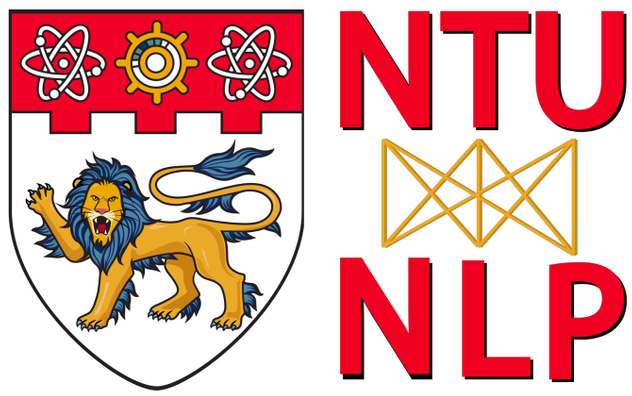
Deep Learning for Natural Language Processing

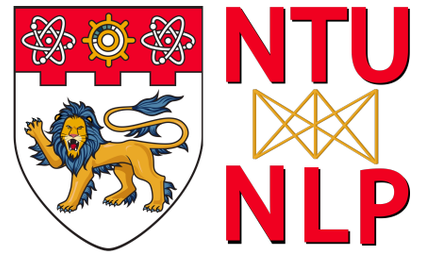
CE/CZ 4045

Shafiq Joty



Lecture 4: Word Vectors

Where we are



Models/Algorithms

- Linear models
- Feed-forward Neural Nets

NLP tasks/applications

- **Word** meaning

Word Meaning

NTU NLP

- The idea that is represented by a word, phrase, etc.
- How do we represent it in a computer?

Denotational Semantics

Common solution: Use e.g. WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

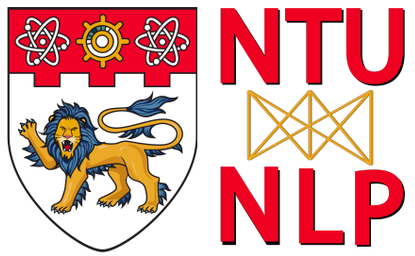
```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

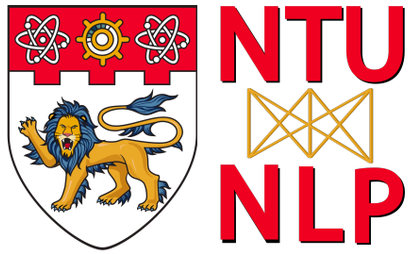
```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems with WordNet



- Great as a lexical resource but missing nuance
  - e.g. "proficient" is listed as a synonym for "good". This is only correct in some contexts.
- Missing new meanings of words, e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
- Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt

Problems with Discrete Representation



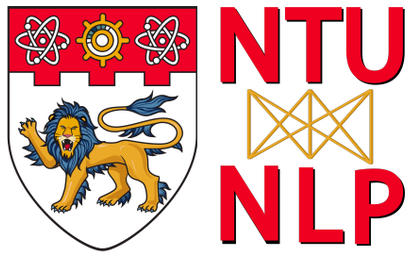
- Hard to compute accurate word similarity
- In traditional NLP, we regard words as discrete symbols: hotel, conference, motel

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
  - How to represent polysemous words?

Representing words by their context



- Instead: learn to encode similarity in the vectors themselves

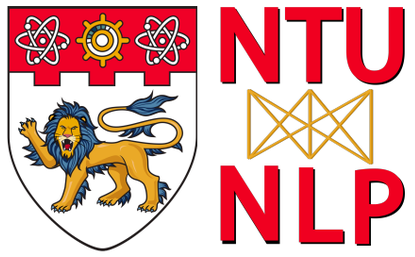
- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
    - "You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of w to build up a representation of w

Representing words by their context



- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of w to build up a representation of w

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Word Vectors

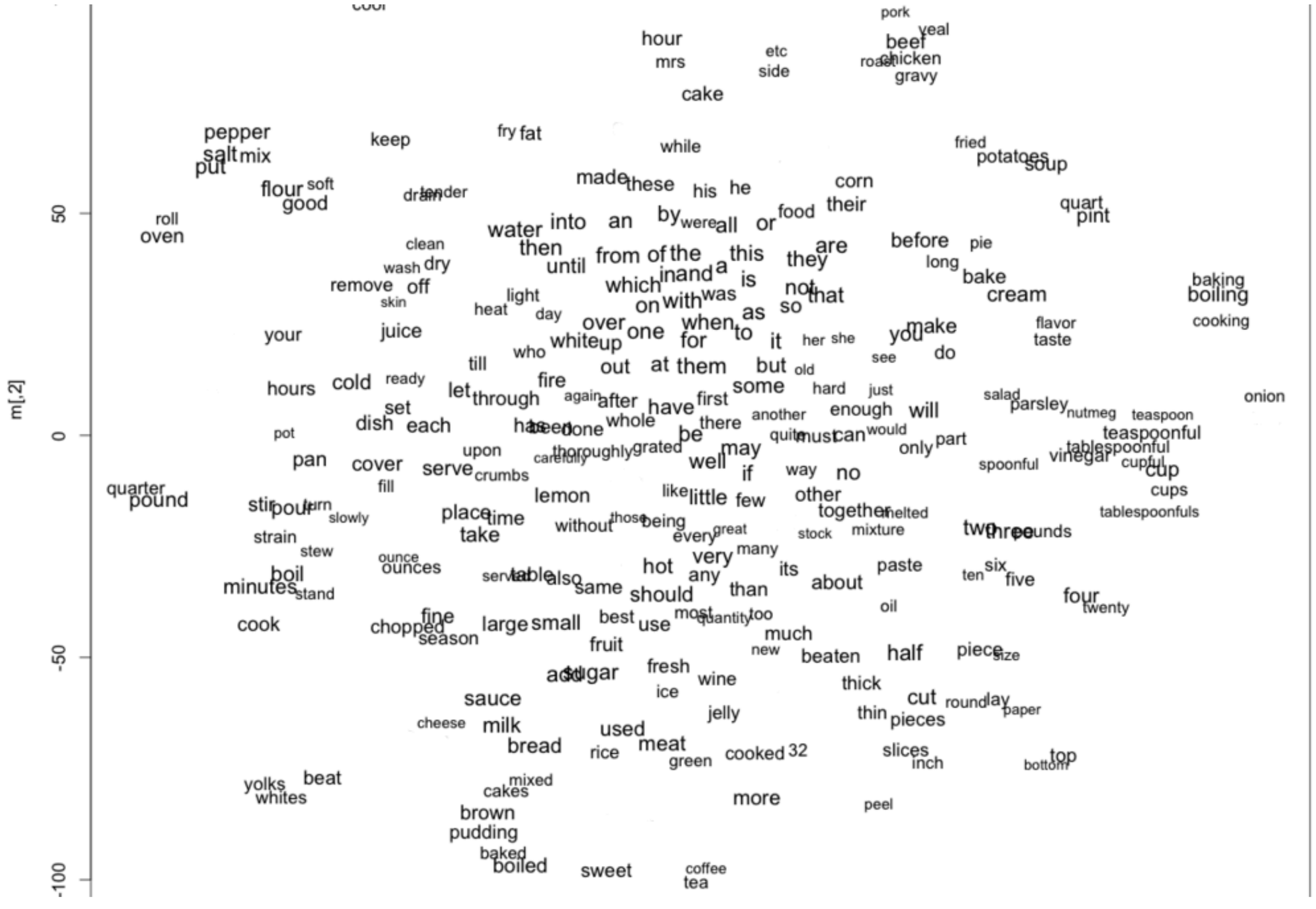
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

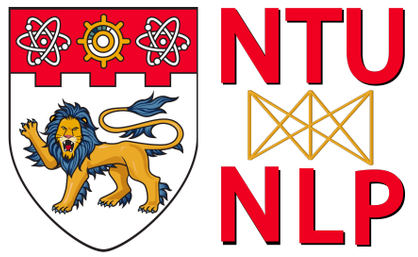
word vectors are sometimes called word embeddings or word representations.
They are a distributed representation.

Source: Stanford 224N

Word Vectors



Lecture Plan



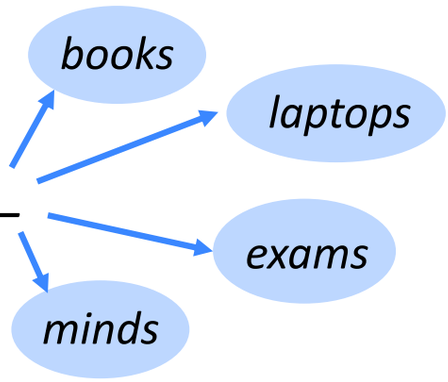
- Word meaning & Distributed representations
- Word2Vec models & Negative sampling
- Incorporating subword information (FastText)
- Glove
- Word vector evaluation
- Cross-lingual word embeddings

The Earliest Neural Model to Learn Word Embeddings

NTU NLP

A **language model** takes a list of words (history), and attempts to predict the word that follows them

the students opened their _____



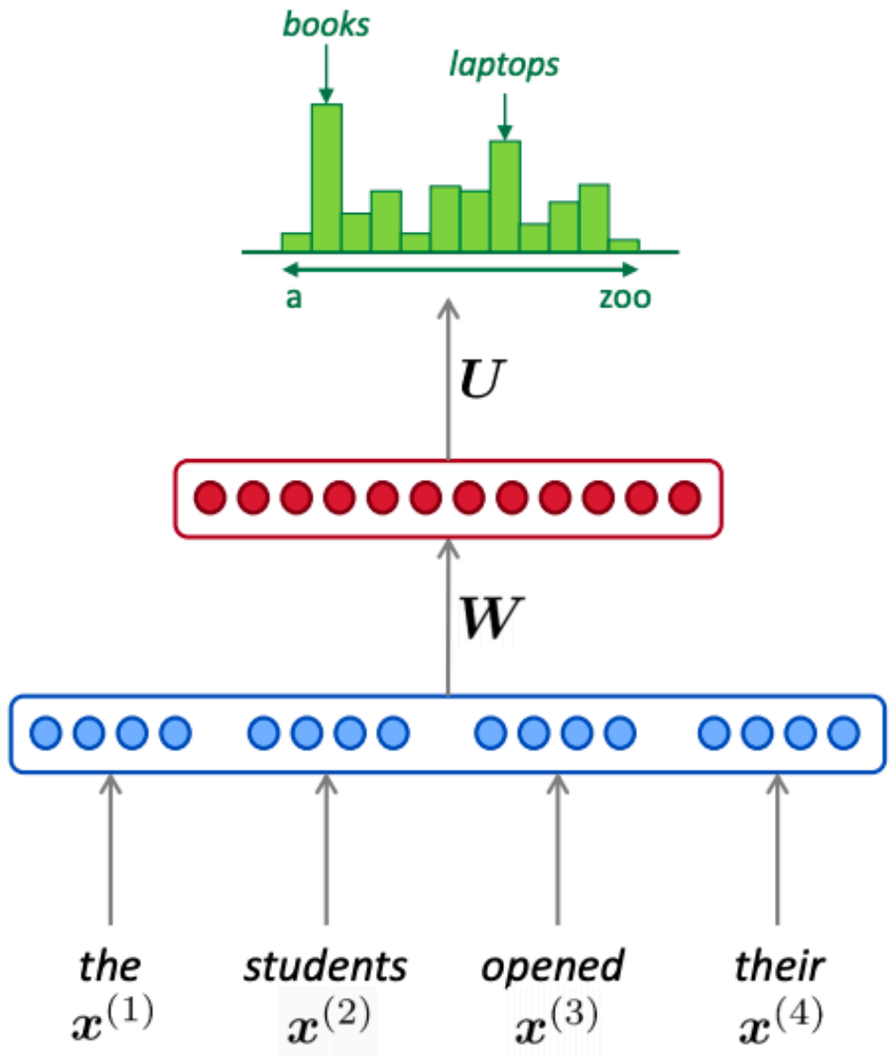
Google

what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

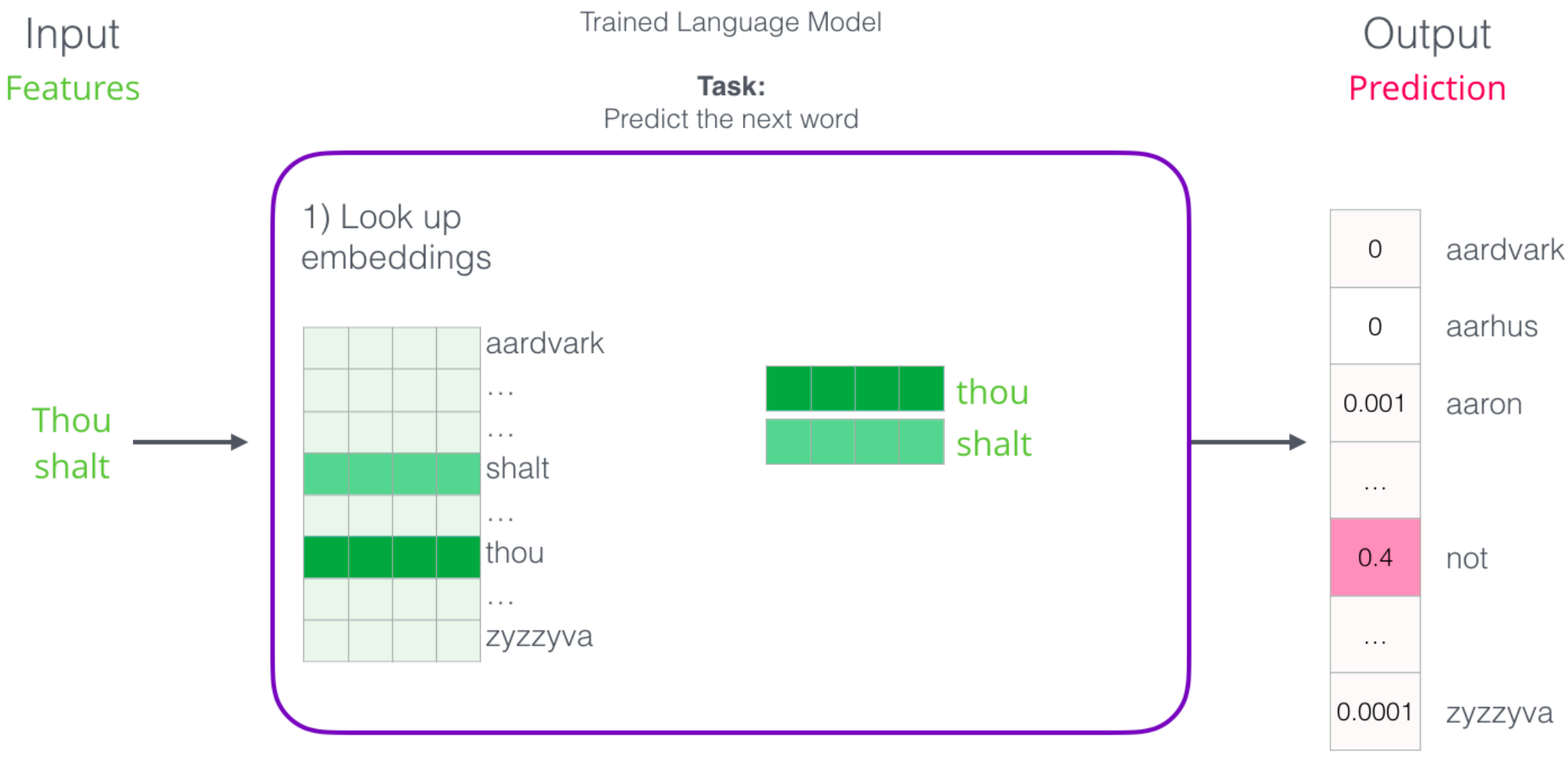
Google Search I'm Feeling Lucky

Assignment # 1



Bengio, 2003

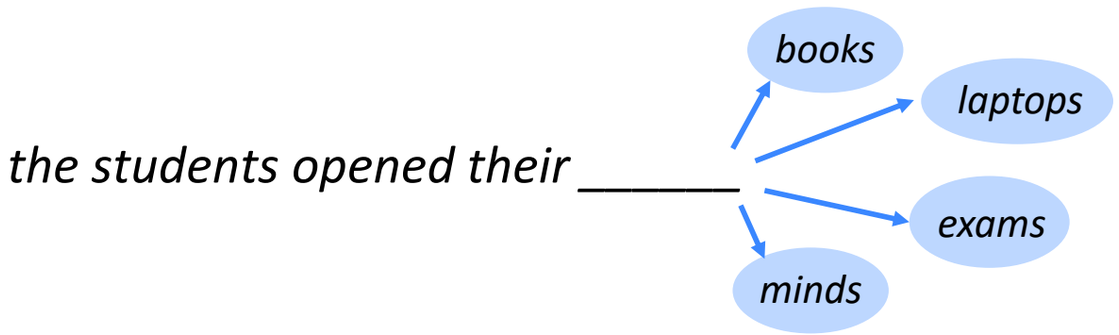
Language Model

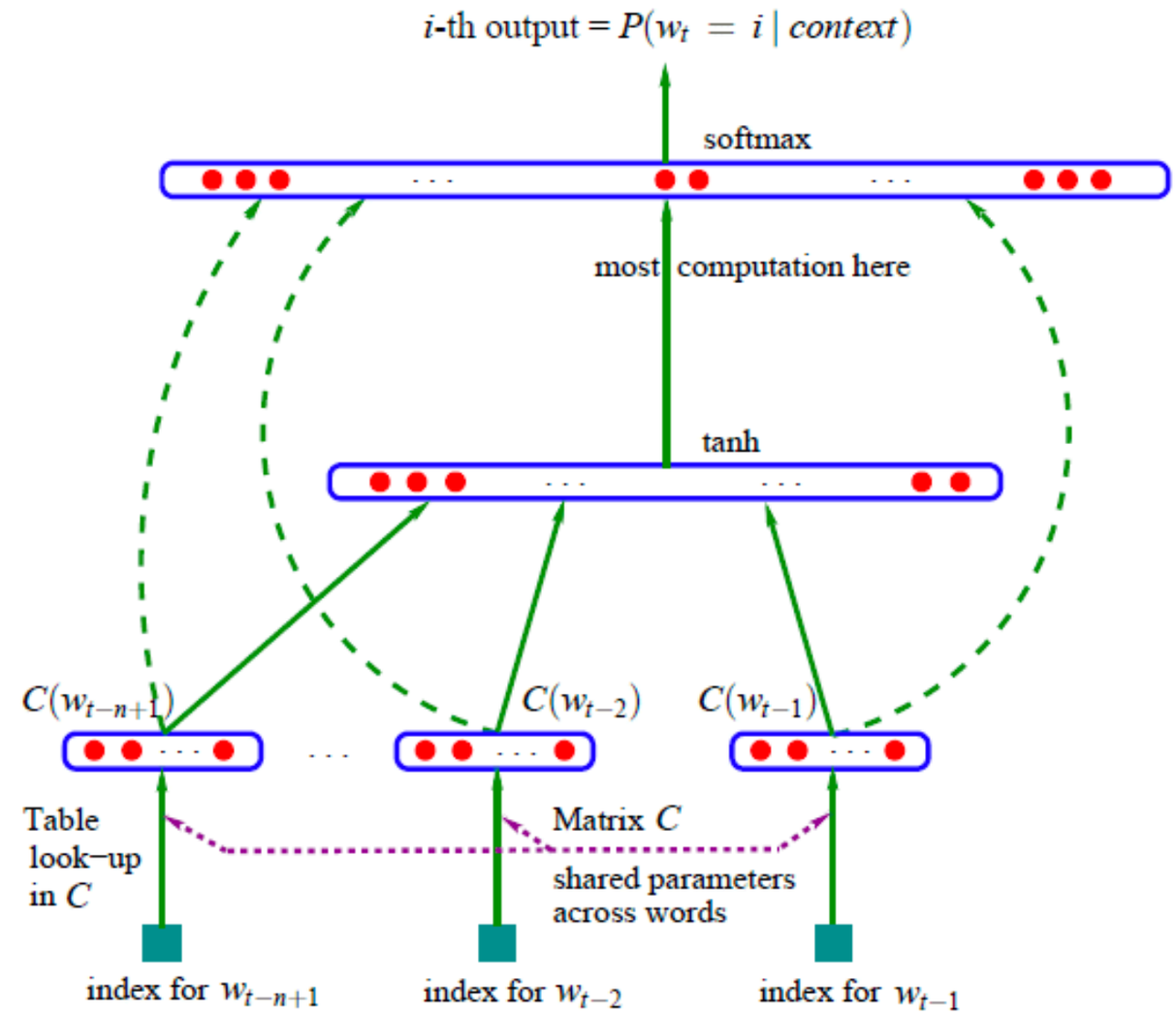


Source:http://jalammar.github.io/

Language Model

A language model takes a list of words (history), and attempts to predict the word that follows them





Source: Bengio, 2003

Word2Vec

NTU NLP

Methods to efficiently create word embeddings

Idea:

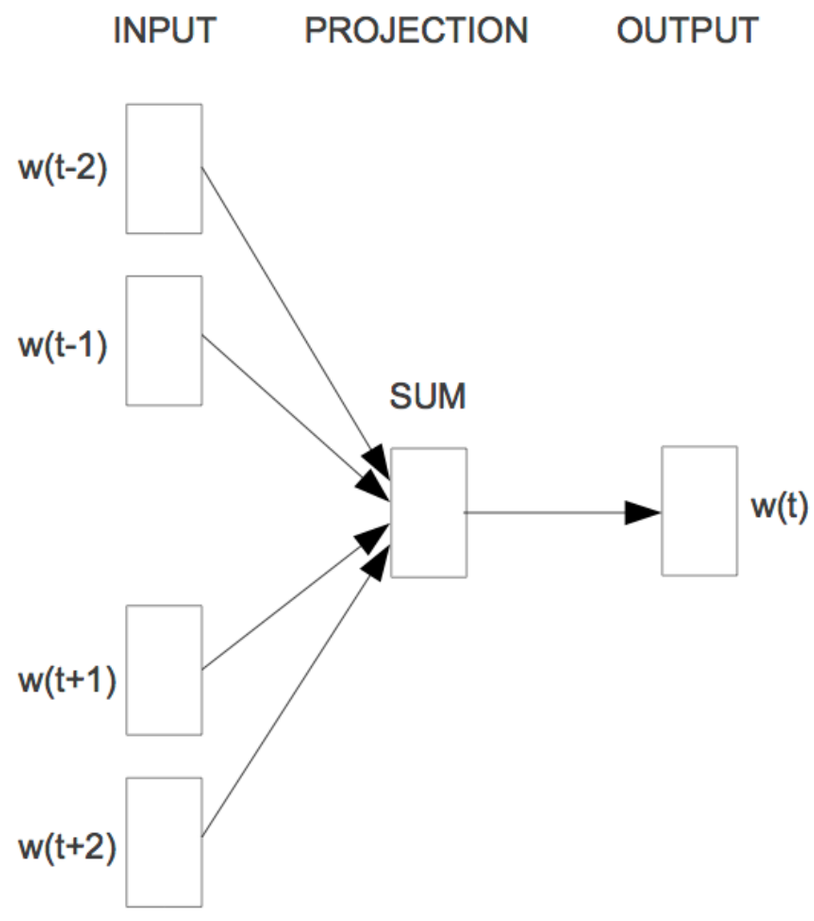
- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Source: Stanford 224N

Word2Vec - CBOW and Skipgram

Methods to efficiently create word embeddings

If our goal is just to learn word embeddings, instead of only looking words before the target word, we can also look at words after it.



This is called a **Continuous Bag of Words (CBOW)** architecture

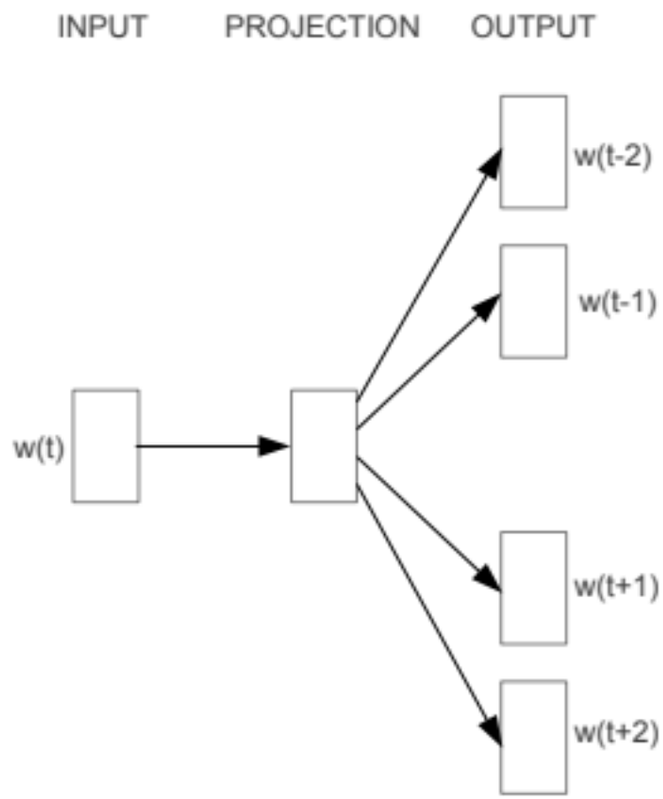
Source: Stanford 224N

Word2Vec - CBOW and Skipgram

NTU NLP

Methods to efficiently create word embeddings

Instead of guessing a word based on its context (the words before and after it), the other architecture tries to guess neighbouring words using the current word.

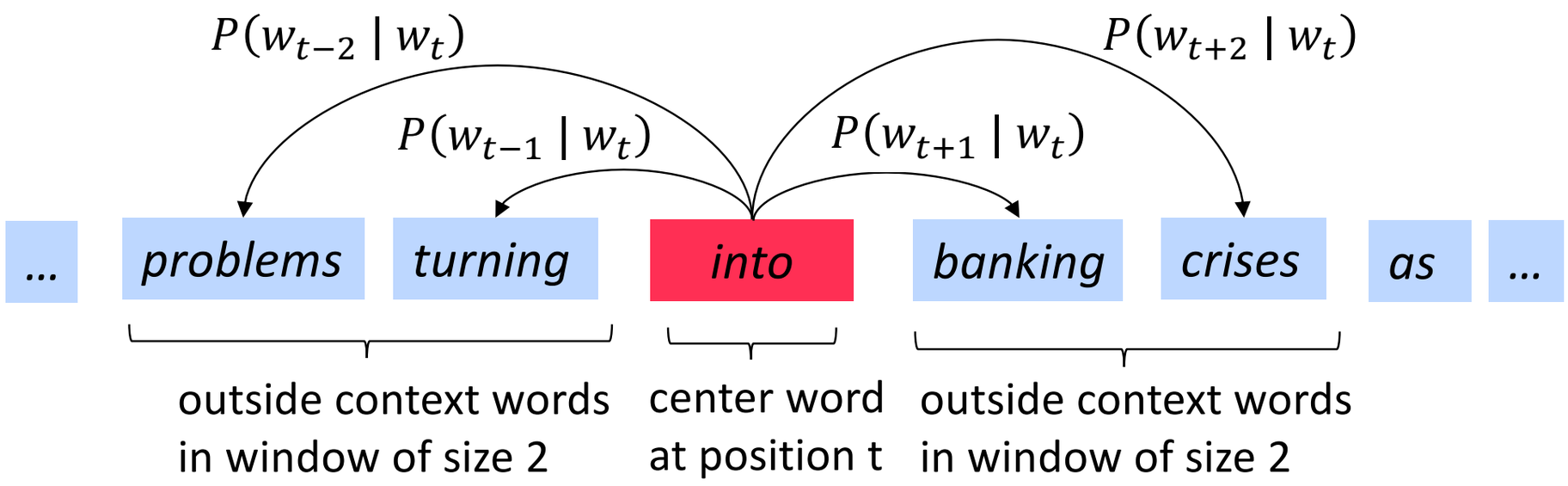


Skip-gram

This is called a Skipgram architecture

Source: Stanford 224N

Word2Vec - skipgram



Source: Stanford 224N

Word2Vec - skipgram

NTU NLP

Input-Output training pairs

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

Source: http://jalammar.github.io/

Word2Vec – prediction

Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

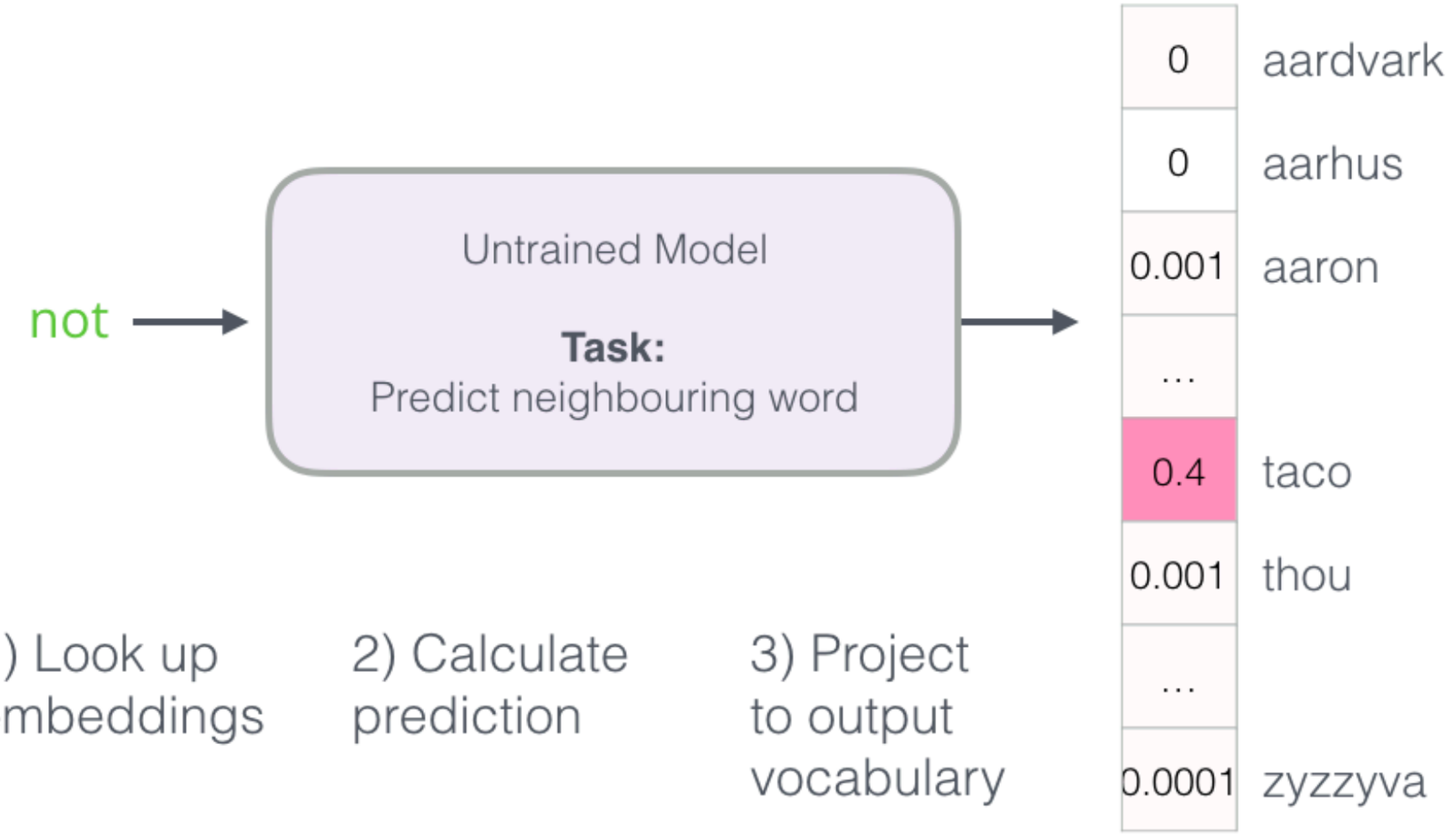
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- "max" because amplifies probability of largest x_i
- "soft" because still assigns some probability to smaller x_i

Source: Stanford 224N

Word2Vec - prediction

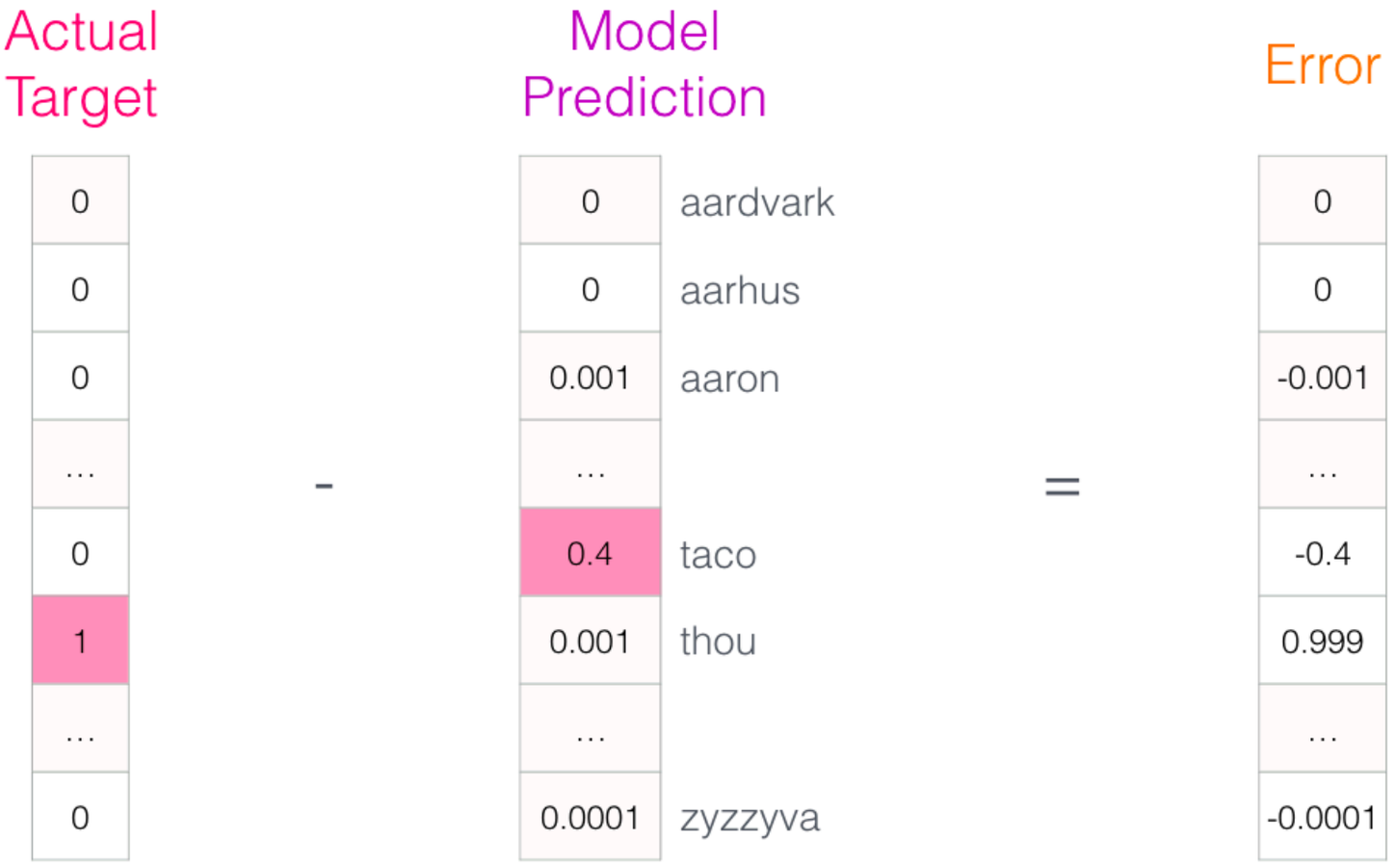
Thou shalt not make a machine in the likeness of a human mind



Word2Vec - loss/Gradient computation

Thou shalt not make a machine in the likeness of a human mind

$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)}$$

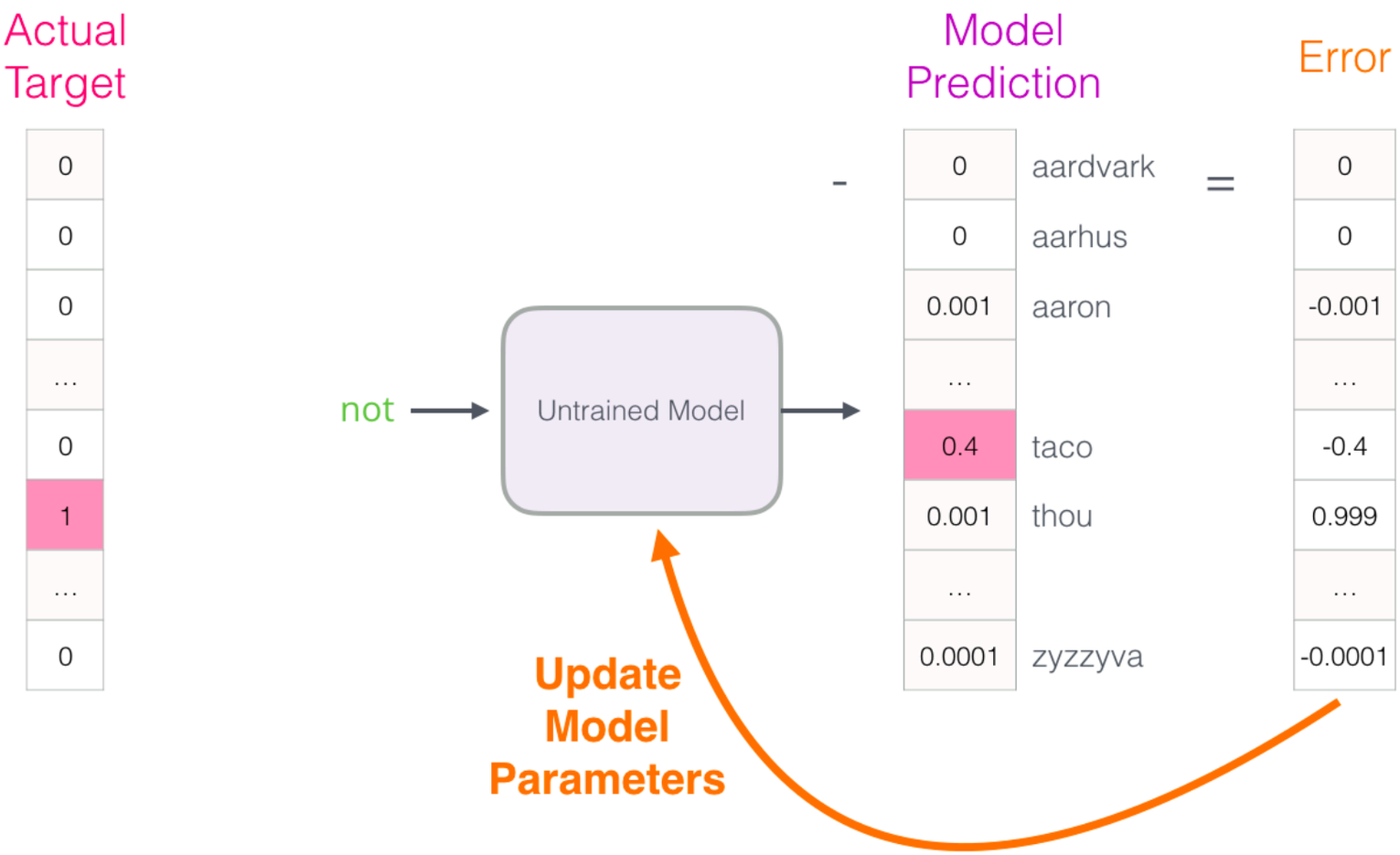


How far off was the model? We subtract the two vectors resulting in an error vector

Source: http://jalammar.github.io/

Word2Vec - training

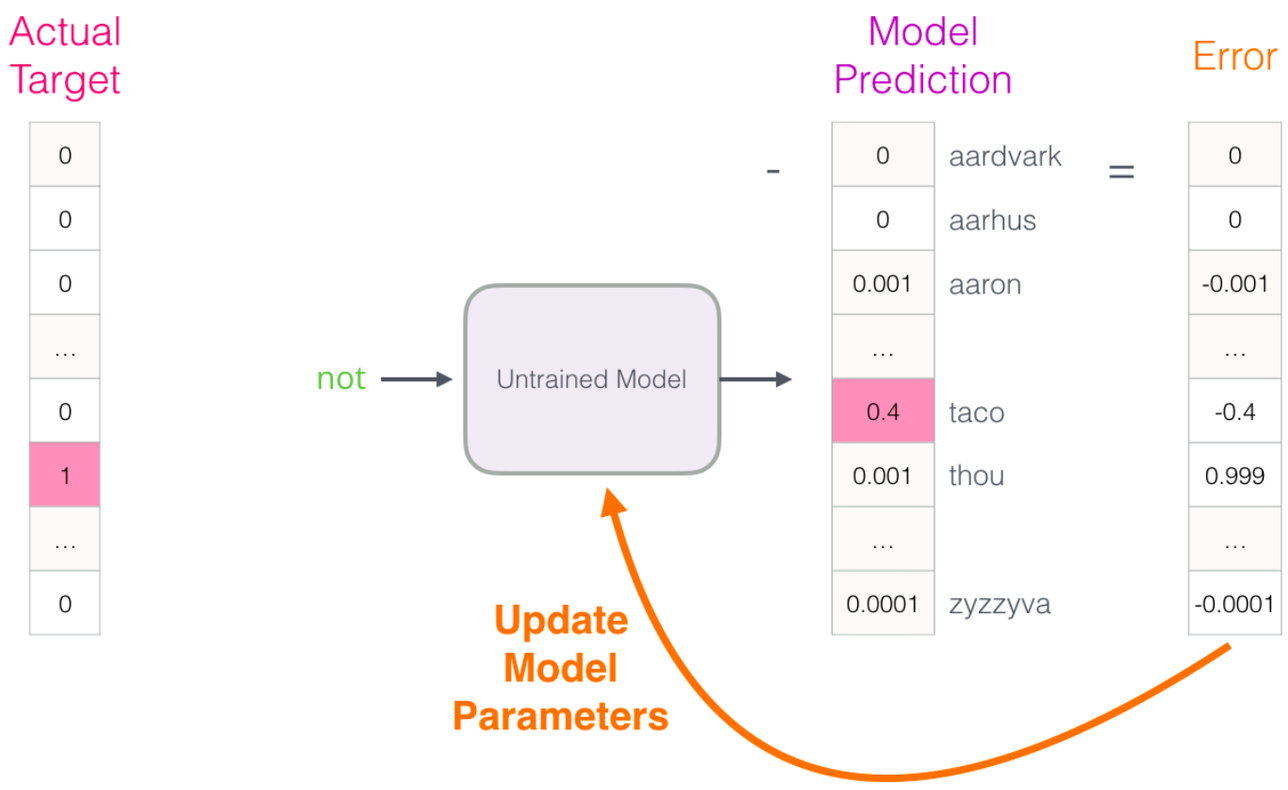
Thou shalt not make a machine in the likeness of a human mind



This error vector can now be used to update the model so the next time, it's a little more likely to guess thou when it gets not as input.

Source: http://jalammar.github.io/

Word2Vec - training



- We proceed to do the same process with the next sample in our dataset, and then the next, until we've covered all the samples in the dataset. That concludes one epoch of training. We do it over again for a number of epochs.
- Then we'd have our trained model and we can extract the embedding matrix from it and use it for any other application.

Source: http://jalammar.github.io/

Word2Vec - training

$$\log p(o|c) = \log \frac{\exp(u_o{}^T v_c)}{\sum_{w=1}^{V} \exp(u_w{}^T v_c)}$$

Actual Target		Model Prediction			Error
0		0	aardvark		0
0		0	aarhus		0
0		0.001	aaron		-0.001
…	-	…		=	…
0		0.4	taco		-0.4
1		0.001	thou		0.999
…		…			…
0		0.0001	zyzzyva		-0.0001

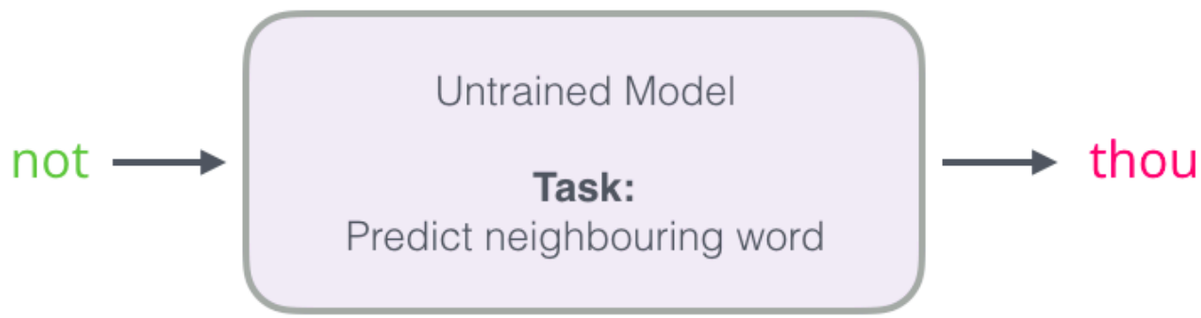
- One Issue: The last step is expensive (because of the denominator). We need to do it once for every training sample in our dataset (easily millions of times)

Source: http://jalammar.github.io/

Negative Sampling

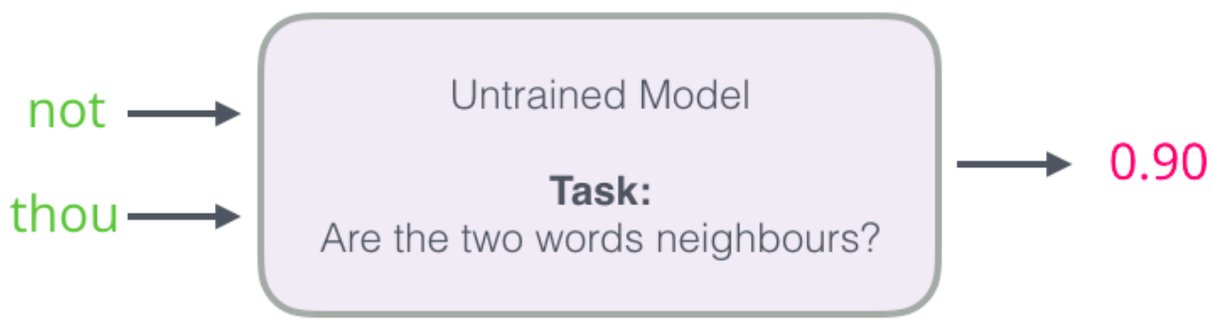
- One Issue: The last step is expensive (because of the denominator). We need to get rid of this!

Change Task from



$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)}$$

To:

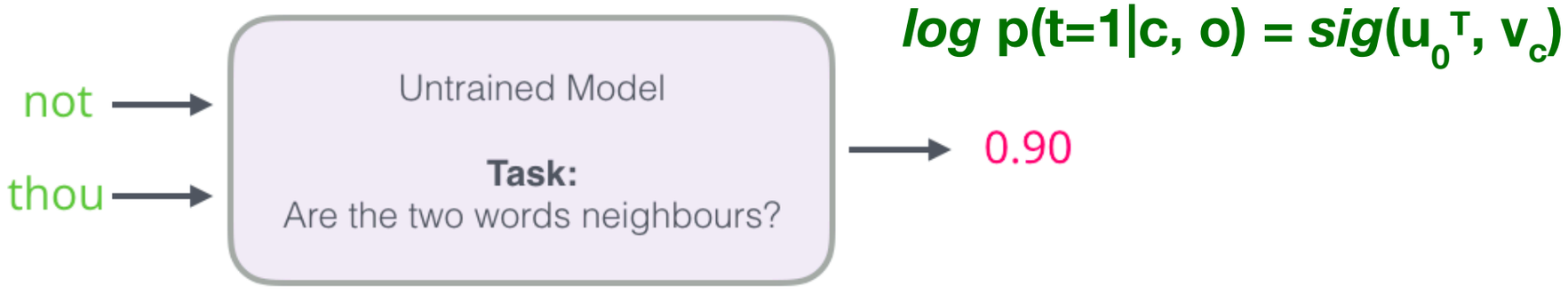


log p (t=1|c, o) = sig (u_0^T, v_c)

Source: http://jalammar.github.io/

Negative Sampling

To:



$$\log p(t=1|c, o) = sig(u_0^T, v_c)$$

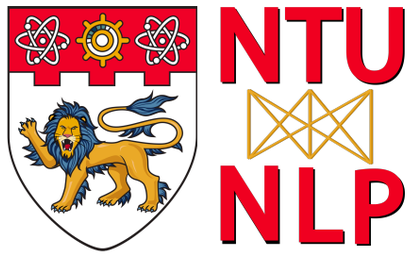
This simple switch changes the model to a logistic regression model (in the output) - thus it becomes much simpler and much faster to calculate.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Source: http://jalammar.github.io/

Negative Sampling



But there's one loophole! All of our examples are positive

input word	output word	target
not	thou	**1**
not		**0**
not		**0**
not	shalt	**1**
not	make	**1**

Negative examples

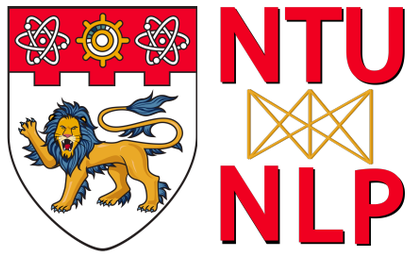
Introduce negative samples to our dataset - samples of words that are not neighbors. Our model needs to return 0 for those samples.

Randomly sample words from our vocabulary

Negative sampling is a version of Noise-contrastive estimation. We are contrasting the actual signal (positive examples of neighboring words) with noise (randomly selected words that are not neighbors)

Source: http://jalammar.github.io/

Skipgram with Negative Sampling



Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

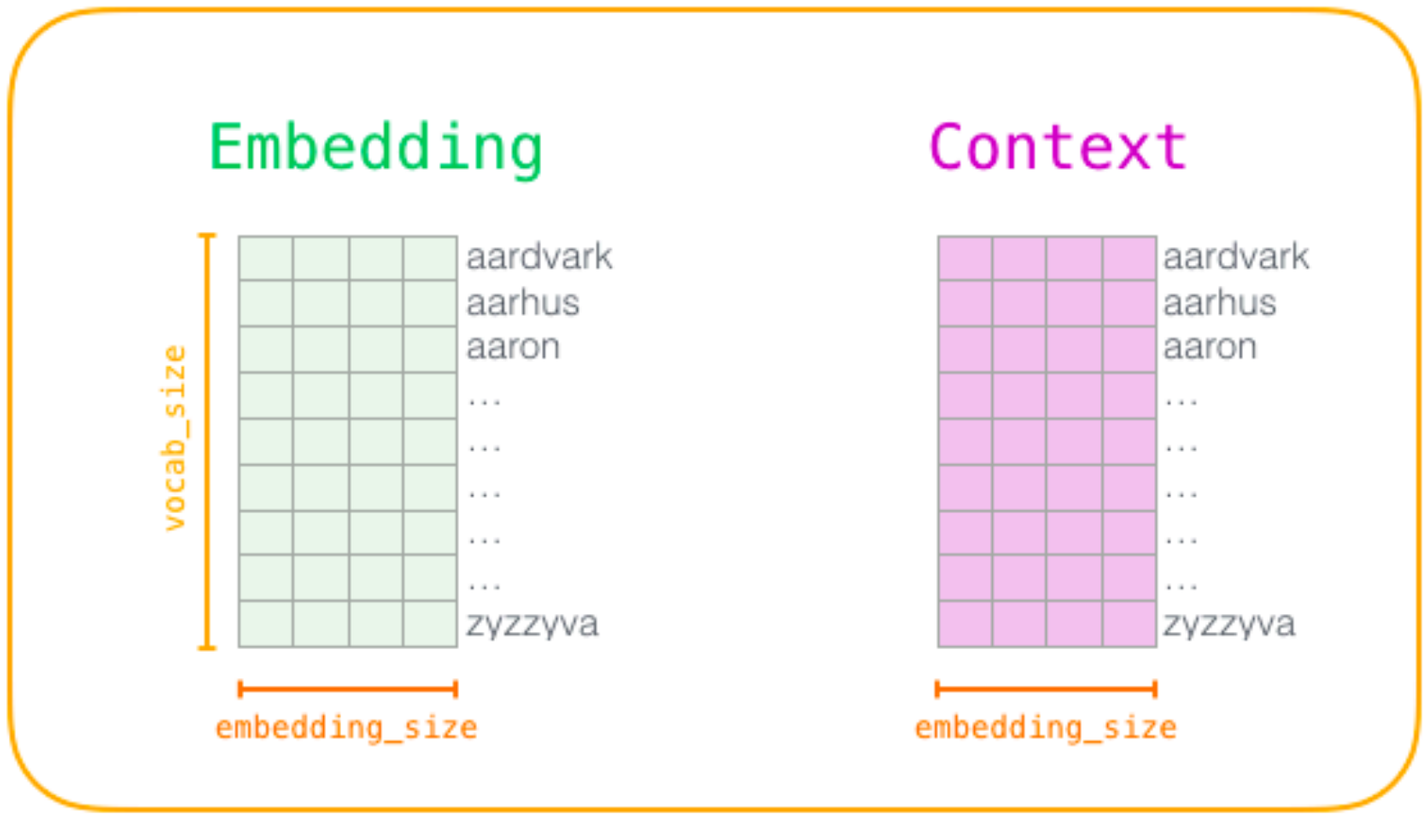
input word	output word	target
make	shalt	**1**
make	aaron	**0**
make	taco	**0**

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

Maximize prob. that real outside word appears (co-occur), minimize prob. that random words appear around center word

Source: http://jalammar.github.io/

Skipgram Training



Create two matrices - an Embedding matrix and a Context matrix.

Initialize these matrices with random values

Source: http://jalammar.github.io/

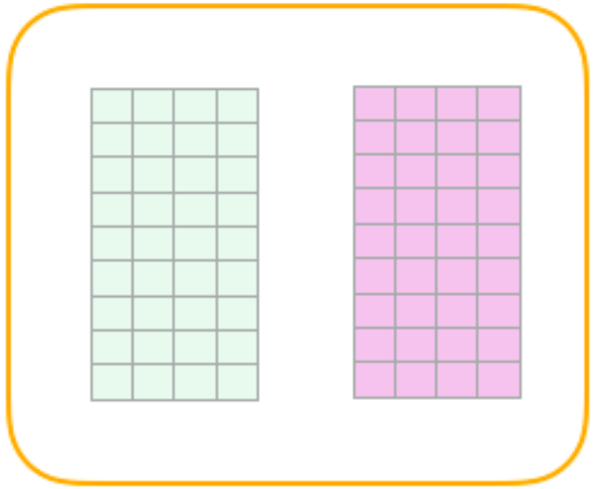
Skipgram Training

NTU NLP

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

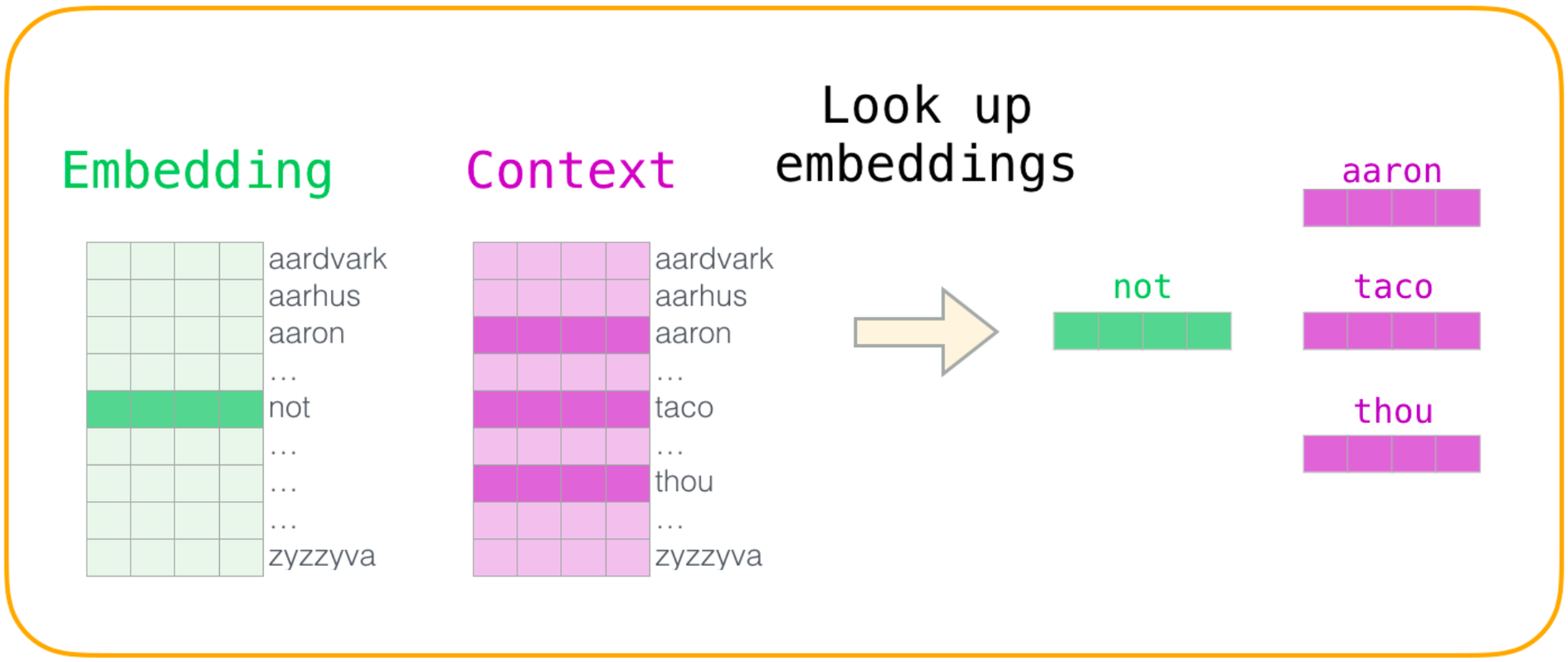
model



In each training step, we take one positive example and its associated negative examples.

Source: http://jalammar.github.io/

Skipgram Training

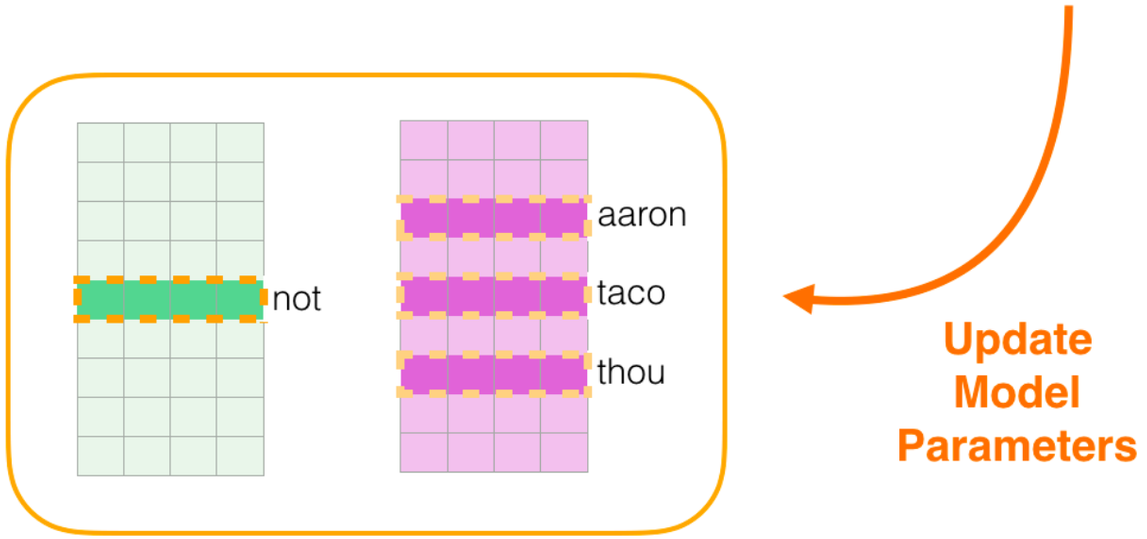


We proceed to look up their embeddings – for the input word, we look in the Embedding matrix. For the context words, we look in the Context matrix (even though both matrices have an embedding for every word in our vocabulary).

Source: http://jalammar.github.io/

Skipgram Training

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



We proceed to compute the predictions, the loss, gradients and model updates

Source: http://jalammar.github.io/

Skipgram Training

Two hyper-parameters:

Window Size and Number of Negative Samples

Window size: 5

Window size: 15

Task dependent
(default is 5)

Negative samples: 2

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

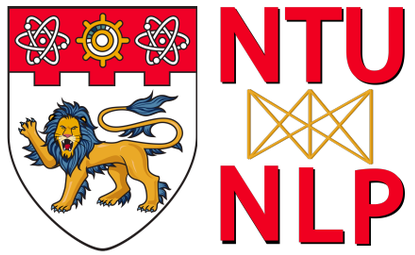
Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

5 - 20
5 is good enough

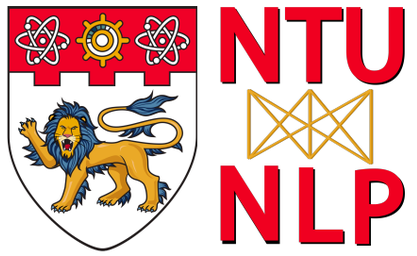
Source: http://jalammar.github.io/

Incorporating Subword Information



- Morphology: Words have internal structures and formation methods; e.g.,"dog", "dogs", "doggy", and "doghouse". Same root, "dog", but use different suffixes to change the meaning of the word.
- This relation can be extended to other words; e.g., "cat" and "cats". The relationship between "boy" and "boyfriend" is just like the relationship between "girl" and "girlfriend".
- Word2vec does not directly use morphology; "dog" and "dogs" are represented by two different vectors.
- FastText [Bojanowski et al., 2017] uses subword embedding in the skip-gram model of word2vec.

FastText



- Each central word is represented as a collection of subwords. E.g., for n = 3, "house" is represented as "<ho", "hou", "ous", "use", "se>", and "<house>".
- For each central word, we record the union of all its subwords with length of 3 to 6 and special subwords (e.g., "<house>"). Thus, the dictionary is the union of the collection of subwords of all words.

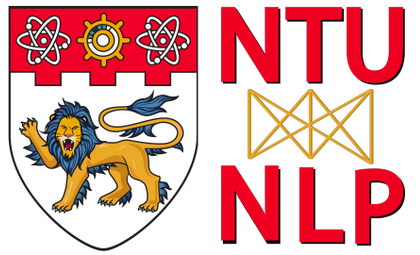
$$\sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top$$

G_w => union of all subwords of w

- Remaining is same as the skipgram model

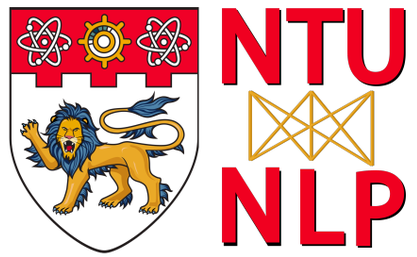
[Bojanowski et al., 2017]

Lecture Plan



- Recap
  - Word meaning & Distributed representations
  - LM for word vector
  - Word2Vec models & Negative sampling
  - Incorporating subword information (FastText)
  - Word vector evaluation

Intrinsic Evaluation of Word Vectors



- Word Vector Analogies:

man:woman :: king: ?

man is to woman as king is to queen

a is to a^* as b is to b^*

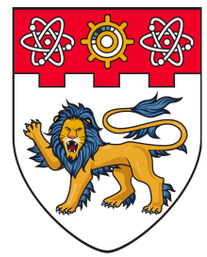
$$a - a^* = b - b^*$$

$$b - a + a^* = b^*$$

king - man + woman = queen

Word vectors should capture "relational similarities"

Intrinsic Evaluation of Word Vectors



NTU NLP

- Word Vector Analogies:

$$\underset{b}{\text{Tokyo}} - \underset{a}{\text{Japan}} + \underset{a^*}{\text{France}} = \underset{b^*}{\text{Paris}}$$

$$\underset{b}{\text{best}} - \underset{a}{\text{good}} + \underset{a^*}{\text{strong}} = \underset{b^*}{\text{strongest}}$$

We wish to find:

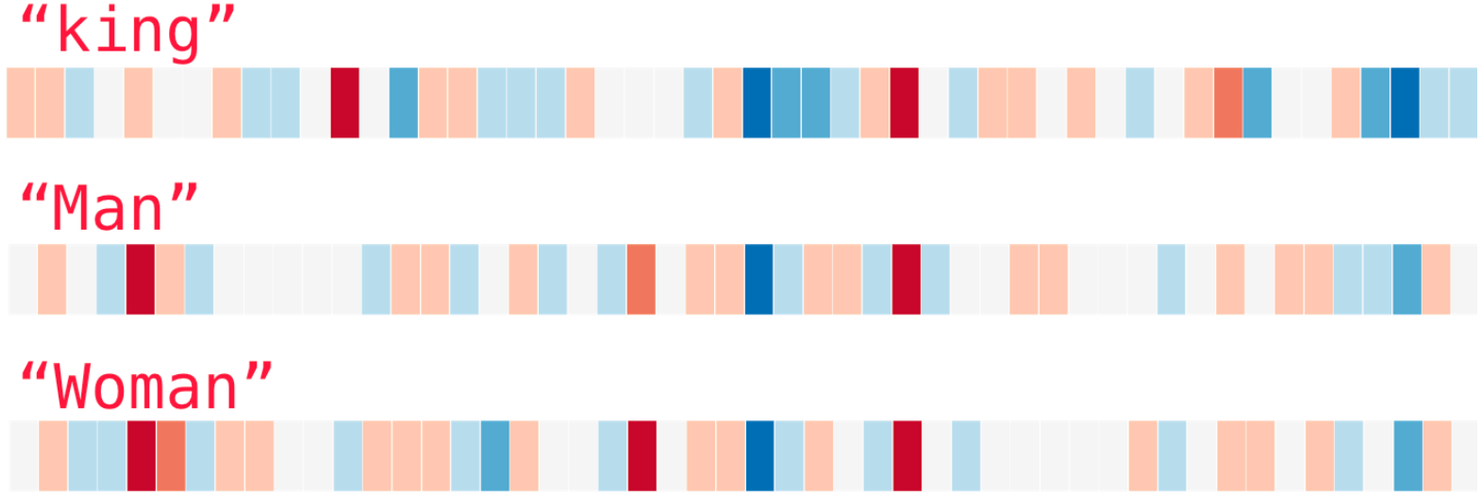
$$\arg\max_{b^*}(\cos(b^*, b - a + a^*))$$

$$=$$

$$\arg\max_{b^*}(\cos(b^*, b) - \cos(b^*, a) + \cos(b^*, a^*))$$

vector arithmetic = similarity arithmetic

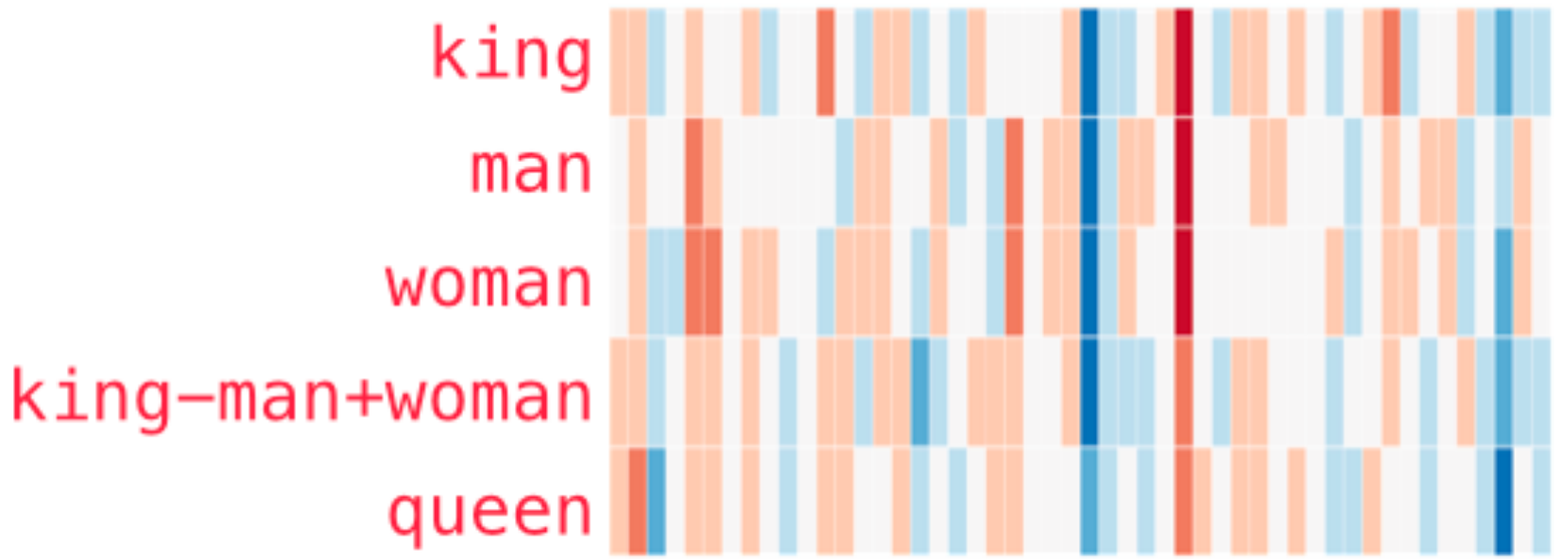
Word2Vec vectors



Source:http://jalammar.github.io/

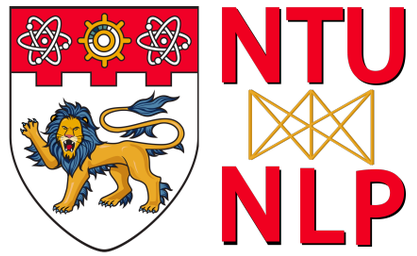
Word2Vec vectors

king − man + woman ~= queen



Source: http://jalammar.github.io/

Intrinsic Evaluation of Word Vectors



- Word Analogy Dataset:

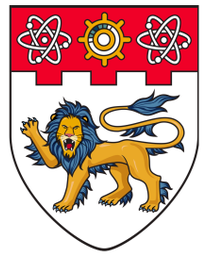
http://code.google.com/p/word2vec/source/browse/trunk/questions- words.txt

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

Source: 224n (Stanford)

Another Intrinsic Evaluation



NTU NLP

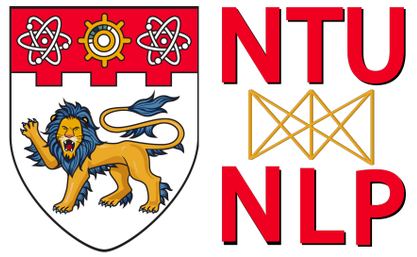
- Word vector distances and their correlation with human judgments

- Example dataset: WordSim353
  http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Source: 224n (Stanford)

Extrinsic Evaluation



- Word vectors are crucial in almost all intermediate and end tasks.

Intermediate tasks:

- Named Entity Recognition (Person, City, Org)
- Parsing (syntactic, semantic, discourse)
- Cross-lingual embeddings
- ….

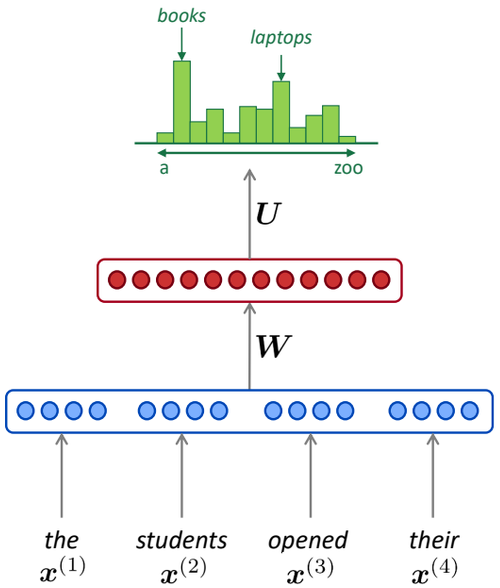
End Applications:

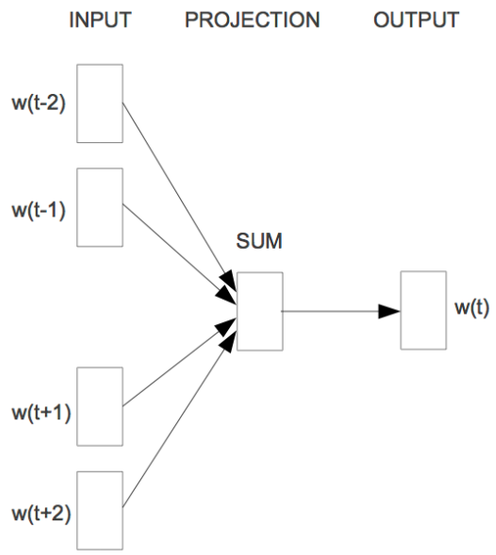
- Machine Translation
- Sentiment Analysis
- Summarization
- Dialogue systems
- ….

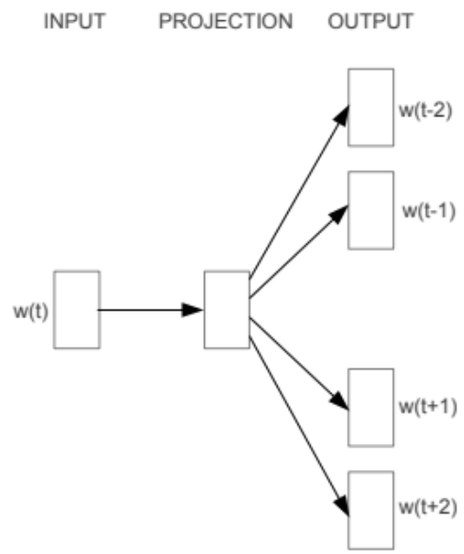
Source: 224n (Stanford)

Summary

Word2Vec







Skip-gram

Negative Sampling

Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Word embedding evaluation

$$a - a^* = b - b^*$$