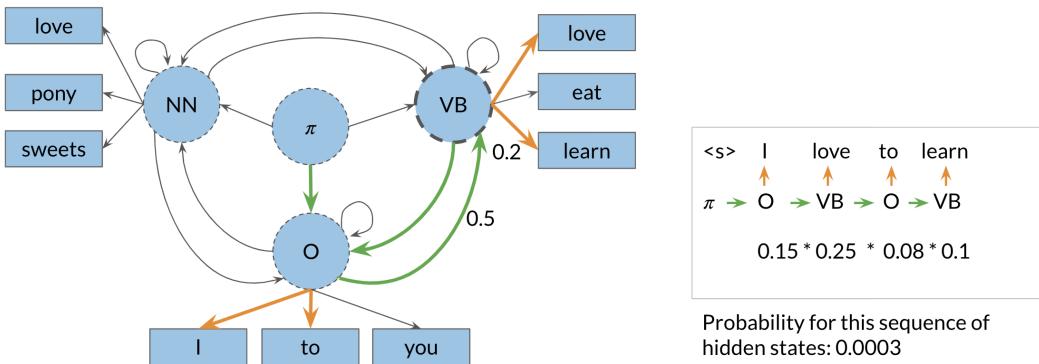
$D =$

	w_1	w_2	...	w_K
t_1				
...				
t_N				

The Viterbi algorithm makes use of the transition probabilities and the emission probabilities as follows.



To go from π to O you need to multiply the corresponding transition probability (0.3) and the corresponding emission probability (0.5), which gives you 0.15. You keep doing that for all the words, until you get the probability of an entire sequence.



You can then see how you will just pick the sequence with the highest probability. We will show you a systematic way to accomplish this (Viterbi!).



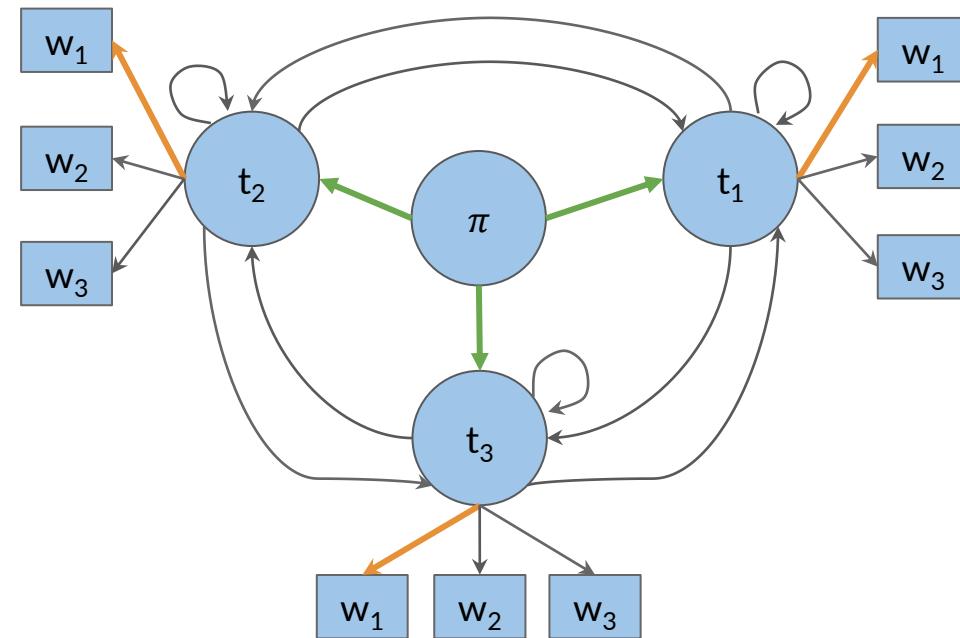
deeplearning.ai

Viterbi: Initialization

Viterbi algorithm – Steps

1. Initialization step

10 Initialization step *the first col of C,D is populated*

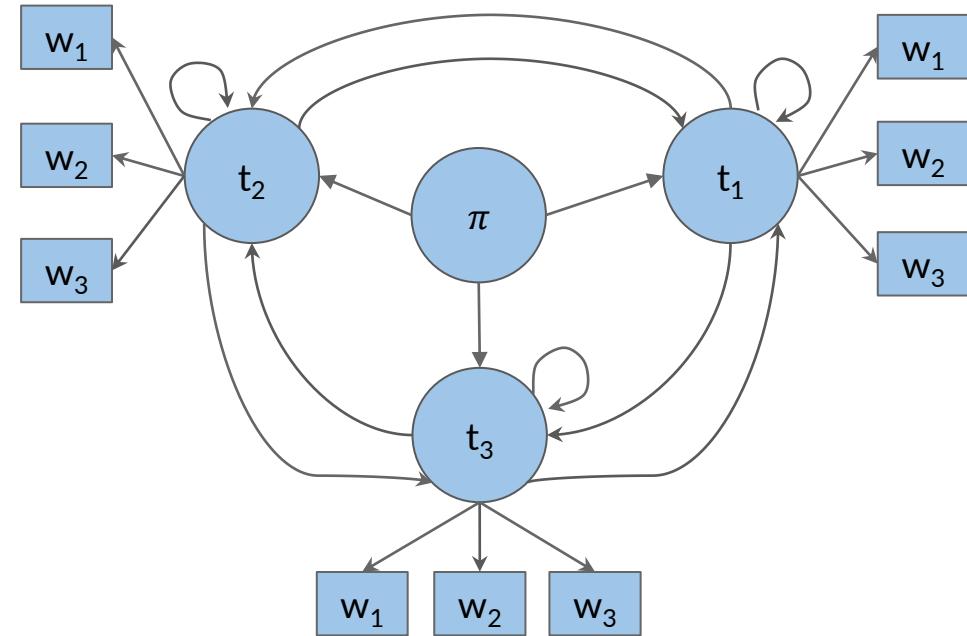


$C =$

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

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

Initialization step

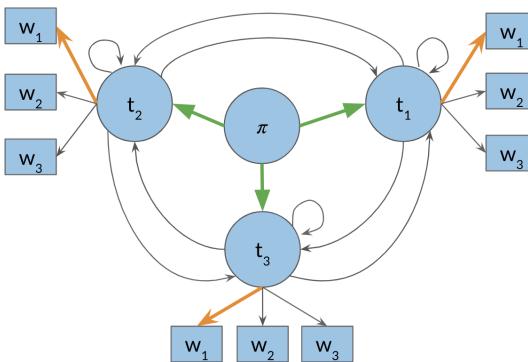


$D =$

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

$$d_{i,1} = 0$$

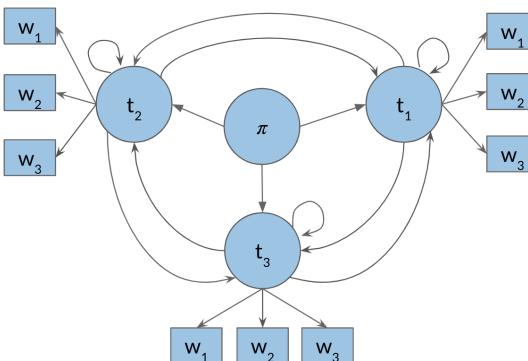
You will now populate a matrix C of dimension (num_tags, num_words). This matrix will have the probabilities that will tell you what part of speech each word belongs to.



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

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

Now to populate the first column, you just multiply the initial π distribution, for each tag, times $b_{i,cindex(w_1)}$. Where the i , corresponds to the tag of the initial distribution and the $cindex(w_1)$, is the index of **word 1** in the emission matrix. And that's it, you are done with populating the first column of your new C matrix. You will now need to keep track what part of speech you are coming from. Hence we introduce a matrix D , which allows you to store the labels that represent the different states you are going through when finding the most likely sequence of POS tags for the given sequence of words w_1, \dots, w_K . At first you set the first column to 0, because you are not coming from any POS tag.



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

$$d_{i,1} = 0$$

These two matrices will make more sense in the next videos.



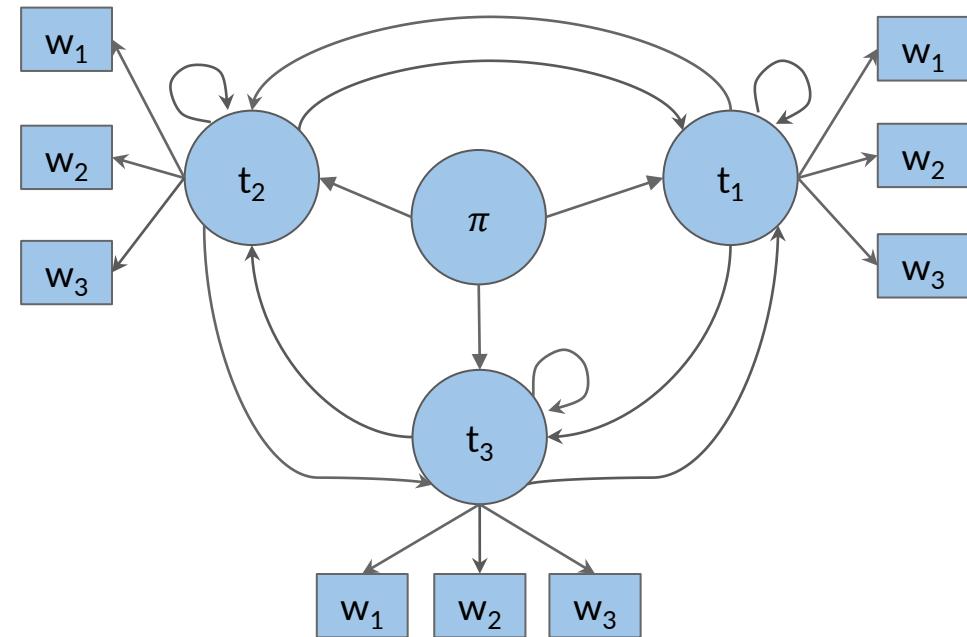
deeplearning.ai

Viterbi: Forward Pass

Viterbi algorithm – Steps

2. Forward pass

Forward pass *Populate col by col*

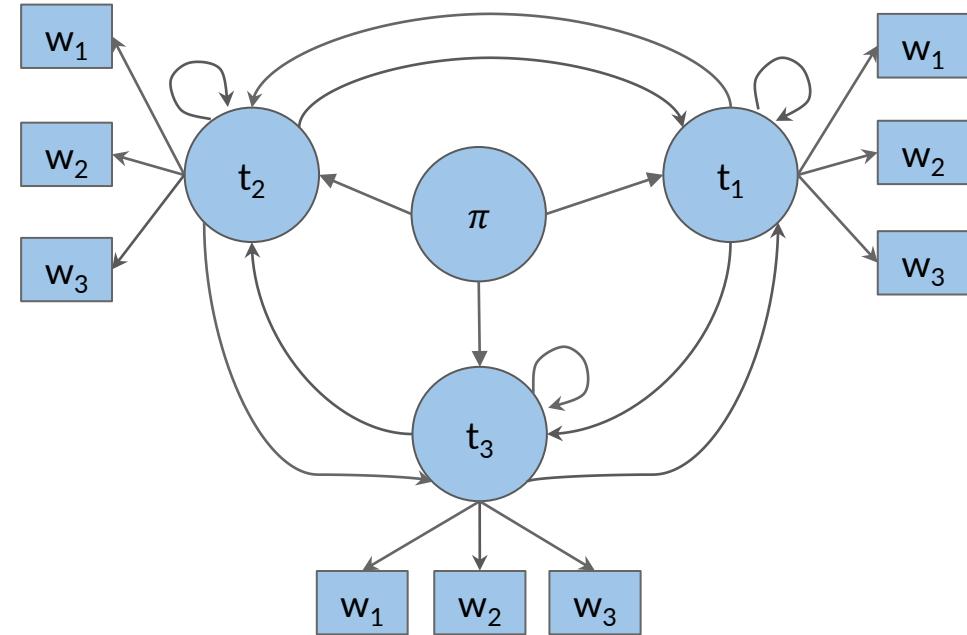


$C =$

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

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

Forward pass

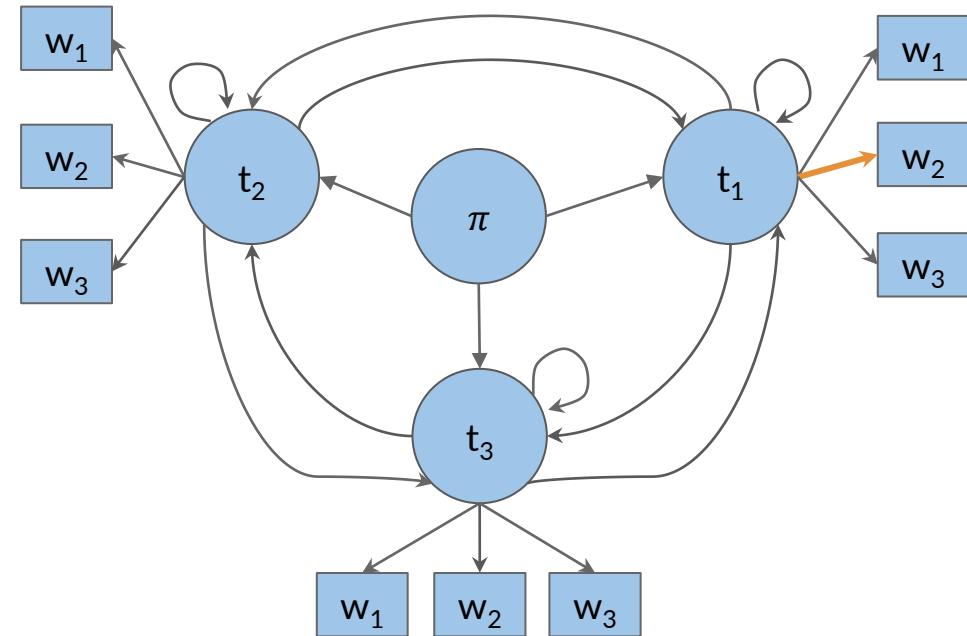


$C =$

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

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

Forward pass

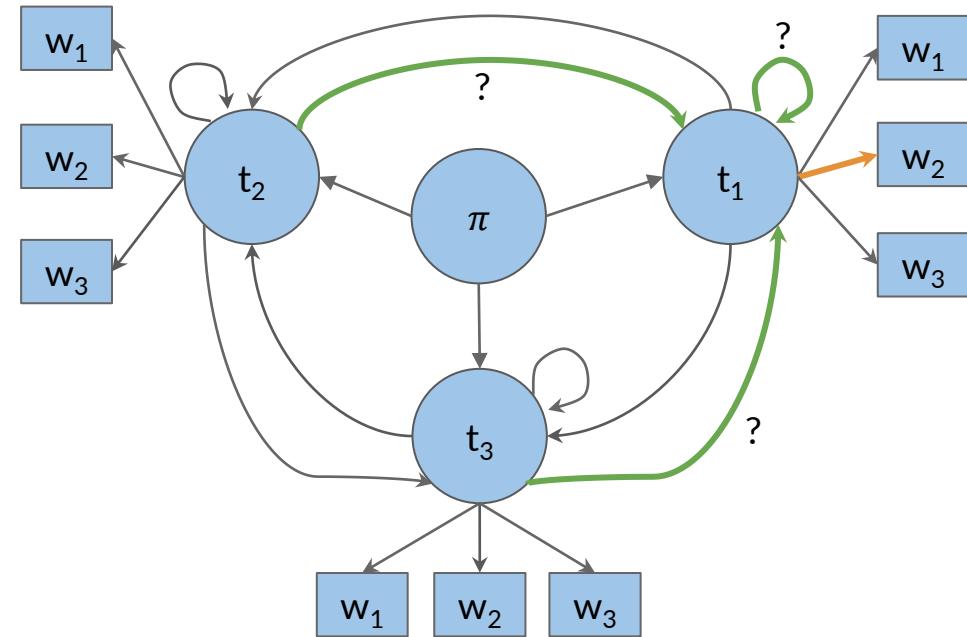


$C =$

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

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

Forward pass

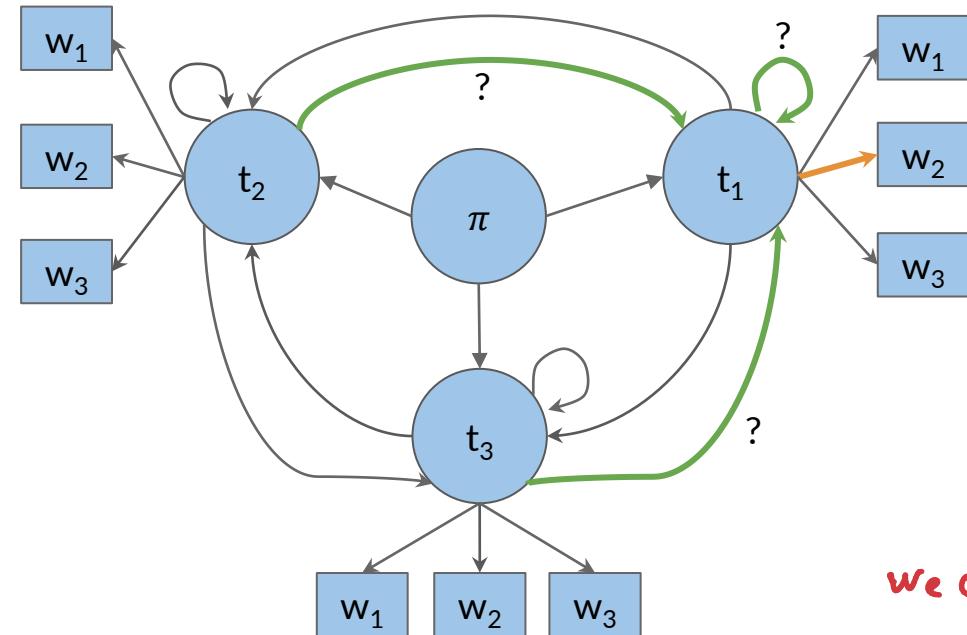


$C =$

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

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

Forward pass



$$C =$$

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

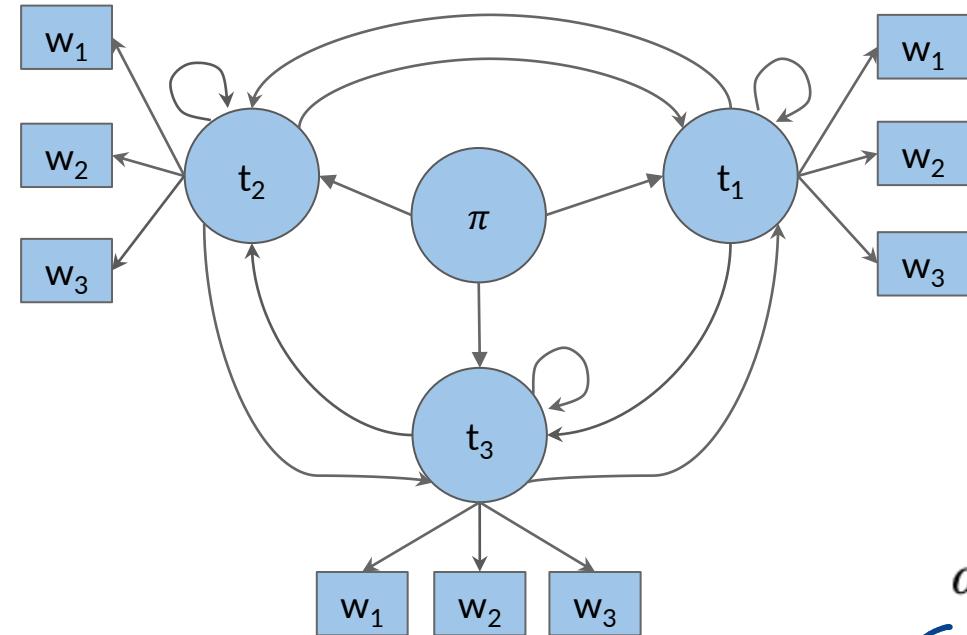
$$c_{1,2} = \max_k c_{k,1} * a_{k,1} * b_{1,cindex(w_2)}$$

we choose the k that
maximize entire
formula

$$k \in \{1, 2, 3\}$$



Forward pass



$$D =$$

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

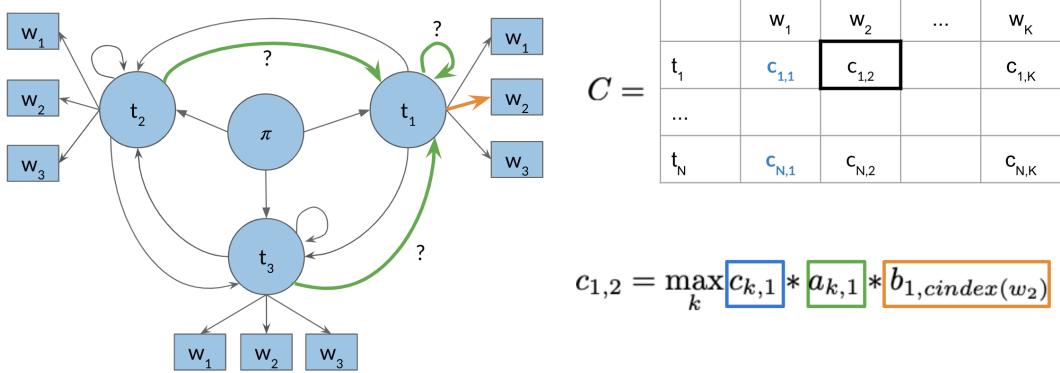
$$c_{i,j} = \max_k c_{k,j-1} * a_{k,i} * b_{i,c\text{index}(w_j)}$$

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

we save k which maximizes $c_{i,j}$

$k \in \{1, 2, 3\}$

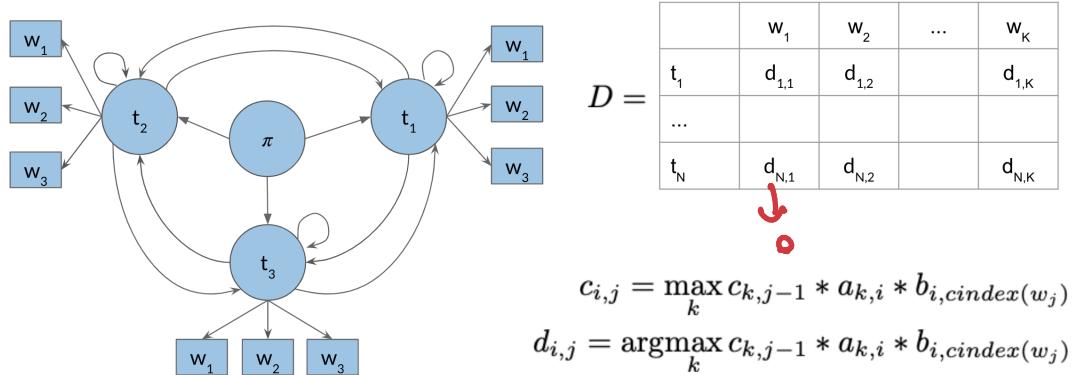
This will be best illustrated with an example:



So to populate a cell (i.e. 1,2) in the image above, you have to take the max of [kth cells in the previous column, times the corresponding transition probability of the kth POS to the first POS times the emission probability of the first POS and the current word you are looking at]. You do that for all the cells. Take a paper and a pencil, and make sure you understand how it is done.

The general rule is $c_{i,j} = \max_k c_{k,j-1} * a_{k,i} * b_{i,c\text{index}(w_j)}$

Now to populate the **D** matrix, you will keep track of the argmax of where you came from as follows:



Note that the only difference between c_{ij} and d_{ij} , is that in the former you compute the probability and in the latter you keep track of the index of the row where that probability came from. So you keep track of which k was used to get that max probability.



deeplearning.ai

Viterbi: Backward Pass

Viterbi algorithm – Steps

3. Backward pass

Backward pass

$C =$

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

$D =$

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

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

Backward pass

$D =$

	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	

Backward pass

$C =$

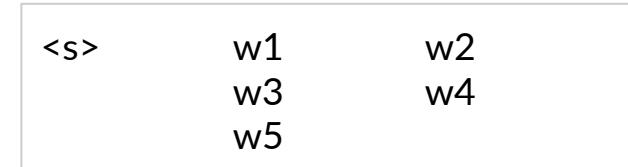
	w_1	w_2	w_3	w_4	w_5
t_1	0.25	0.125	0.025	0.0125	0.01
t_2	0.1	0.025	0.05	0.01	0.003
t_3	0.3	0.05	0.025	0.02	0.0000
t_4	0.2	0.1	0.000	0.0025	0.0003

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

Backward pass

$D =$

	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



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

Backward pass

$D =$

	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	

t_1

Backward pass

$D =$

	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

$\langle s \rangle \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

$t_3 \leftarrow t_1$

$\langle s \rangle$	w_1	w_2
	w_3	w_4
	w_5	

t_1

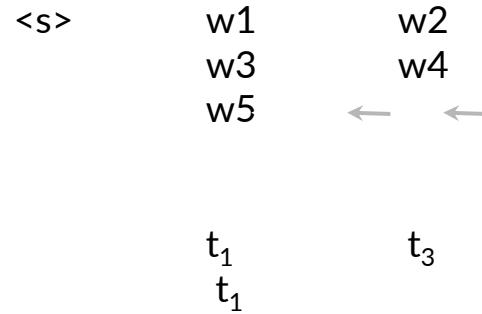
t_3

Backward pass

$D =$

	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

$\langle s \rangle \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$
 $\eta \leftarrow t_3 \leftarrow t_1$

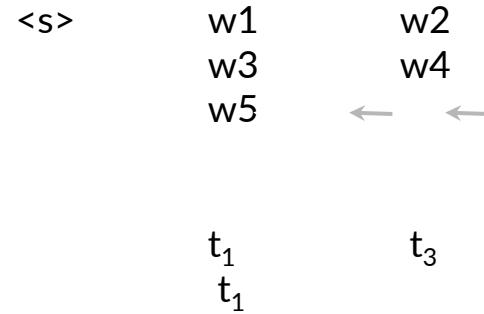


Backward pass

$D =$

	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

$\langle s \rangle \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$
 $t_3 \leftarrow t_1 \leftarrow t_3 \leftarrow t_1$



Backward pass

$D =$

	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

$\langle s \rangle \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$
 $t_2 \leftarrow t_3 \leftarrow q_1 \leftarrow t_3 \leftarrow q_1$

$\langle s \rangle$

w_1
 w_3
 w_5

w_2
 w_4

t_3
 t_3

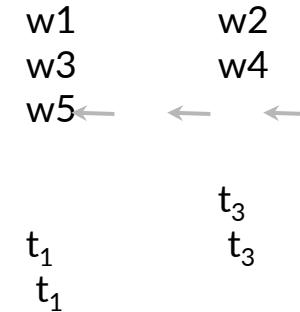
t_1
 t_1

Backward pass

$D =$

	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

$\langle s \rangle$



Sequence of pos tags
with highest prob

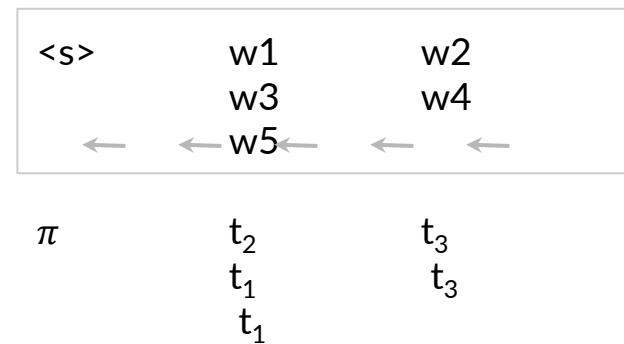
$\langle s \rangle \leftarrow t_2 \leftarrow t_3 \leftarrow t_1 \leftarrow t_3 \leftarrow t_1$

Backward pass

$D =$

	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

$\langle s \rangle \quad w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$
 $t_2 \leftarrow t_3 \leftarrow t_1 \leftarrow t_3 \leftarrow t_1$



Implementation notes

1. In Python index starts with 0!
2. Use log probabilities

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


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

Summary

1. From word sequence to POS tag sequence
2. Viterbi algorithm
3. Log probabilities

Great, now that you know how to compute A, B, C, and D, we will put it all together and show you how to construct the path that will give you the part of speech tags for your sentence.

	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

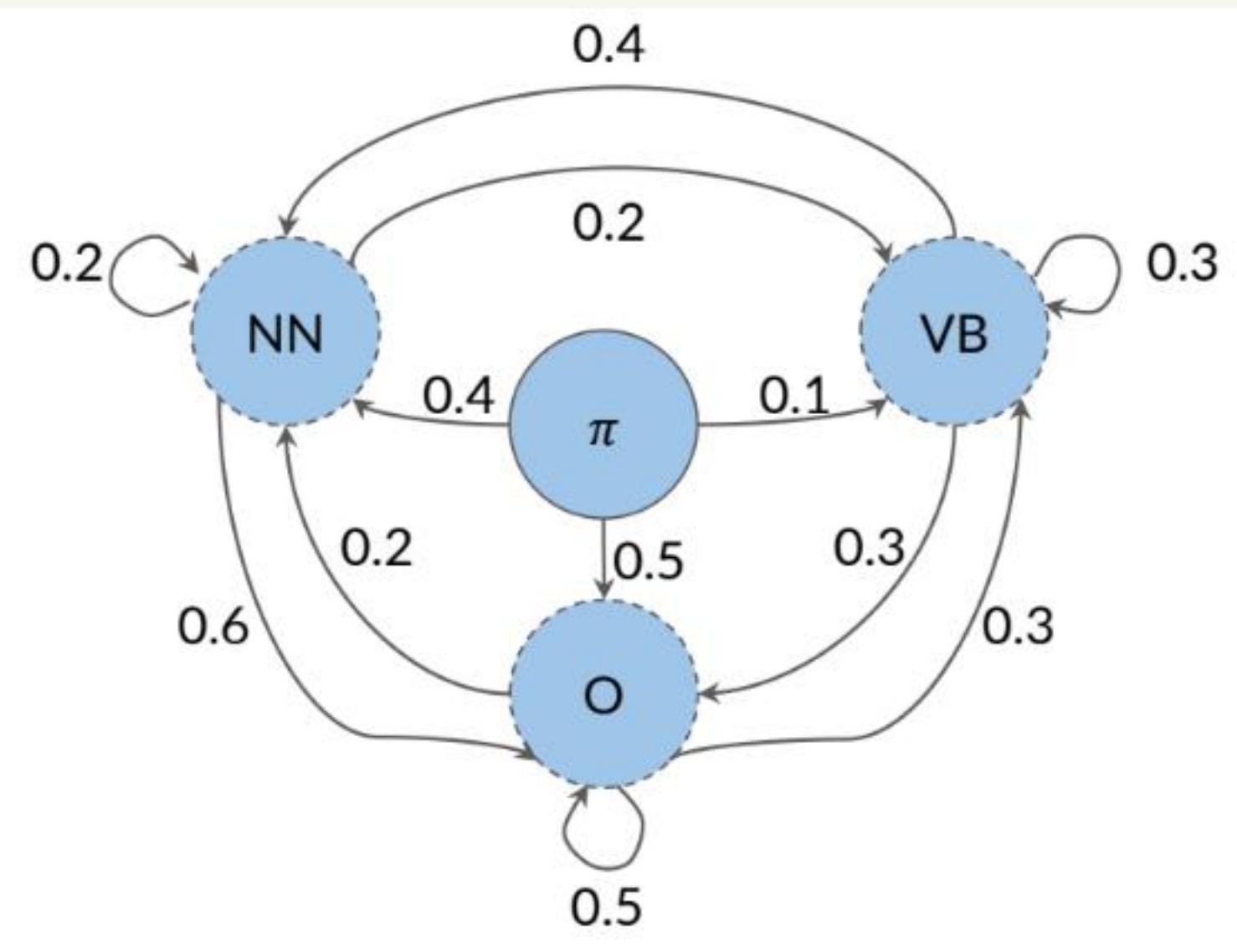
$$s = \operatorname{argmax}_i c_{i,K} = 1$$

The equation above just gives you the index of the highest row in the last column of C. Once you have that, you can go ahead and start using your D matrix as follows:

	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

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

Note that since we started at index one, hence the last word (w_5) is t_1 . Then we go to the first row of D and whatever that number is, it indicated the row of the next part of speech tag. Then next part of speech tag indicates the row of the next and so forth. This allows you to reconstruct the POS tags for your sentence. You will be implementing this in this week's programming assignment. Good luck!



$A =$

	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

5

$B \leftarrow$

	w_1	w_2	w_3
NN	0.5	0.1	0.4
VB	0.2	0.4	0.4
O	0.1	0.7	0.2

C₂

	w_1	w_2	w_3
NN	0.2	0.007	
VB	0.02		
O	0.05		

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

$$c_{1,2} = \max \left\{ \begin{array}{l} c_{1,1} \times a_{1,1} \times b_{1,2} = 0.2 \times 0.2 \times 0.1 = 0.004 \\ c_{2,1} \times a_{2,1} \times b_{2,2} = 0.02 \times 0.4 \times 0.4 = 0.0032 \\ c_{3,1} \times a_{3,1} \times b_{3,2} = 0.05 \times 0.2 \times 0.7 = 0.007 \end{array} \right\}$$

↓
k=3

D₂

	w_1	w_2	w_3
NN	0	3	
VB	0		
O	0		

and so on...

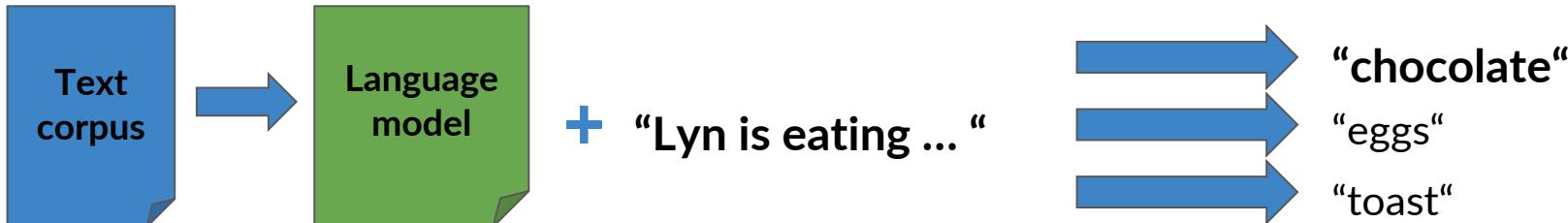


deeplearning.ai

N-Grams: Overview

What you'll be able to do!

- Create **language model (LM)** from text corpus to
 - Estimate probability of word sequences
 - Estimate probability of a word following a sequence of words
- Apply this concept to **autocomplete a sentence** with most likely suggestions



Other Applications

Speech recognition



$P(I \text{ saw a van}) > P(\text{eyes awe of an})$

Spelling correction



“He entered the **ship** to buy some groceries” - “ship” a dictionary word
• $P(\text{entered the shop to buy}) > P(\text{entered the ship to buy})$

Augmentative communication

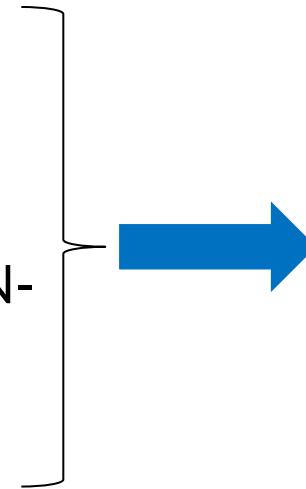


Predict most likely word from menu for people unable to physically talk or sign.
(Newell et al., 1998)

→ Used by Stephen Hawking! (RIP)

Learning objectives

- Process text corpus to N-gram language model
- Out of vocabulary words
- Smoothing for previously unseen N-grams
- Language model evaluation



Sentence
auto-complete

N-grams are fundamental and give you a foundation that will allow you to understand more complicated models in the specialization. These models allow you to calculate probabilities of certain words happening in a specific sequence. Using that, you can build an auto-correct or even a search suggestion tool.

Other applications of N-gram language modeling include:

Speech recognition



$P(\text{I saw a van}) > P(\text{eyes awe of an})$

Spelling correction



- $P(\text{entered the shop to buy}) > P(\text{entered the ship to buy})$

Augmentative communication



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

This week you are going to learn to:

- Process a text corpus to N-gram language model
- Handle out of vocabulary words
- Implement smoothing for previously unseen N-grams
- Language model evaluation



deeplearning.ai

N-grams and Probabilities

Outline

- What are N-grams?
- N-grams and conditional probability from corpus

★ Punctuation is treated like words

N-gram

An N-gram is a sequence of N words → word order matters!

Corpus: I am happy because I am learning

Unigrams: { I, am , happy , because , learning } → Unique single words

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

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

Sequence notation

Corpus: $w_1 \ w_2 \ w_3$... $w_{498} \ w_{499} \ w_{500}$ $m = 500$

$$w_1^m = w_1 \ w_2 \ ... \ w_m$$

$$w_1^3 = w_1 \ w_2 \ w_3$$

$$w_{m-2}^m = w_{m-2} \ w_{m-1} \ w_m$$

Unigram probability

Corpus: I am happy because I am learning

1 2 3 4 5 6 7
Size of corpus m = 7

$$P(I) = \frac{2}{7}$$

$$P(happy) = \frac{1}{7}$$

Probability of unigram:

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

Bigram probability

Corpus: I am happy because I am learning

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

$$P(happy|I) = \frac{C(I happy)}{C(I)} = \frac{0}{2} = 0 \quad \times \text{ I happy}$$

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

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

Only works if x followed by another word

Trigram Probability

Corpus: I am happy because I am learning

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

Probability of a trigram:

$$P(w_3|w_1^2) = \frac{C(w_1^2 w_3)}{C(w_1^2)}$$

$$C(w_1^2 w_3) = C(w_1 w_2 w_3) = C(w_1^3)$$

N-gram probability

Probability of N-gram:

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

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

$$C(w_1^{N-1} w_N) = C(w_1^N)$$

Quiz

Objective: Apply n-gram probability calculation on sample corpus and 3-gram.

Question:

Corpus: "In every place of great resort the monster was the fashion. They sang of it in the cafes, ridiculed it in the papers, and represented it on the stage. " (Jules Verne, Twenty Thousand Leagues under the Sea)

In the context of our corpus, what is the probability of word "papers" following the phrase "it in the".

Type: Multiple Choice, single answer

Options and solution:

1. $P(\text{papers}|\text{it in the}) = 0$
3. $P(\text{papers}|\text{it in the}) = 2/3$

2. $P(\text{papers}|\text{it in the}) = 1$

4. $P(\text{papers}|\text{it in the}) =$
 $= C(\text{it in the papers})/C(\text{it in the})$

(A blue arrow points from option 4 to the formula.)

Before we start computing probabilities of certain sequences, we need to first define what is an N-gram language model:

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 ...} X I happy

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

Now given those definitions, we can label a sentence as follows:

Corpus: This is great ... teacher drinks tea. $m = 500$
 $w_1 \ w_2 \ w_3 \dots \ w_{498} \ w_{499} \ w_{500}$

In other notation you can write:

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

Given the following corpus: I am happy because I am learning.

- 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

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

$$P(happy|I) = \frac{C(I happy)}{C(I)} = \frac{0}{2} = 0 \quad \text{X} \quad \text{I happy}$$

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

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:

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

N-gram Probability:

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



deeplearning.ai

Sequence Probabilities

Outline

- Sequence probability
- Sequence probability shortcomings
- Approximation by N-gram probabilities

Probability of a sequence

- Given a sentence, what is its probability?

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

- Conditional probability and chain rule reminder

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

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

$$P(A)P(B|A) \geq P(A, B) \quad P(C|A, B) \geq P(C|A, B, C) \quad \checkmark$$

$$P(A, B)P(C|A, B) \geq P(A, B, C)$$

Probability of a sequence

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

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

Sentence not in corpus

- Problem: Corpus almost never contains the exact sentence we're interested in or even its longer subsequences!

Input: **the teacher drinks tea**

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

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

← Both likely 0

Approximation of sequence probability

the teacher drinks tea

$$P(\text{tea}|\text{the teacher drinks}) \approx P(\text{tea}|\text{drinks})$$

$$\begin{aligned} &P(\text{teacher}|\text{the}) \\ &P(\text{drinks}|\text{teacher}) \\ &P(\text{tea}|\text{drinks}) \end{aligned}$$

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

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



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

Approximation of sequence probability

- Markov assumption: only last N words matter
- Bigram $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$
- N-gram $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$
- Entire sentence modeled with bigram $P(w_1^n) \approx \prod_{i=1}^n P(w_i|w_{i-1})$
 $P(w_1^n) \approx \boxed{P(w_1)} P(w_2|w_1) \dots P(w_n|w_{n-1})$

Quiz

Objective: Apply sequence probability approximation with bigrams.

Question:

Given these conditional probabilities

$$\begin{array}{lll} P(\text{Mary})=0.1; & P(\text{likes})=0.2; & P(\text{cats})=0.3 \\ P(\text{Mary}|\text{likes}) =0.2; & P(\text{likes}|\text{Mary}) =0.3; & P(\text{cats}|\text{likes})=0.1; \\ & & P(\text{likes}|\text{cats})=0.4 \end{array}$$

Approximate the probability of the following sentence with bigrams: "Mary likes cats"

Type: Multiple Choice, single answer

Options and solution:

1. $P(\text{Mary likes cats}) = 0$
2. $P(\text{Mary likes cats}) = 1$
3. $P(\text{Mary likes cats}) = 0.003$
4. $P(\text{Mary likes cats}) = 0.008$

You just saw how to compute sequence probabilities, their short comings, and finally how to approximate N-gram probabilities. In doing so, you try to approximate the probability of a sentence. For example, what is the probability of the following sentence: *The teacher drinks tea*. To compute it, you will make use of the following:

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

To compute the probability of a sequence, you can compute the following:

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

One of the main issues with computing the probabilities above is the corpus rarely contains the exact same phrases as the ones you computed your probabilities on. Hence, you can easily end up getting a probability of 0. The *Markov assumption* indicates that only the last word matters. Hence:

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

You can model the entire sentence as follows:

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



deeplearning.ai

Starting and Ending Sentences

Outline

- Start of sentence symbols <s>
- End of sentence symbol </s>

Start of sentence token < s >

the teacher drinks tea

$$P(\text{the teacher drinks tea}) \approx P(\text{the})P(\text{teacher}|\text{the})P(\text{drinks}|\text{teacher})P(\text{tea}|\text{drinks})$$



< s > the teacher drinks tea

$$P(<\text{s}> \text{the teacher drinks tea}) \approx P(\text{the}|<\text{s}>)P(\text{teacher}|\text{the})P(\text{drinks}|\text{teacher})P(\text{tea}|\text{drinks})$$

↳ Now everything is Bigram

Start of sentence token < s > for N-grams

- Trigram:

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

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

the teacher drinks tea => < s > < s > the teacher drinks tea

$$P(w_1^n) \approx P(w_1|<\text{s}><\text{s}>)P(w_2|<\text{s}> w_1)...P(w_n|w_{n-2} w_{n-1})$$

- N-gram model: add N-1 start tokens < s >

End of sentence token </s> - motivation

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

Corpus:

< s > Lyn drinks chocolate

< s > John drinks



$$\sum_w C(drinks \ w) = 1 \quad \textcircled{1}$$

$$C(drinks) = 2 \quad \textcircled{2}$$

In this situation $\textcircled{1} \neq \textcircled{2}$

End of sentence token </s> - motivation

Corpus

< s > yes no

< s > yes yes

< s > no no

Sentences of length 2:

< s > yes yes

< s > yes no

< s > no no

< s > no yes

$$P(< s > \text{ yes yes}) =$$

$$P(\text{yes} | < s >) \times P(\text{yes} | \text{yes}) =$$

$$\frac{C(< s > \text{ yes})}{\sum_w C(< s > w)} \times \frac{C(\text{yes yes})}{\sum_w C(\text{yes } w)} =$$

$$\frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$$

End of sentence token </s> - motivation

Corpus

< s > yes no

< s > yes yes

< s > no no

Sentences of length 2:

< s > yes yes

< s > yes no

< s > no no

< s > no yes

$$P(< s > \text{ yes yes}) = \frac{1}{3}$$

$$P(< s > \text{ yes no}) = \frac{1}{3}$$

$$P(< s > \text{ no no}) = \frac{1}{3}$$

$$P(< s > \text{ no yes}) = 0$$

$$\sum_{\text{2 word}} P(\dots) = 1$$

End of sentence token </s> - motivation

Corpus

< s > yes no

< s > yes yes

< s > no no

Sentences of length 3:

< s > yes yes yes

< s > yes yes no

...

< s > no no no

$$P(< s > \text{ yes yes yes}) = \dots$$

$$P(< s > \text{ yes yes no}) = \dots$$

$$\dots = \dots$$

$$P(< s > \text{ no no no}) = \dots$$

$$\sum_{\text{3 word}} P(\dots) = 1$$

End of sentence token </s> - motivation

Corpus

< s > yes no

< s > yes yes

< s > no no

bupt
we want this →

$$\sum_{\text{2 word}} P(\dots) + \sum_{\text{3 word}} P(\dots) + \dots = 1$$

End of sentence token </s> - solution

- Bigram

< s > the teacher drinks tea => < s > the teacher drinks tea < /s >

$P(\text{the}|\text{<} \text{s} \text{>})P(\text{teacher}|\text{the})P(\text{drinks}|\text{teacher})P(\text{tea}|\text{drinks})P(\text{<} \text{/s} \text{>}|\text{tea})$

Corpus:

< s > Lyn drinks chocolate < /s >

< s > John drinks < /s >

$$\sum_w C(\text{drinks } w) = 2 \quad \textcircled{1}$$
$$C(\text{drinks}) = 2 \quad \textcircled{2}$$

$$\textcircled{1} = \textcircled{2}$$

End of sentence token </s> for N-grams

- N-gram => just one </s>

E.g. Trigram:

the teacher drinks tea => <s> <s> the teacher drinks tea </s>

Example - bigram

Corpus

< s > Lyn drinks chocolate < /s >



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

< s > John drinks tea < /s >

< s > Lyn eats chocolate < /s >

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

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

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

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

Quiz

Objective: Apply sequence probability approximation with bigrams after adding start and end word.

Question:

Given these conditional probabilities

$$\begin{array}{lll} P(\text{Mary})=0.1; & P(\text{likes})=0.2; & P(\text{cats})=0.3 \\ P(\text{Mary}|\langle s \rangle)=0.2; & P(\langle /s \rangle|\text{cats})=0.6 \\ P(\text{likes}|\text{Mary}) =0.3; & P(\text{cats}|\text{likes})=0.1 \end{array}$$

Approximate the probability of the following sentence with bigrams: " $\langle s \rangle$ Mary likes cats $\langle /s \rangle$ "

Type: Multiple Choice, single answer

Options and solution:

1. $P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle) = 0$

3. $P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle) = 0.003$

2.

$P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle) = 0.0036$

4. $P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle) = 1$

$0.2 \times 0.3 \times 0.1 \times 0.6 = 0.0036$

We usually start and end a sentence with the following tokens respectively: <s> </s>.

When computing probabilities using a unigram, you can append an <s> in the beginning of the sentence. To generalize to an N-gram language model, you can add N-1 start tokens <s>.

For the end of sentence token </s>, you only need one even if it is an N-gram. Here is an example:

Corpus

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

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

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

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

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

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

Make sure you know how to compute the probabilities above!



deeplearning.ai

The N-gram Language Model

Outline

- Count matrix
- Probability matrix
- Language model
- Log probability to avoid underflow
- Generative language model

Count matrix

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

- Rows: unique corpus (N-1)-grams
- Columns: unique corpus words

- Bigram count matrix

“study |” bigram

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

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

Second word

Probability matrix

- Divide each cell by its row sum

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

Count matrix (bigram)

	<s>	</s>	I	study	learn	sum
<s>	0	0	1	0	0	1
</s>	0	0	0	0	0	0
I	0	0	0	1	1	2
study	0	0	1	0	0	1
learn	0	1	0	0	0	1

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

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

Probability matrix

	<s>	</s>	I	study	learn
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	0.5	0.5
study	0	0	1	0	0
learn	0	1	0	0	0

Language model

- probability matrix => language model
 - Sentence probability
 - Next word prediction

	< <i>s</i> >	</ <i>s</i> >	I	study	learn
< <i>s</i> >	0	0	1	0	0
</ <i>s</i> >	0	0	0	0	0
I	0	0	0	0.5	0.5
study	0	0	1	0	0
learn	0	1	0	0	0

Sentence probability:
*<*s*> I learn </*s*>*

$$\begin{aligned}P(\textit{sentence}) &= \\ P(I|<\textit{s}>)P(\textit{learn}|I)P(</\textit{s}>|\textit{learn}) &= \\ 1 \times 0.5 \times 1 &= \\ 0.5\end{aligned}$$

Log probability

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

- All probabilities in calculation ≤ 1 and multiplying them brings risk of underflow

- Logarithm properties reminder

$$\log(a * b) = \log(a) + \log(b)$$

- Use log of the probabilities in Probability matrix and calculations

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

- Converts back from log
 $P(w_1^n) = \exp(\log(P(w_1^n)))$

Generative Language 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 \rightarrow base on Prob
2. Choose next bigram starting with previous word
3. Continue until < /s > is picked

Quiz

Objective: Apply sum when calculating log probability instead of product.

Question:

Given the logarithm of these conditional probabilities:

$$\log(P(\text{Mary}|\langle s \rangle)) = -2; \quad \log(P(\langle /s \rangle|\text{cats})) = -1$$

$$\log(P(\text{likes}|\text{Mary})) = -10; \quad \log(P(\text{cats}|\text{likes})) = -100$$

Approximate the log probability of the following sentence with bigrams : “ $\langle s \rangle$ Mary likes cats $\langle /s \rangle$ ”

Type: Multiple Choice, single answer

Options and solution:

1.  $\log(P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle)) = -113$

2. $\log(P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle)) = 2000$

3. $\log(P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle)) = 113$

4. $\log(P(\langle s \rangle \text{ Mary likes cats } \langle /s \rangle)) = -112$

$$\begin{aligned} & -2 - 10 - 100 - 1 \approx -113 \end{aligned}$$

We covered a lot of concepts in the previous video. You have seen:

- Count matrix
- Probability matrix
- Language model
- Log probability to avoid underflow
- Generative language model

In the count matrix:

- Rows correspond to the unique corpus N-1 grams.
- Columns correspond to the unique corpus words.

Here is an example of the count matrix of a **bigram**.

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

• Bigram count matrix

“study I” bigram →

	<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

To convert it into a probability matrix, you can use the following formula:

- $P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})}$
- $\text{sum}(row) = \sum_{w \in V} C(w_{n-N+1}^{n-1}, w) = C(w_{n-N+1}^{n-1})$

Now given the probability matrix, you can generate the language model. You can compute the sentence probability and the next word prediction.

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

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

To avoid underflow, you can multiply by the log.

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

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



deeplearning.ai

Language Model Evaluation

Outline

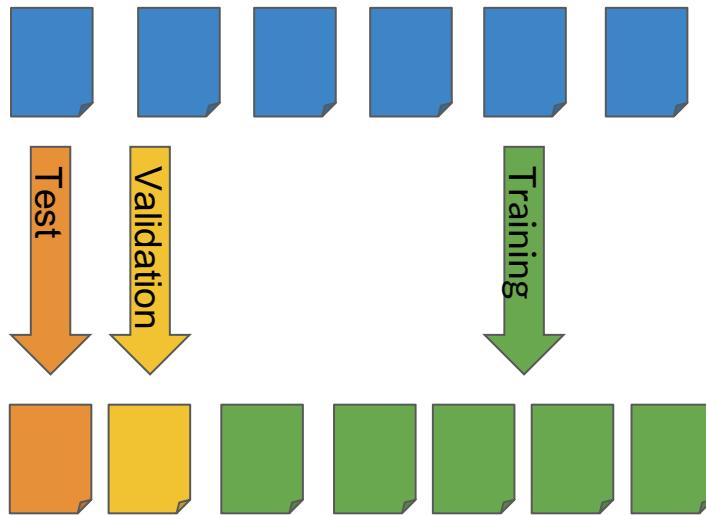
- Train/Validation/Test split
- Perplexity

Test data

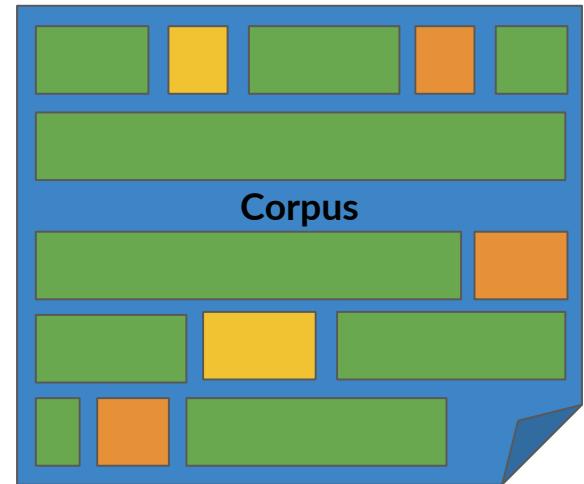
- Split corpus to Train/Validation/Test  Evaluate on Training dataset
- For smaller corpora
 - 80% Train
 - 10% Validation
 - 10% Test
- For large corpora (typical for text)
 - 98% Train
 - 1% Validation
 - 1% Test

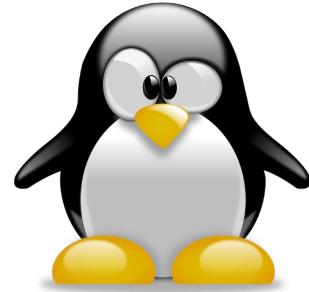
Test data - split method

- Continuous text



- Random short sequences





Perplexity → measure of complexity in a sample of text

↳ closely related to entropy (measure uncertainty)

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

W → test set containing m sentences s

s_i → i-th sentence in the test set, each ending with $\langle /s \rangle$

m → number of all words in entire test set W including

$\langle /s \rangle$ but not including $\langle s \rangle$

text generated by random
→ higher Perplexity

Texts by humans
→ lower Perplexity score

Perplex → confused by some very complex



Perplexity

E.g. m=100 words

$$P(W) = 0.9 \Rightarrow PP(W) = 0.9^{-\frac{1}{100}} = 1.00105416$$

$$P(W) = 10^{-250} \Rightarrow PP(W) = (10^{-250})^{-\frac{1}{100}} \approx 316$$

- Smaller perplexity = better model
Good model / ~ 20 - 60
- Character level models PP < word-based models PP

Perplexity for bigram models

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

$w_j^{(i)}$ → j-th word in i-th sentence

- concatenate all sentences in W

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \frac{1}{P(w_i | w_{i-1})}}$$

w_i → i-th word in test set

Log perplexity

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \frac{1}{P(w_i|w_{i-1})}}$$



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

Examples

Training 38 million words, test 1.5 million words, WSJ corpus
Perplexity Unigram: 962 Bigram: 170 Trigram: 109

Unigram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

Trigram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

[Figure from *Speech and Language Processing* by Dan Jurafsky et. al.]

Sense
more
makes

Quiz

Objective: Calculate log perplexity from log probabilities using sum and correct normalization coefficient (not including $\langle s \rangle$).

Question:

Given the logarithm of these conditional probabilities:

$$\log(P(\text{Mary}|\langle s \rangle)) = -2; \quad \log(P(\langle /s \rangle|\text{cats})) = -1$$

$$\log(P(\text{likes}|\text{Mary})) = -10; \quad \log(P(\text{cats}|\text{likes})) = -100$$

Assuming our test set is $W = \langle s \rangle \text{ Mary likes cats } \langle /s \rangle$, what is the model's perplexity.

Type: Multiple Choice, single answer

$M < 4$

Options and solution:

1. $\log \text{PP}(W) = -113$

3. $\log \text{PP}(W) = (-1/5)*(-113)$

2.

$\log \text{PP}(W) = (-1/4)*(-113)$

4. $\log \text{PP}(W) = (-1/5)*113$

Splitting the Data

We will now discuss the train/val/test splits and perplexity.

Train/Val/Test splits

Smaller Corpora:

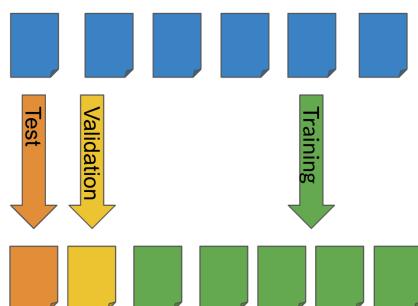
- 80% train
- 10% val
- 10% test

Larger Corpora:

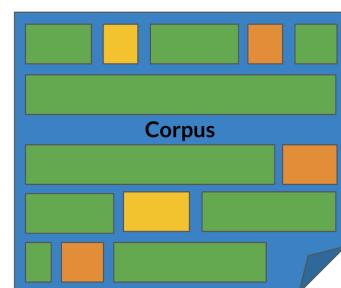
- 98% train
- 1% val
- 1% test

There are two main methods for splitting the data:

- Continuous text



- Random short sequences



Perplexity

Perplexity is used to tell us whether a set of sentences look like they were written by humans rather than by a simple program choosing words at random. A text that is written by humans is more likely to have lower perplexity, where a text generated by random word choice would have a higher perplexity.

Concretely, here are the formulas to calculate perplexity.

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

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

$w_j^{(i)} \rightarrow j$ corresponds to the j th word in the i th sentence. If you were to concatenate all the sentences then w_i is the i th word in the test set. To compute the log perplexity, you go from

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \frac{1}{P(w_i | w_{i-1})}}$$

To

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



deeplearning.ai

Out of Vocabulary Words

Outline

- Unknown words
- Update corpus with <UNK>
- Choosing vocabulary

Out of vocabulary words

- Closed vs. Open vocabularies
- Unknown word = Out of vocabulary word (OOV)
- special tag <UNK> in corpus and in input

Using <UNK> in corpus

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

Example

Corpus

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

6>

Corpus

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

Min frequency f=2

Vocabulary

Lyn, drinks, chocolate

$$\text{freq} \geq 2$$

Input query

< s > Adam drinks chocolate </ s >

< s > < UNK > drinks chocolate </ s >

How to create vocabulary V

- Criteria:
 - Min word frequency f (Usually small number)
 - Max $|V|$, include words by frequency
- Use <UNK> sparingly
 - size of vocab
 - highest freqs
- Perplexity - only compare LMs with the same V

Using UNK will lower perplexity but if we have too many UNKs, the model would generate a seq of UNK words

Quiz

Objective: Create corpus vocabulary based on minimum frequency.

Question:

Given the training corpus and minimum word frequency=2, how would the vocabulary for corpus preprocessed with <UNK> look like?

“<s> I am happy I am learning </s> <s> I am happy I can study </s>”

Type: Multiple Choice, single answer

Options and solution:

1. $V = (\text{I}, \text{am}, \text{happy})$

2. $V = (\text{I}, \text{am}, \text{happy}, \text{learning}, \text{can}, \text{study})$

3. $V = (\text{I}, \text{am}, \text{happy}, \text{I}, \text{am})$

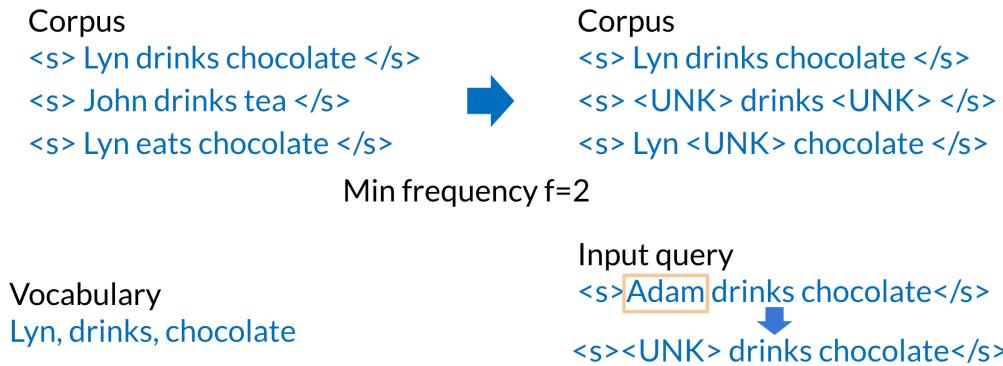
4. $V =$
 $(\text{I}, \text{am}, \text{happy}, \text{learning}, \text{can}, \text{study}, \text{<UNK>})$

Many times, you will be dealing with unknown words in the corpus. So how do you choose your vocabulary? What is a vocabulary?

A vocabulary is a set of unique words supported by your language model. In some tasks like speech recognition or question answering, you will encounter and generate words only from a fixed set of words. Hence, a **closed vocabulary**.

Open vocabulary means that you may encounter words from outside the vocabulary, like a name of a new city in the training set. Here is one recipe that would allow you to handle unknown words.

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



The example above shows how you can use *min_frequency* and replace all the words that show up fewer times than *min_frequency* by UNK. You can then treat UNK as a regular word.

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



deeplearning.ai

Smoothing

Outline

- Missing N-grams in corpus
- Smoothing
- Backoff and interpolation

Missing N-grams in training corpus

- Problem: N-grams made of known words still might be missing in the training corpus “John”, “eats” in corpus  “John eats”
- Their counts cannot be used for probability estimation

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

← Can be 0

Smoothing

would only work if
Count > +1

- Advanced methods:
Kneser-Ney smoothing
Good-Turing smoothing

- Add-one smoothing (Laplacian smoothing)

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

- Add-k smoothing

$$\rightarrow \text{for large corpus}$$
$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{\sum_{w \in V} (C(w_{n-1}, w) + k)} = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + k * V}$$

Makes Probs smoother

Backoff

- If N-gram missing => use (N-1)-gram, ...
 - Probability discounting e.g. Katz backoff
 - “Stupid” backoff

N-2 grams and so on...

Corpus

< s > Lyn drinks chocolate < /s >

< s > John drinks tea < /s >

< s > Lyn eats chocolate < /s >

$$P(\text{chocolate} | \text{John drinks}) = ?$$

Experimentally shown to work well



$$0.4 \times P(\text{chocolate} | \text{drinks})$$

Interpolation

$$\begin{aligned}\hat{P}(\text{chocolate} | \text{John drinks}) &= 0.7 \times P(\text{chocolate} | \text{John drinks}) \\ &\quad + 0.2 \times P(\text{chocolate} | \text{drinks}) + 0.1 \times P(\text{chocolate})\end{aligned}$$

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

Learned from validation data corpus $\sum_i \lambda_i = 1$

Quiz

Objective: Apply n-gram probability with add-k smoothing for phrase not present in the corpus.

Question:

Corpus: "I am happy I am learning"

$$|V| \approx 4$$

In the context of our corpus, what is the estimated probability of word "can" following the word "I" using the bigram model and add-k-smoothing where k=3.

$$\frac{0+3}{2+3*4} = \frac{3}{2+3*4}$$

Type: Multiple Choice, single answer

Options and solution:

1. $P(\text{can}|I) = 0$

2. $P(\text{can}|I) = 1$

3. ✓ $P(\text{can}|I) = 3/(2+3*4)$

4. $P(\text{can}|I) = 3/(3*4)$

The three main concepts covered here are dealing with missing n-grams, smoothing, and Backoff and interpolation.

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

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

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

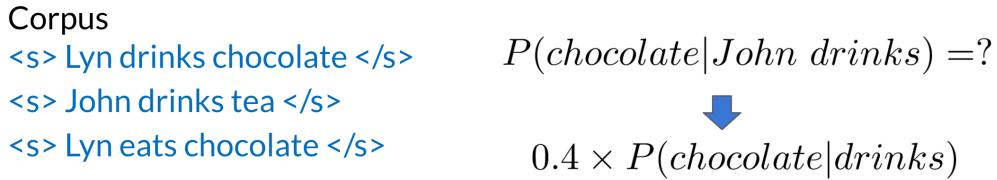
Add-k smoothing is very similar:

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

When using 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.

Here is a visualization:



You can also use interpolation when computing probabilities as follows:

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

Where

$$\sum_i \lambda_i = 1$$



deeplearning.ai

Week Summary

Summary

- N-Grams and probabilities
- Approximate sentence probability from N-Grams
- Build language model from corpus
- Fix missing information
 - Out of vocabulary words with <UNK>
 - Missing N-Gram in corpus with smoothing, backoff and interpolation
- Evaluate language model with perplexity
- Coding assignment!

This week you learned the following concepts

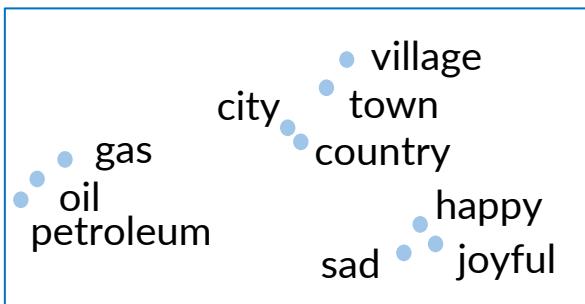
- N-Grams and probabilities
- Approximate sentence probability from N-Grams
- Build a language model from a corpus
- Fix missing information
- Out of vocabulary words with <UNK>
- Missing N-Gram in corpus with smoothing, backoff and interpolation
- Evaluate language model with perplexity
- Coding assignment!



deeplearning.ai

Overview

Some basic applications of word embeddings



Semantic analogies
and similarity



Sentiment analysis

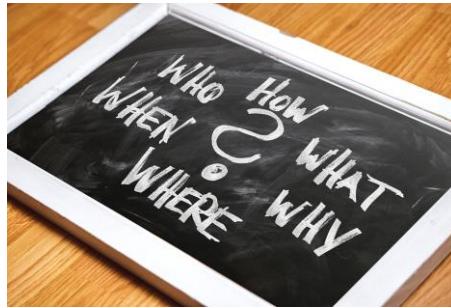


Classification of
customer feedback

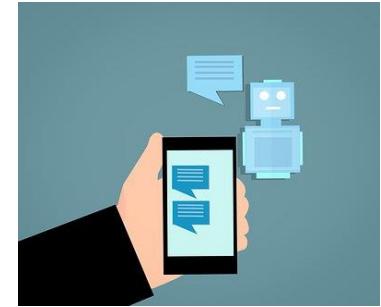
Advanced applications of word embeddings



Machine translation



Information extraction



Question answering

Learning objectives

Prerequisite: neural networks

- Identify the key concepts of word representations
- Generate word embeddings
- Prepare text for machine learning
- Implement the continuous bag-of-words model

Word embeddings are used in most NLP applications. Whenever you are dealing with text, you first have to find a way to encode the words as numbers. Word embedding are a very common technique that allows you to do so. Here are a few applications of word embeddings that you should be able to implement by the time you complete the specialization.



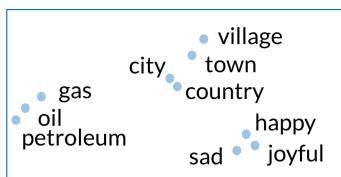
Machine translation



Information extraction



Question answering



Semantic analogies
and similarity



Sentiment analysis



Classification of
customer feedback

By the end of this week you will be able to:

- Identify the key concepts of word representations
- Generate word embeddings
- Prepare text for machine learning
- Implement the continuous bag-of-words model



deeplearning.ai

Basic Word Representations

Outline

- Integers
- One-hot vectors
- Word embeddings

Integers

Assigns unique int
to each word

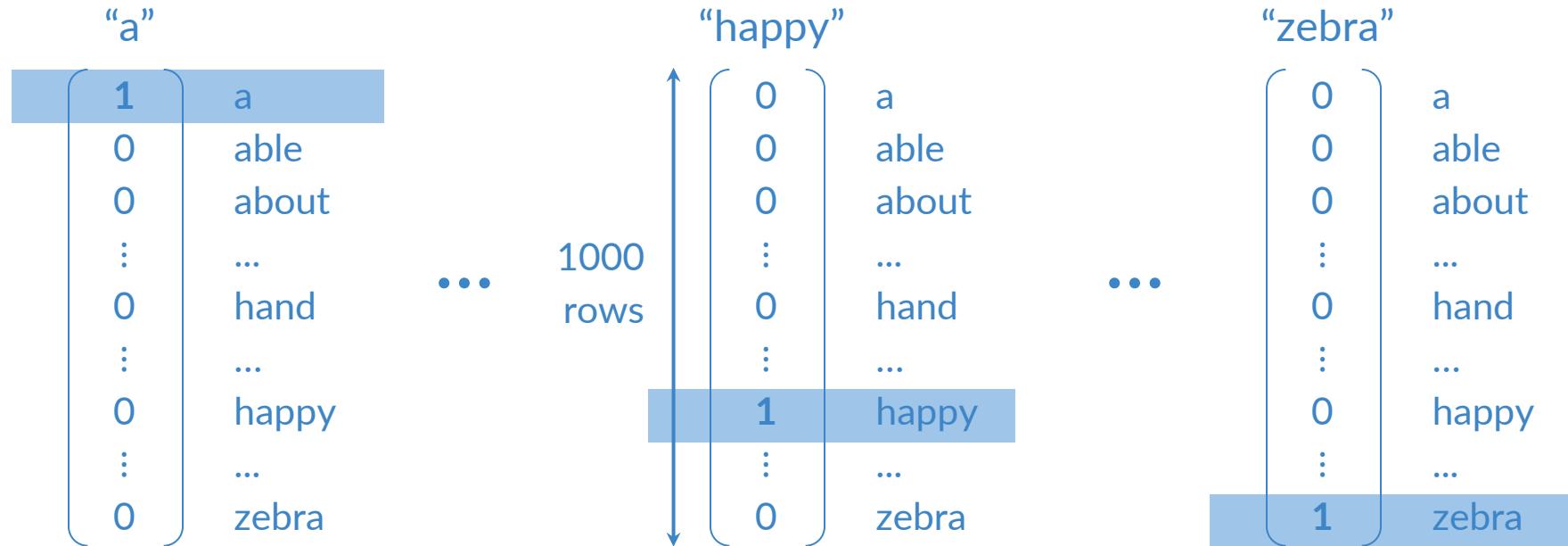
Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

Integers

- + Simple
- Ordering: little semantic sense

hand < happy < zebra
615 621 1000
?! ?!

One-hot vectors



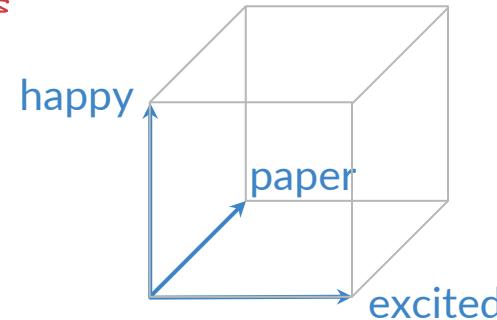
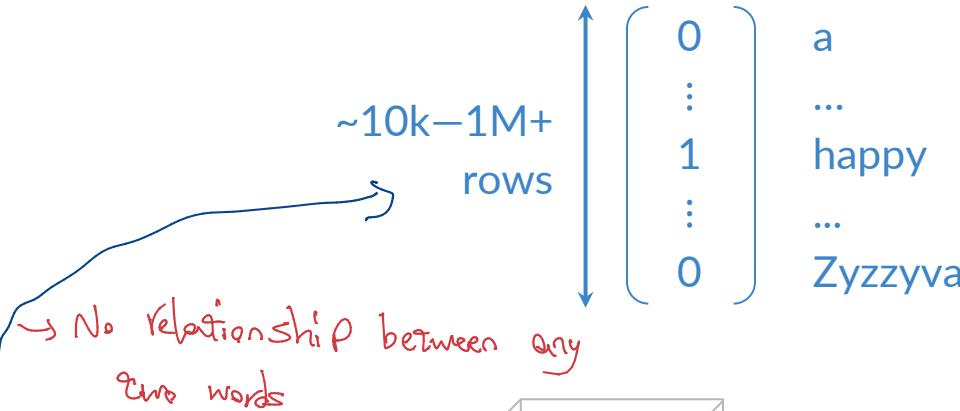
One-hot vectors

Integer \leftrightarrow One-hot vectors

Word	Number		"happy"	
a	1		1	0
able	2		2	0
about	3		3	0
...	⋮
hand	615		615	0
...	⋮
happy	621	↔	1	happy
...	...		⋮	...
zebra	1000		1000	0

One-hot vectors

- + Simple
- + No implied ordering
- Huge vectors
- No embedded meaning



$$\begin{aligned}d(\text{paper}, \text{excited}) \\= d(\text{paper}, \text{happy}) \\= d(\text{excited}, \text{happy})\end{aligned}$$

Basic word representations could be classified into the following:

- Integers
- One-hot vectors
- Word embeddings

Word	Number		"happy"	
a	1		1 [0]	a
able	2		2 [0]	able
about	3		3 [0]	about
... [:]	...
hand	615		615 [0]	hand
... [:]	...
happy	621	←→	621 [1]	happy
... [:]	...
zebra	1000		1000 [0]	zebra

To the left, you have an example where you use integers to represent a word. The issue there is that there is no reason why one word corresponds to a bigger number than another. To fix this problem we introduce one hot vectors (diagram on the right). To implement one hot vectors, you have to initialize a vector of **zeros** of dimension V and then put a 1 in the index corresponding to the word you are representing.

The **pros** of one-hot vectors: simple and require no implied ordering.

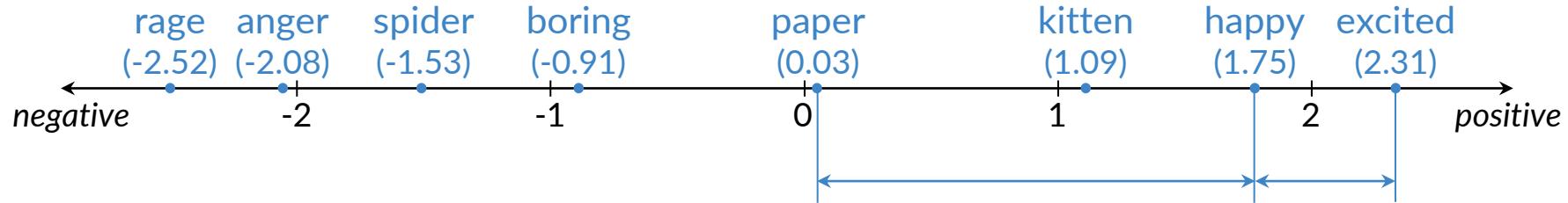
The **cons** of one-hot vectors: huge and encode no meaning.



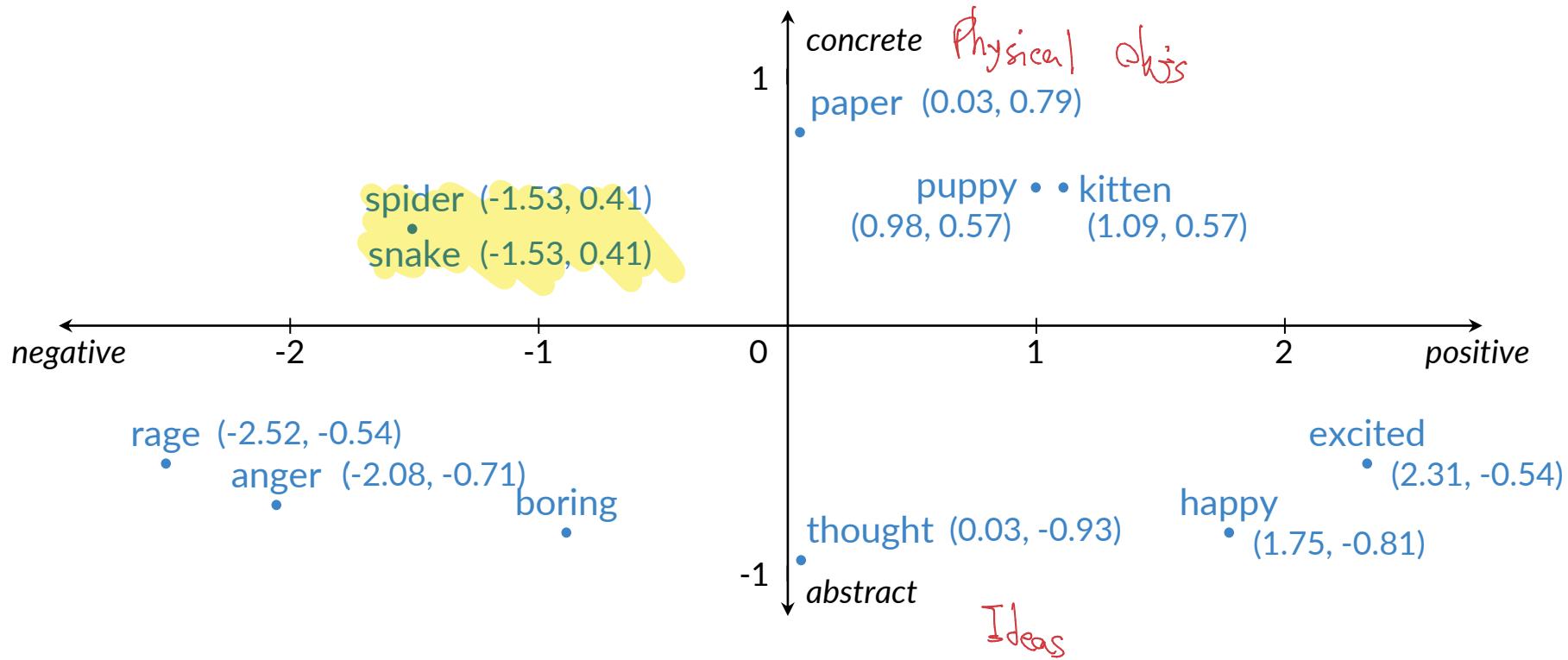
deeplearning.ai

Word Embeddings

Meaning as vectors



Meaning as vectors



Word embedding vectors

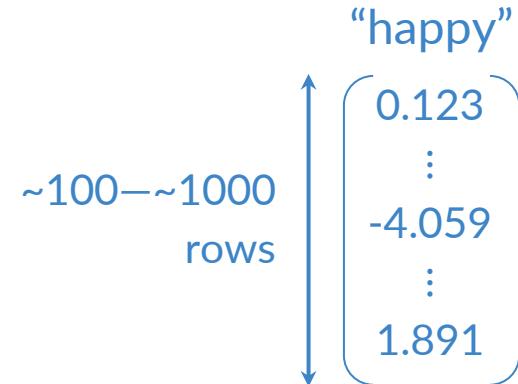
- + Low dimension
- + Embed meaning
 - o e.g. semantic distance

forest \approx tree forest \approx ticket

- o e.g. analogies

Paris:France :: Rome:?

Talk



Terminology

integers

word vectors

one-hot vectors

both ↗
are ↙

word embedding vectors

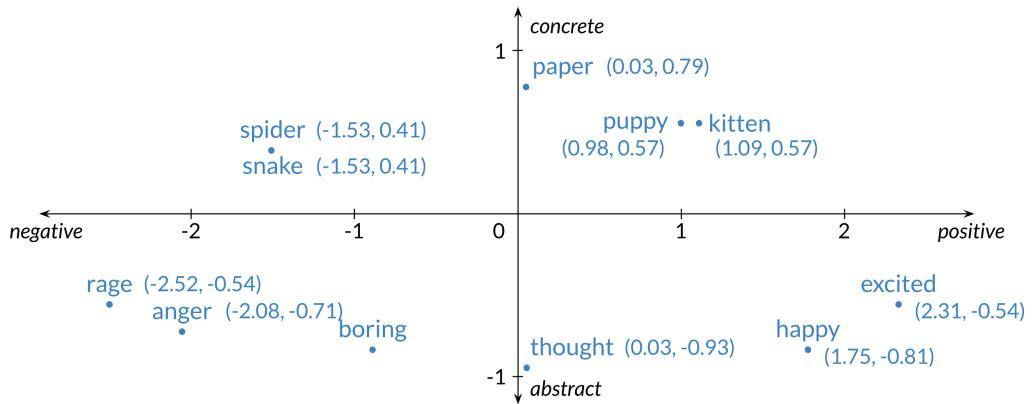
“word vectors”

word embeddings

Summary

- Words as integers
- Words as vectors
 - One-hot vectors
 - Word embedding vectors
- Benefits of word embeddings for NLP

So why use word embeddings? Let's take a look.



From the plot above, you can see that when encoding a word in 2D, similar words tend to be found next to each other. Perhaps the first coordinate represents whether a word is positive or negative. The second coordinate tell you whether the word is abstract or concrete. This is just an example, in the real world you will find embeddings with hundreds of dimensions. You can think of each coordinate as a number telling you something about the word.

The pros :

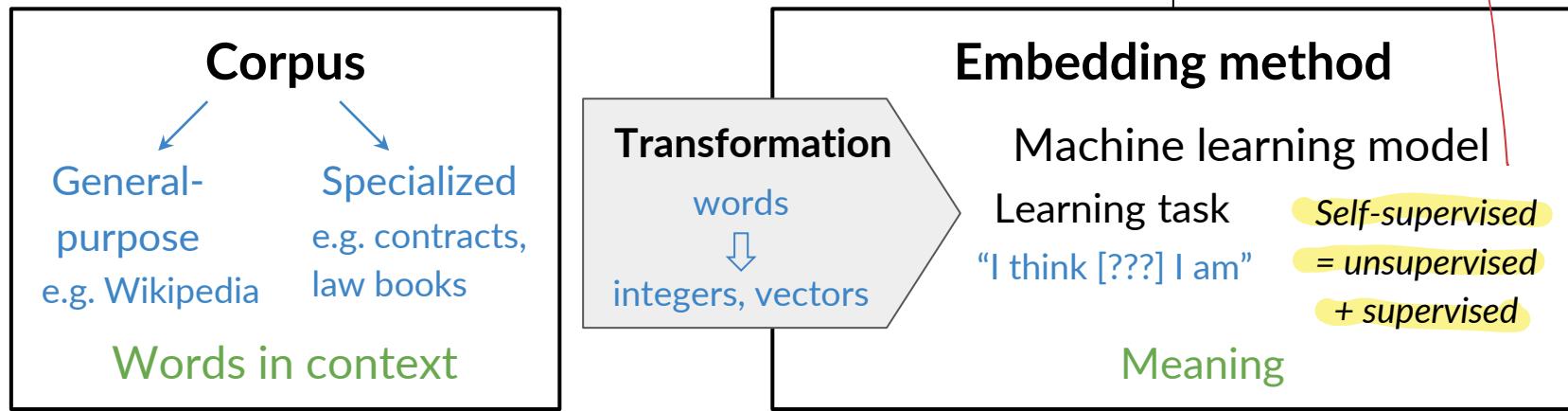
- Low dimensions (less than V)
- Allow you to encode meaning



deeplearning.ai

How to Create Word Embeddings

Word embedding process



↳ Organized in the same way as they would be used in the context of interest



To create word embeddings you always need a corpus of text, and an embedding method.

The context of a word tells you what type of words tend to occur near that specific word. The context is important as this is what will give meaning to each word embedding.

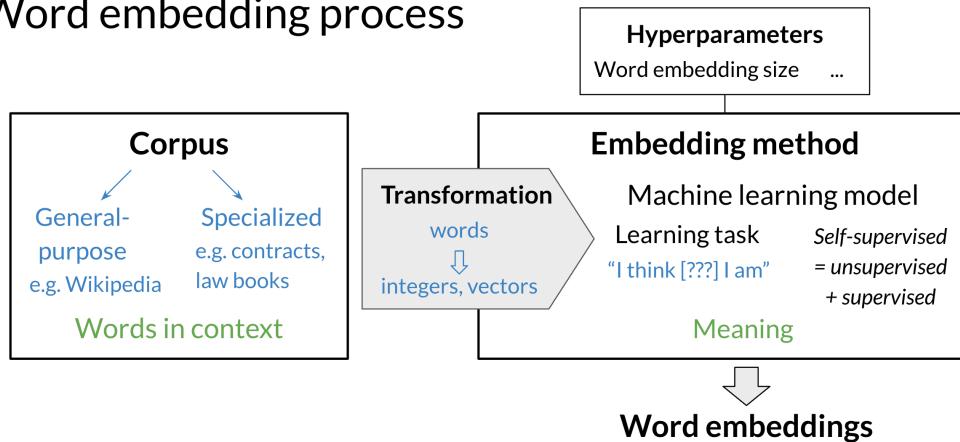
Embeddings

There are many types of possible methods that allow you to *learn* the word embeddings. The machine learning model performs a learning task, and the main by-products of this task are the word embeddings. The task could be to learn to predict a word based on the surrounding words in a sentence of the corpus, as in the case of the continuous bag-of-words.

The task is **self-supervised** : it is both unsupervised in the sense that the input data — the corpus — is unlabelled, and supervised in the sense that the data itself provides the necessary context which would ordinarily make up the labels.

When training word vectors, there are some parameters you need to tune. (i.e. the dimension of the word vector)

Word embedding process





deeplearning.ai

Word Embedding Methods

Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)
 - Continuous skip-gram / Skip-gram with negative sampling (SGNS)
- Global Vectors (GloVe) (Stanford, 2014)
- fastText (Facebook, 2016)
 - Supports out-of-vocabulary (OOV) words

Advanced word embedding methods

Deep learning, contextual embeddings

- BERT (Google, 2018)
 - ELMo (Allen Institute for AI, 2018)
 - GPT-2 (OpenAI, 2018)
- 
- Tunable pre-trained
models available

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.

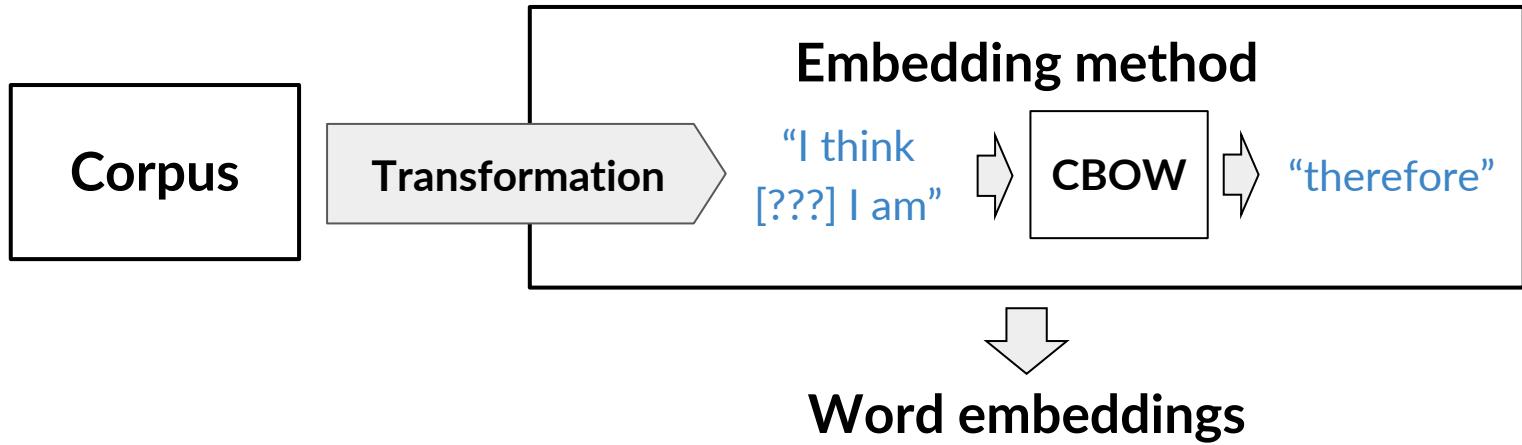
- BERT (Google, 2018):
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)



deeplearning.ai

Continuous Bag-of-Words Model

Continuous bag-of-words word embedding process





Center word prediction: rationale

The little _____ ? is barking



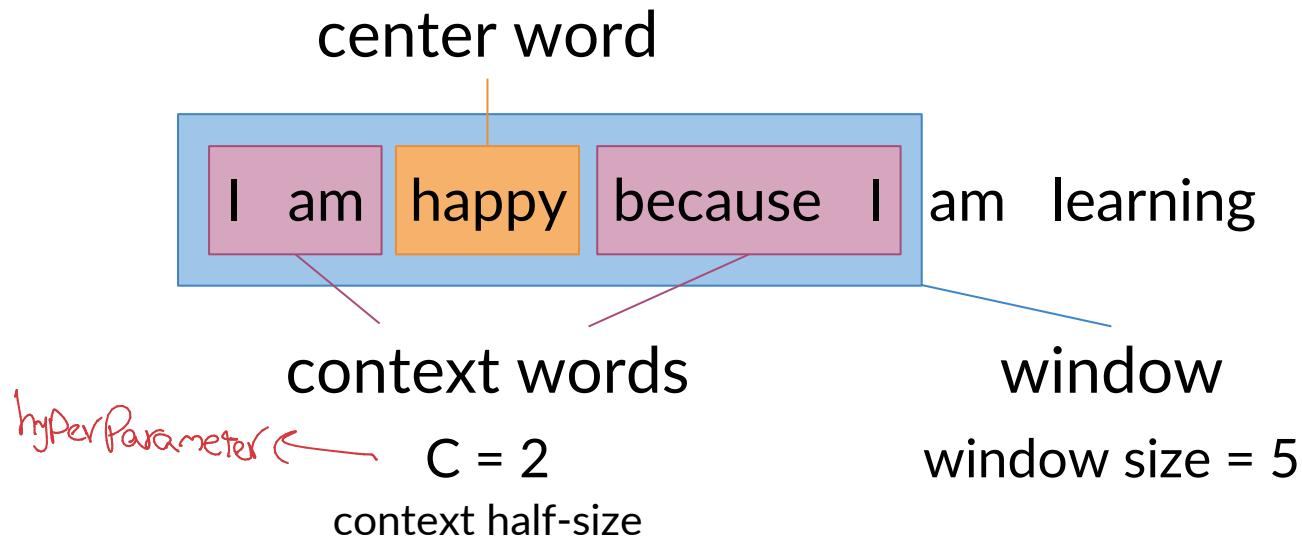
dog
puppy
hound
terrier

...

They are related
Semantically

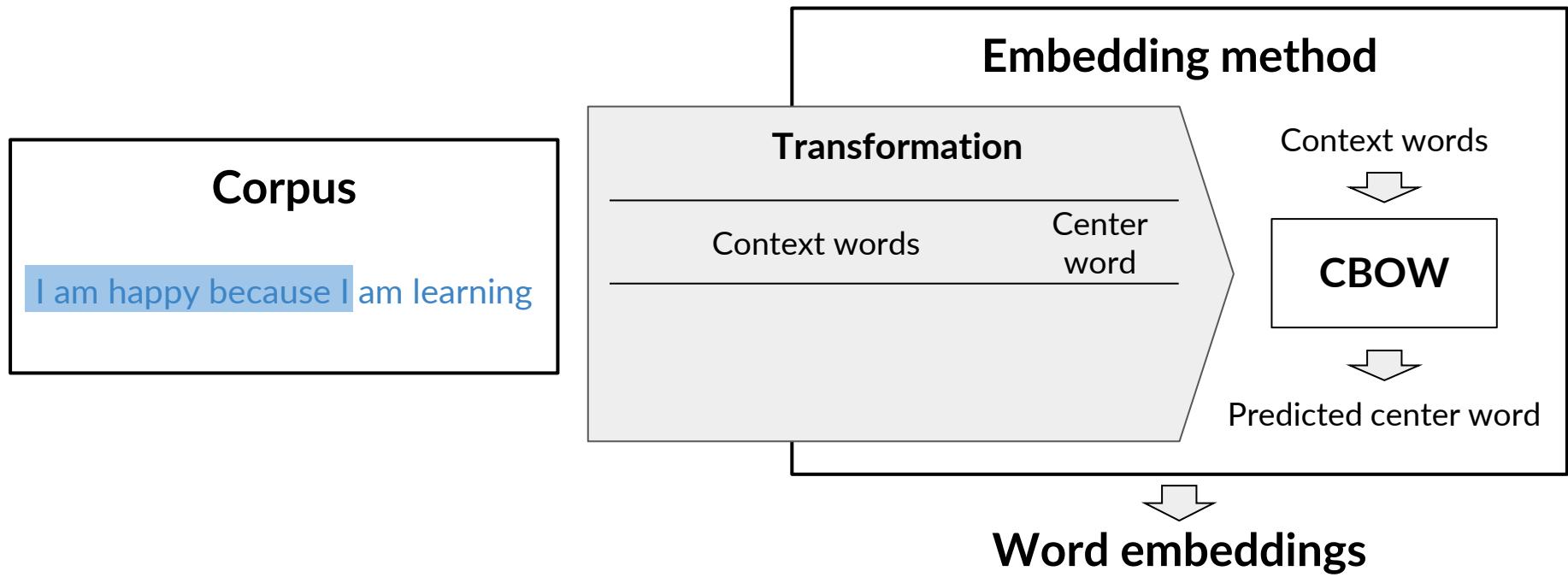


Creating a training example



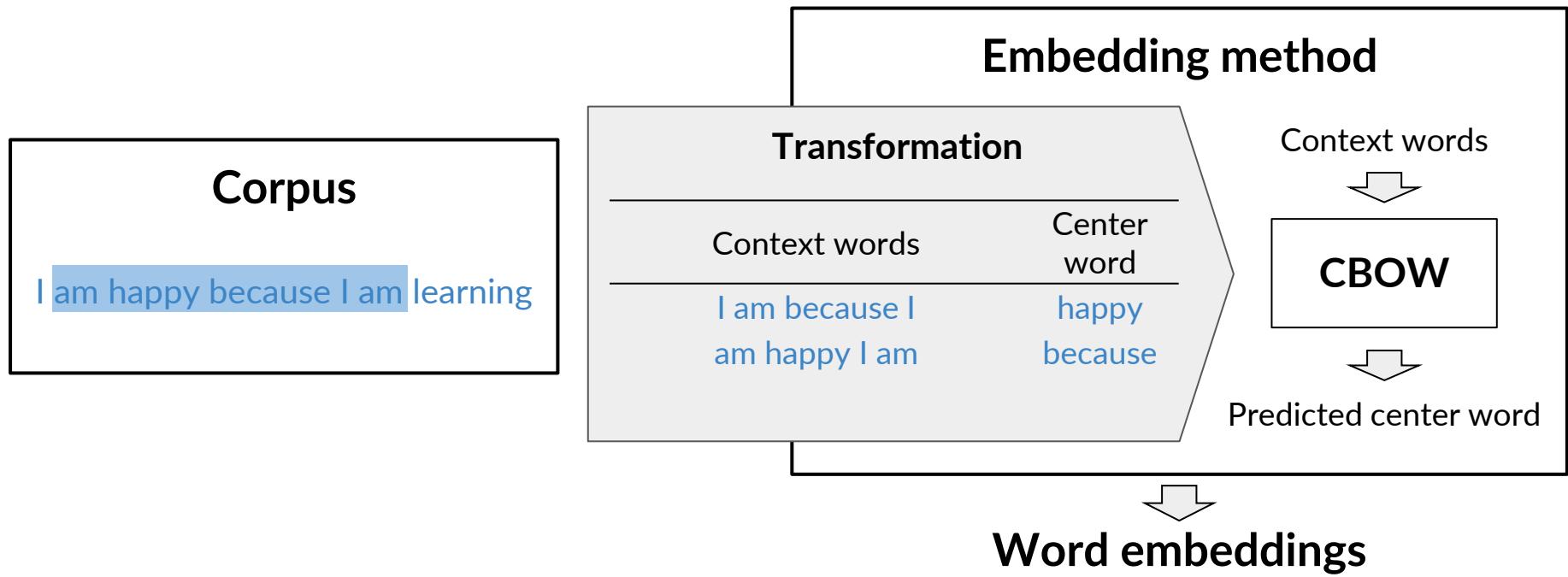


From corpus to training





From corpus to training

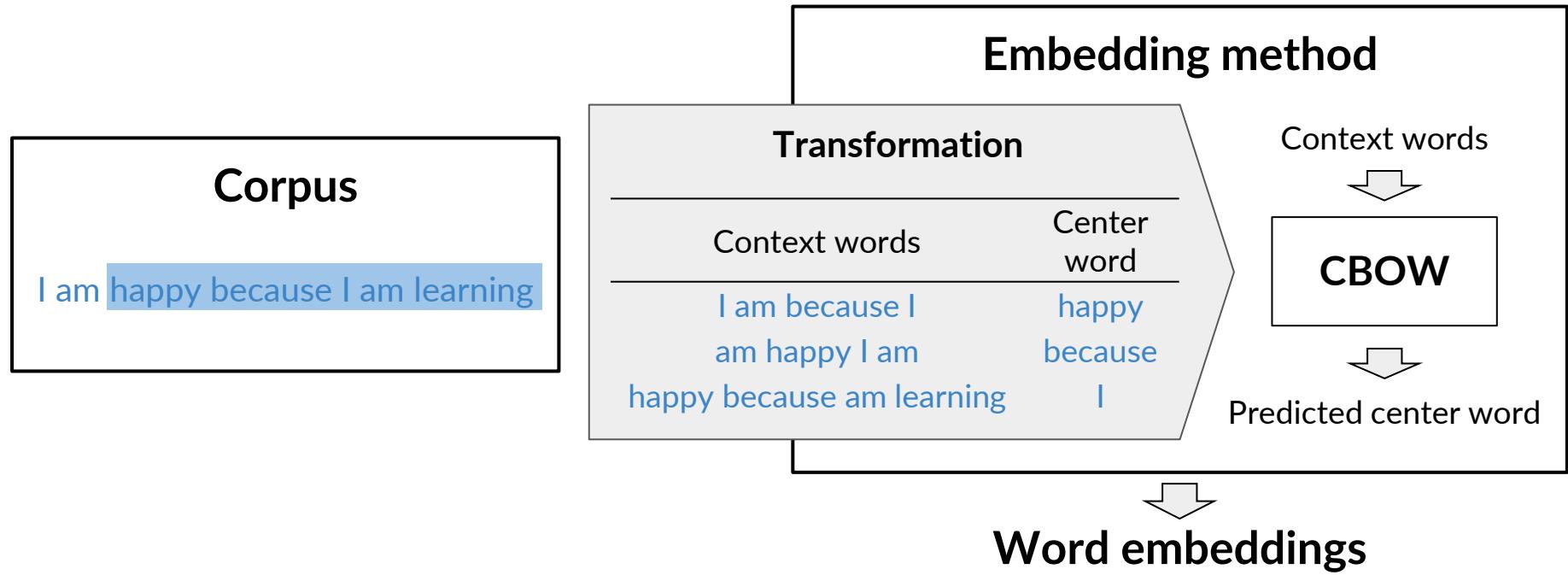


Corpus

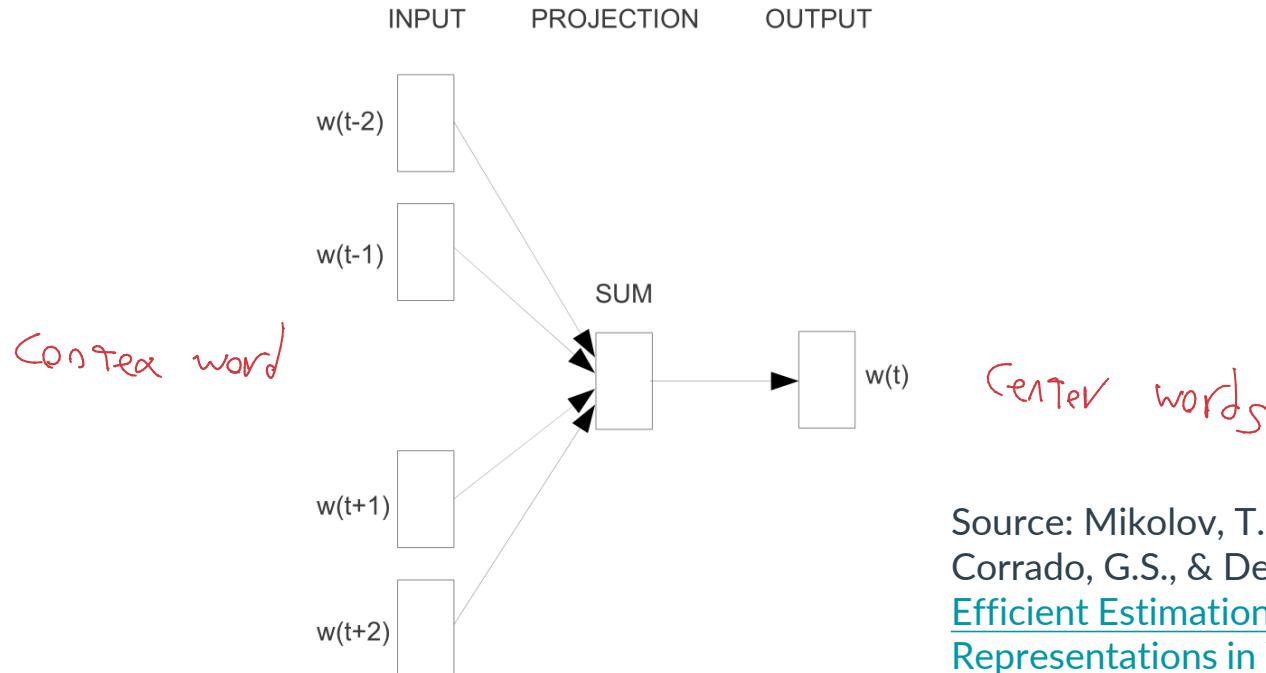
Transformation

CBOW

From corpus to training

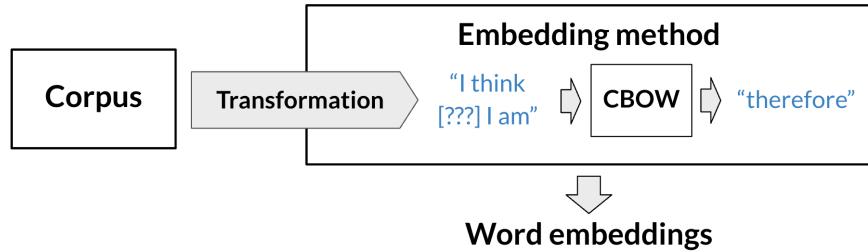


CBOW in a nutshell

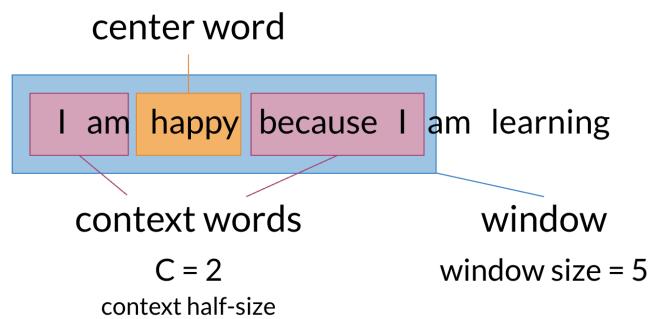


Source: Mikolov, T., Chen, K., Corrado, G.S., & Dean, J. (2013).
[Efficient Estimation of Word Representations in Vector Space](#)

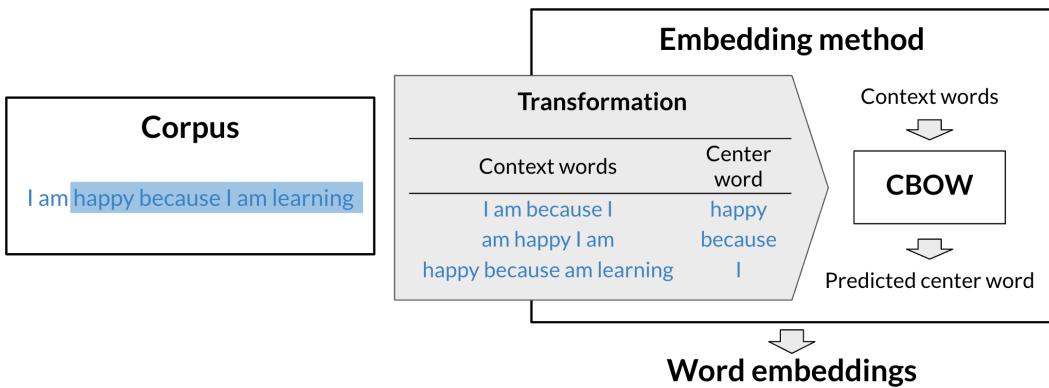
To create word embeddings, you need a corpus and a learning algorithm. The by-product of this task would be a set of word embeddings. In the case of the continuous bag-of-words model, the objective of the task is to predict a missing word based on the surrounding words.



Here is a visualization that shows you how the model works.



As you can see, the window size in the image above is 5. The context size, C, is 2. C usually tells you how many words before or after the center word the model will use to make the prediction. Here is another visualization that shows an overview of the model.





deeplearning.ai

Cleaning and Tokenization

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / uppercase*

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / uppercase*
- Punctuation , ! . ? → . “ ‘ ‘ ‘ ’ ” → ∅ ... !! ??? → .

↪ *Ignore*

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / uppercase*
- Punctuation , ! . ? → . “ ‘ ‘ ‘ ’ ” → ∅ ... !! ??? → .
- Numbers 1 2 3 5 8 → ∅ 3.14159 90210 → *as is / <NUMBER>*
 or ↗

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / uppercase*
- Punctuation , ! . ? → . “ ‘ ‘ ‘ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → *as is / <NUMBER>*
- Special characters ₣ \$ € § ¶ ** → Ø

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / uppercase*
- Punctuation , ! . ? → . “ ‘ ‘ ‘ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → *as is / <NUMBER>*
- Special characters ₣ \$ € § ¶ ** → Ø
- Special words 😊 #nlp → :happy: #nlp

Example in Python: corpus

Who ❤️ "word embeddings" in 2020? I do!!!

emoji

punctuation

number

Example in Python: libraries

```
# pip install nltk
# pip install emoji

import nltk
from nltk.tokenize import word_tokenize
import emoji

nltk.download('punkt') # download pre-trained Punkt tokenizer for English
```

Example in Python: code

```
corpus = 'Who ❤ "word embeddings" in 2020? I do!!!'  
  
data = re.sub(r'[,!?;-]+', '.', corpus)
```

→ Who ❤ "word embeddings" in 2020. I do.

Example in Python: code

```
corpus = 'Who ❤ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words

→ ['Who', '❤', '', 'word', 'embeddings', "", 'in', '2020', '.', 'I',
'do', '.']
```

Example in Python: code

```
corpus = 'Who ❤ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words
data = [ ch.lower() for ch in data
         if ch.isalpha()
         or ch == '.'
         or emoji.get_emoji_regexp().search(ch)
     ]
→ ['who', '❤', 'word', 'embeddings', 'in', '.', 'i', 'do', '.']
```

Before implementing any natural language processing algorithm, you might want to clean the data and tokenize it. Here are a few things to keep track of when handling your data.

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

You can clean data using python as follows:

```
corpus = 'Who ❤️ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words
data = [ ch.lower() for ch in data
        if ch.isalpha()
        or ch == '.'
        or emoji.get_emoji_regexp().search(ch)
    ]
→ ['who', '❤️', 'word', 'embeddings', 'in', '.', 'i', 'do', '.']
```

You can add as many conditions as you want in the lines corresponding to the green rectangle above.

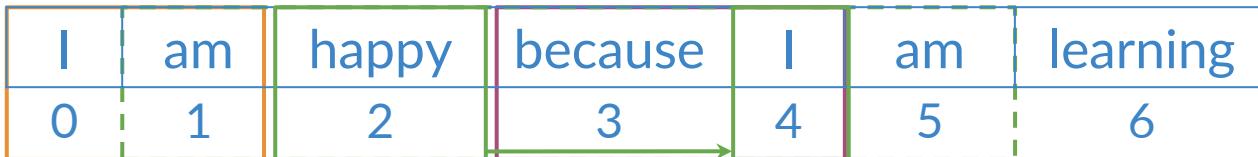


deeplearning.ai

Sliding Window of Words in Python

Sliding window of words in Python

```
def get_windows(words, C):    →generator func  
    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  
would return ←  
and then would continue to run if more values is needed
```



Sliding window of words in Python

```
def get_windows(words, C):
    ...
    yield context_words, center_word
```

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

Sliding window of words in Python

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

```
→ ['I', 'am', 'because', 'I']          happy  
['am', 'happy', 'I', 'am']              because  
['happy', 'because', 'am', 'learning']   I
```

```

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

```

I	am	happy	because	I	am	learning
0	1	2	3	4	5	6

The code above shows you a function which takes in two parameters.

- Words: a list of words.
- C: the context size.

We first start by setting i to C . Then we single out the `center_word`, and the `context_words`. We then yield those and increment i .



deeplearning.ai

Transforming Words into Vectors

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot vector	am	because	happy	I	learning
am	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$
because	$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
happy	$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
I	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
learning	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Transforming context words into vectors

Average of individual one-hot vectors

$$\left(\begin{array}{c} I \\ am \\ because \\ happy \\ learning \end{array} \right) + \left(\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right) / 4 = \left(\begin{array}{c} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{array} \right)$$

Final prepared training set

Context words	Context words vector	Center word	Center word vector
<i>I am because I</i>	[0.25; 0.25; 0; 0.5; 0]	<i>happy</i>	[0; 0; 1; 0; 0]

To transform the context vectors into one single vector, you can use the following.

$$\left(\begin{array}{c} \text{I} \\ \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{array} \right) \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right) + \left(\begin{array}{c} \text{am} \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} \text{because} \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} \text{I} \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right) \right) / 4 = \left(\begin{array}{c} \text{I am because I} \\ 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{array} \right)$$

As you can see, we started with one-hot vectors for the context words and we transform them into a single vector by taking an average. As a result you end up having the following vectors that you can use for your training.

Context words	Context words vector	Center word	Center word vector
<i>I am because I</i>	[0.25; 0.25; 0; 0.5; 0]	<i>happy</i>	[0; 0; 1; 0; 0]



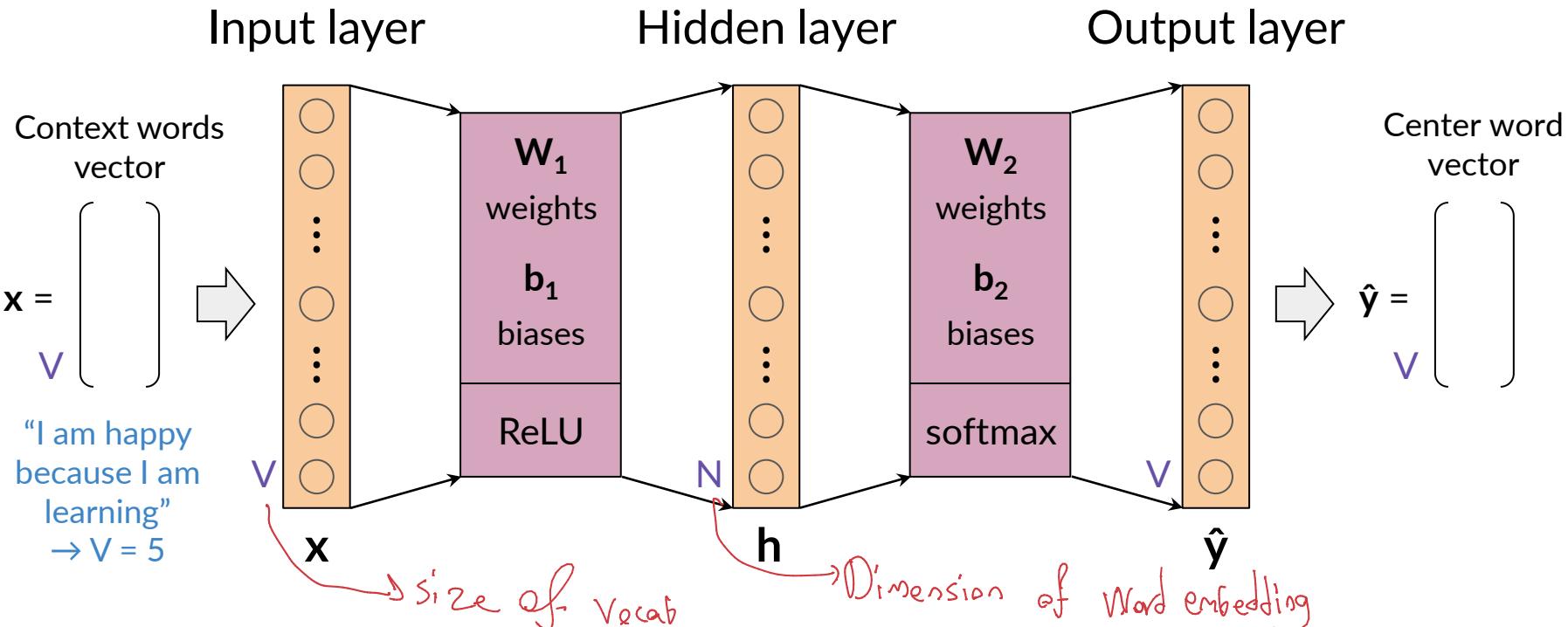
deeplearning.ai

Architecture of the CBOW Model

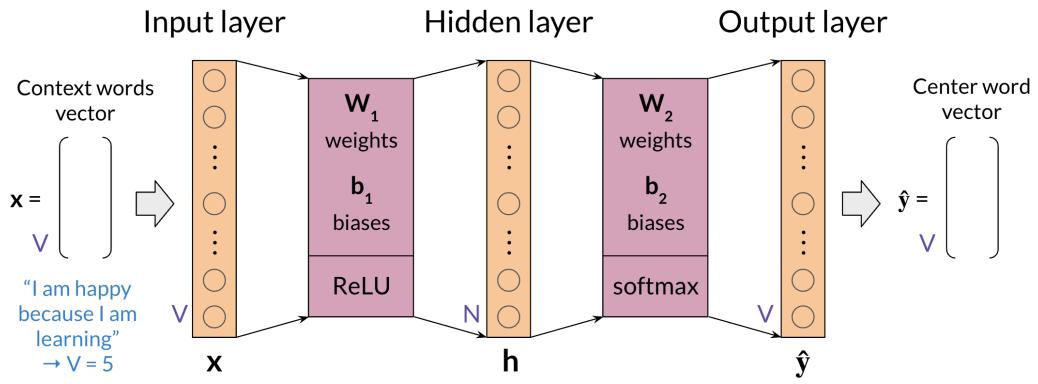
Architecture of the CBOW model

Hyperparameters

N : Word embedding size ...



The architecture for the CBOW model could be described as follows



You have an input, X , which is the average of all context vectors. You then multiply it by W_1 and add b_1 . The result goes through a ReLU function to give you your hidden layer. That layer is then multiplied by W_2 and you add b_2 . The result goes through a softmax which gives you a distribution over V vocabulary words. You pick the vocabulary word that corresponds to the arg-max of the output.

★ we use Softmax because we have V possible outcomes

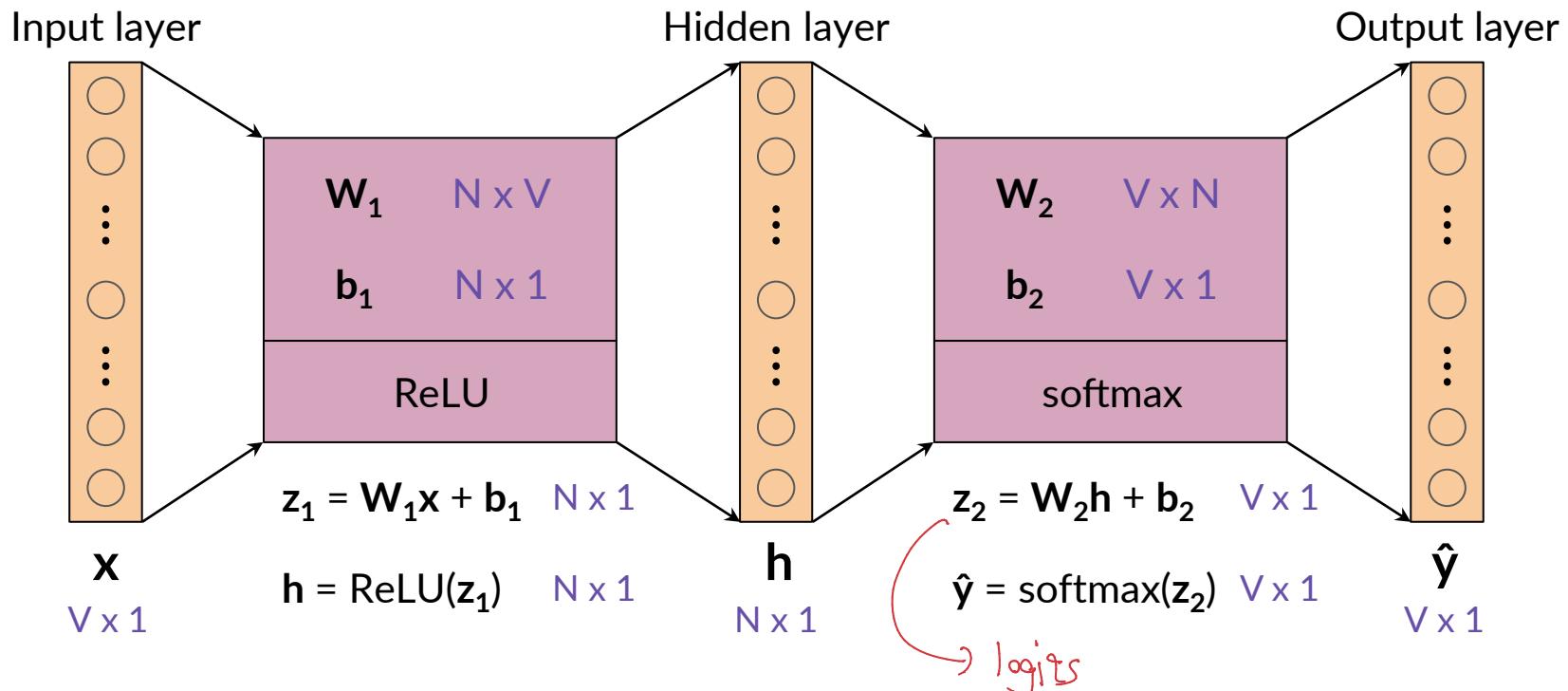


deeplearning.ai

Architecture of the CBOW Model:

Dimensions

Dimensions (single input)



Dimensions (single input)

Column vectors

$$\mathbf{z}_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$\mathbf{z}_1 = \begin{pmatrix} \end{pmatrix} \quad \mathbf{W}_1 = \begin{pmatrix} & N \times V \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} \end{pmatrix} \quad \mathbf{b}_1 = \begin{pmatrix} \end{pmatrix}$$

$N \times 1$ $V \times 1$ $N \times 1$

Row vectors

$$\mathbf{z}_1 = \mathbf{x} \mathbf{W}_1^T + \mathbf{b}_1$$

$$\mathbf{b}_1 = \begin{pmatrix} 1 \times N \end{pmatrix} \quad \mathbf{W}_1 = \begin{pmatrix} & N \times V \end{pmatrix} \quad \mathbf{b}_1 = \begin{pmatrix} 1 \times N \end{pmatrix}$$
$$\mathbf{x} = \begin{pmatrix} 1 \times V \end{pmatrix} \quad \mathbf{w}_1^T \in \begin{pmatrix} V \times N \end{pmatrix}$$

The equations for the previous model are:

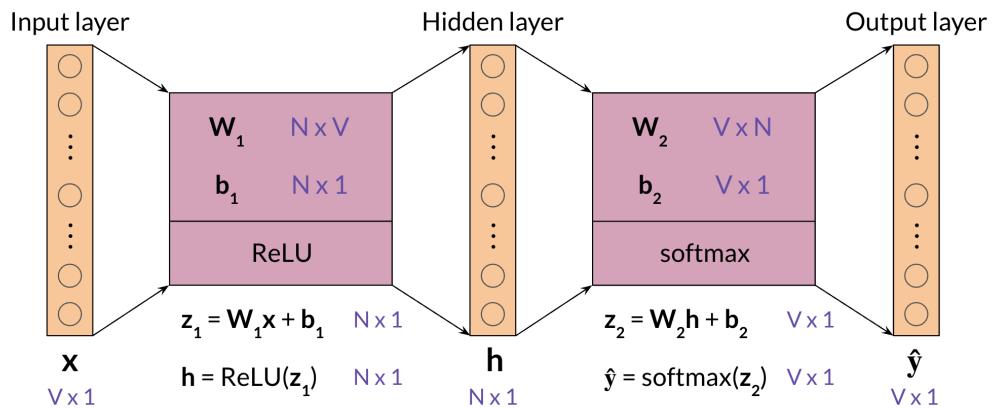
$$z_1 = W_1x + b_1$$

$$h = \text{ReLU}(z_1)$$

$$z_2 = W_2h + b_2$$

$$\hat{y} = \text{softmax}(z_2)$$

Here, you can see the dimensions:



Make sure you go through the matrix multiplications and understand why the dimensions make sense.



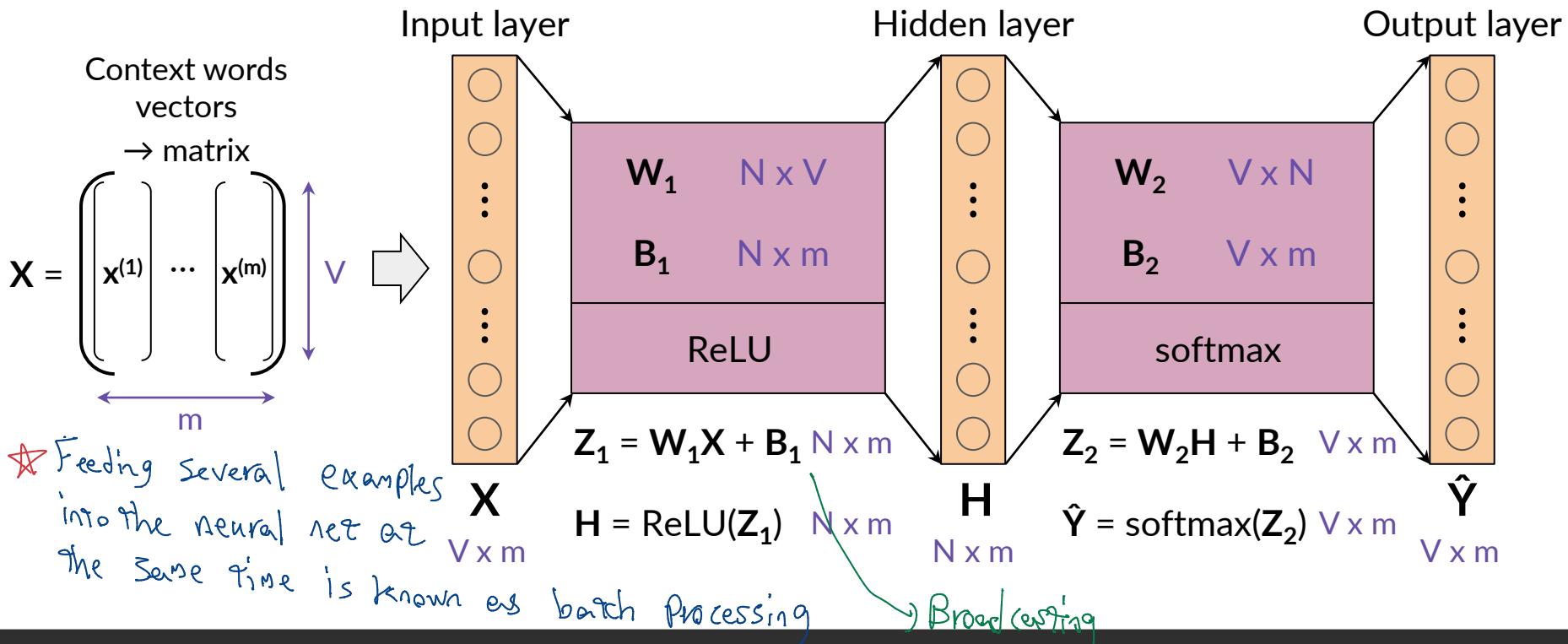
deeplearning.ai

Architecture of the CBOW Model:

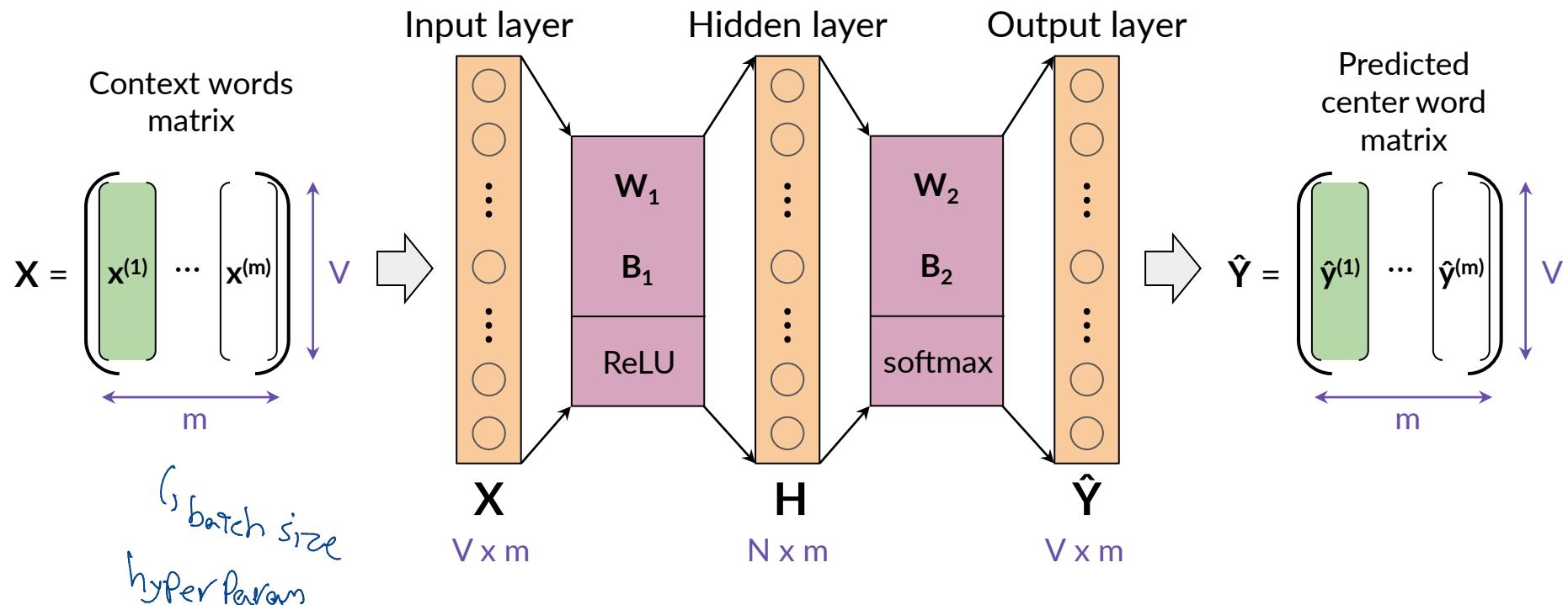
Dimensions 2

Dimensions (batch input)

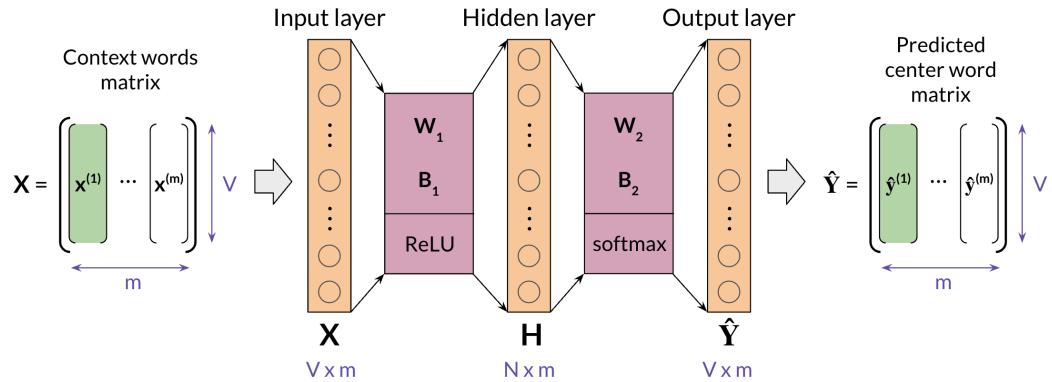
$$\begin{bmatrix} \mathbf{b}_1 \end{bmatrix} \rightarrow \mathbf{B}_1 = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_1 \end{pmatrix}_{m \times N} \quad \text{broadcasting}$$



Dimensions (batch input)



When dealing with batch input, you can stack the examples as columns. You can then proceed to multiply the matrices as follows:



In the diagram above, you can see the dimensions of each matrix. Note that your \hat{Y} is of dimension V by m . Each column is the prediction of the column corresponding to the context words. So the first column in \hat{Y} is the prediction corresponding to the first column of X .

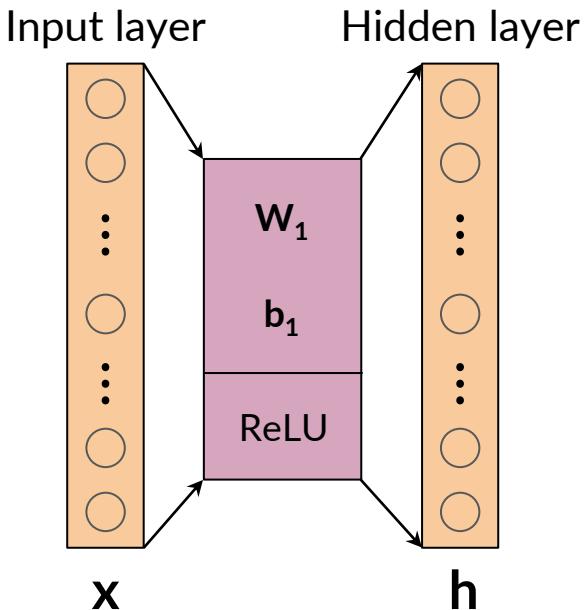


deeplearning.ai

Architecture of the CBOW Model

Activation Functions

Rectified Linear Unit (ReLU)

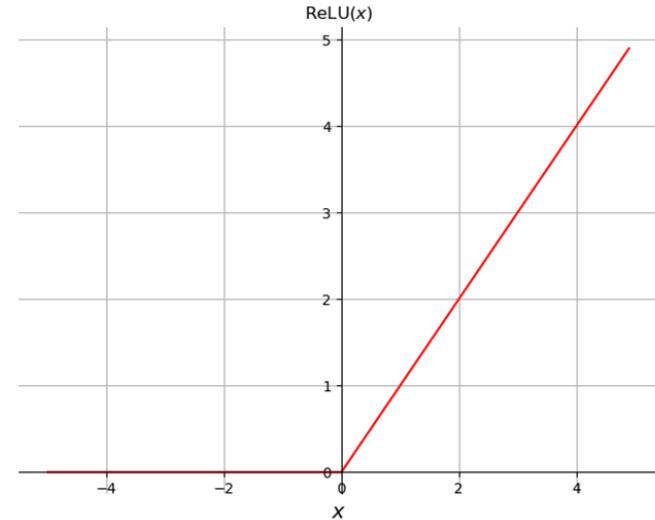


$$z_1 = W_1 x + b_1$$

$$h = \text{ReLU}(z_1)$$

$$\begin{array}{c} z_1 \\ \hline 5.1 \\ -0.3 \\ \vdots \\ -4.6 \\ 0.2 \end{array} \xrightarrow{\text{ReLU}} \begin{array}{c} h \\ \hline 5.1 \\ 0 \\ \vdots \\ 0 \\ 0.2 \end{array}$$

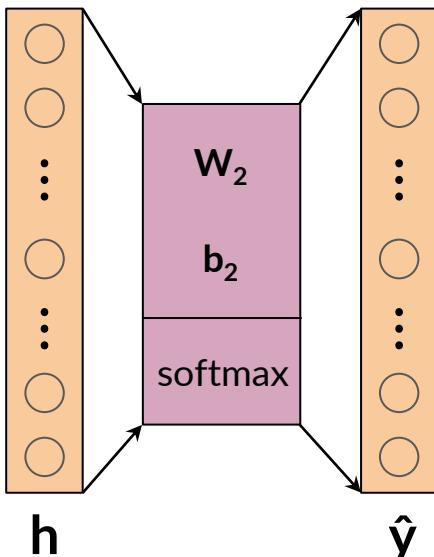
$$\text{ReLU}(x) = \max(0, x)$$



Softmax

Hidden layer

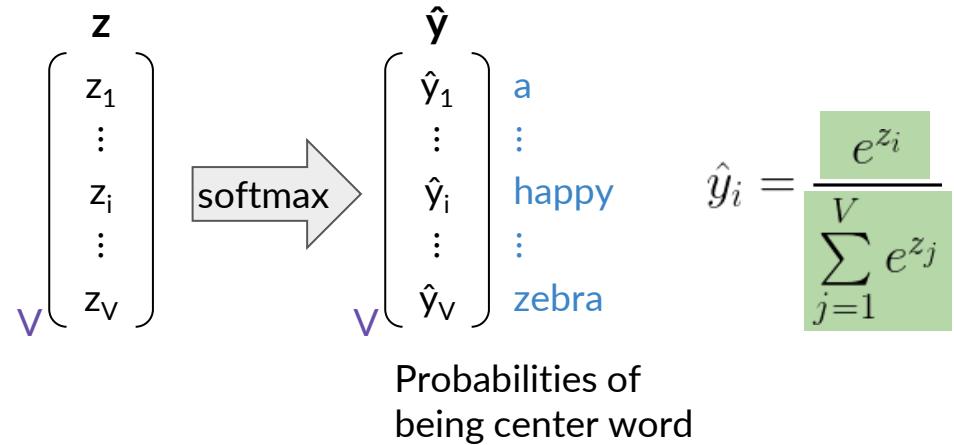
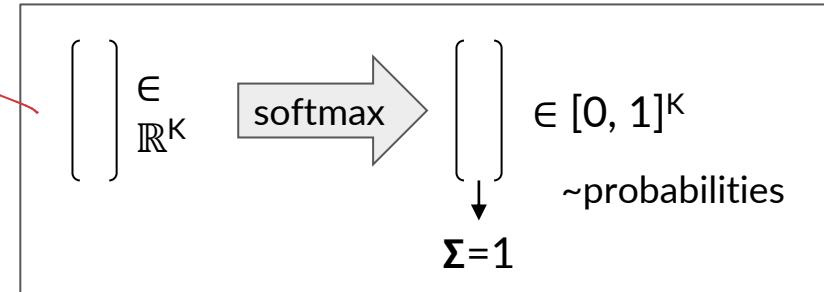
Output layer



$$z = W_2 h + b_2$$

$$\hat{y} = \text{softmax}(z)$$

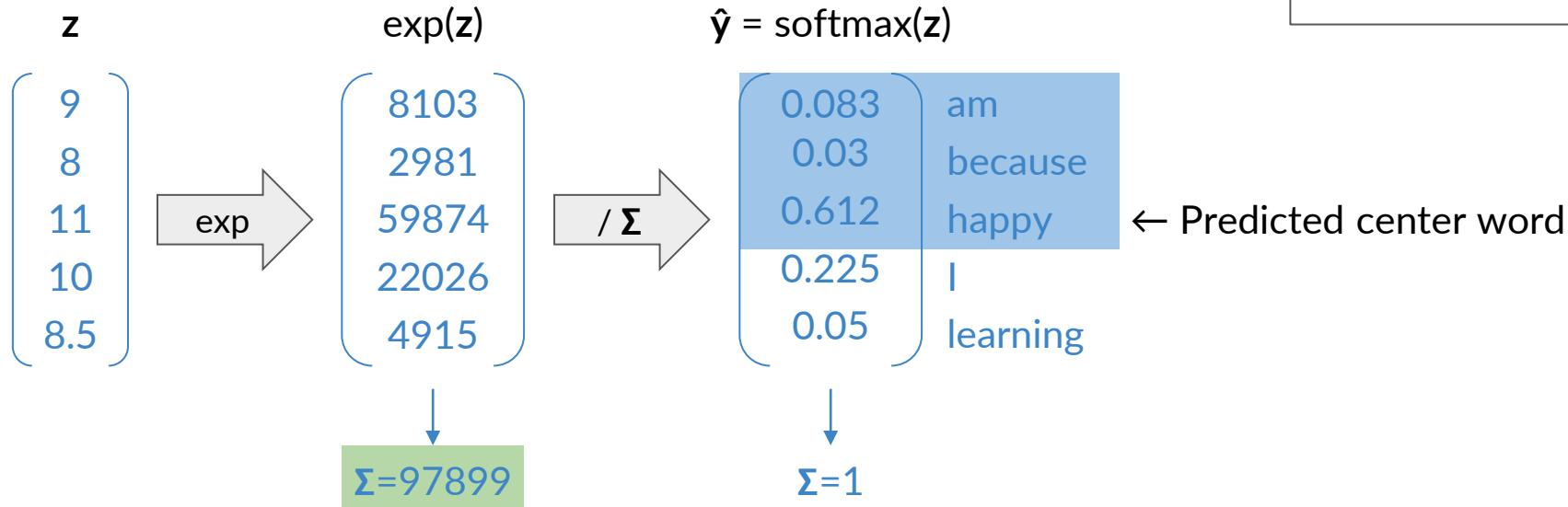
Vector of
Real numbers



Softmax: example

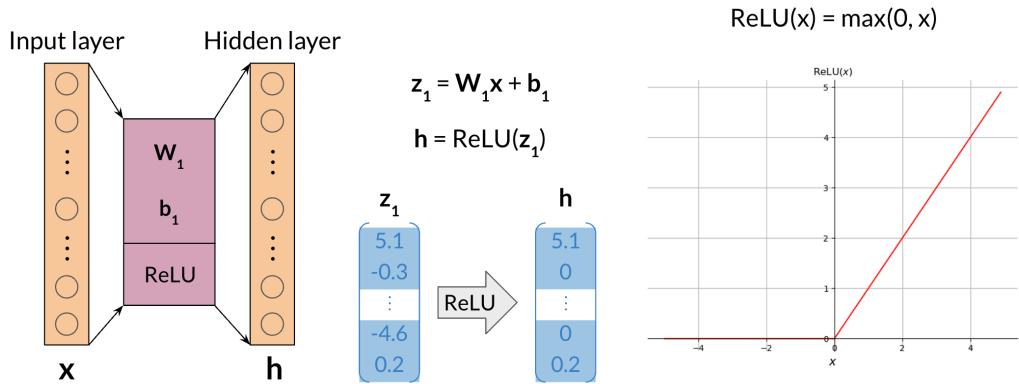
Positive numbers

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



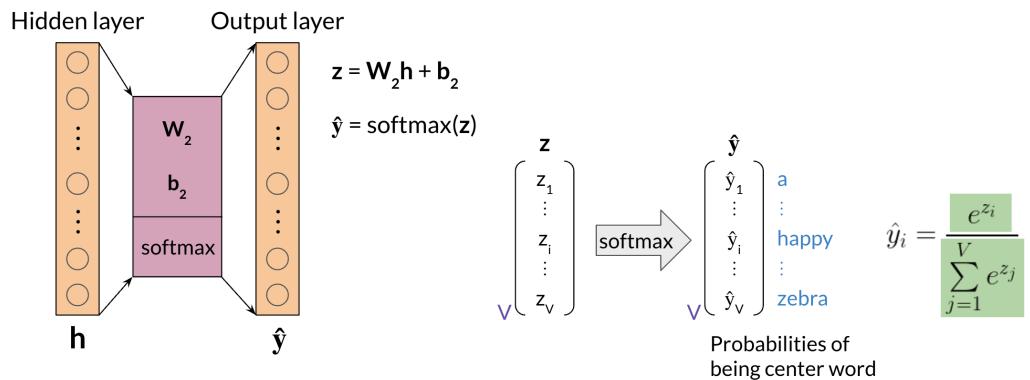
ReLU function

The rectified linear unit (ReLU), is one of the most popular activation functions. When you feed a vector, namely x , into a ReLU function. You end up taking $x = \max(0, x)$. This is a drawing that shows ReLU.



Softmax function

The softmax function takes a vector and transforms it into a probability distribution. For example, given the following vector z , you can transform it into a probability distribution as follows.



As you can see, you can compute $\hat{y} = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$.

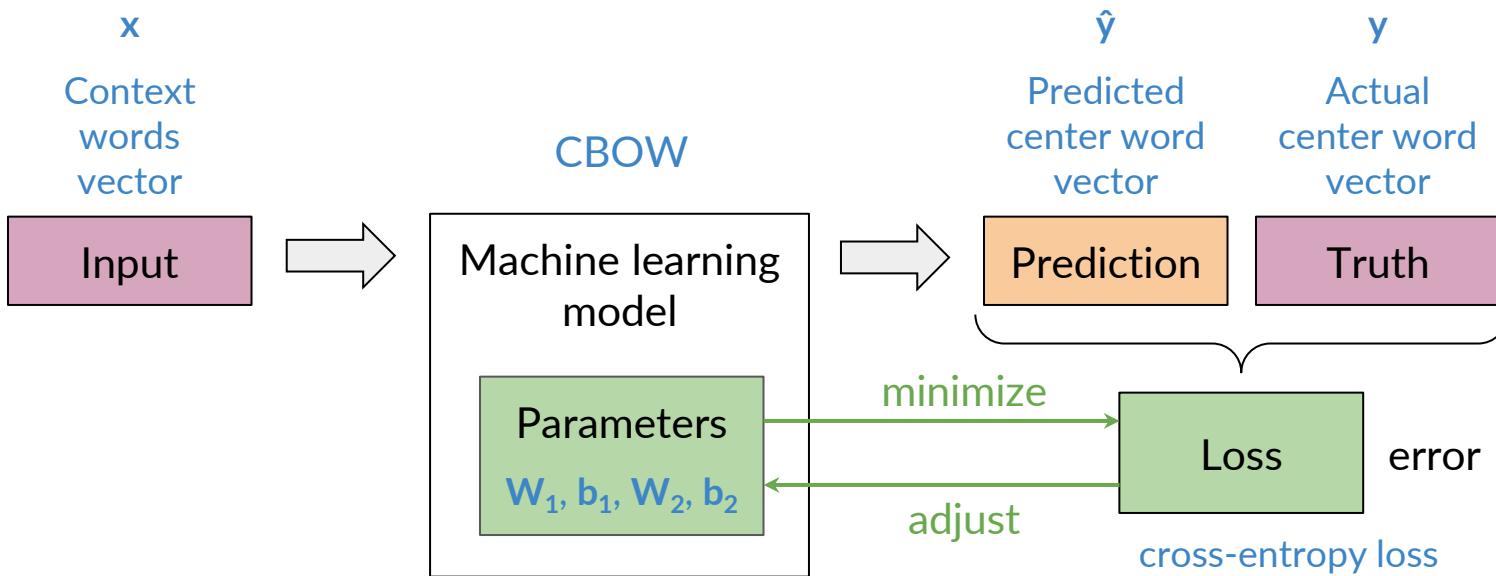


deeplearning.ai

Training a CBOW Model

Cost Function

LOSS



Cross-entropy loss

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

Actual
 $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_V \end{pmatrix}$

Predicted
 $\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_V \end{pmatrix}$

I am **happy** because I am learning

\mathbf{y}	$\hat{\mathbf{y}}$	
0	am	0.083
0	because	0.03
1	happy	0.611
0	I	0.225
0	learning	0.05

\rightarrow log

$\log(\hat{\mathbf{y}})$

-2.49
-3.49
-0.49
-1.49
-2.49

$\mathbf{y} \odot \log(\hat{\mathbf{y}})$

0
0
-0.49
0
0

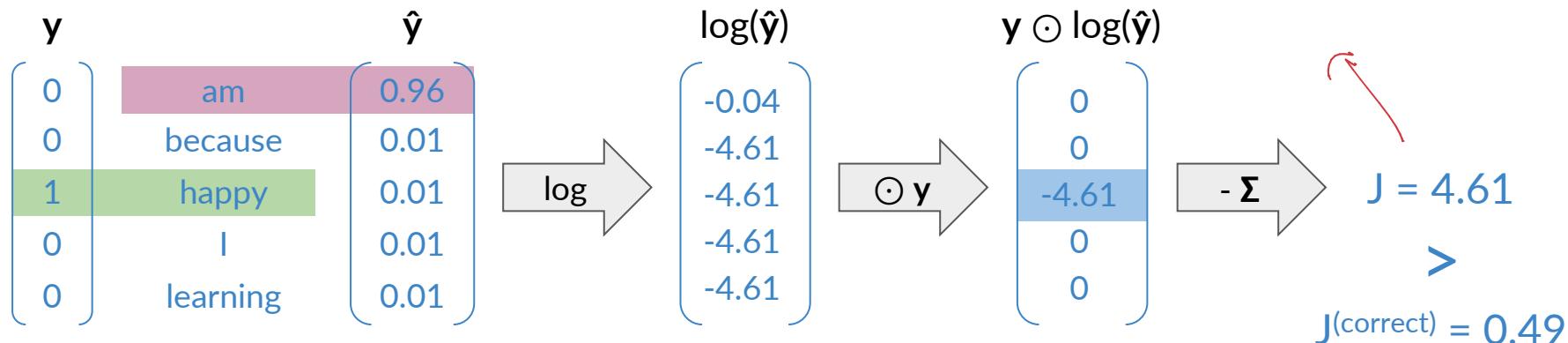
$\rightarrow \odot \mathbf{y}$

$\rightarrow -\Sigma$ $J = 0.49$

Cross-entropy loss

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

Loss is larger when Prediction is incorrect



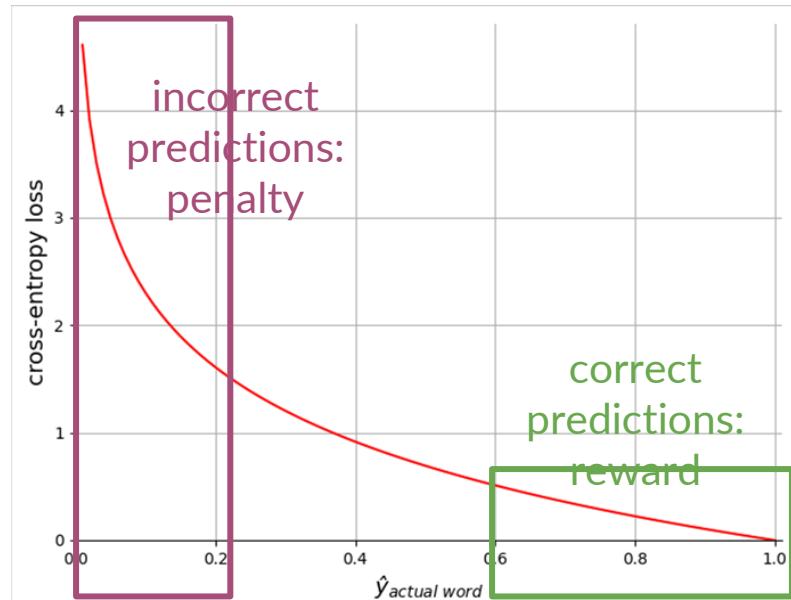
Cross-entropy loss

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

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

$$\rightarrow J = 4.61$$

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



The cost function for the CBOW model is a cross-entropy loss defined as:

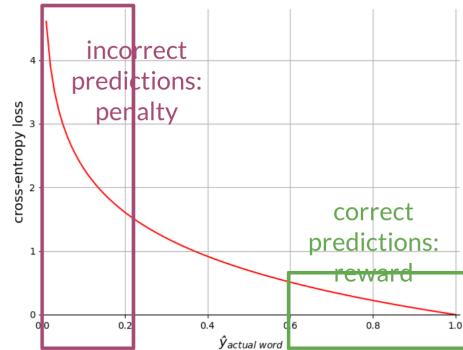
$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Here is an example where you use the equation above.

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

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

$$\rightarrow J = 4.61$$



Why is the cost 4.61 in the example above?

Wrong Prediction



deeplearning.ai

Training a CBOW Model

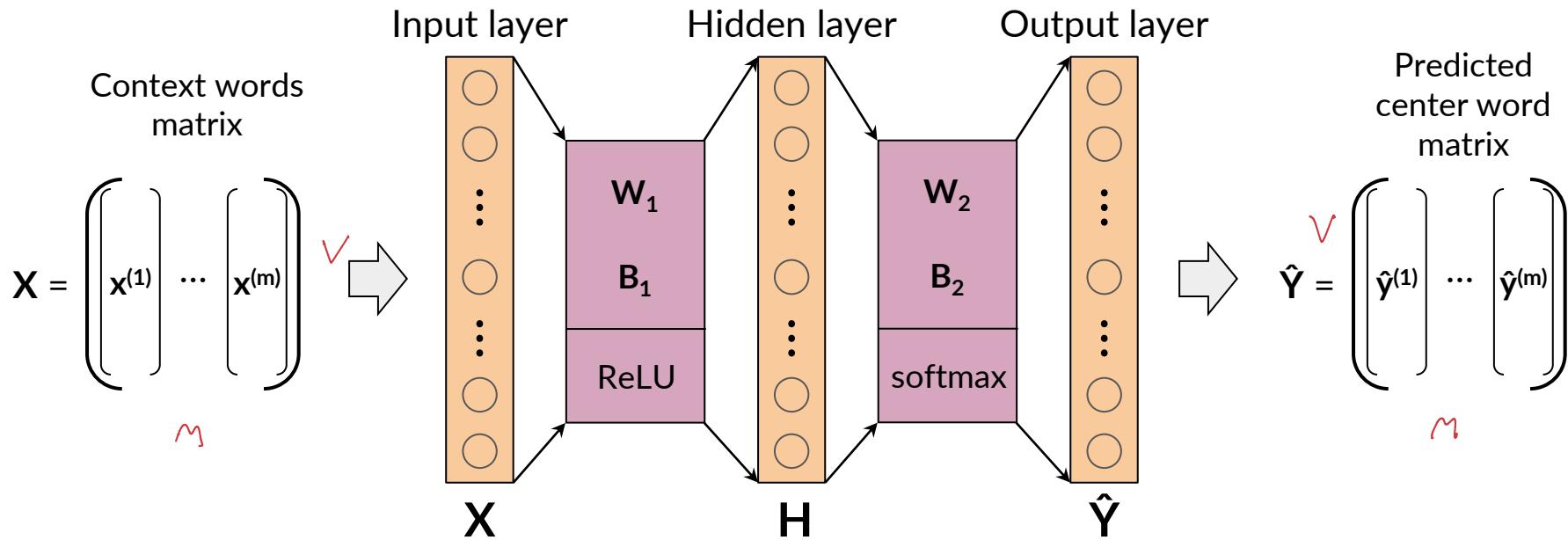
Forward Propagation

Training process

- Forward propagation
- Cost
- Backpropagation and gradient descent

Forward propagation

$$\begin{aligned} Z_1 &= W_1 X + B_1 & Z_2 &= W_2 H + B_2 \\ H &= \text{ReLU}(Z_1) & \hat{Y} &= \text{softmax}(Z_2) \end{aligned}$$



Cost

loss = Single example

cost = a batch of examples

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

Cost: mean of losses

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

Predicted center word matrix

Actual center word matrix

$$\hat{\mathbf{Y}} = \begin{pmatrix} \hat{\mathbf{y}}^{(1)} \\ \vdots \\ \hat{\mathbf{y}}^{(m)} \end{pmatrix}$$

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}^{(1)} \\ \vdots \\ \mathbf{y}^{(m)} \end{pmatrix}$$

$$J_{batch} = -\frac{1}{m} \sum_{i=1}^m J^{(i)}$$

loss for example i

Forward propagation is defined as:

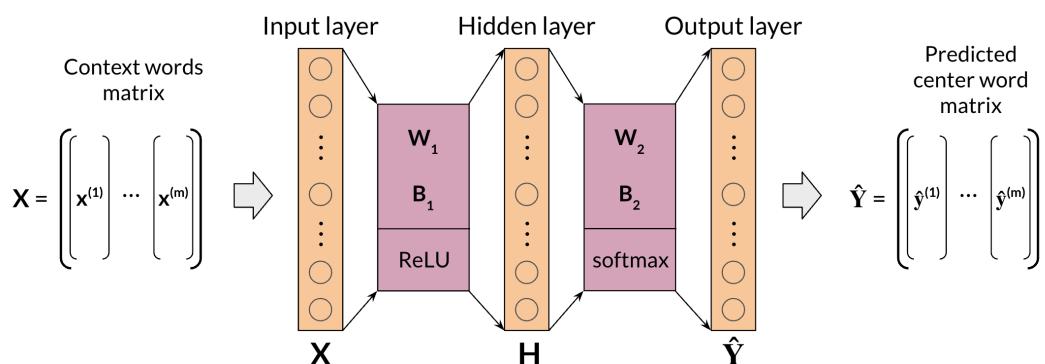
$$Z_1 = W_1 X + B_1$$

$$H = \text{ReLU}(Z_1)$$

$$Z_2 = W_2 H + B_2$$

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

In the image below you start from the left and you forward propagate all the way to the right.



To calculate the loss of a batch, you have to compute the following:

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

Given, your predicted center word matrix, and actual center word matrix, you can compute the loss.

Cosine

Predicted center word matrix $\hat{Y} = \begin{pmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(m)} \end{pmatrix}$	Actual center word matrix $Y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix}$
--	---



deeplearning.ai

Training a CBOW Model

Backpropagation and
Gradient Descent

Minimizing the cost

$$J_{batch} = f(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)$$

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Minimizing the cost

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

- Gradient descent: update weights and biases

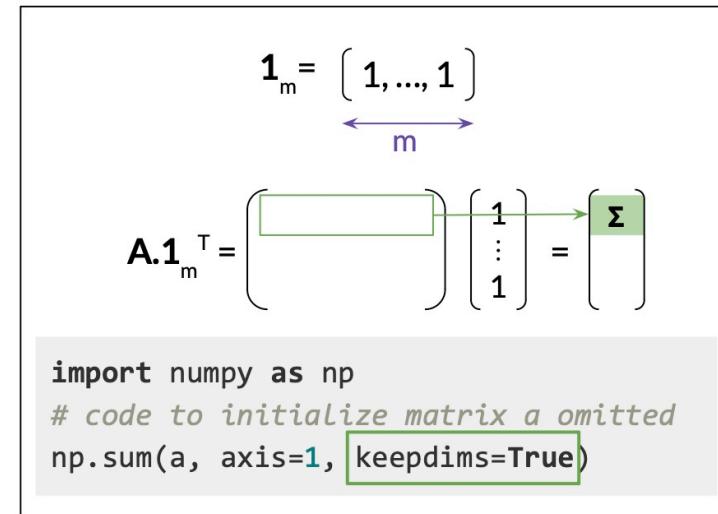
Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} (\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \cdot \text{step}(\mathbf{Z}_1)) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} (\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \cdot \text{step}(\mathbf{Z}_1)) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{1}_m^\top$$



Gradient descent

Hyperparameter: learning rate $\alpha \rightarrow$ *hyperParam*

$$\mathbf{W}_1 := \mathbf{W}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\mathbf{W}_2 := \mathbf{W}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\mathbf{b}_1 := \mathbf{b}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\mathbf{b}_2 := \mathbf{b}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

- **Backpropagation** : calculate partial derivatives of cost with respect to weights and biases.

When computing the back-prop in this model, you need to compute the following:

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

- **Gradient descent**: update weights and biases

Now to update the weights you can iterate as follows:

$$\mathbf{W}_1 := \mathbf{W}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\mathbf{W}_2 := \mathbf{W}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\mathbf{b}_1 := \mathbf{b}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\mathbf{b}_2 := \mathbf{b}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

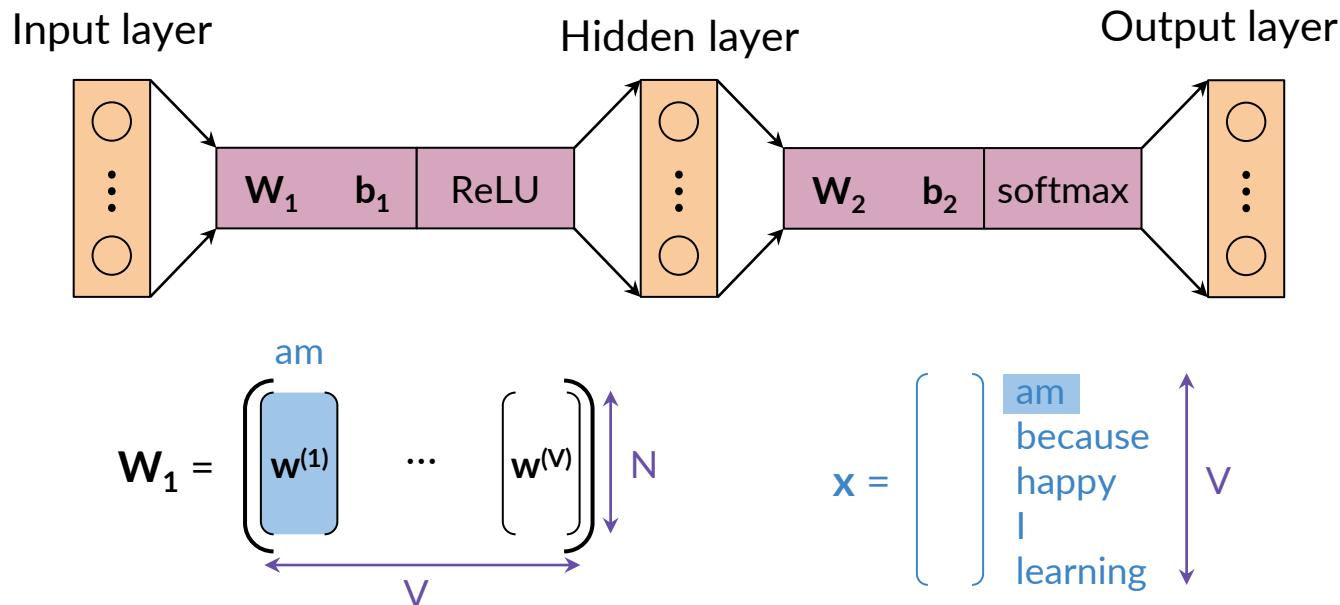
A smaller alpha allows for more gradual updates to the weights and biases, whereas a larger number allows for a faster update of the weights. If α is too large, you might not learn anything, if it is too small, your model will take forever to train.



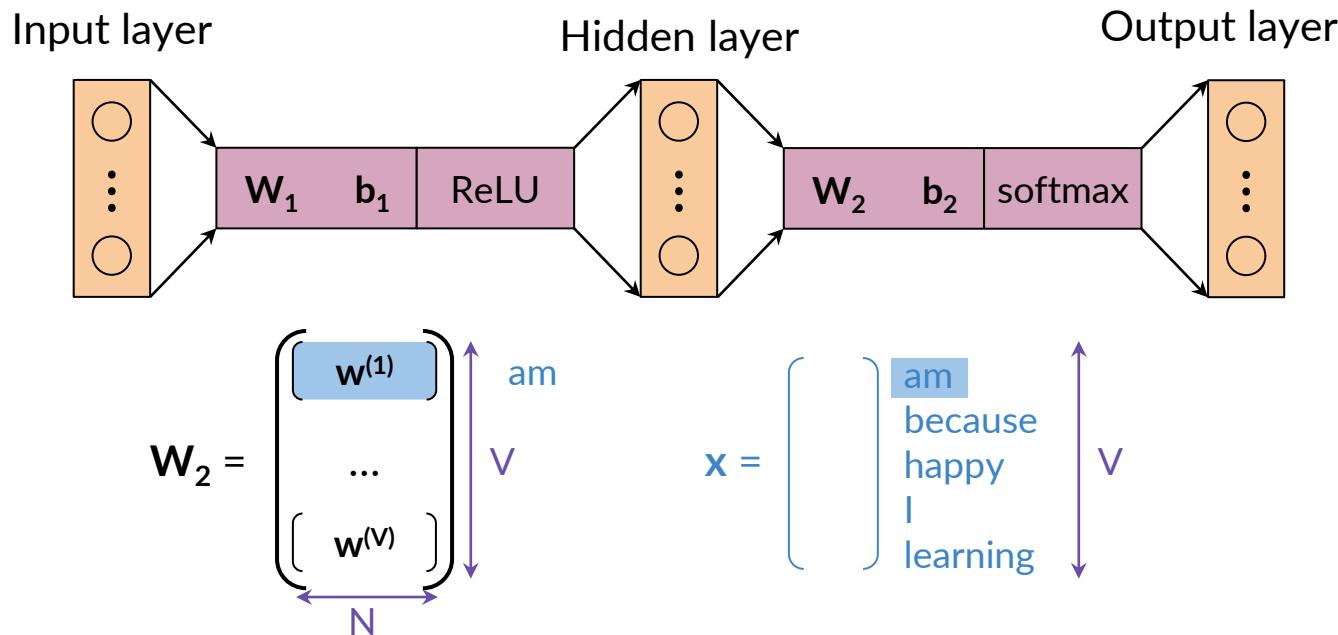
deeplearning.ai

Extracting Word Embedding Vectors

Extracting word embedding vectors: option 1



Extracting word embedding vectors: option 2



Extracting word embedding vectors: option 3

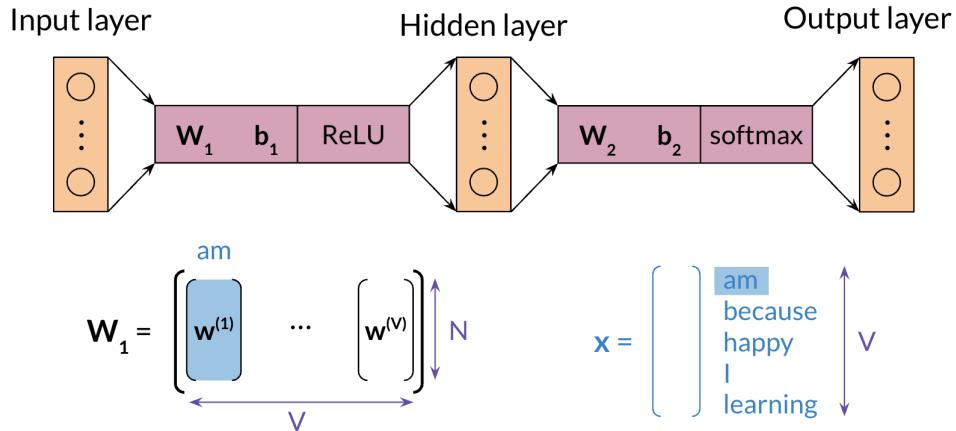
$$\mathbf{W}_1 = \begin{pmatrix} \mathbf{w}_1^{(1)} \\ \vdots \\ \mathbf{w}_1^{(V)} \end{pmatrix} \quad \mathbf{W}_2 = \begin{pmatrix} \mathbf{w}_2^{(1)} \\ \vdots \\ \mathbf{w}_2^{(V)} \end{pmatrix}$$

$\mathbf{W}_3 = 0.5 (\mathbf{W}_1 + \mathbf{W}_2^T) = \begin{pmatrix} \mathbf{w}_3^{(1)} \\ \vdots \\ \mathbf{w}_3^{(V)} \end{pmatrix}$

$\mathbf{x} = \begin{pmatrix} \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{pmatrix}$

Diagram illustrating the extraction of word embedding vectors. Two weight matrices, \mathbf{W}_1 and \mathbf{W}_2 , are shown as vertical stacks of vectors $\mathbf{w}_1^{(1)}, \dots, \mathbf{w}_1^{(V)}$ and $\mathbf{w}_2^{(1)}, \dots, \mathbf{w}_2^{(V)}$ respectively. An arrow points from these matrices to the equation for \mathbf{W}_3 . \mathbf{W}_3 is defined as the average of \mathbf{W}_1 and the transpose of \mathbf{W}_2 . This results in a vector $\mathbf{w}_3^{(1)}, \dots, \mathbf{w}_3^{(V)}$ where each component is the average of the corresponding components in \mathbf{W}_1 and \mathbf{W}_2 . The dimension of \mathbf{W}_3 is indicated by a double-headed arrow labeled V below it. To the right, a vector \mathbf{x} is shown with components "am", "because", "happy", "I", and "learning". A double-headed arrow labeled V indicates the dimension of \mathbf{x} .

There are two options to extract word embeddings after training the continuous bag of words model. You can use w_1 as follows:



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:

$$W_2 = \begin{pmatrix} w^{(1)} \\ \dots \\ w^{(N)} \end{pmatrix} \quad x = \begin{pmatrix} am \\ because \\ happy \\ I \\ learning \end{pmatrix}$$

The final option is to take an average of both matrices as follows:

$$W_3 = 0.5 (W_1 + W_2^T) = \begin{pmatrix} w_3^{(1)} \\ \dots \\ w_3^{(N)} \end{pmatrix}$$



deeplearning.ai

Evaluating Word Embeddings

Intrinsic Evaluation

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

“France” is to “Paris” as “Italy” is to <?>

Rome

Syntactic analogies

“seen” is to “saw” as “been” is to <?>

⚡ Ambiguity

“wolf” is to “pack” as “bee” is to <?> → swarm? colony?

Intrinsic evaluation

Test relationships between words

- Analogies

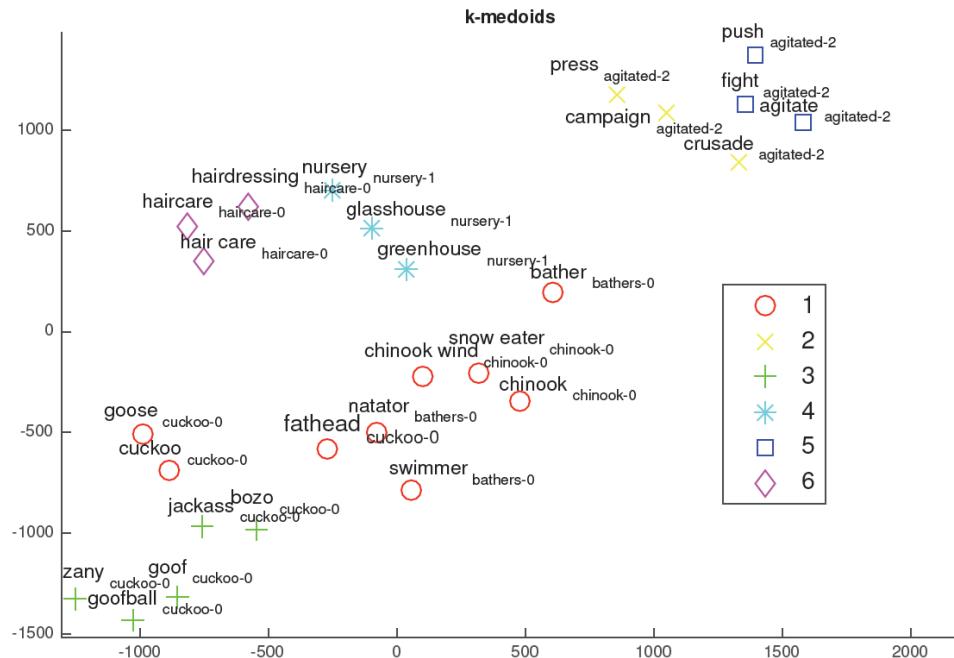
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Intrinsic evaluation

Test relationships between words

- Analogies
 - Clustering

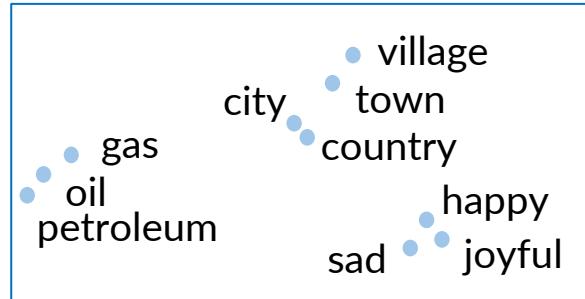
Source: Michael Zhai, Johnny Tan, and Jinho D. Choi. 2016. [Intrinsic and extrinsic evaluations of word embeddings](#)



Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering
- Visualization



Intrinsic evaluation allows you to test relationships between words. It allows you to capture semantic analogies as, “France” is to “Paris” as “Italy” is to <?> and also syntactic analogies as “seen” is to “saw” as “been” is to <?>.

Ambiguous cases could be much harder to track:

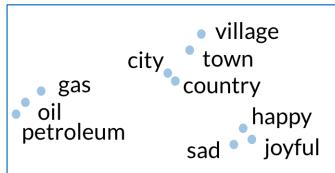
⚡ Ambiguity
“wolf” is to “pack” as “bee” is to <?> → swarm? colony?

Rome
was/were

Here are a few ways that allow to use intrinsic evaluation.

Test relationships between words

- Analogies
- Clustering
- Visualization





deeplearning.ai

Evaluating Word Embeddings

Extrinsic Evaluation

Extrinsic evaluation

evaluates actual usefulness
of the word embedding

Test word embeddings on external task

e.g. named entity recognition, parts-of-speech tagging

Named entity

Andrew works at deeplearning.ai

person

organization

Extrinsic evaluation

Test word embeddings on external task

e.g. named entity recognition, parts-of-speech tagging

- + Evaluates actual usefulness of embeddings
- Time-consuming
- More difficult to troubleshoot

Extrinsic evaluation tests word 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

So now you know both **intrinsic** and **extrinsic** evaluation.



deeplearning.ai

Conclusion

Recap and assignment

- Data preparation
- Word representations
- Continuous bag-of-words model
- Evaluation

Going further

- Advanced language modelling and word embeddings
- NLP and machine learning libraries

Keras

```
# from keras.layers.embeddings import Embedding  
embed_layer = Embedding(10000, 400)
```

PyTorch

```
# import torch.nn as nn  
embed_layer = nn.Embedding(10000, 400)
```

This week you learned the following concepts.

- Data preparation
- Word representations
- Continuous bag-of-words model
- Evaluation

You have all the foundations now. From now on, you will start using some advanced AI libraries in the next courses. Congratulations and good luck with the assignment :)