

# COMP 472: Artificial Intelligence Natural Language Processing Word Embeddings

*part 5*

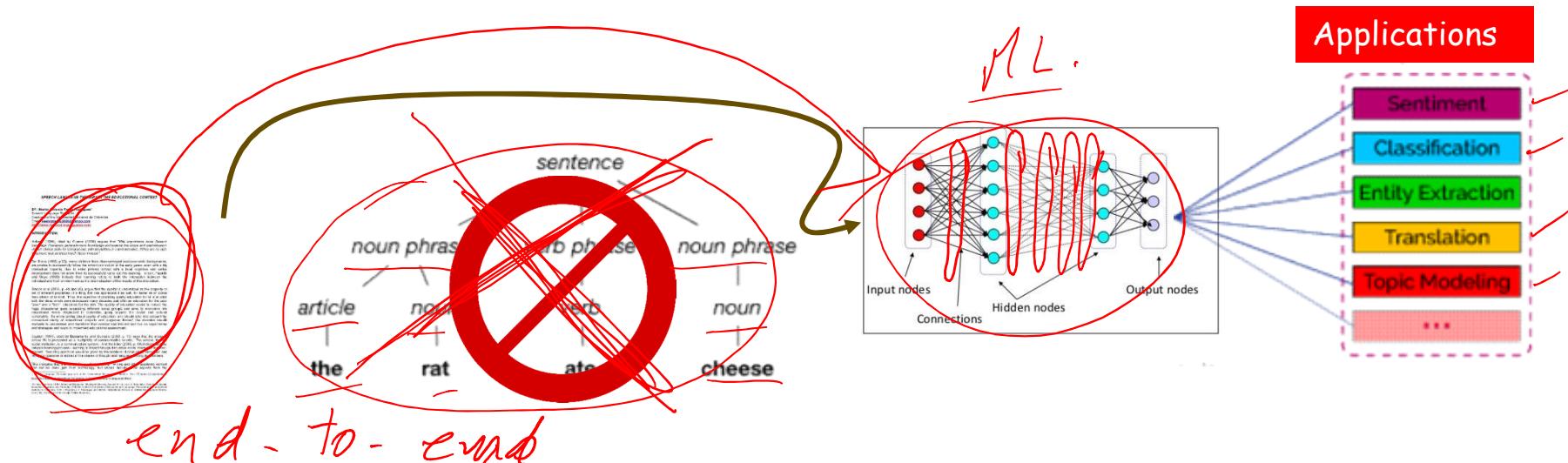
*videos*

- Russell & Norvig: Section 24.1

# Today

1. Introduction
2. Bag of word model
3. n-gram models
4. Deep Learning for NLP
  1. Word Embeddings 
  2. Recurrent Neural Networks

# Deep Language Processing (circa 2010-today)



## Deep Neural Networks applied to NLP problems

- Rules are developed automatically (using machine learning) 😊
- And the linguistic features are found automatically! 😊

# Deep Learning for NLP

Deep learning models for NLP use

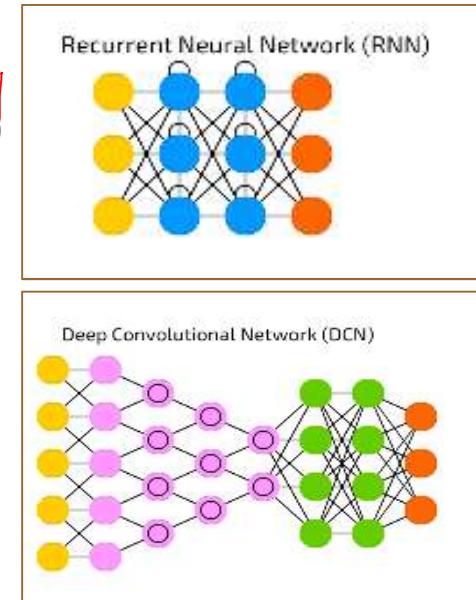
#1 Vector representation of words

- ❑ i.e., word embeddings



#2 Neural network structures

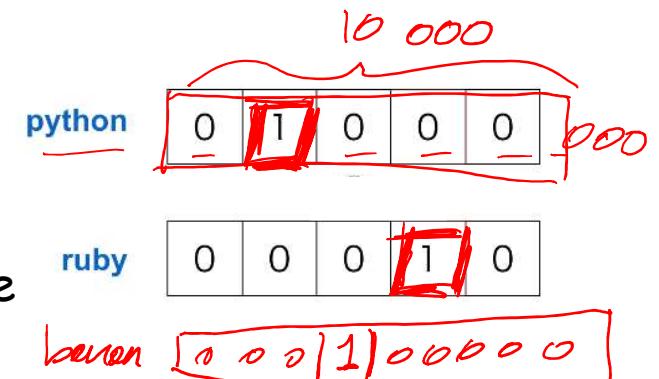
- ❑ Recurrent Neural Networks (RNNs)
- ❑ Convolutional Networks (CNNs)
- ❑ Recursive Neural Networks
- ❑ ...



# Word Embeddings

- To do NLP with neural networks, words need to be represented as vectors
- Traditional approach: "one hot vector"

- Binary vector
- Length = | vocab |  $|V| = 10\,000$
- 1 in the position of the word id, the rest are [0, 0, 0, 1, 0, 0, 0, ...]



- However, this does not represent word meaning 😔

- Similar words such as python and ruby should have similar vector representations

*python and banane*

- However, similarity/distance between all "one hot vectors" are the same for all pairs of words



# Word Embeddings

- We would like:
  - ~~cat/kitty/dog~~ ... to have similar representations
  - ~~cat/orange/train/python~~ ... to have dissimilar representations
- Word embeddings:
  - aka. word representations
  - Represent each word by a shorter/condensed vector (eg, 50 to 300)
  - Like a point in n-dimensional space



# Word2vec

- Developed in 2013 Tomas Mikolov et al, from Google
- Very fast to train
- Code available on the web
  - <https://code.google.com/archive/p/word2vec>
- Idea:
  - an n-gram model counts... word2vec predicts
  - Instead of **counting** how often each word  $w$  occurs near "apricot"
  - Train a classifier on a binary **prediction task**:
    - Is  $w$  likely to show up near "cat"?
  - In the end, we do not actually care about this task
    - But we will take the learned weights as the word embeddings
  - Use as training set readily available texts, so no need for hand-labeled supervision !



# Word2vec Models



*"A word is known by the company it keeps"* - J. R. Firth

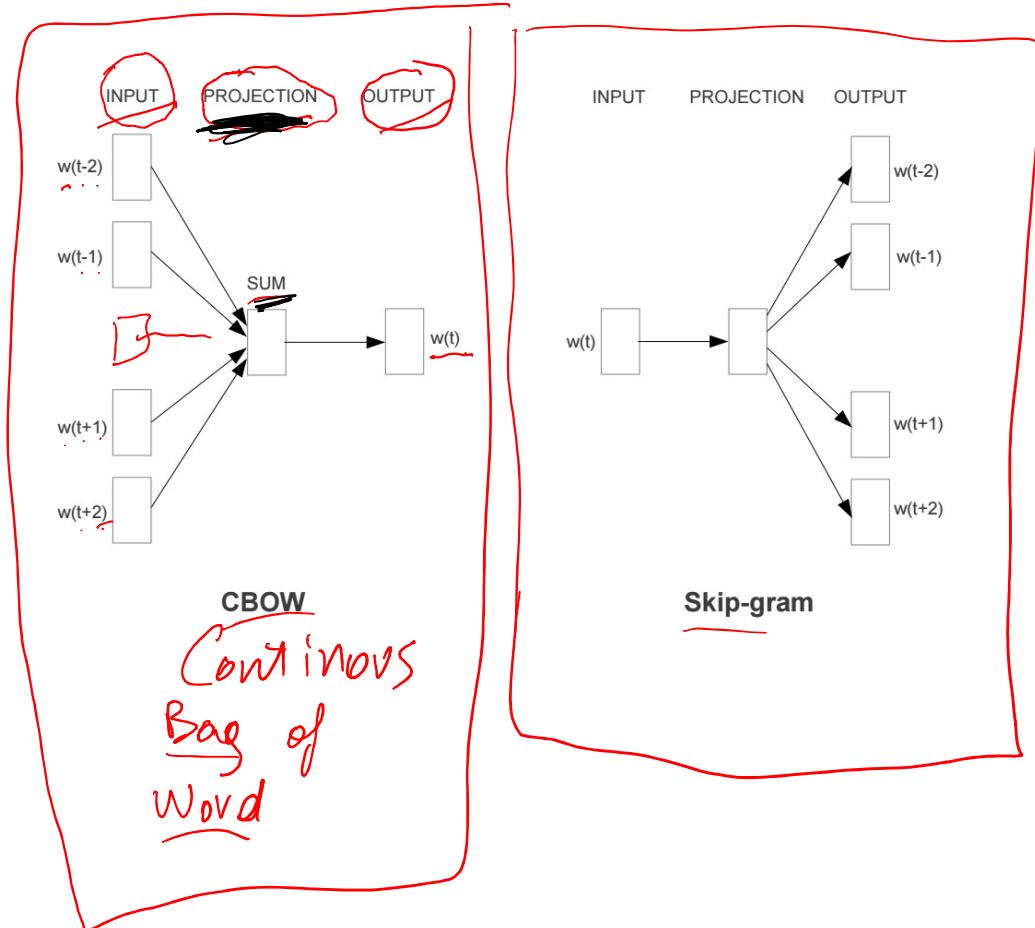
## Basic Idea:

1. Similar words should have similar contexts (surrounding words)
  2. So we can use the contexts to guess the word (or vice-versa).
    - The cat/kitty/dog hunts for mice.
    - The brown furry cat/kitty/dog is eating.
    - John's cat/kitty purrs.
  3. Train an ANN to guess a word given its context (or vice-versa)
- CBOW  
skip-gram

# Word2Vec models

Word2Vec has 2 models:

1. CBOW: given context words, guess the word
2. Skip-gram: given a word, guess one of its surrounding words



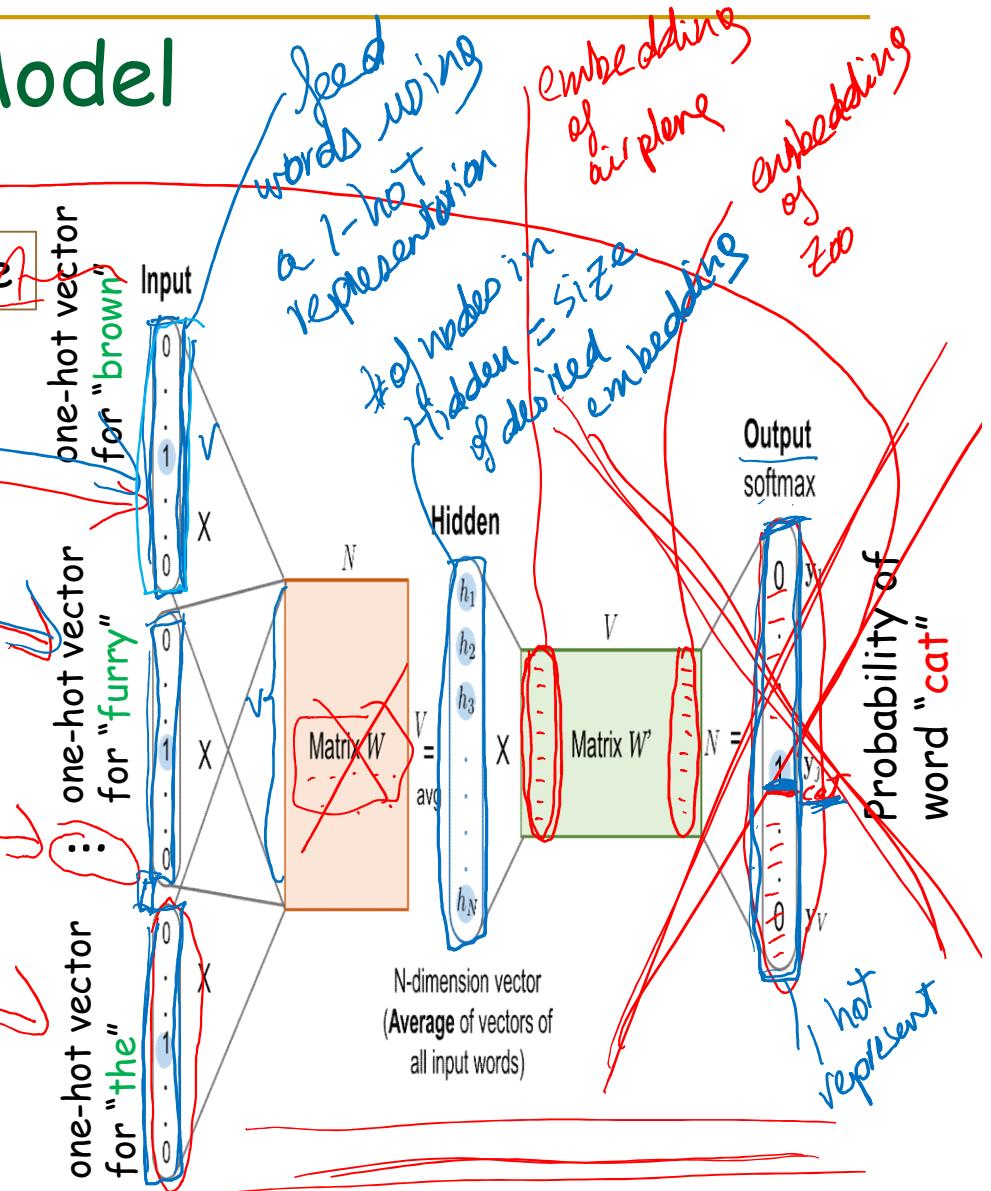
# Word2Vec: CBOW Model

The brown furry cat chases the mouse

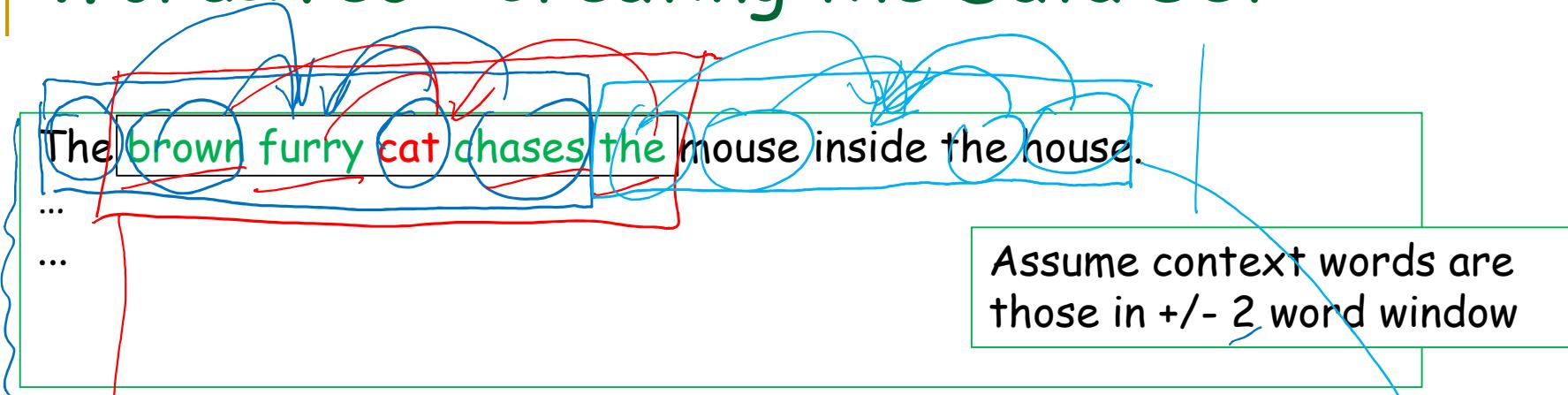
Uses a shallow neural network with only 3 layers:

1. One input layer
2. One hidden layer and
3. One output layer.

goal: predict the probability of a target word (**cat**) given a context (**brown** **furry** **chases** **the**).

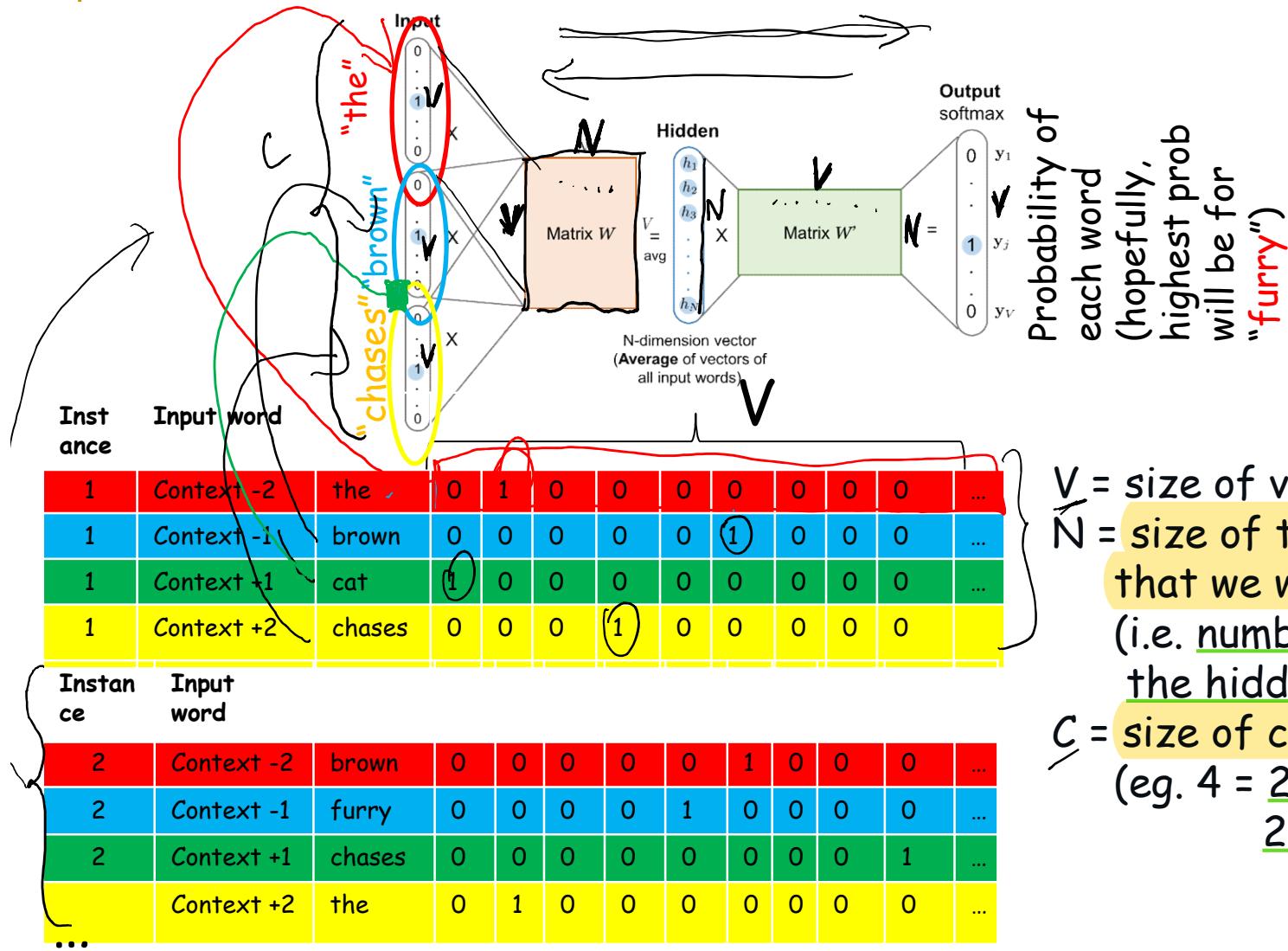


# Word2Vec - Creating the Data Set



Instance	Context word -2	Context word -1	Context word +1	Context word +2	To Predict
1	the	brown	cat	chases	furry
2	brown	furry	chases	the	cat
3	furry	cat	the	mouse	chases
4	cat	chases	mouse	inside	the
5	chases	the	inside	the	mouse
6	the	mouse	the	house	inside

# Word2Vec - Input to the Network



$V$  = size of vocabulary

$N$  = size of the embedding that we want

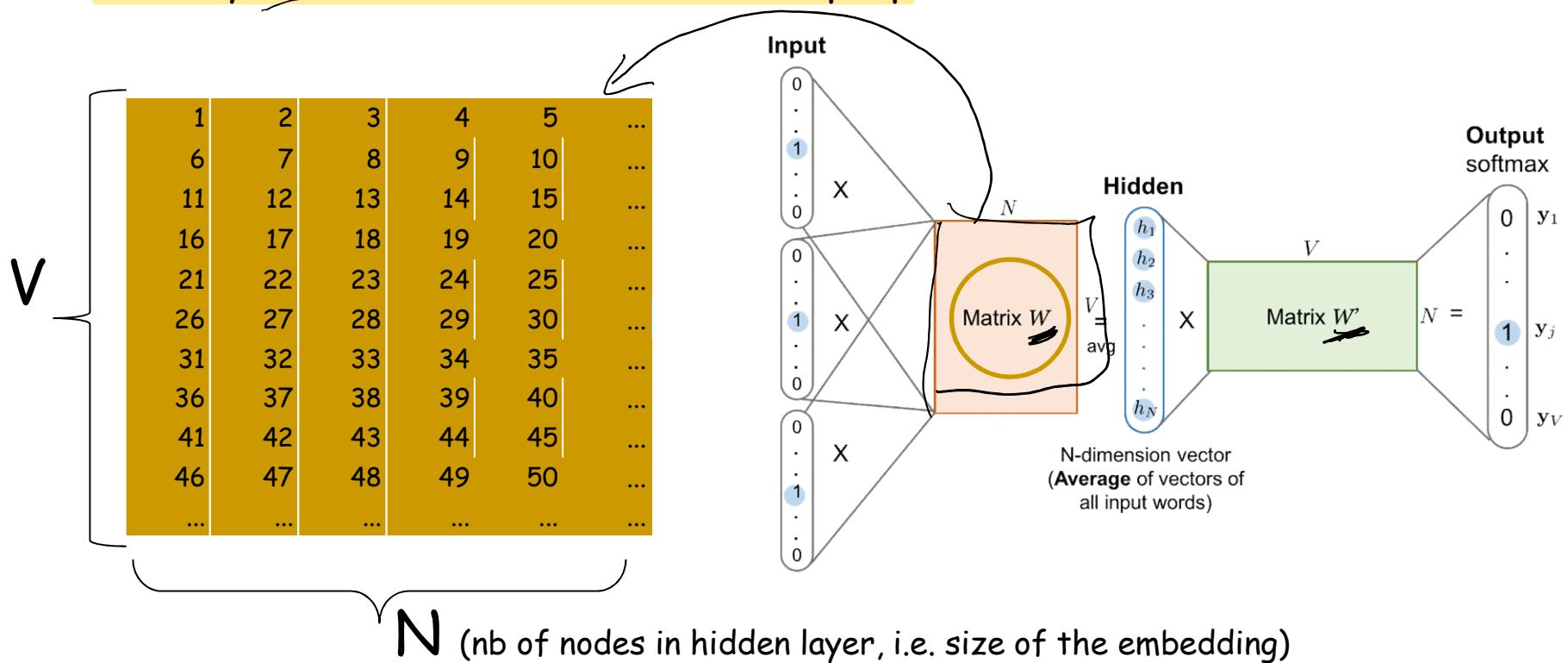
(i.e. number of neurons in the hidden layer)

$C$  = size of context

(eg. 4 = 2 words before + 2 words after)

# Word2Vec - Weights W

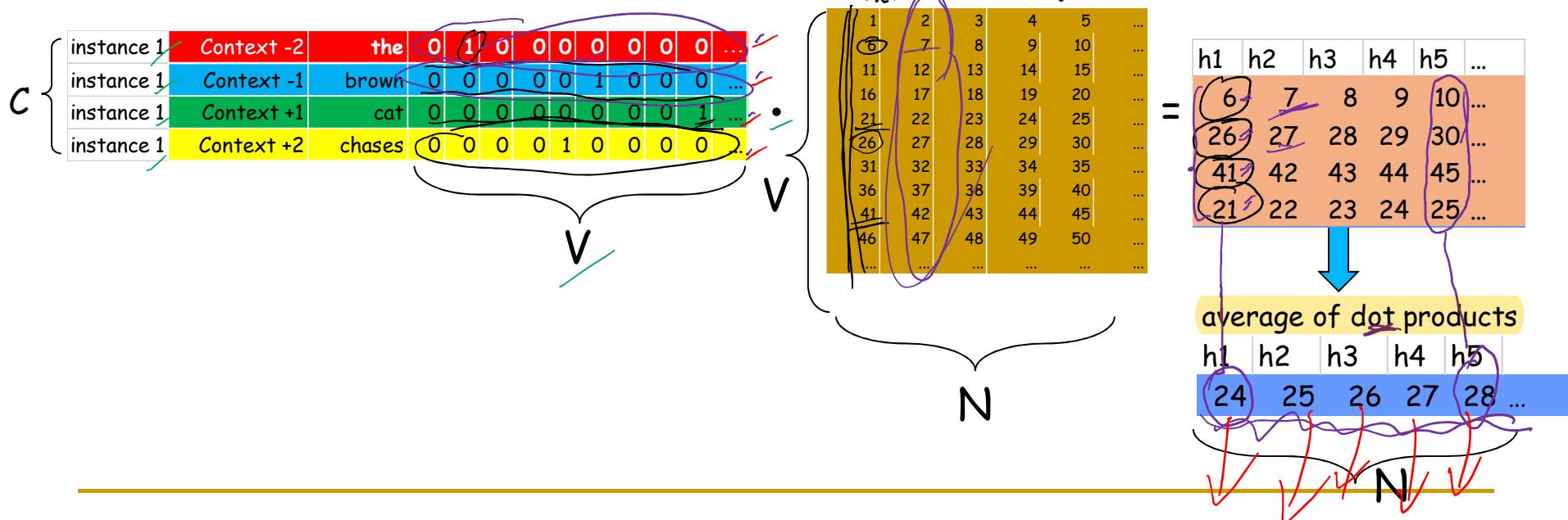
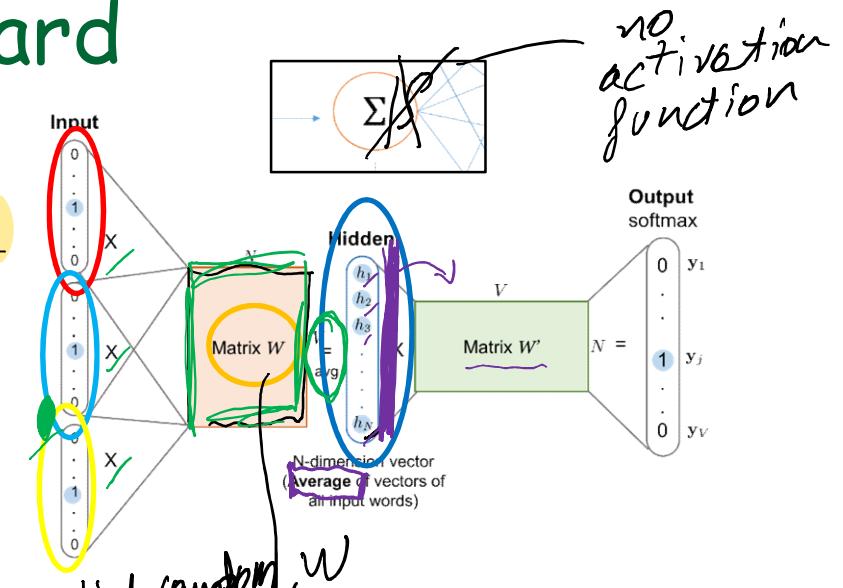
- Weight Matrix W between input & hidden layer
- W is a VxN matrix...
- Initially random but modified via backprop



# Word2Vec - Feedforward

- Calculate the output of each of the N hidden nodes for each context word  
Note: there is no activation function.  
Output of  $h_i$  is just the dot product

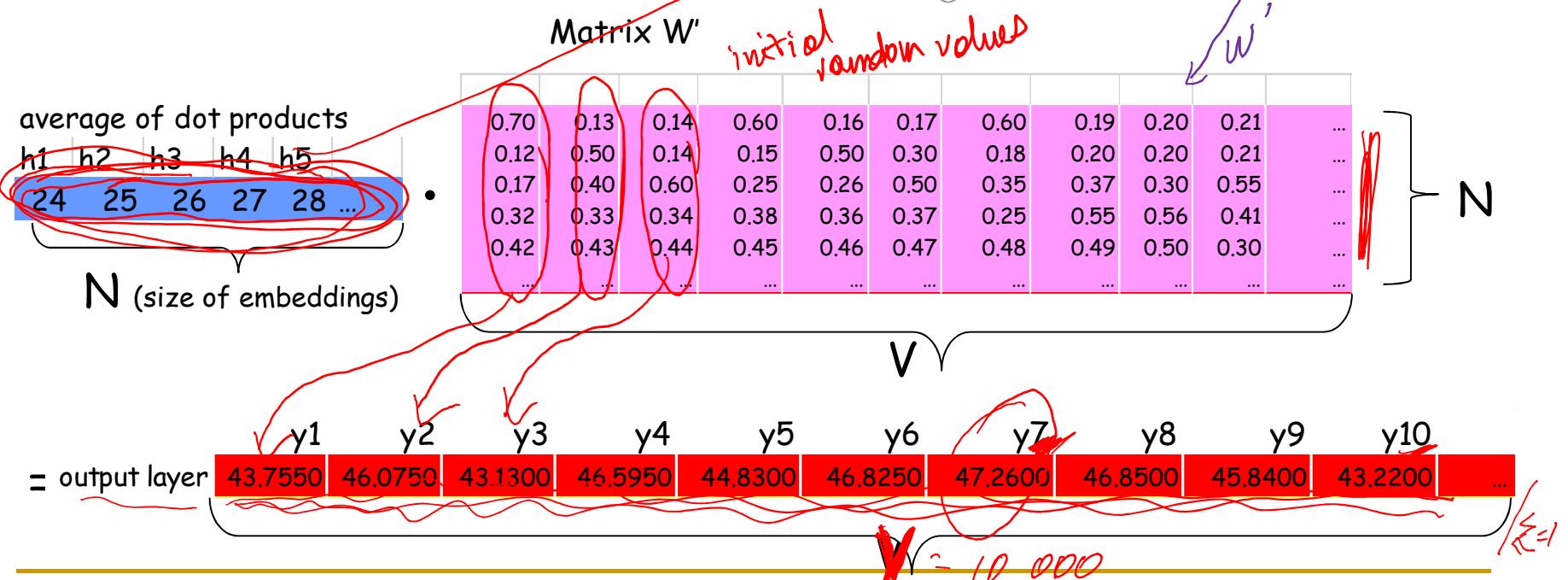
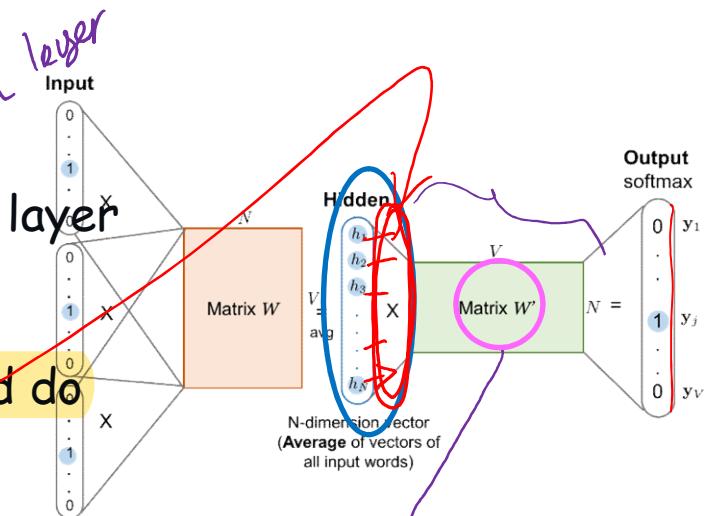
- Then take the average



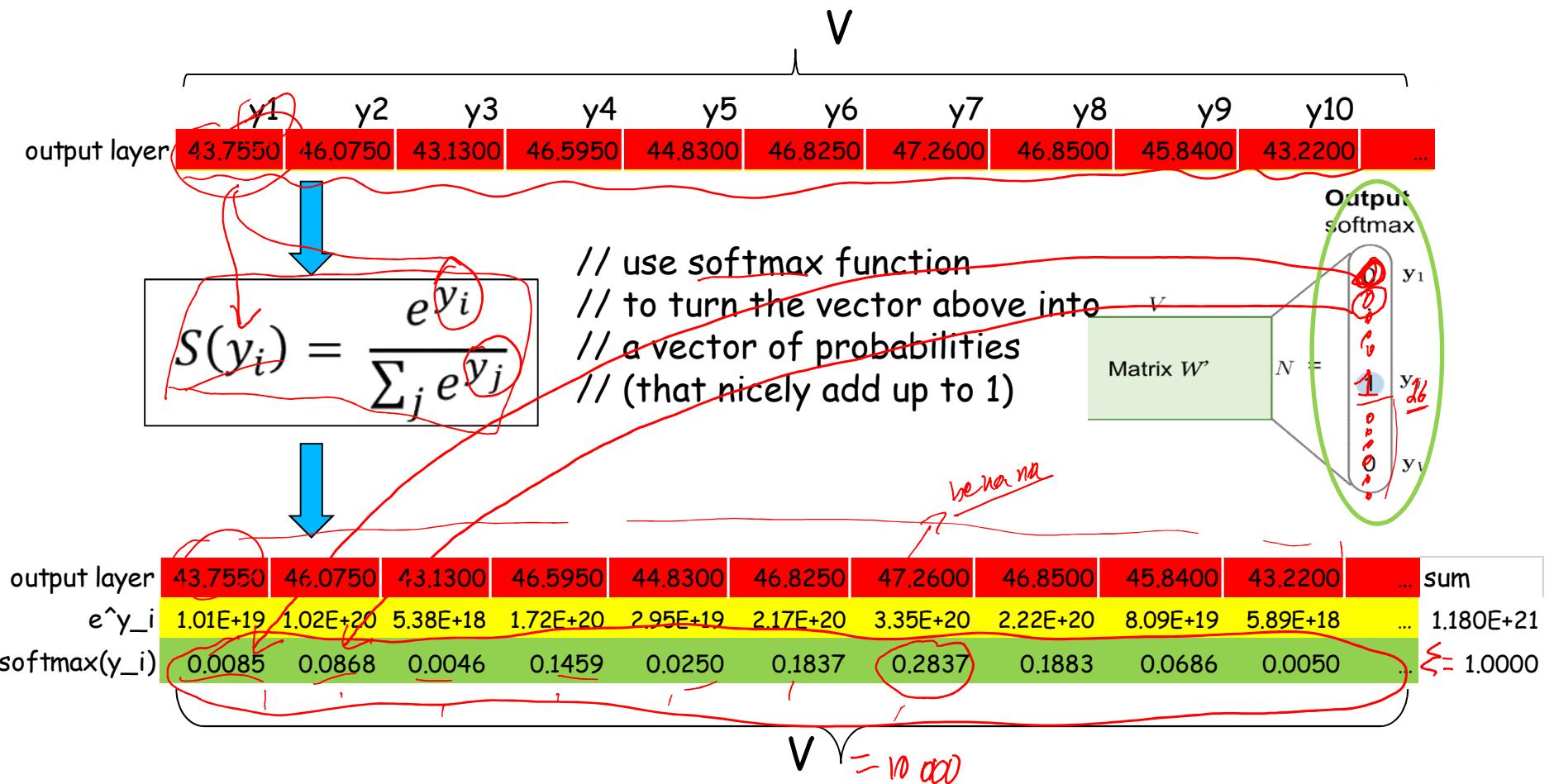
# Word2Vec - Weights $W'$

- Weight Matrix  $W'$  between hidden & output layer
- $W'$  is a  $N \times V$  matrix...
- $W'$  Initially random but modified via backprop
- Feed forward average of hidden neurons and do dot product with matrix  $W'$

size of defined embeddings = # of nodes in the hidden layer



# Turn dot product into probabilities

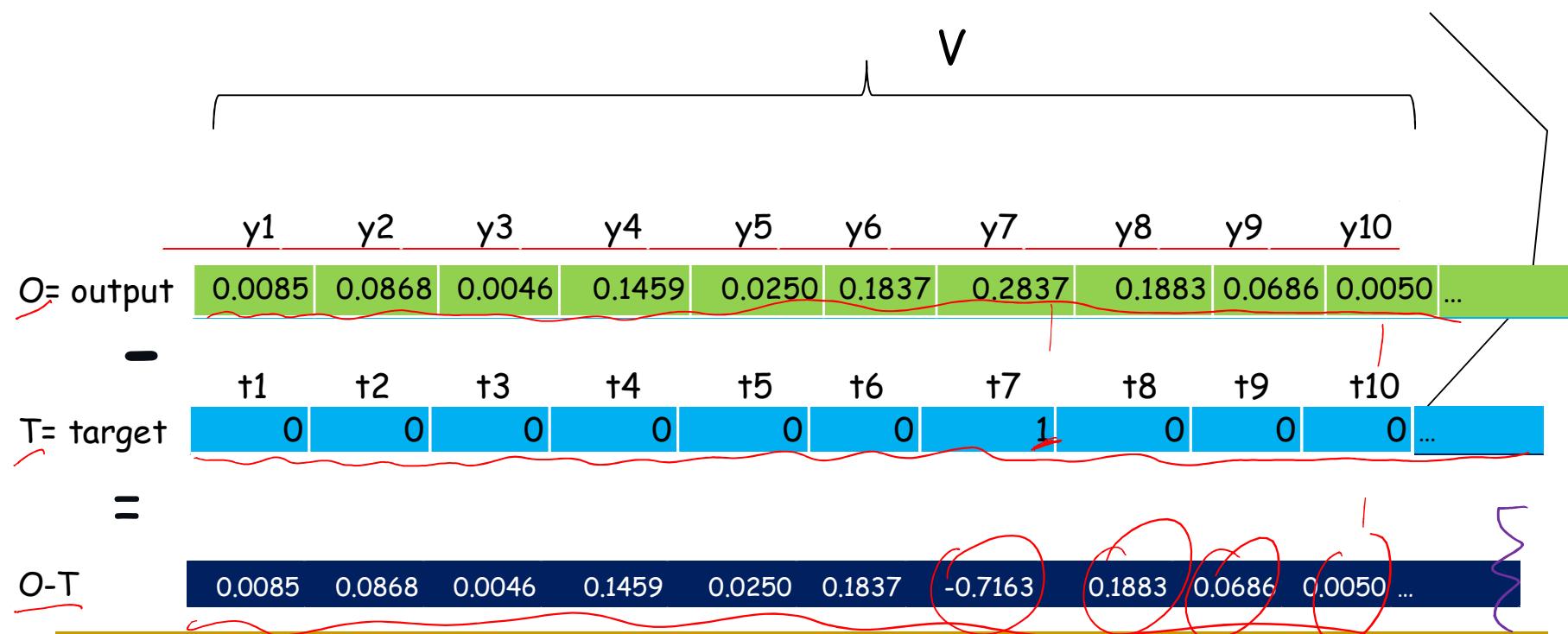


# Compute Error of Output Layer

remember the training set:

Instance	Context word -2	Context word -1	Context word +1	Context word +2	To Predict
1	the	brown	cat	chases	furry
2	brown	furry	chases	the	cat
...	...	...	...	...	...

```
// target = 1 hot representation of the target (furry) in the training set
```



# Backpropagate errors to adjust $W'$ and $W$

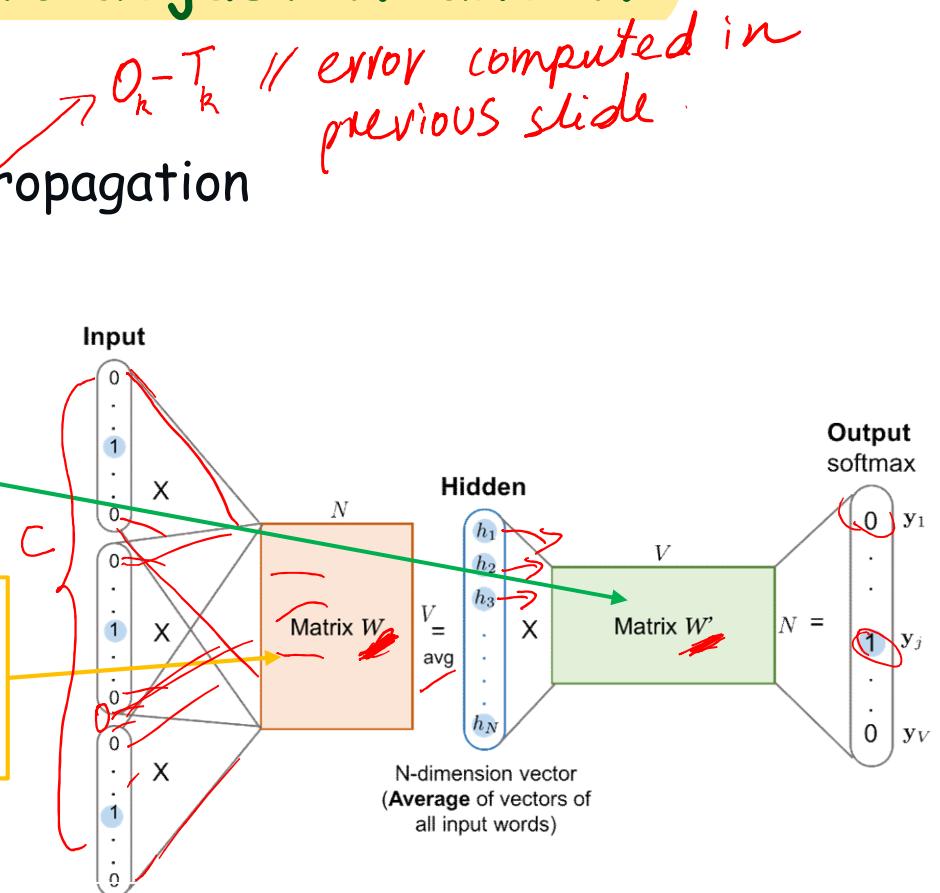
- Adjust  $W'$  and  $W$  using backpropagation
- <after a bit of math>, we get:

$$\text{new } w'_{jk} = \text{old } w'_{jk} - \eta (y_k - t_k) h_j$$

learning rate

$$\text{new } w_{ij} = w_{ij} - \left( \eta \frac{1}{C} \sum_{k=1}^V (y_k - t_k) w'_{jk} \right)$$

learning rate



where

- $\eta$ : learning rate
- $C$ : size of context (eg 4)

# Backpropagate errors to adjust $W$ and $W'$

$$w'_{jk} = w'_{jk} - \eta(y_k - t_k)h_j$$

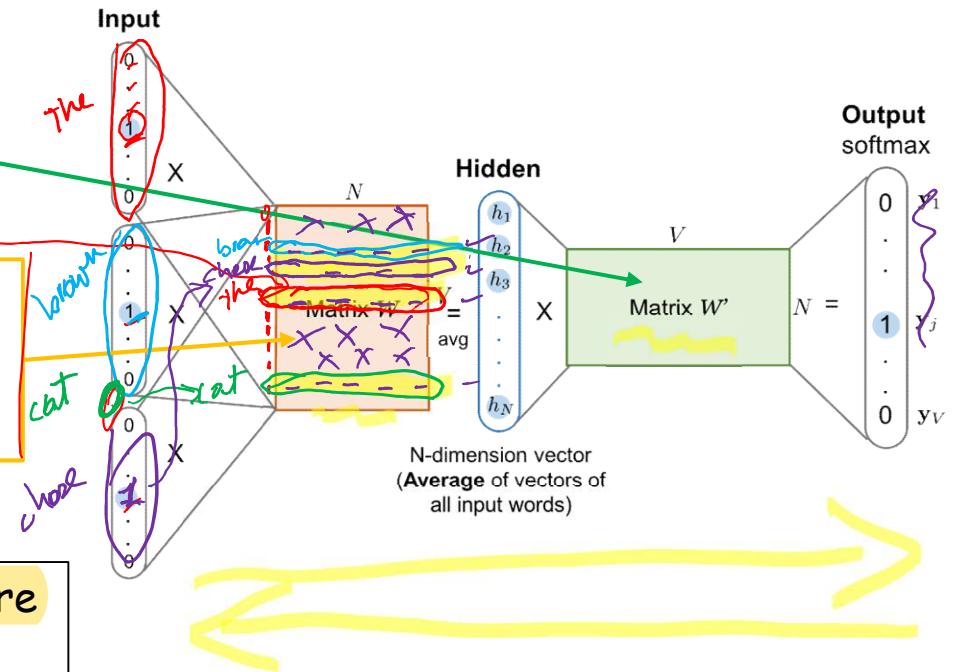
$$w_{ij} = w_{ij} - \eta \frac{1}{C} \sum_{k=1}^V (y_k - t_k) w'_{jk}$$

update ONLY the  $w_{ij}$  for the inputs where  $x_1=1$

i.e weight updates are ONLY done on the "rows" of  $W$  that correspond to the input word, not for all elements of  $W$ .

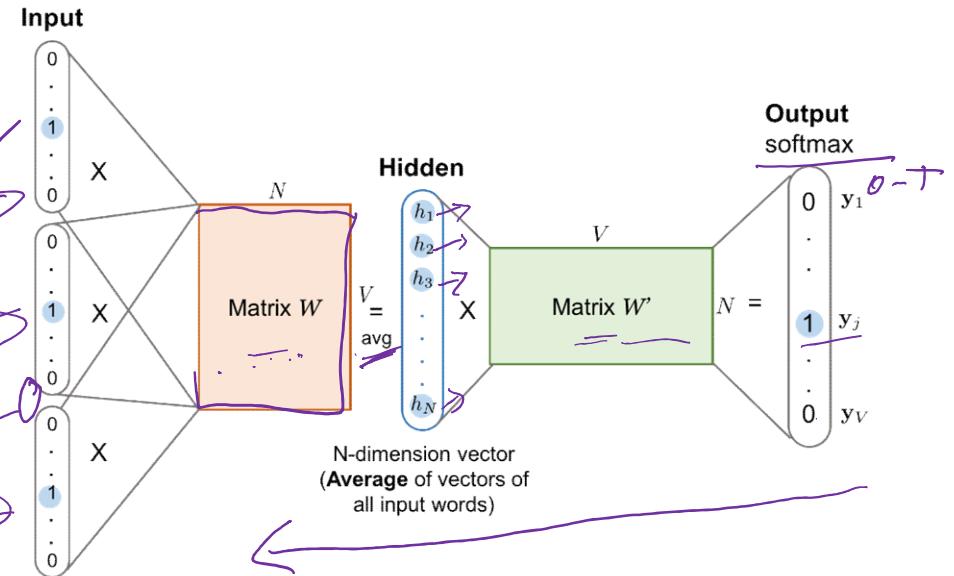
Remember that the input words are represented as 1-hot vectors, so only the weights of the word that has a "1" are updated.

This makes intuitive sense, as we want to update the weight only of the context word since its previous weights lead to an error.



# Word2Vec - FeedForward next data

Instance	Context t word -2	Context t word -1	Context t word +1	Context t word +2	To Predict
1	the	brown	eat	chases	furry ✓
2	brown	furry	chases	the	cat
3	furry	cat	the	mouse	chases
4	cat	chases	mouse	inside	the
5	chases	the	inside	the	mouse
6	the	mouse	the	house	inside



Instance	Input word	Context -2	Context -1	Context +1	Context +2
2	brown	0	0	0	0
2	furry	0	0	0	1
2	chases	0	0	0	0
1	the	0	1	0	0

- Iterate feedforward/ backprop until error is minimized
- Trained on Google News dataset (about 100 billion words).
- See: <https://code.google.com/archive/p/word2vec/>

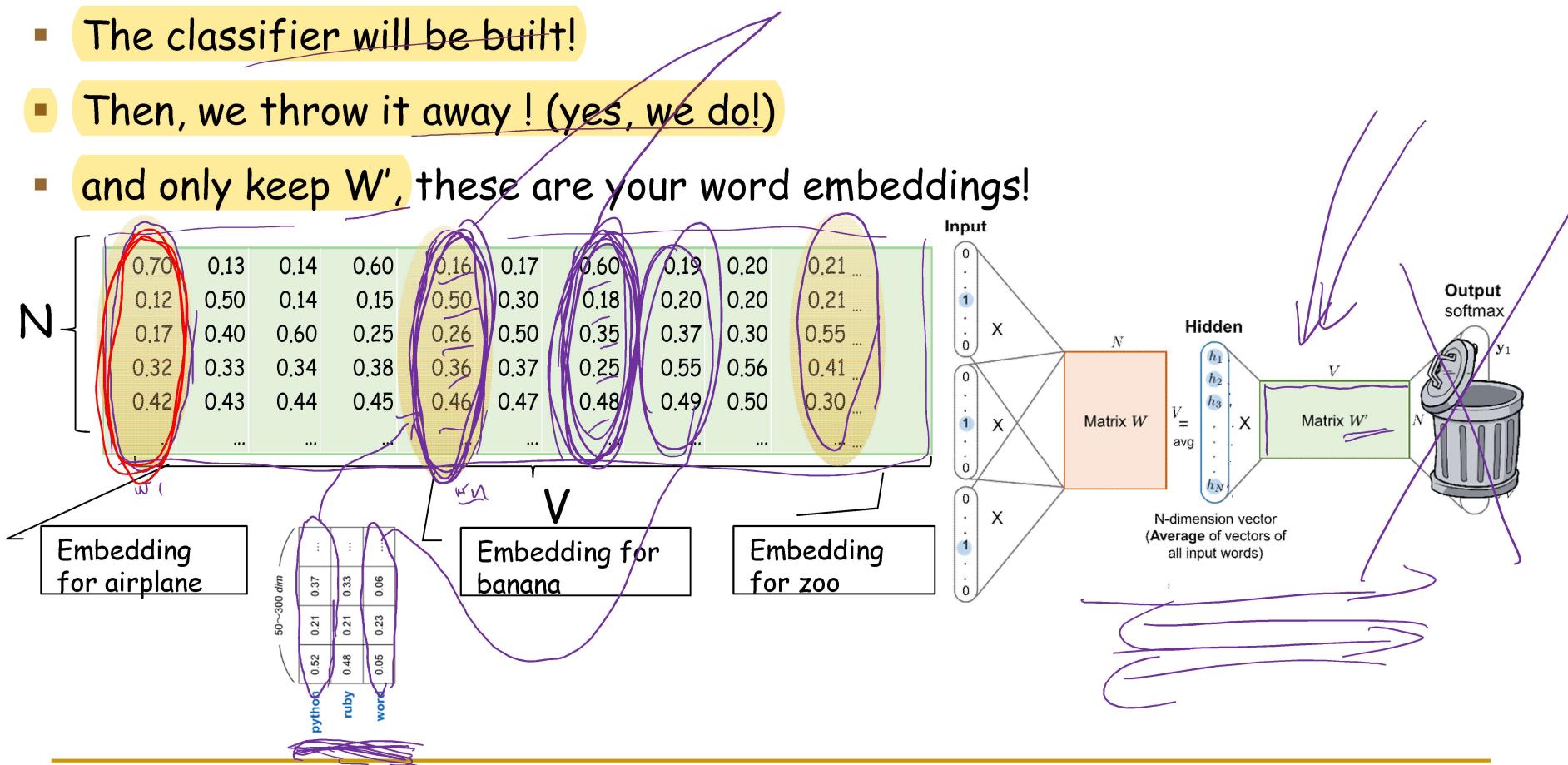


almost...

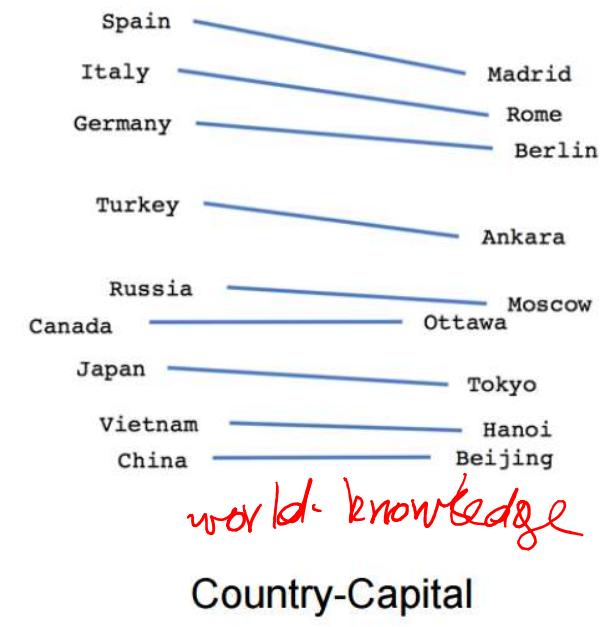
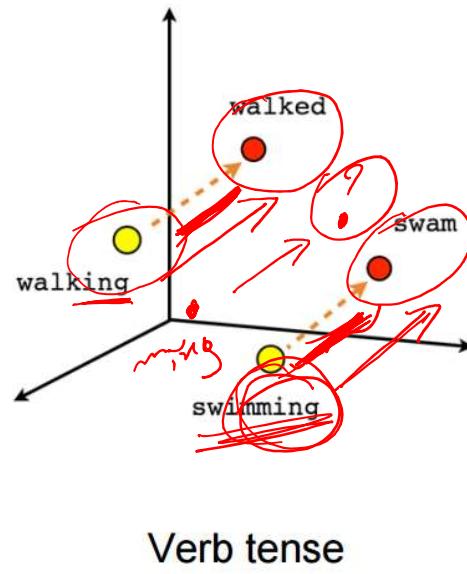
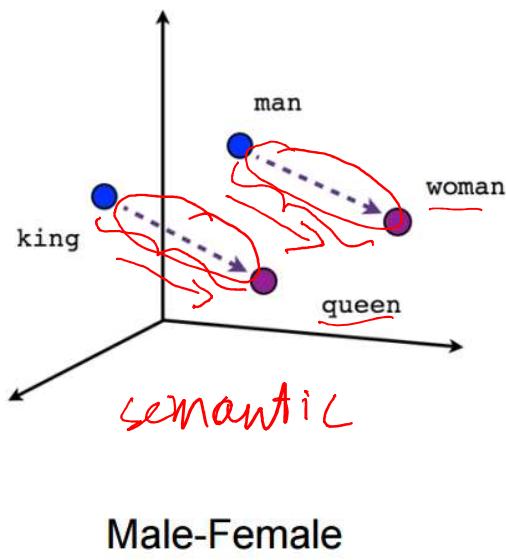
remember, we did all this to get embeddings...  
I'm not leaving 'till I get my embeddings!

# Word2Vec- Get the embeddings

- After many iterations of feedforward, backpropagation on the entire training set ...
  - The classifier will be built!
  - Then, we throw it away ! (yes, we do!)
  - and only keep  $W'$ , these are your word embeddings!

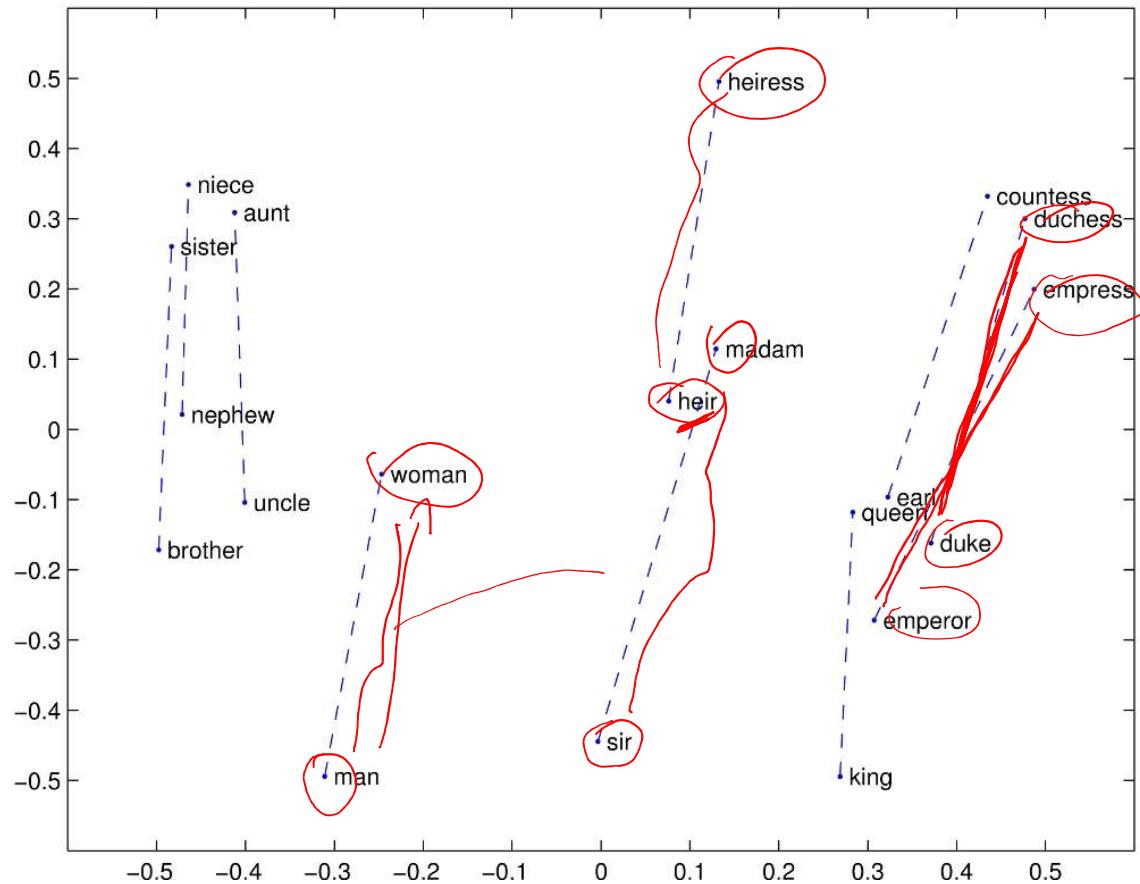


# Results

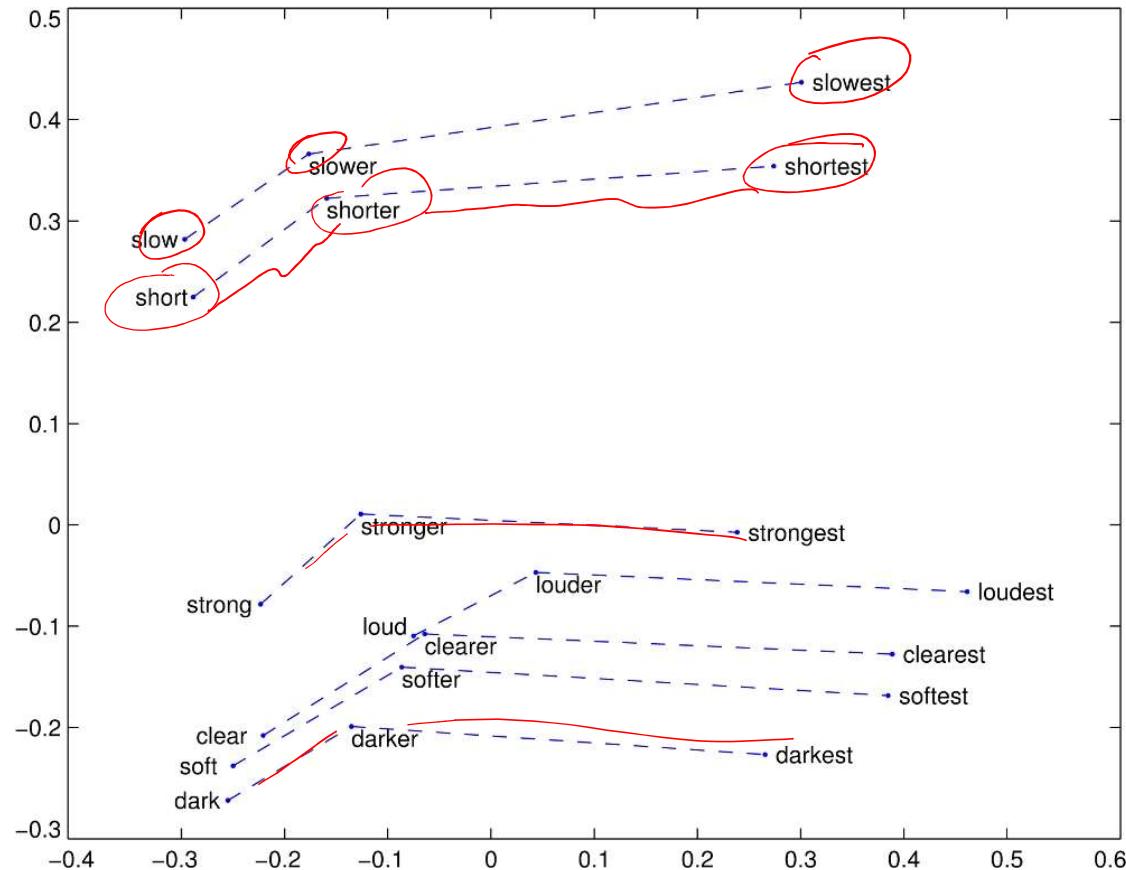


$$\begin{aligned}
 & (\text{vector}[\text{king}] - \text{vector}[\text{man}] + \text{vector}[\text{woman}]) \sim \text{vector}[\text{queen}] \\
 & (\text{vector}[\text{swam}] - \text{vector}[\text{walked}] + \text{vector}[\text{walking}]) \sim \text{vector}[\text{swimming}] \\
 & (\text{vector}[\text{Madrid}] - \text{vector}[\text{Spain}] + \text{vector}[\text{Italy}]) \sim \text{vector}[\text{Rome}]
 \end{aligned}$$

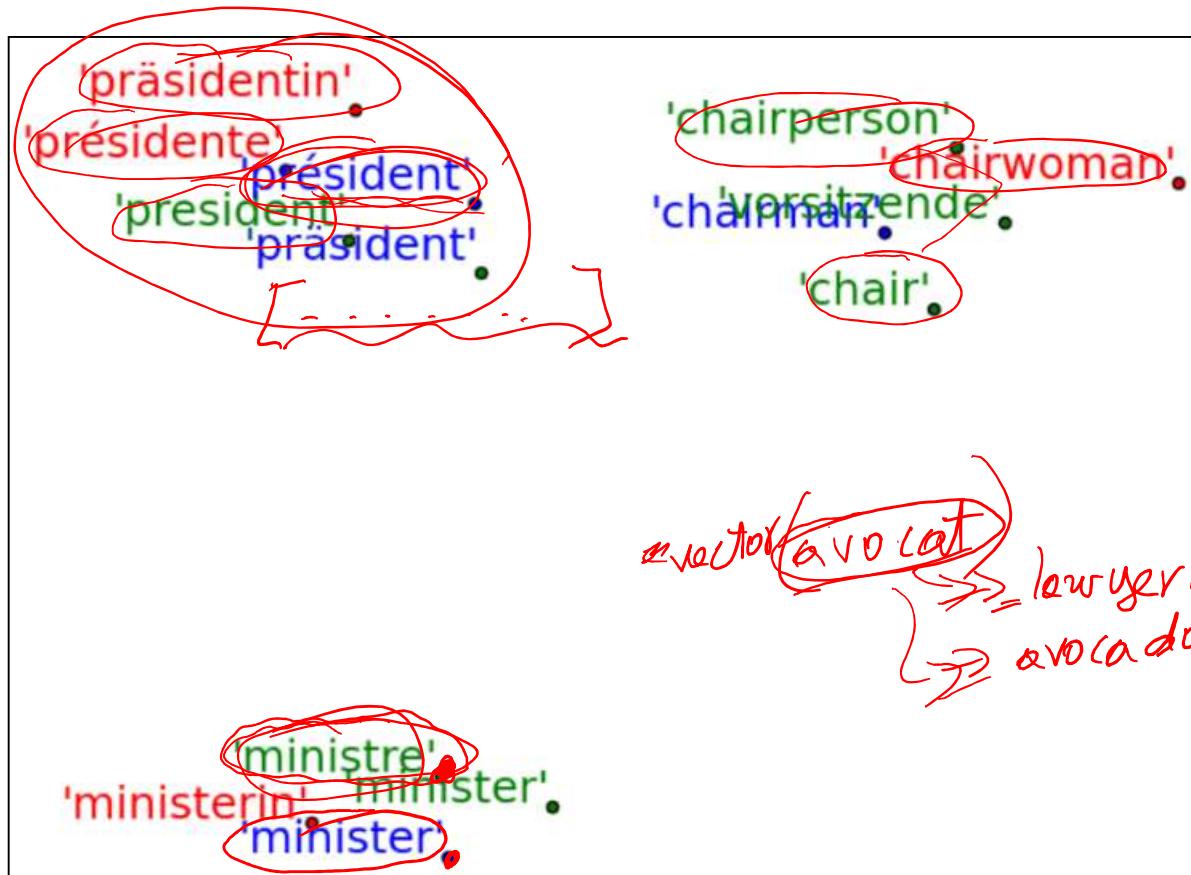
# Results



# Results



# Multilingual Word Embeddings



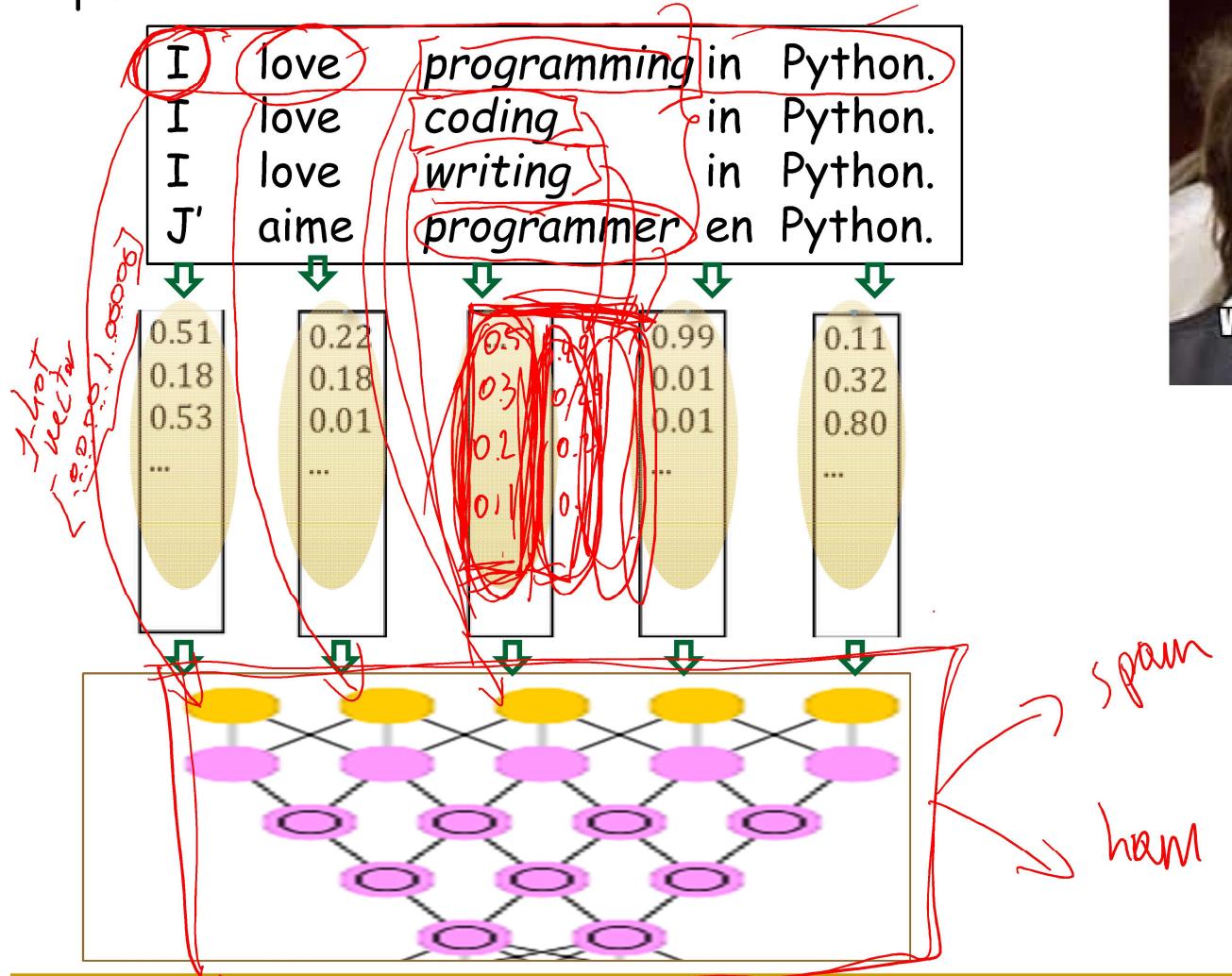
Words in different languages but with similar meanings (i.e. translations) are represented by similar vectors

urj

Used in Machine Translation

# Why are we doing this, again?

Input sentence:

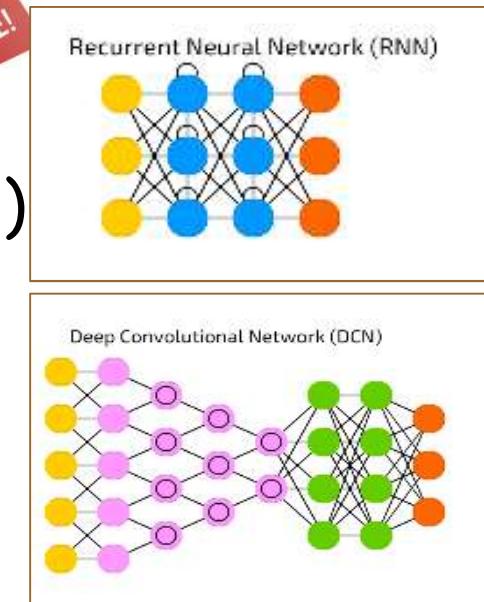


# Deep Learning for NLP

Deep learning models for NLP use:

- Vector representation of words
  - i.e., word embeddings *Word2Vec*

- Neural network structures
  - Recurrent Neural Networks (RNNs)
  - Recursive Neural Networks
  - Convolutional Networks (CNNs)
  - ...



# Today

1. Introduction 
2. Bag of word model 
3. n-gram models 
4. Deep Learning for NLP
  1. Word Embeddings 
  2. Recurrent Neural Networks

# Up Next

1. Introduction
2. Bag of word model
3. n-gram models
4. Deep Learning for NLP
  1. Word Embeddings
  2. Recurrent Neural Networks ↵