COMP 472: Artificial Intelligence
Natural Language Processing part 5
Word Embeddings video 5

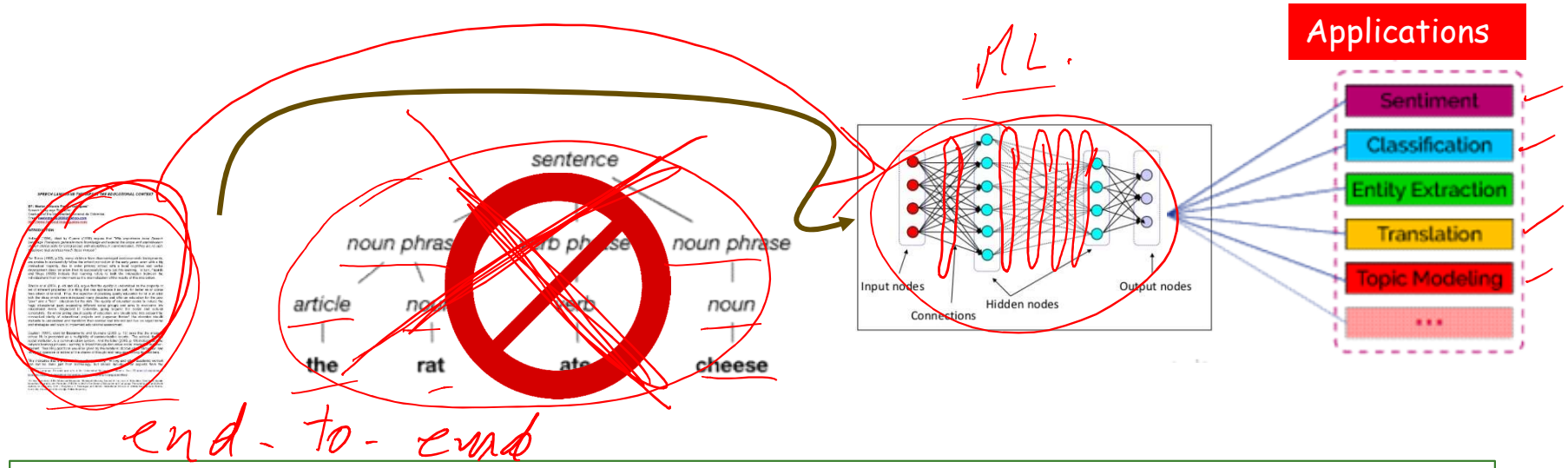
- Russell & Norvig: Section 24.1

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

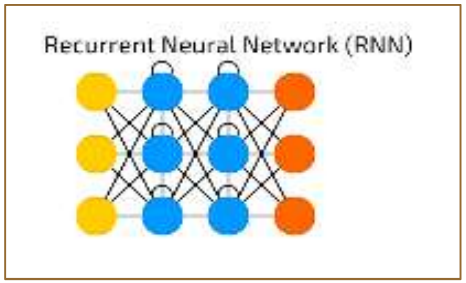
- Rules are developed automatically (using machine learning)
- And the linguistic features are found automatically!

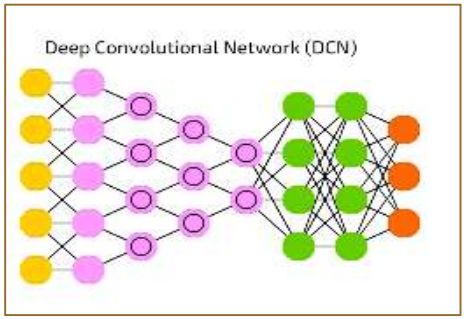
Deep Learning for NLP

Deep learning models for NLP use

#1 Vector representation of words
- i.e., word embeddings

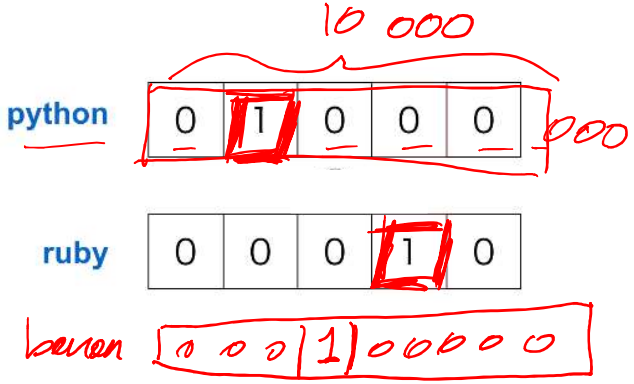
#2 Neural network structures
- Recurrent Neural Networks (RNNs)
- Convolutional Networks (CNNs)
- Recursive Neural Networks
- …



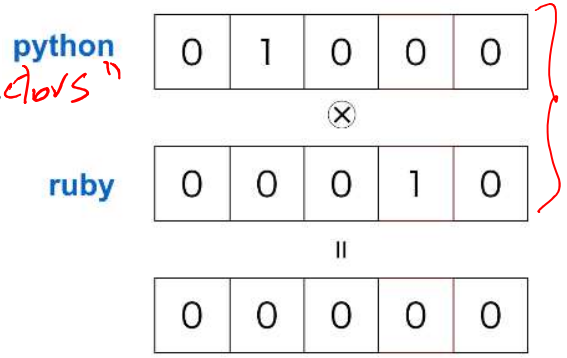


Word Embeddings

- To do NLP with neural networks, words need to be represented as vectors
- Traditional approach: "one hot vector"
  - Binary vector
  - Length = | vocab |
  - 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
  - However, similarity/distance between all "one hot vectors" are the same



http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/

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 !

https://en.wikipedia.org/wiki/Tomas_Mikolov

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

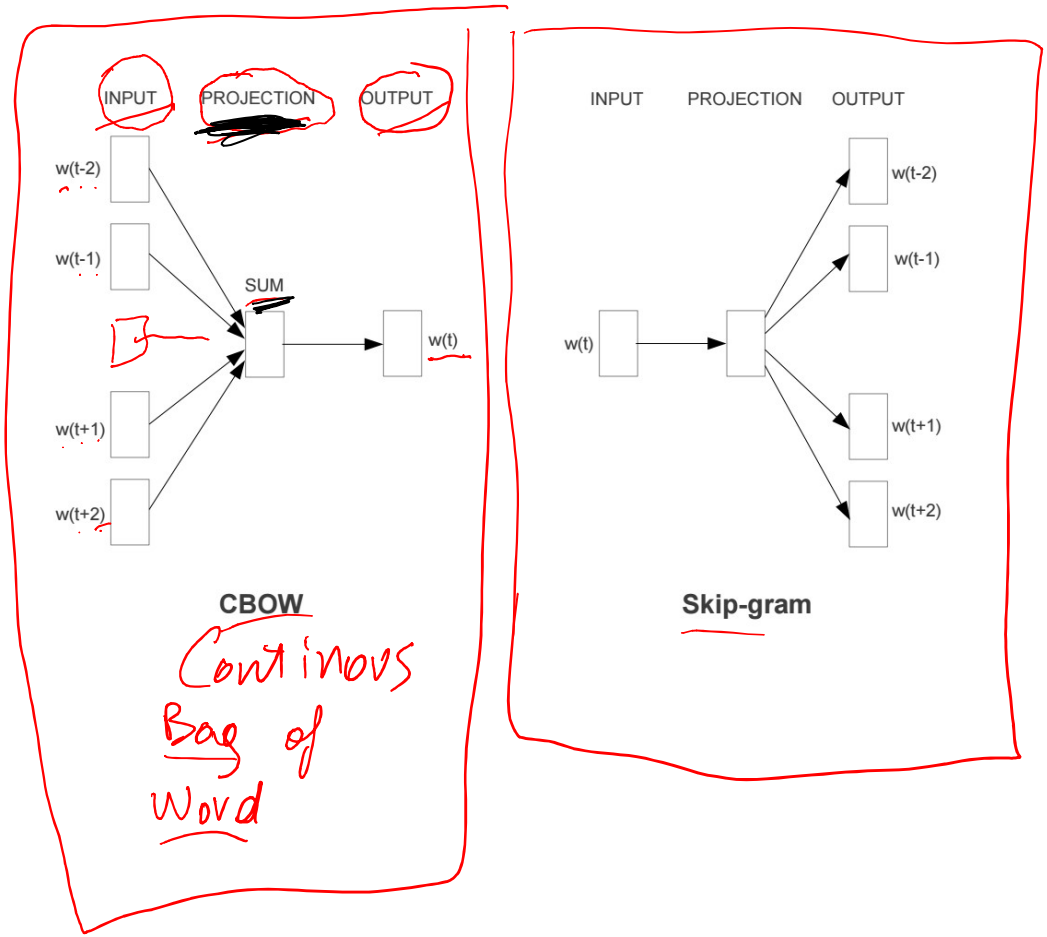
skip-gram

https://en.wikipedia.org/wiki/John_Rupert_Firth

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 word



http://rohanvarma.me/Word2Vec/

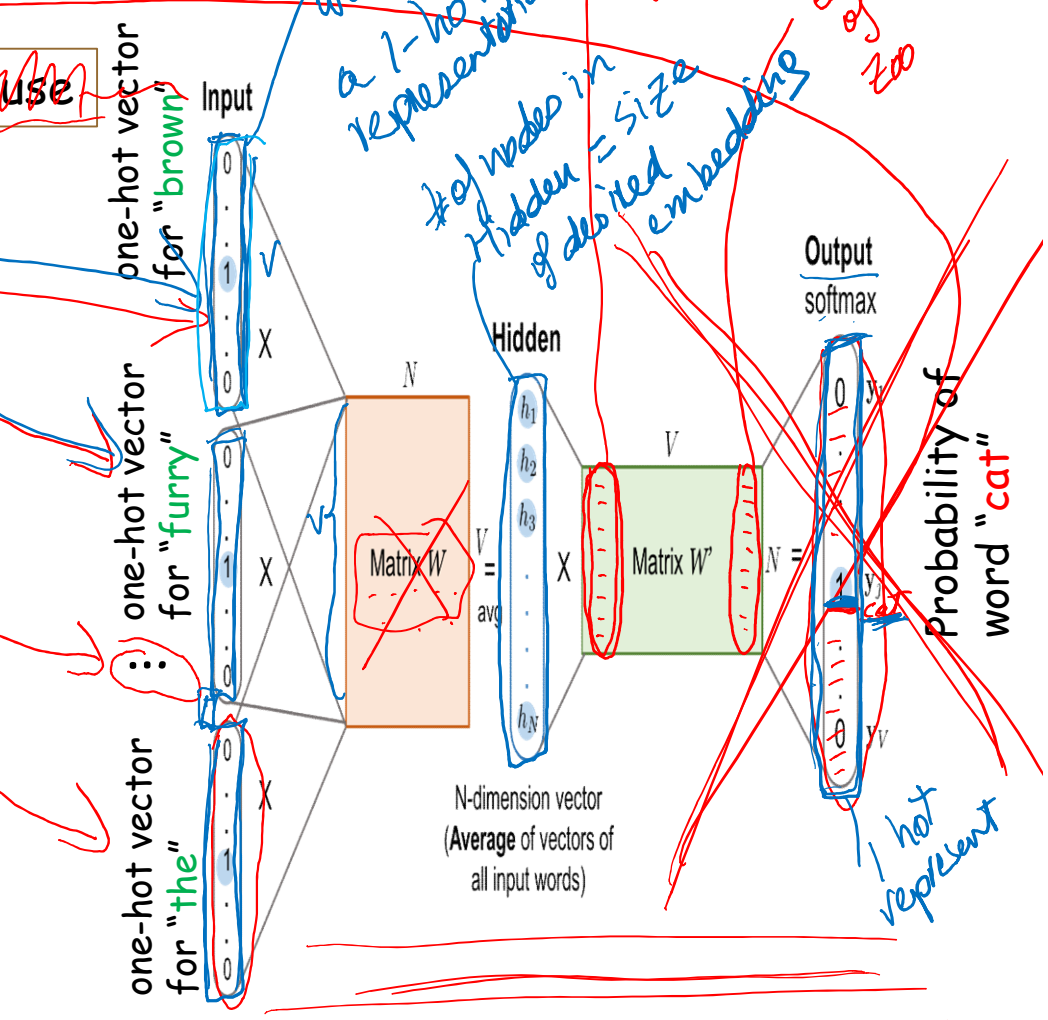
Word2Vec: CBOW Model

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).



https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

Word2Vec – Creating the Data Set

The brown furry cat chases the mouse inside the house.

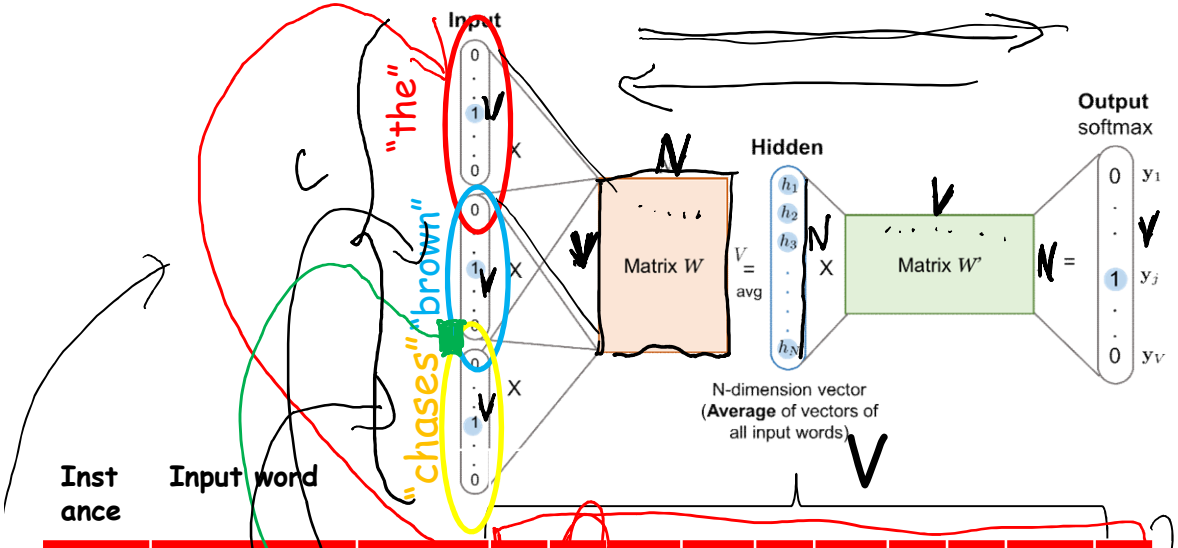
…

…

Assume context words are those in +/- 2 word window

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



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

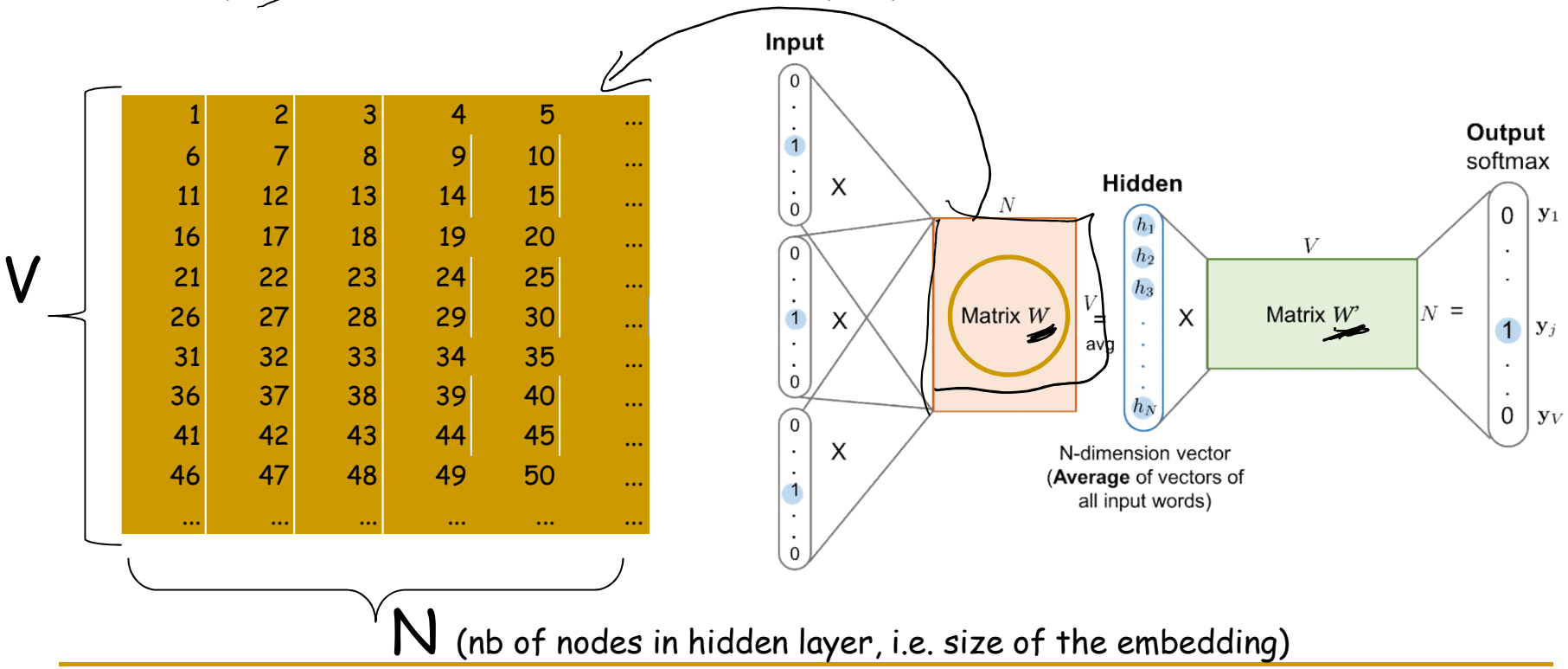
Instance	Input word										
2	Context -2	brown	0	0	0	0	0	1	0	0	0	...
2	Context -1	furry	0	0	0	0	1	0	0	0	0	...
2	Context +1	chases	0	0	0	0	0	0	0	0	1	...
	Context +2	the	0	1	0	0	0	0	0	0	0	...

...

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

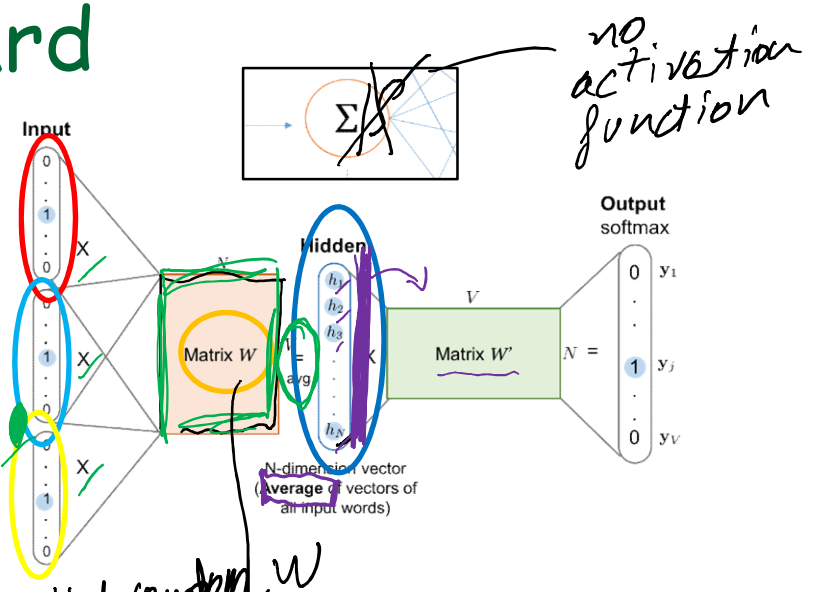
- Weight Matrix W between input & hidden layer
- W is a VxN matrix…
- Initially random but modified via backprop

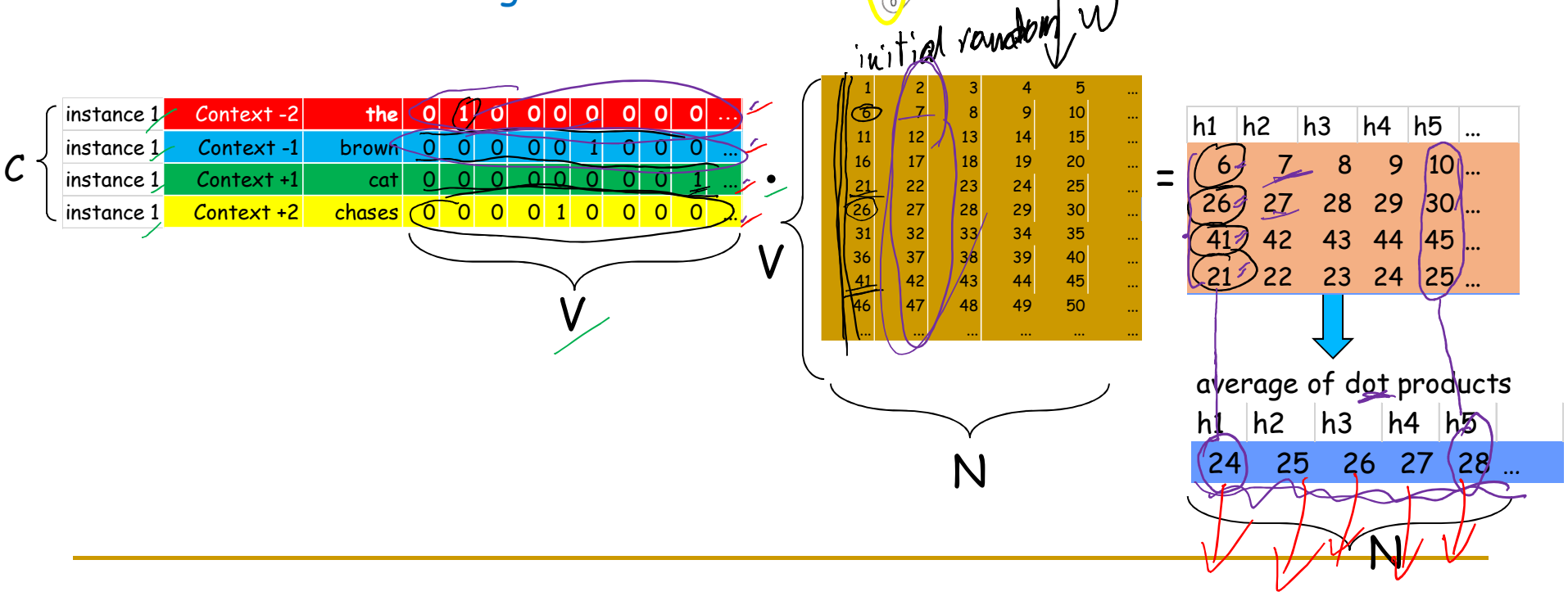


Word2Vec – Feedforward

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

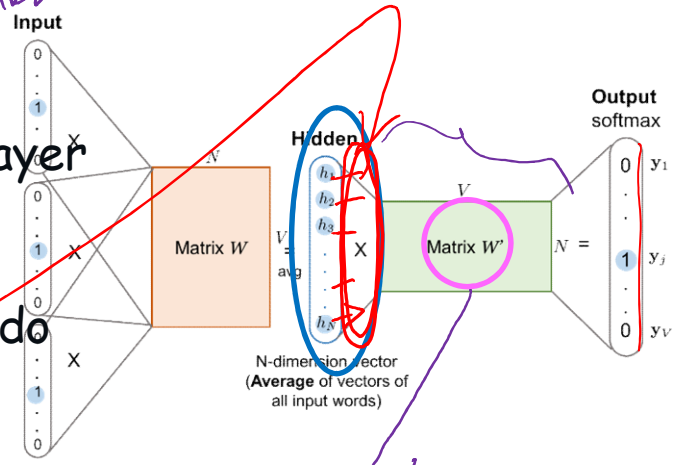
2. Then take the average





Word2Vec – Weights W'

- Weight Matrix W' between hidden & output layer
- W' is a NxV matrix…
- Initially random but modified via backprop
- Feed forward average of hidden neurons and do dot product with matrix W'



Matrix W'

0.70	0.13	0.14	0.60	0.16	0.17	0.60	0.19	0.20	0.21	…
0.12	0.50	0.14	0.15	0.50	0.30	0.18	0.20	0.20	0.21	…
0.17	0.40	0.60	0.25	0.26	0.50	0.35	0.37	0.30	0.55	…
0.32	0.33	0.34	0.38	0.36	0.37	0.25	0.55	0.56	0.41	…
0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.30	…
…	…	…	…	…	…	…	…	…	…	…

N

V

average of dot products

h1	h2	h3	h4	h5	
24	25	26	27	28	…

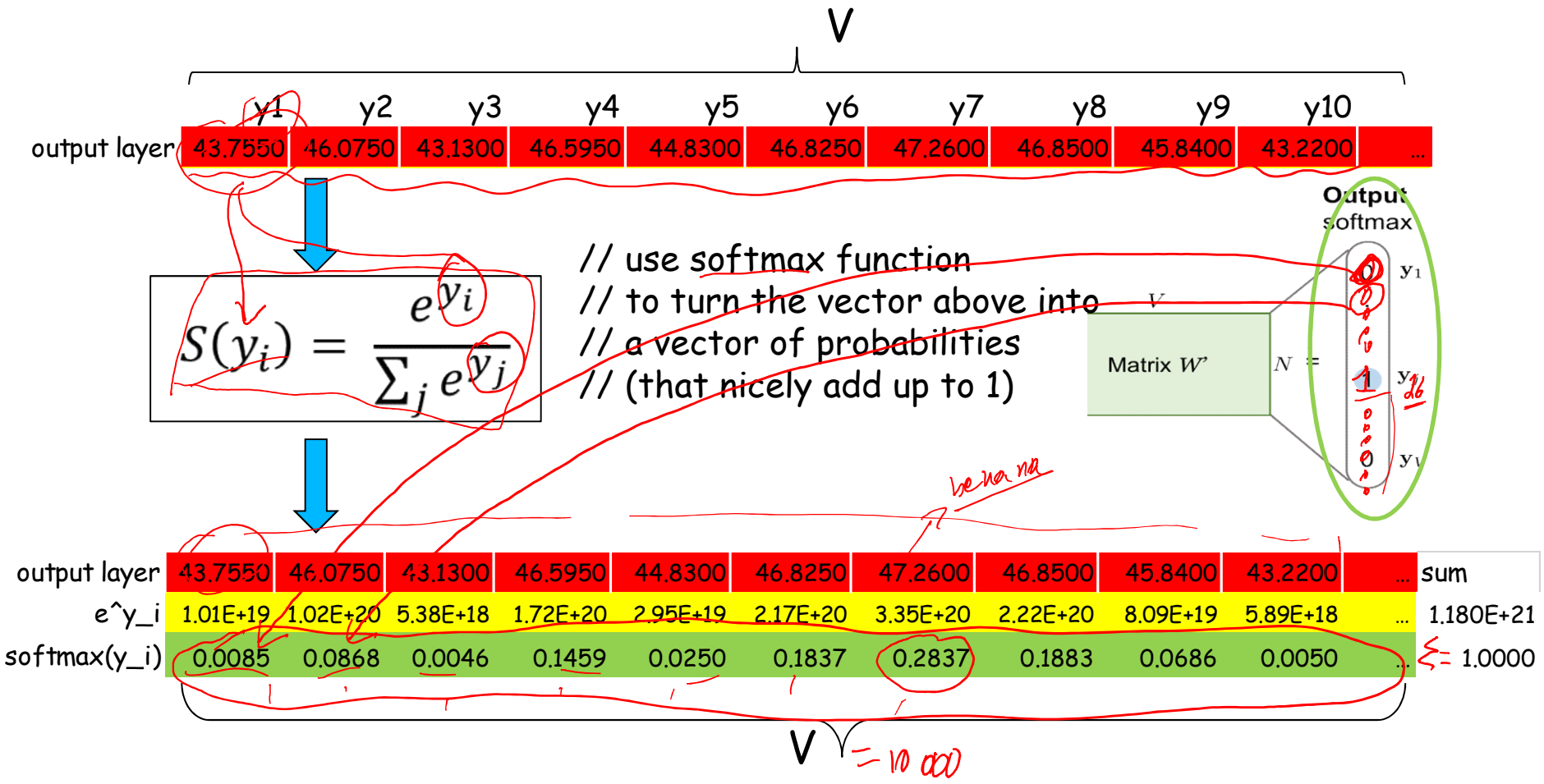
N (size of embeddings)

= output layer

y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	
43.7550	46.0750	43.1300	46.5950	44.8300	46.8250	47.2600	46.8500	45.8400	43.2200	…

V = 10 000

Turn dot product into probabilities

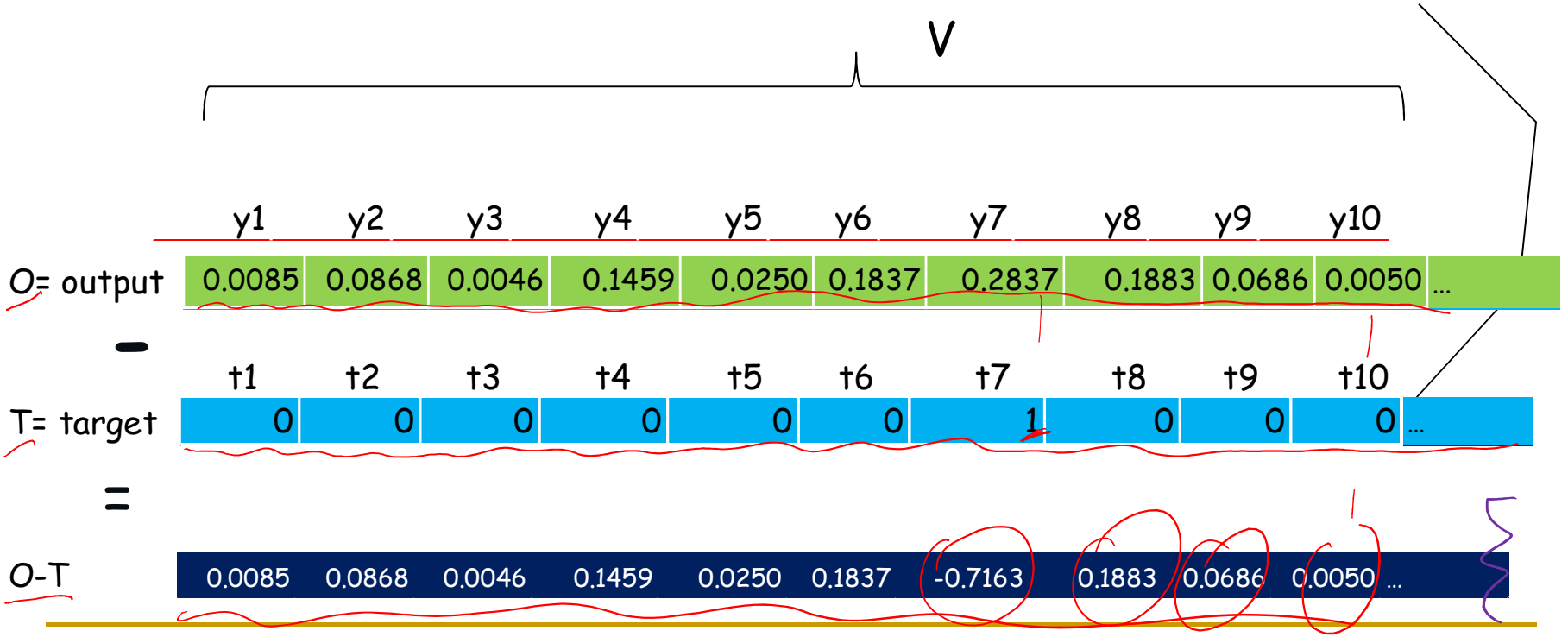


Compute Error of Output Layer

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

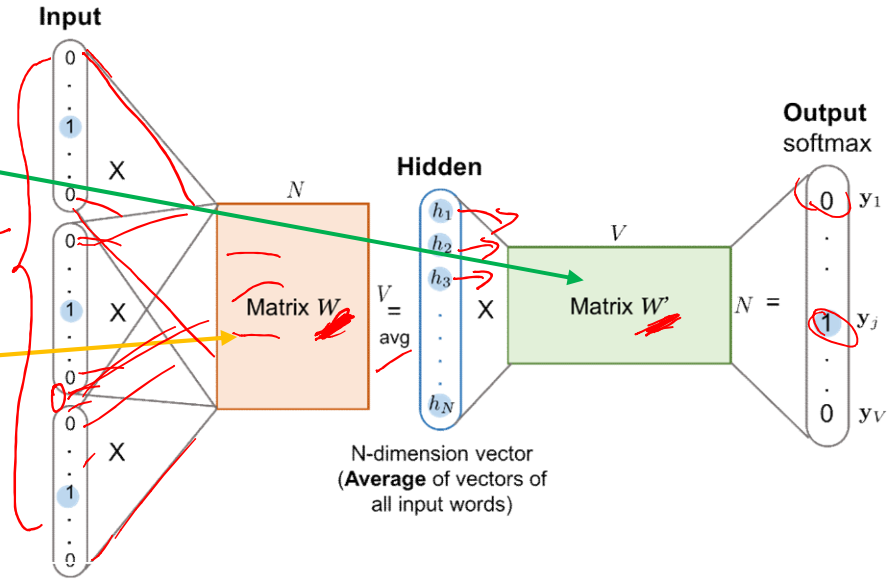


Backpropagate errors to adjust W and W'

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

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

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



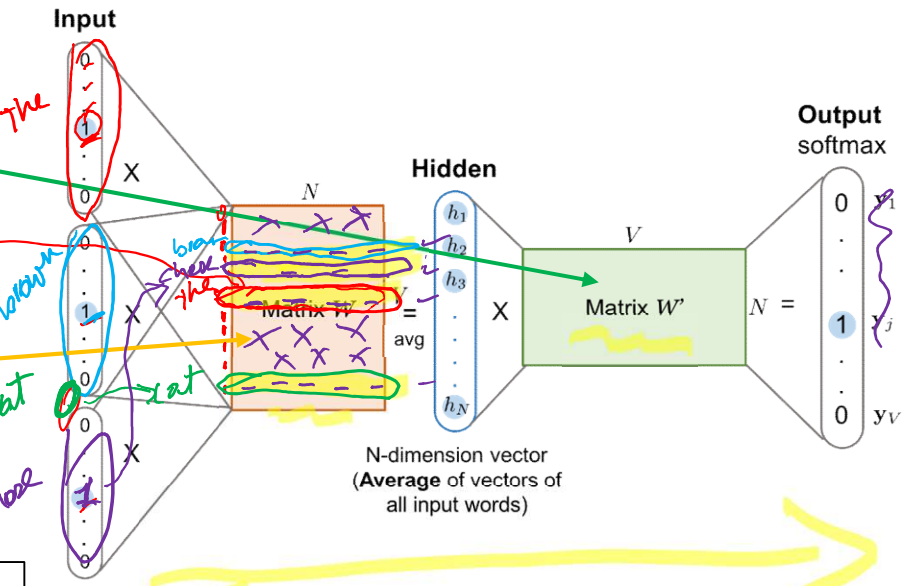
where

- η: 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

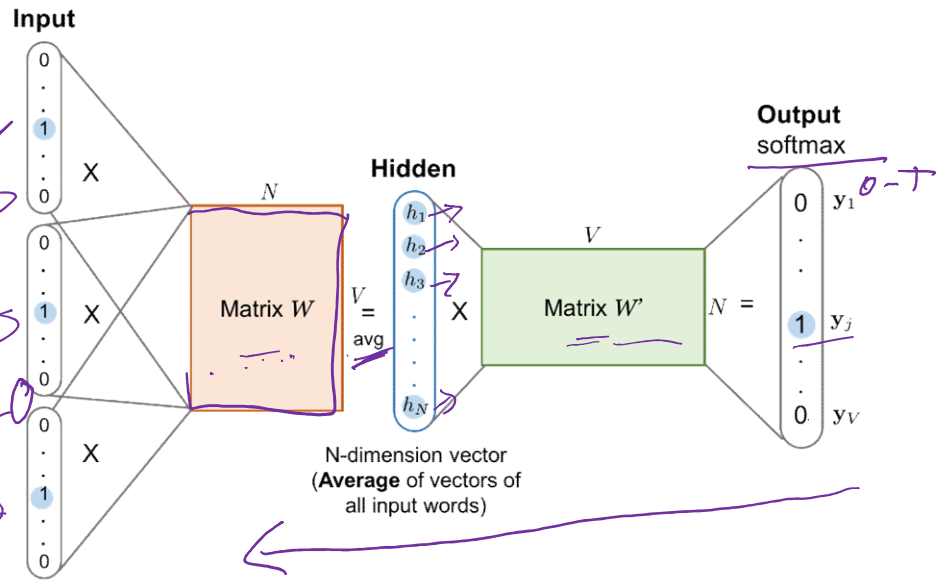
i.e weight updates are ONLY done on the "rows" of W that correspond to the input word, not for all elements of W.

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 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



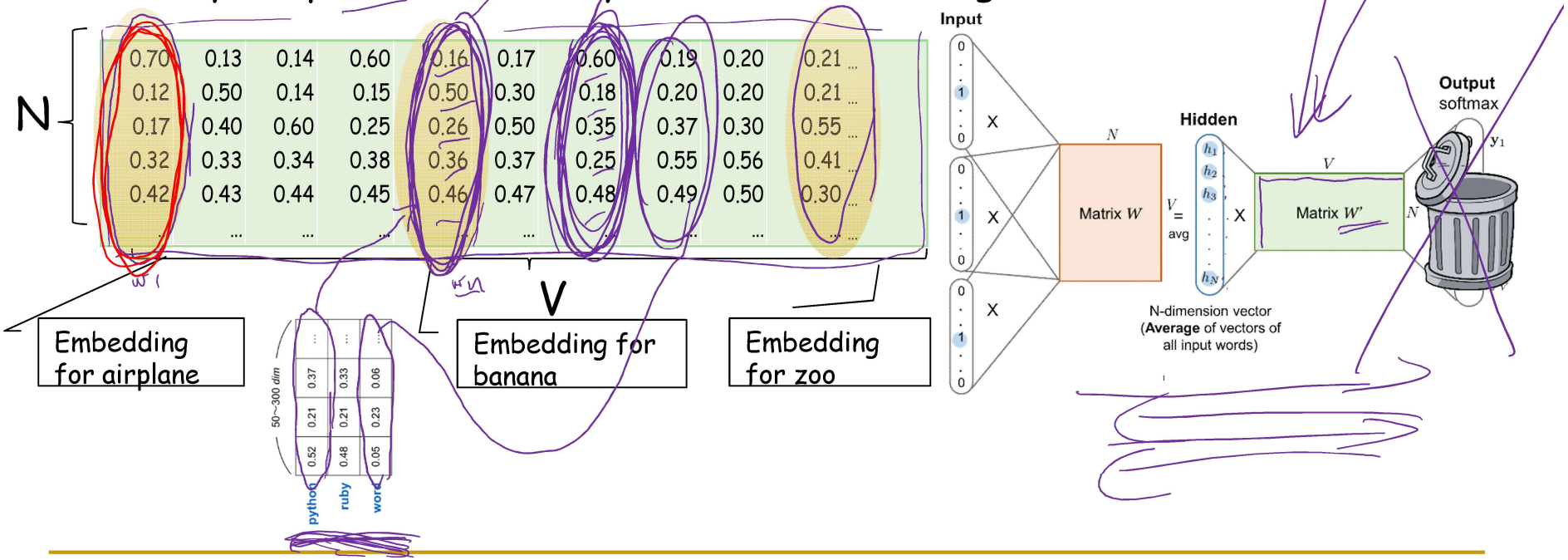
Instance	Input word											
2	Context -2	brown	0	0	0	0	0	1	0	0	0	...
2	Context -1	furry	0	0	0	0	1	0	0	0	0	...
2	Context +1	chases	0	0	0	0	0	0	0	0	1	...
1	Context +2	the	0	1	0	0	0	0	0	0	0	...

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

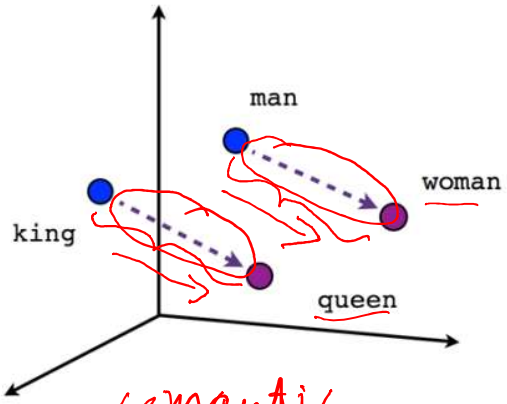
almost...
remember, we did all this to get embeddings...
I'm not leaving 'till I get my embeddings!

Word2Vec- Get the embeddings

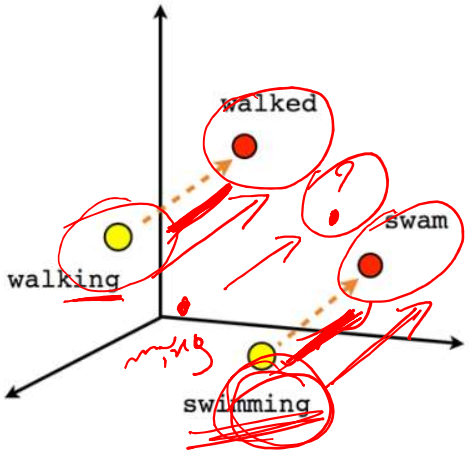
- After many iterations of feedforward, backpropagation on the entire training set …
- The classifier will be built!
- Then, we throw it away ! (yes, we do!)
- and only keep W', these are your word embeddings!



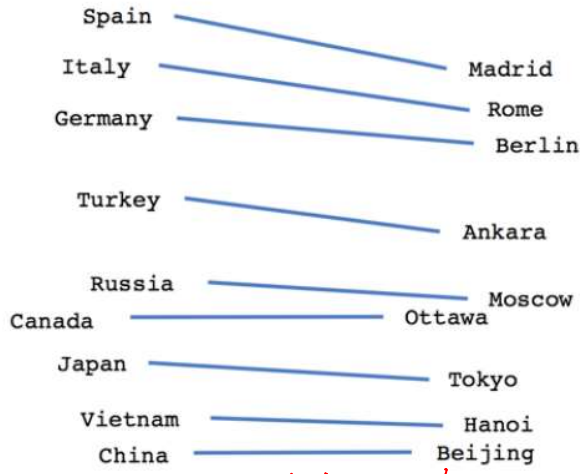
Results



Male-Female



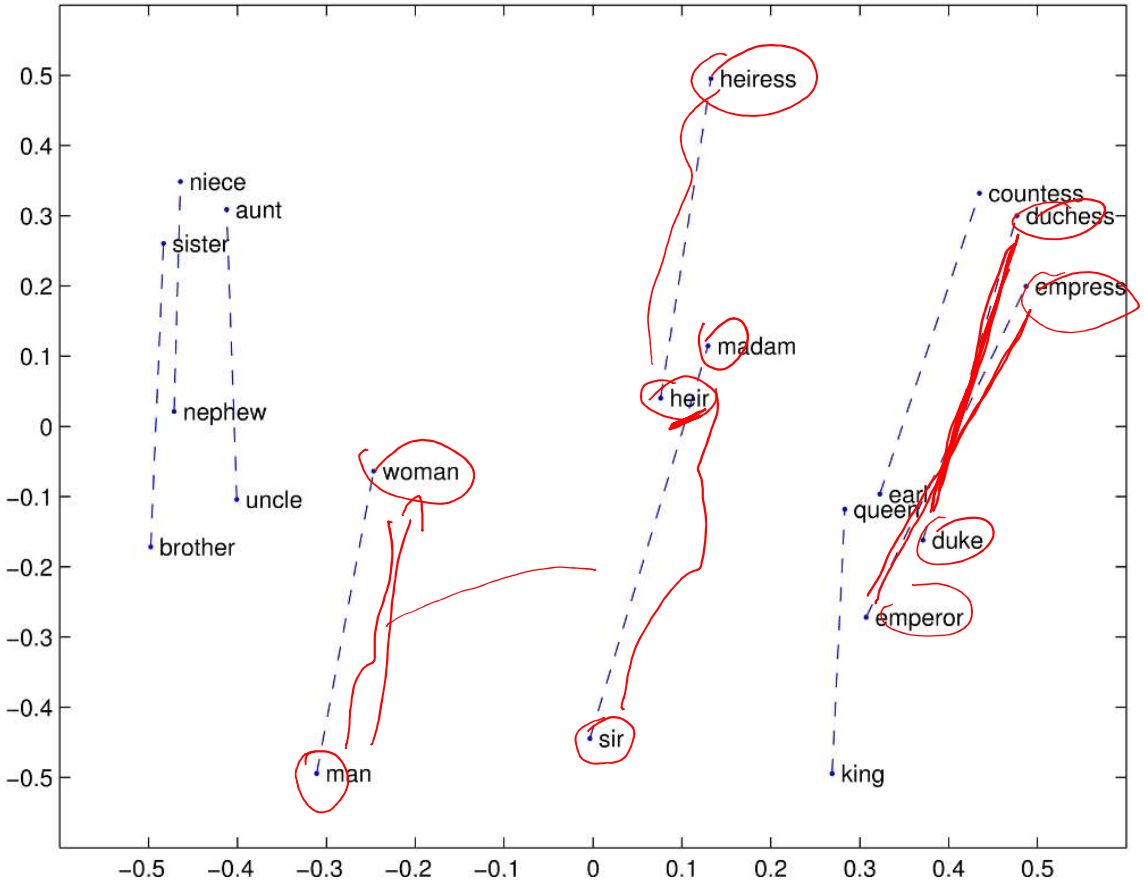
Verb tense



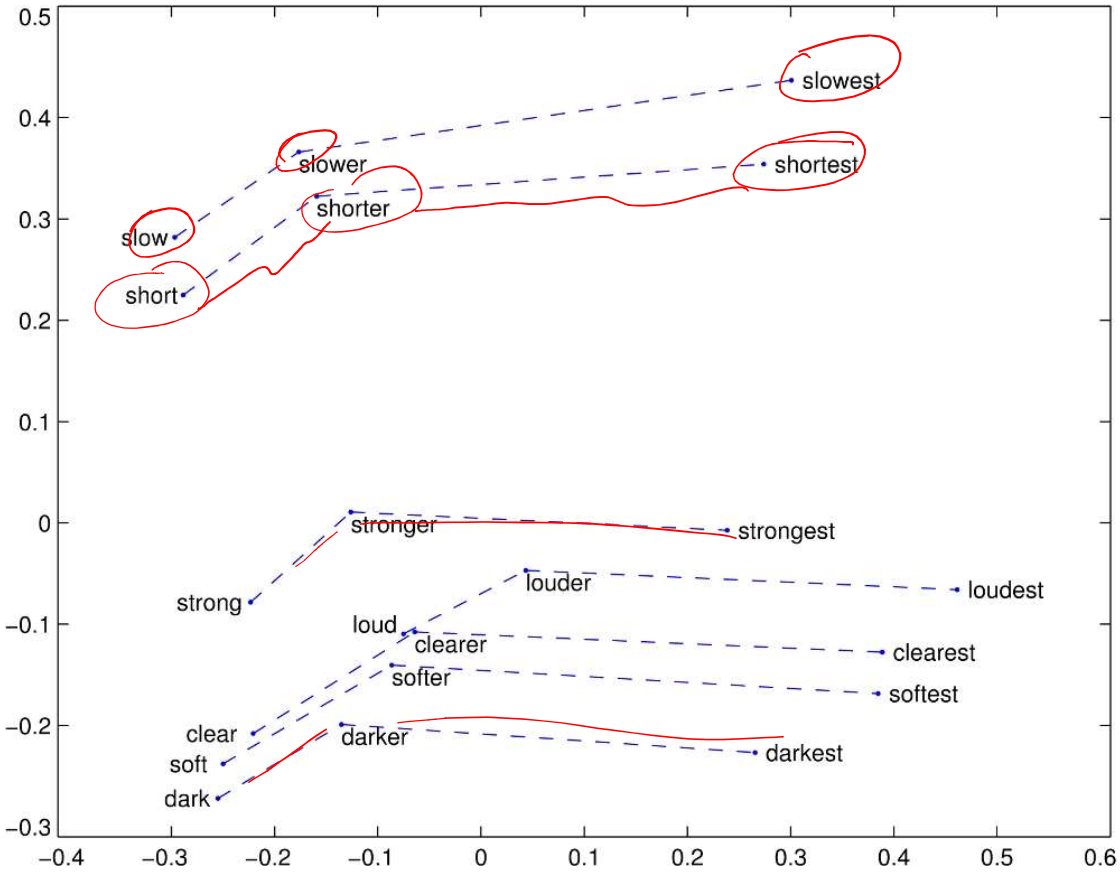
Country-Capital

(vector[king] - vector[man] + vector[woman]) ~ vector[queen]
(vector[swam] - vector[walked] + vector[walking]) ~ vector[swimming]
(vector[Madrid] - vector[Spain] + vector[Italy]) ~ vector[Rome]

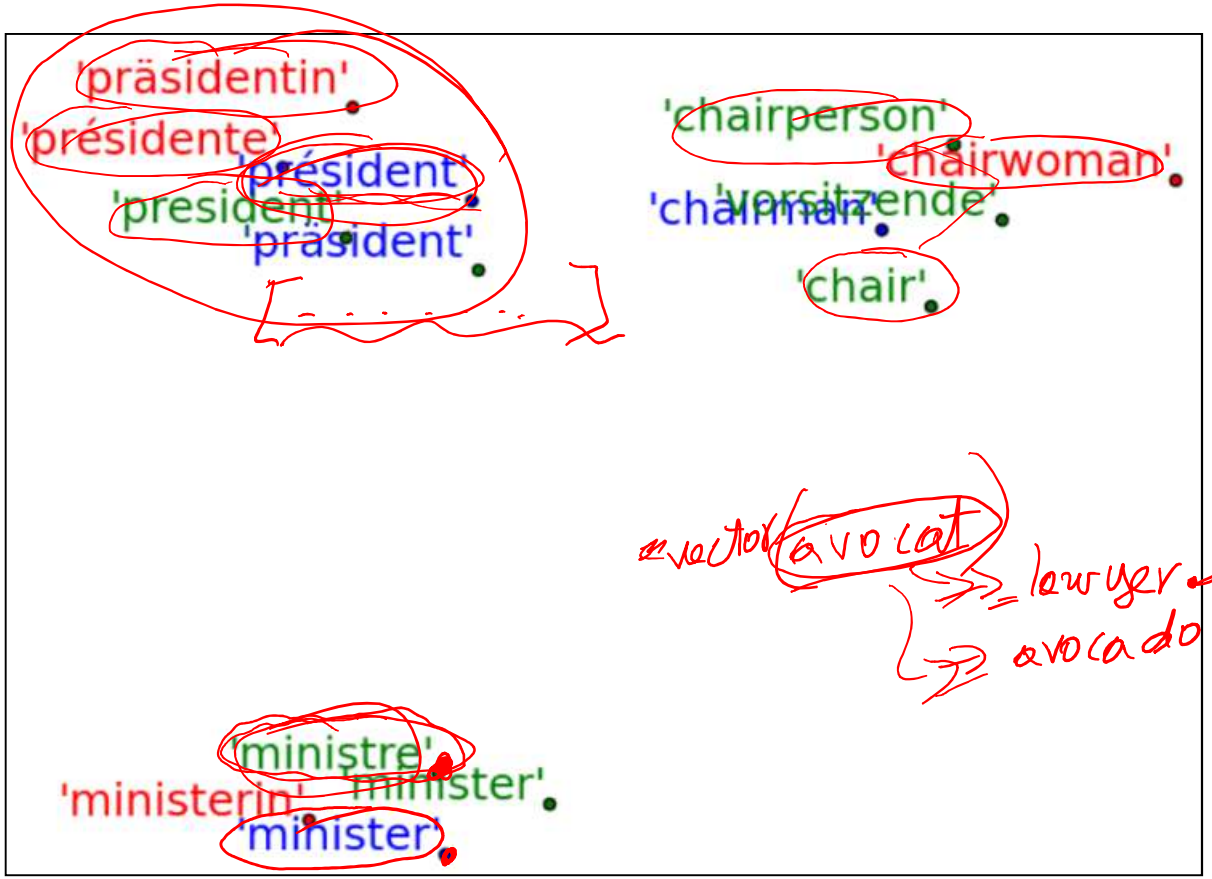
Results


niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

Results


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
slowest
slower
shortest
shorter
slow
short
stronger
strongest
strong
louder
loudest
loud
clearer
clearest
softer
softest
clear
darker
soft
darkest
dark

Multilingual Word Embeddings

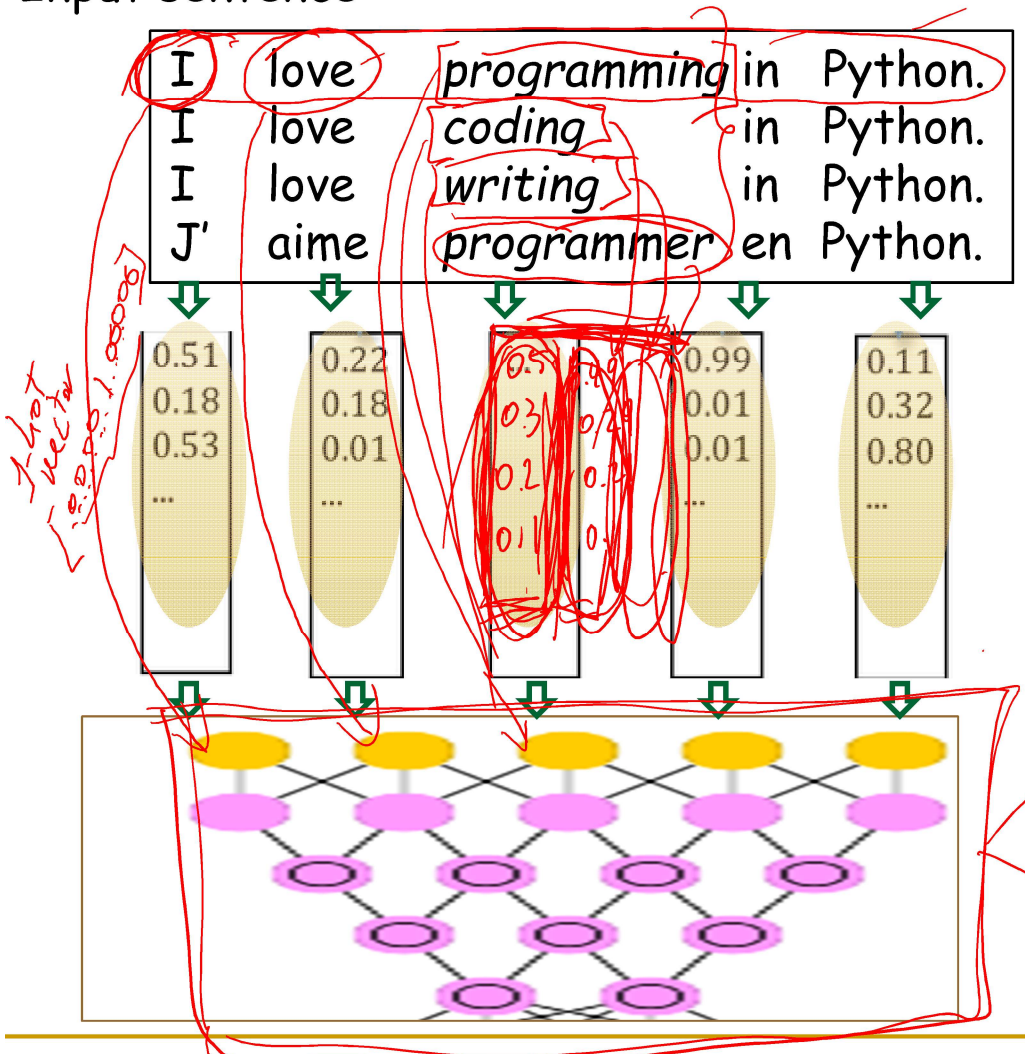


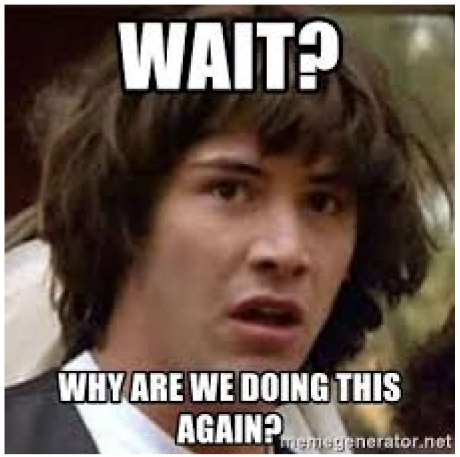
Words in different languages but with similar meanings (i.e. translations) are represented by similar vectors

Used in Machine Translation

Why are we doing this, again?

Input sentence:





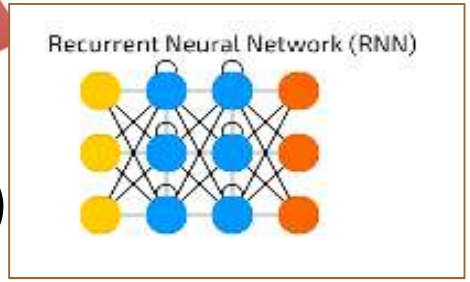
https://steemit.com/newbie/@clumsysilverdad/ain-t-about-followers-it-about-you-thou-shall-not-follow-in-vain

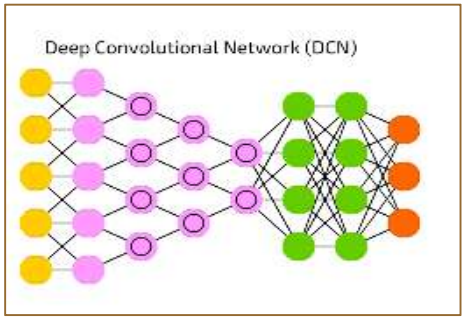
Deep Learning for NLP

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)
  - …

YOU ARE HERE!





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