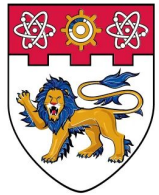


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Natural Language Processing

SC4002 / CE4045 / CZ4045

by Wang Wenya

Email: wangwy@ntu.edu.sg

Contents adapted from Dr. Joty Shafiq's notes

Click here for Lecture 2

Modules we will cover

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Transformer

Attention mechanism, encoder/decoder

Sequence

Sequence modeling, seq2seq learning

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

Outline for today

01 Word meaning & Word vectors

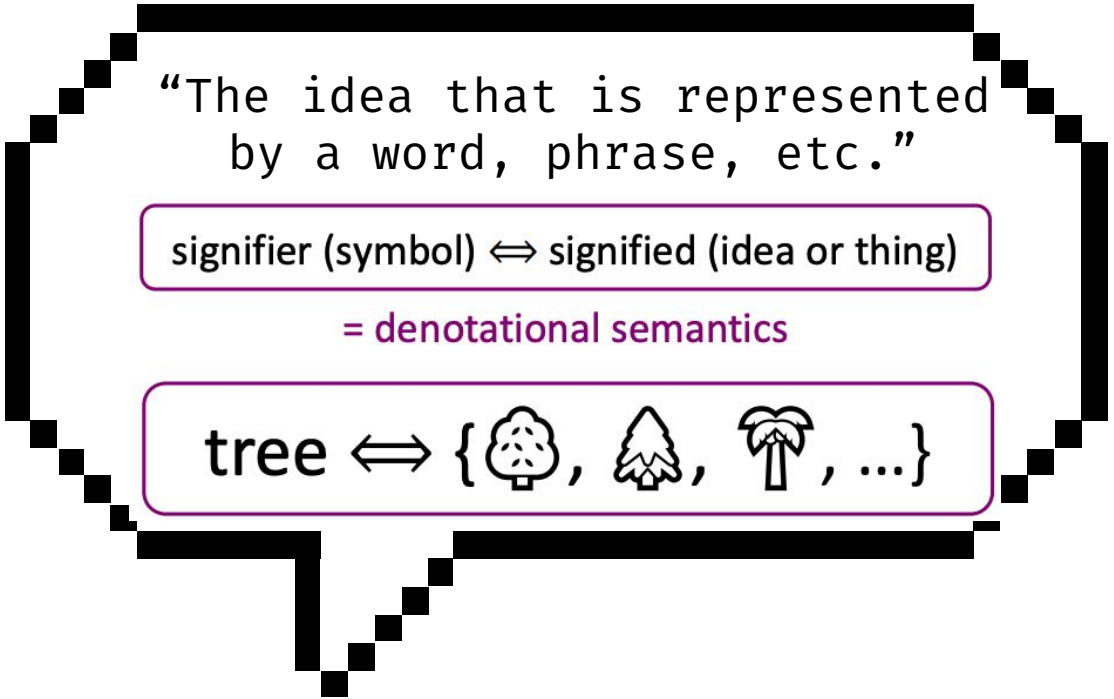
02 Learning word vectors – word2vec

03 Evaluate word vectors

04 Language modeling

01

Word meaning



Word meaning in a computer

Source: stanford 224n

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, requires human labor to create and adapt.
- Subjective

Discrete representation

- In traditional NLP, we regard words as discrete symbols: hotel, motel
- Words can be represented by one-hot vectors

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]

- Vector dimension = number of words in vocabulary (e.g. 500,000)

Problems with discrete representation

- Hard to compute accurate word similarity.

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.

- Use WordNet's list of synonyms to get similarity? Incomplete, not good for polysemous words

Word vectors

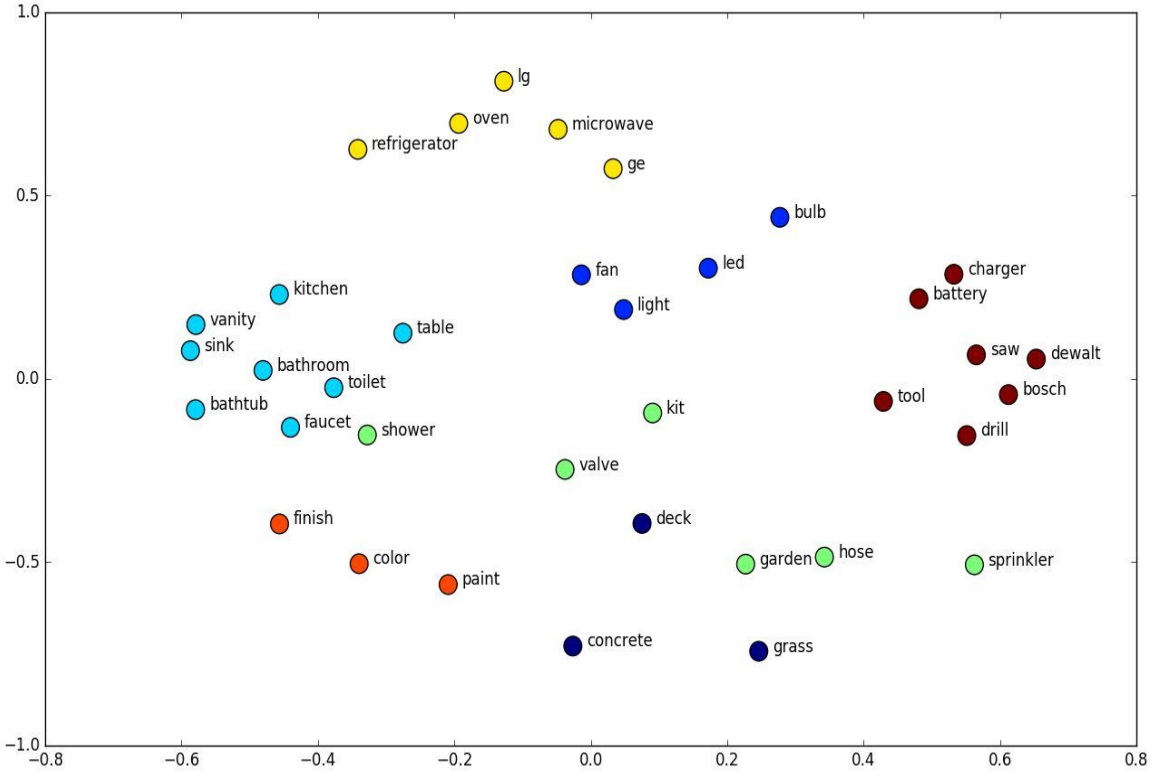
Source: stanford 224n

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.

Word vectors



Outline for today

01 Word meaning & Word vectors

02 Learning word vectors – word2vec

03 Evaluate word vectors

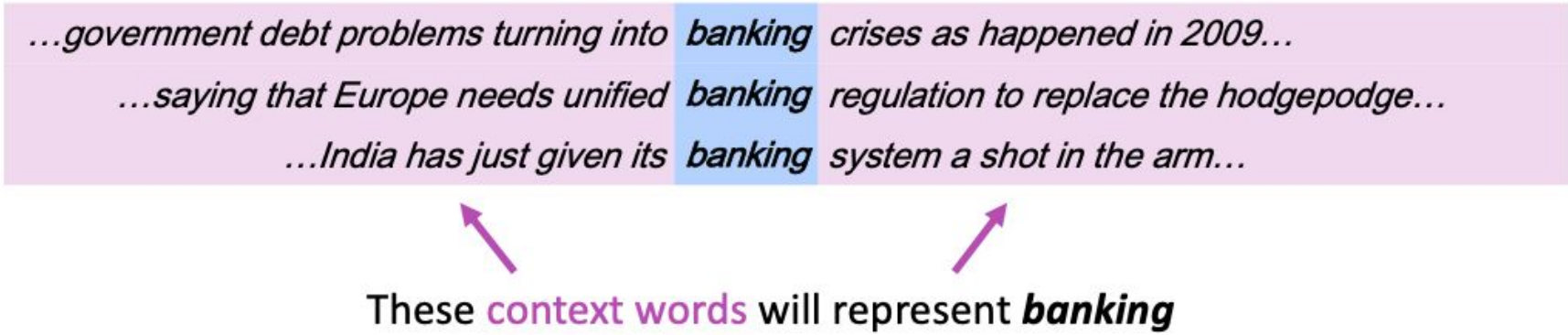
04 Language modeling

Representing words by their context

- Learn to encode similarity in the vectors themselves.
- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by.
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Idea: 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).
- Idea: Use the many contexts of w to build up a representation of w.



Word2vec overview

Source: stanford 224n

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors. The idea includes:

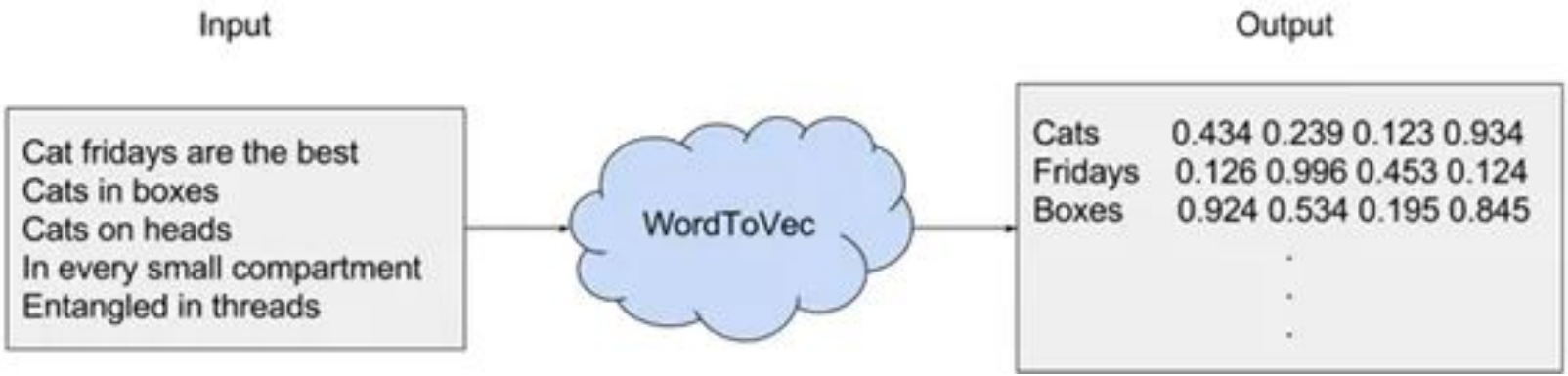
- We have a large corpus of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position **t** in the text, which has a center word **c** and context ("outside") words **o**
- Use the similarity of the word vectors for **c** and **o** to calculate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

Word2vec overview

Source: stanford 224n

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

- Input is a large corpus
- Conduct learning based on some objective
- After training, every word in the dictionary corresponds to a vector

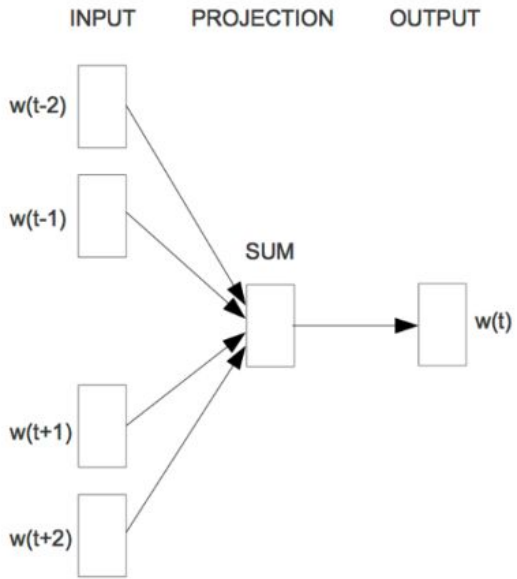


Word2vec – CBOW

Source: stanford 224n

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.



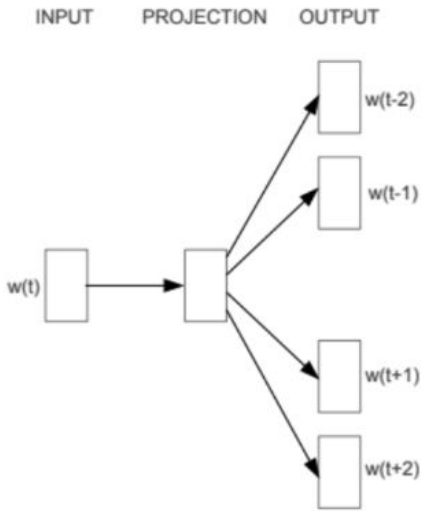
This is called a **Continuous Bag of Words (CBOW)** architecture

Word2vec – Skipgram

Source: stanford 224n

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.



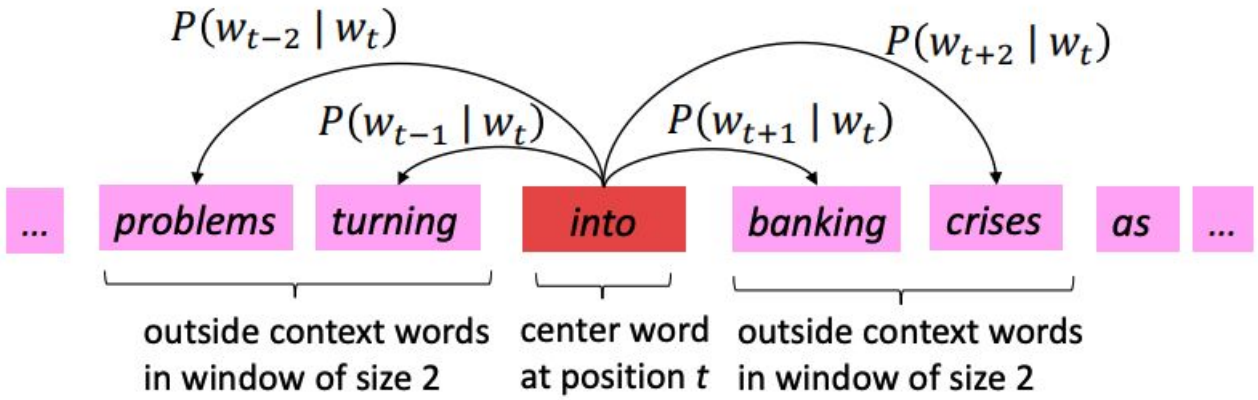
Skip-gram

This is called a Skipgram architecture

Word2vec – Skipgram

Source: stanford 224n

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec – Skipgram

Source: stanford 224n

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec – Skipgram

Source: stanford 224n

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

Word2vec – Skipgram

Source: stanford 224n

② Exponentiation makes anything positive

① 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

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

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ ← Open region

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

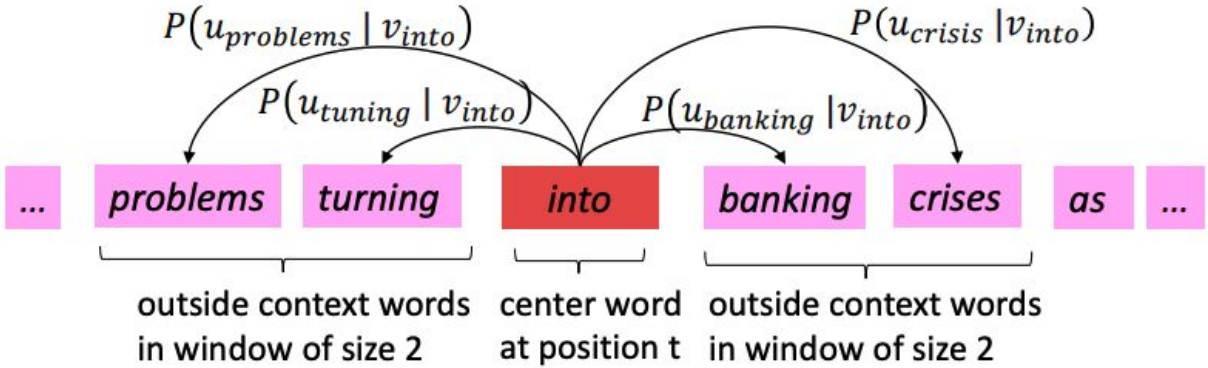
- The softmax function maps arbitrary values x_i to a probability distribution p_i

Word2vec – Skipgram

Source: stanford 224n

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\ ; u_{problems}, v_{into}, \theta)$

All words vectors θ appear in denominator



Word2vec – Skipgram

Source: stanford 224n

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\ ; u_{problems}, v_{into}, \theta)$

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

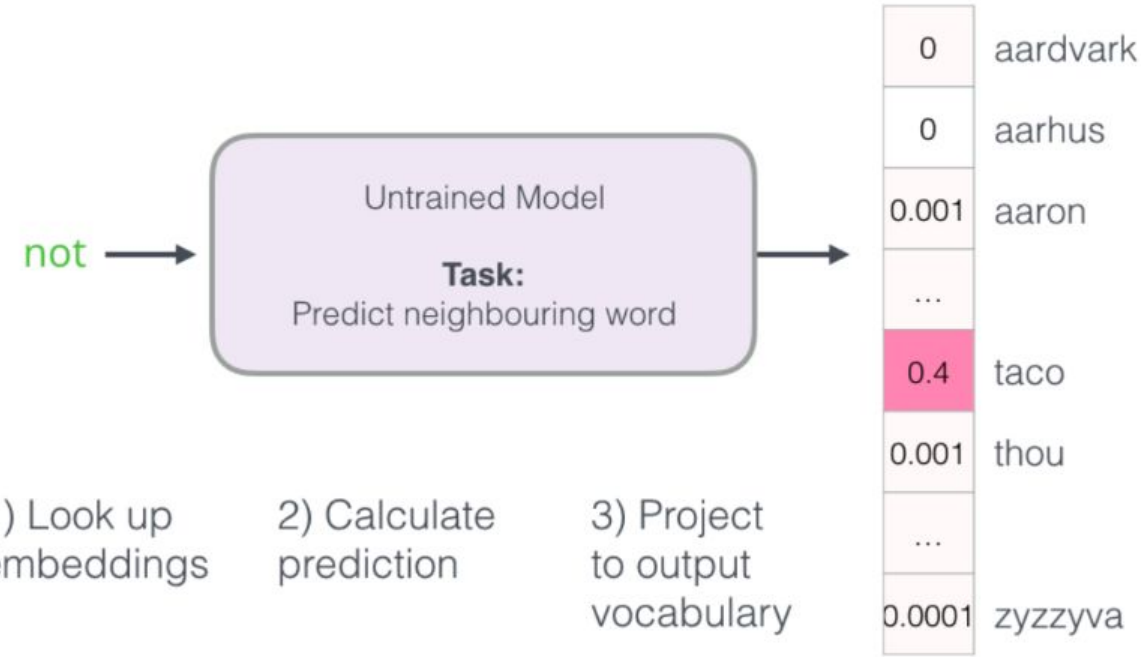
Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

Example

http://jalammar.github.io/illustrated-word2vec/

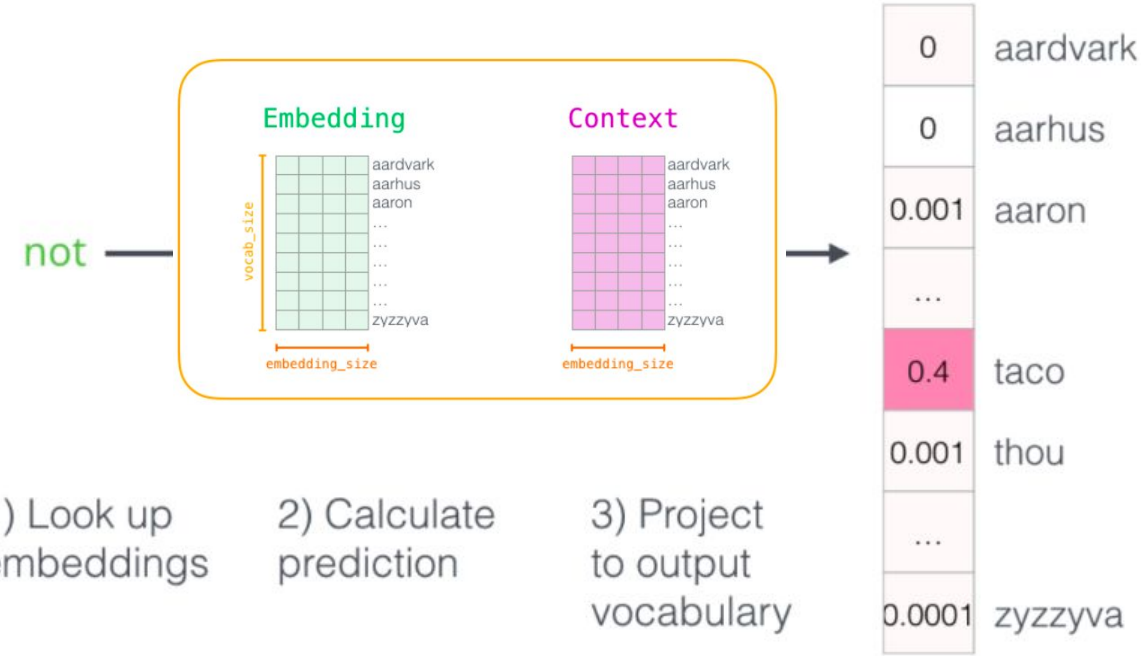
Thou shalt not make a machine in the likeness of a human mind



Example

http://jalammar.github.io/illustrated-word2vec/

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



Example

http://jalammar.github.io/illustrated-word2vec/

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

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

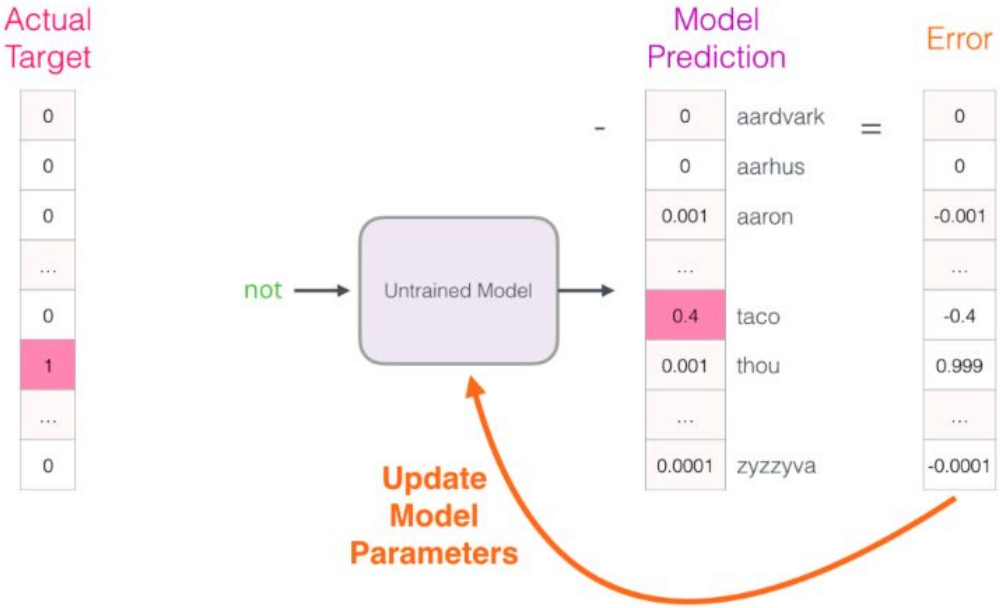
Actual Target		Model Prediction			Error
0		0	aardvark		0
0		0	aarhus		0
0		0.001	aaron		-0.001
...	-	...		=	...
0		0.4	taco		-0.4
1		0.001	thou		0.999
...		...			...
0		0.0001	zyzzyva		-0.0001

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

Example

http://jalammar.github.io/illustrated-word2vec/

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

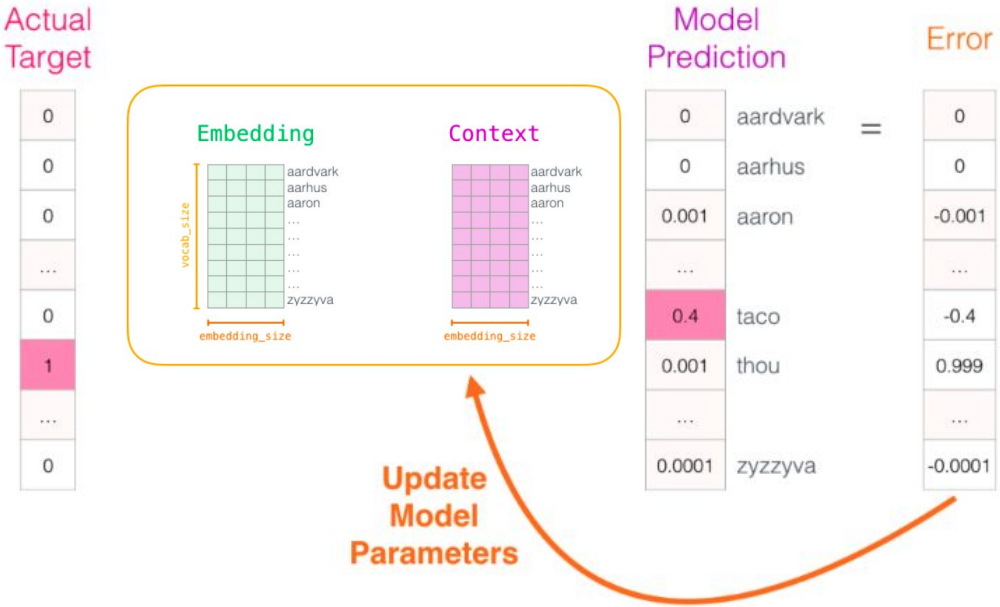


This error vector can now be used to update the model so the next time, it's a little more likely to guess thou when it gets not as input.

Example

http://jalammar.github.io/illustrated-word2vec/

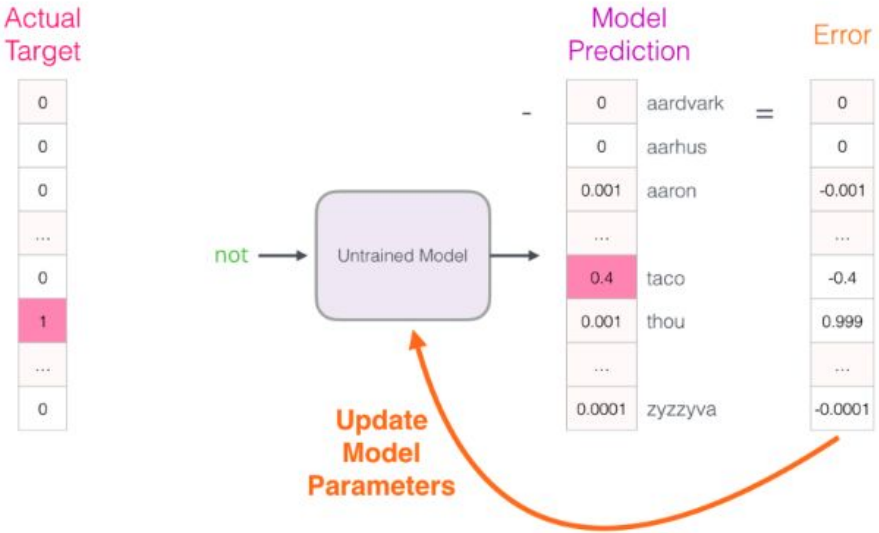
Thou shalt not make a machine in the likeness of a human mind



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.

Example

http://jalammar.github.io/illustrated-word2vec/



- 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.

Skipgram with negative sampling

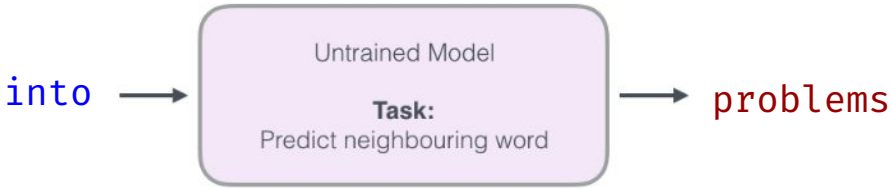
- $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

A big sum over words

Skipgram with negative sampling

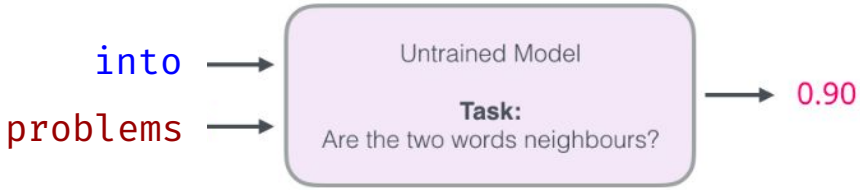
- $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$ ← A big sum over words

Change Task from



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

To:



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

Skipgram with negative sampling

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/

Skipgram with negative sampling

http://jalammar.github.io/illustrated-word2vec/

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

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

Negative examples

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)

Skipgram with negative sampling

Source: stanford 224n

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

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \log \sigma(-u_{o_i}^T v_c)$$

$\{o_1, \ldots, o_k\}$ are random words

Skipgram with negative sampling

Source: stanford 224n

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

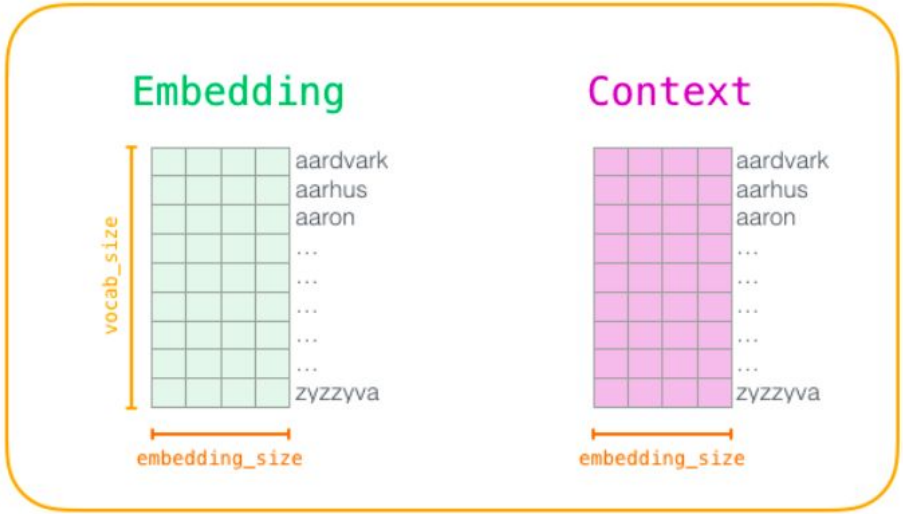
$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \log \sigma(-u_{o_i}^T v_c)$$

$\{o_1, \ldots, o_k\}$ are random words

$$\begin{aligned} J_t(\theta) &= \log[p(t=1|c,o) \prod_{i=1}^{k} p(t=0|c,o_i)] \\ &= \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \log(1 - \sigma(u_{o_i}^T v_c)) \\ &= \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \log \sigma(-u_{o_i}^T v_c) \end{aligned}$$

Skipgram Training

http://jalammar.github.io/illustrated-word2vec/



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

Initialize these matrices with random values

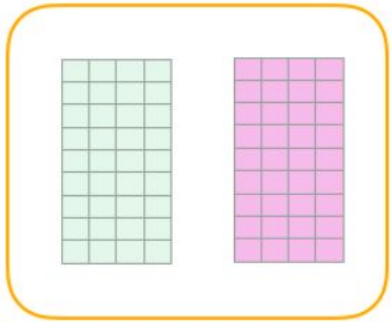
Skipgram Training

http://jalammar.github.io/illustrated-word2vec/

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
...	...	...

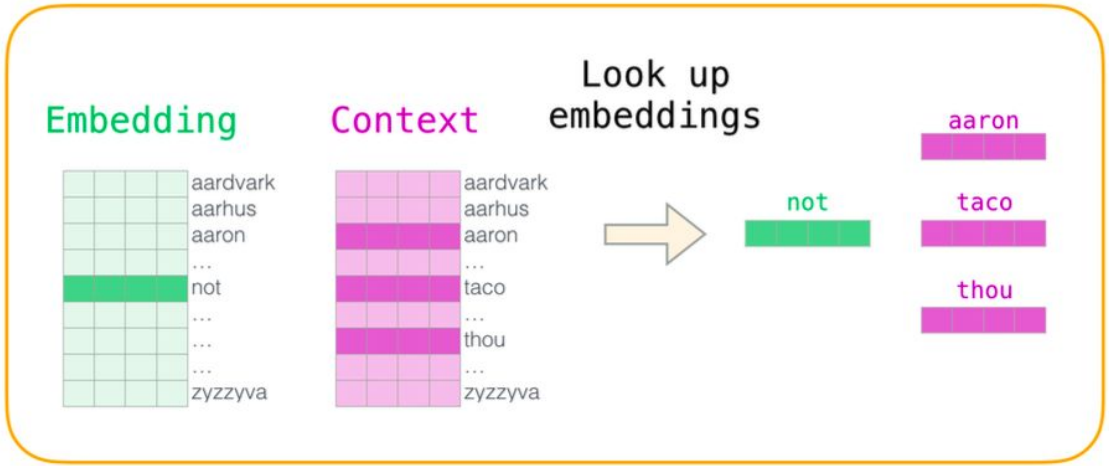
model



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

Skipgram Training

http://jalammar.github.io/illustrated-word2vec/

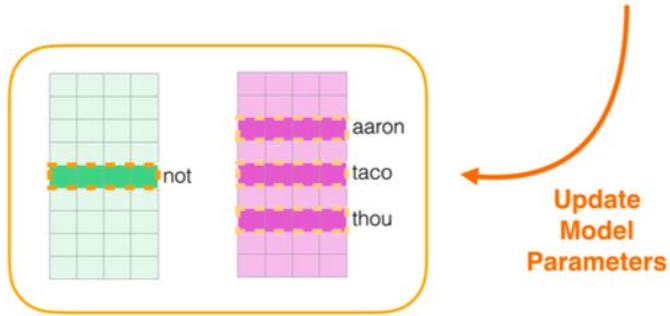


We proceed to look up their embeddings – for the input word, we look in the Embedding matrix. For the context words, we look in the Context matrix (even though both matrices have an embedding for every word in our vocabulary).

Skipgram Training

http://jalammar.github.io/illustrated-word2vec/

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

Which embedding to use for downstream tasks???

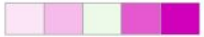
Skipgram Training

http://jalammar.github.io/illustrated-word2vec/

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

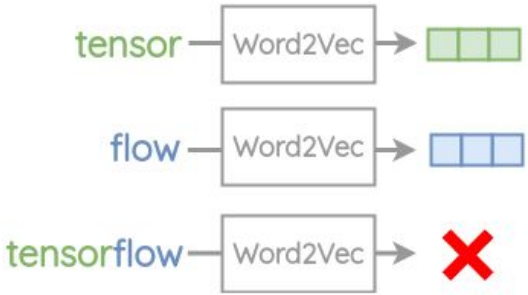
Negative samples: 5

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

Problem with Word2Vec and Glove

- Out-Of-Vocabulary (OOV) words: In word2vec, an embedding is created for each word. As such, it can't handle any words it has not encountered during its training.
- Morphology: For words with same radicals such as "eat" and "eaten", Word2Vec doesn't do any parameter sharing. Each word is learned uniquely based on the context it appears in.



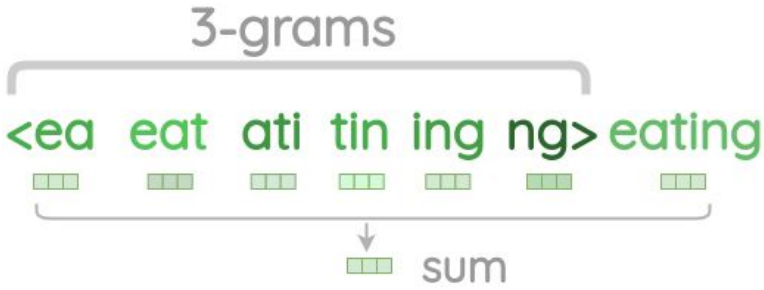
Shared radical

eat eats eaten eater eating

FastText

- Subword generation: For a word, we generate character n-grams of length 3 to 6 present in it.

- Compose embeddings



Outline for today

01 Word meaning & Word vectors

02 Learning word vectors – word2vec

03 Evaluate word vectors

04 Language modeling

Intrinsic Evaluation of Word Vectors

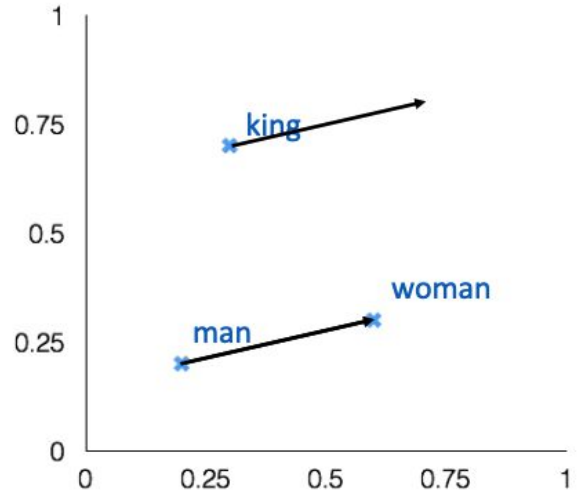
man is to woman as king is to queen

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

- 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:

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

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

Intrinsic Evaluation of Word Vectors

Source: stanford 224n

- 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

Word Vectors are Cool

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Extrinsic Evaluation

Source: stanford 224n

- 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

Supervised Tasks in NLP

In a supervised setup, we have a training dataset

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

- x_i are inputs, e.g. words, sentences, documents, etc.
- y_i are labels (one of C classes) we try to predict, e.g.,
  - Other words: Language model
  - Classes: POS tag, sentiment, named entities
  - Multi-word sequences: Translation, Summarisation

Named Entity Recognition (NER)

- Task: find and classify names in text, by labeling word tokens, for example

Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

- Possible application: tracking mentions of particular entities in documents, build knowledge base

Named Entity Recognition (NER)

- Idea: classify each word in its context window of neighboring words
- Example: decide if "Paris" belongs to **location**

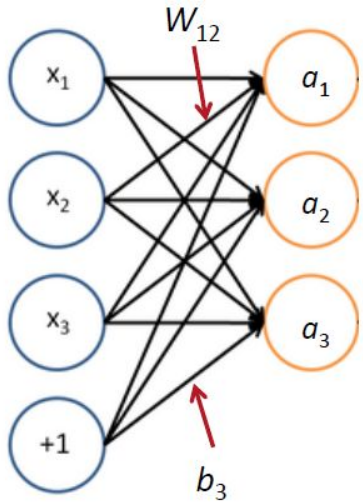
the museums in Paris are amazing to see .

$X_{window} = [\ x_{museums}\quad x_{in}\quad x_{Paris}\quad x_{are}\quad x_{amazing}\ ]^T$

We can formalize it as a supervised binary/multiclass prediction problem, and use logistic regression/softmax.

Recall: Neural Networks

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$
.....



In Matrix Notation $z = Wx + b$

$a = f(z)$

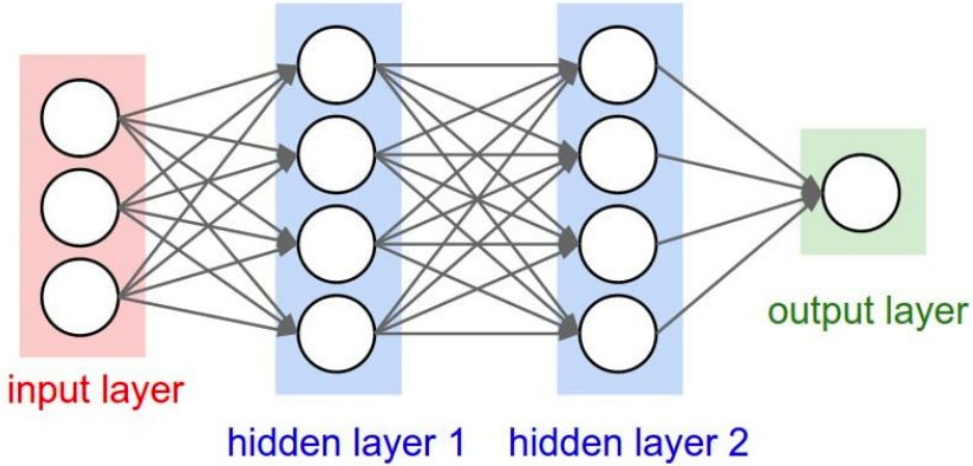
Where f() is applied element-wise

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

Activation function

Recall: Neural Networks

... which we can feed into another logistic regression function



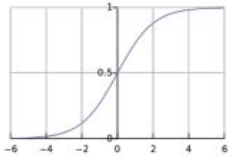
Output layer can be a classification layer or a regression layer

Named Entity Recognition (NER)

Source: stanford 224n

$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

predicted model probability of class



Binary prediction

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

$$\boldsymbol{x} \quad \text{(input)}$$

$x = [\ x_{museums}\ \ x_{in}\ \ x_{Paris}\ \ x_{are}\ \ x_{amazing}\ ]$

Named Entity Recognition (NER)

Source: stanford 224n

~~$J_t(\theta) = \sigma(s)$~~ $J_t(\theta) = softmax(s_c) = \frac{\exp(s_c)}{\sum_{c' \in C} \exp(s_{c'})}$

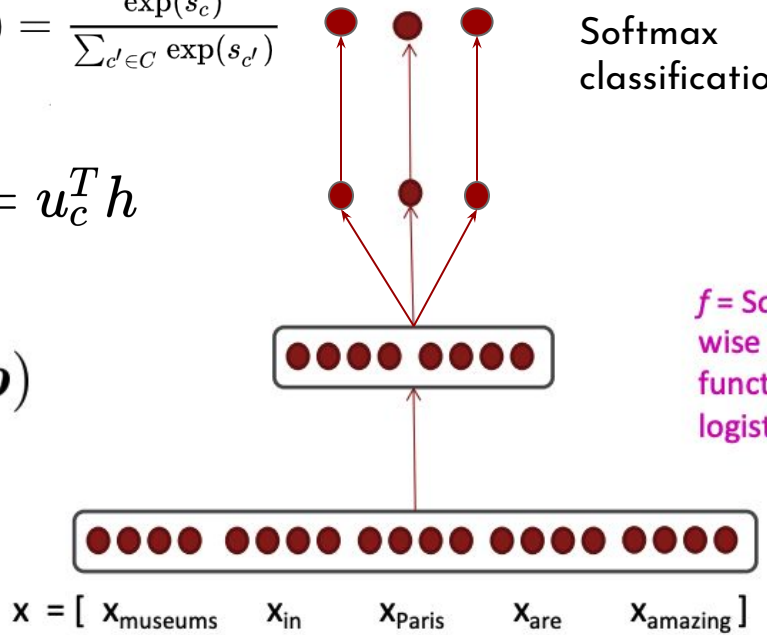
Softmax classification

predicted model probability of class

~~$s = u^T h$~~ $s_c = u_c^T h$

$h = f(Wx + b)$

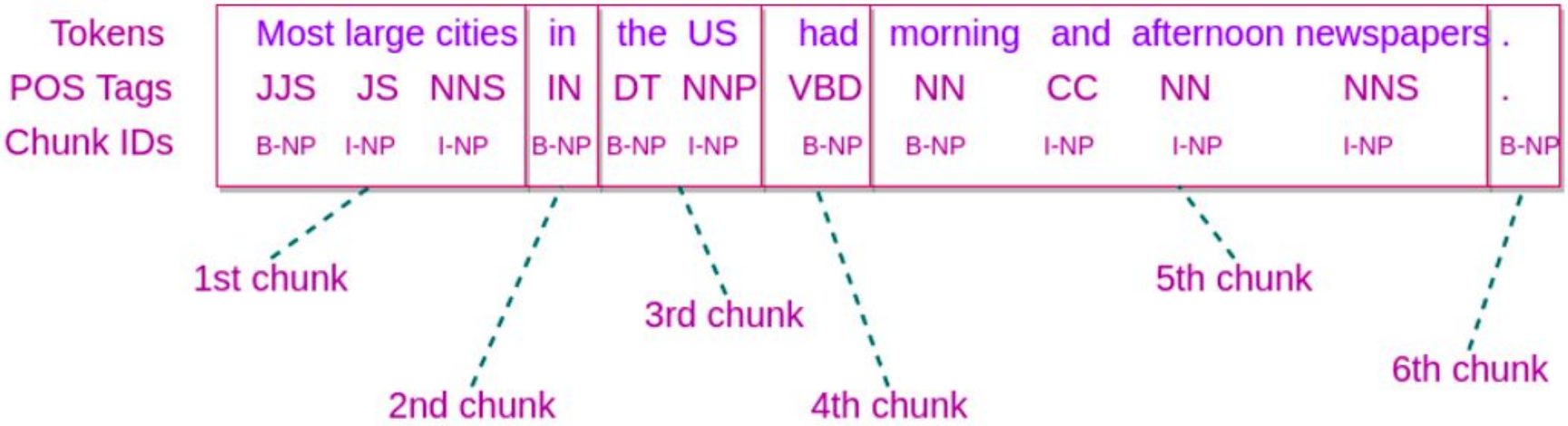
x (input)



f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

Example of NLP Tasks

Chunking



Example of NLP Tasks

Part-of-Speech (POS) Tagging

The_DT first_JJ time_NN he_PRP was_VBD shot_VBN in_IN the_DT hand_NN as_IN he_PRP chased_VBD the_DT robbers_NNS outside_RB ...

first	time	shot	in	hand	as	chased	outside
JJ	NN	NN	IN	NN	IN	JJ	IN
RB	VB	VBD	RB	VB	RB	VBD	JJ
		VBN	RP			VBN	NN
							RB

Outline for today

01 Word meaning & Word vectors

02 Learning word vectors – word2vec

03 Evaluate word vectors

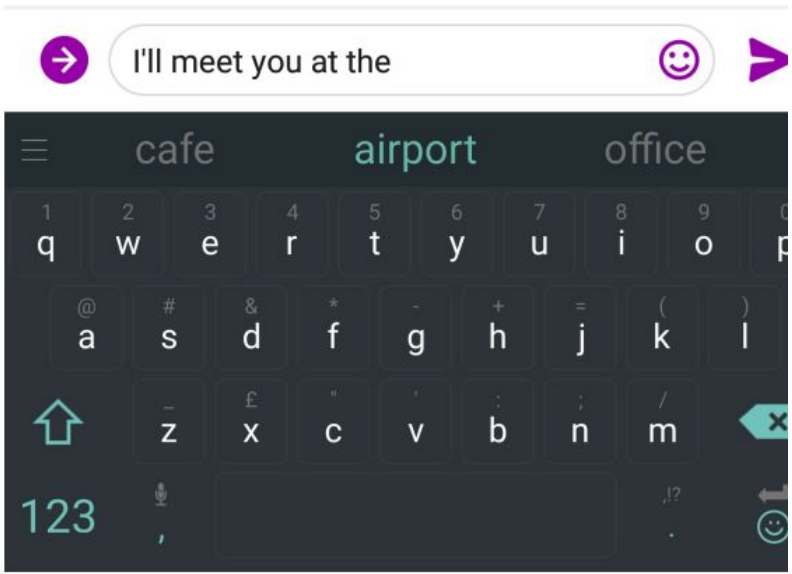
04 Language modeling

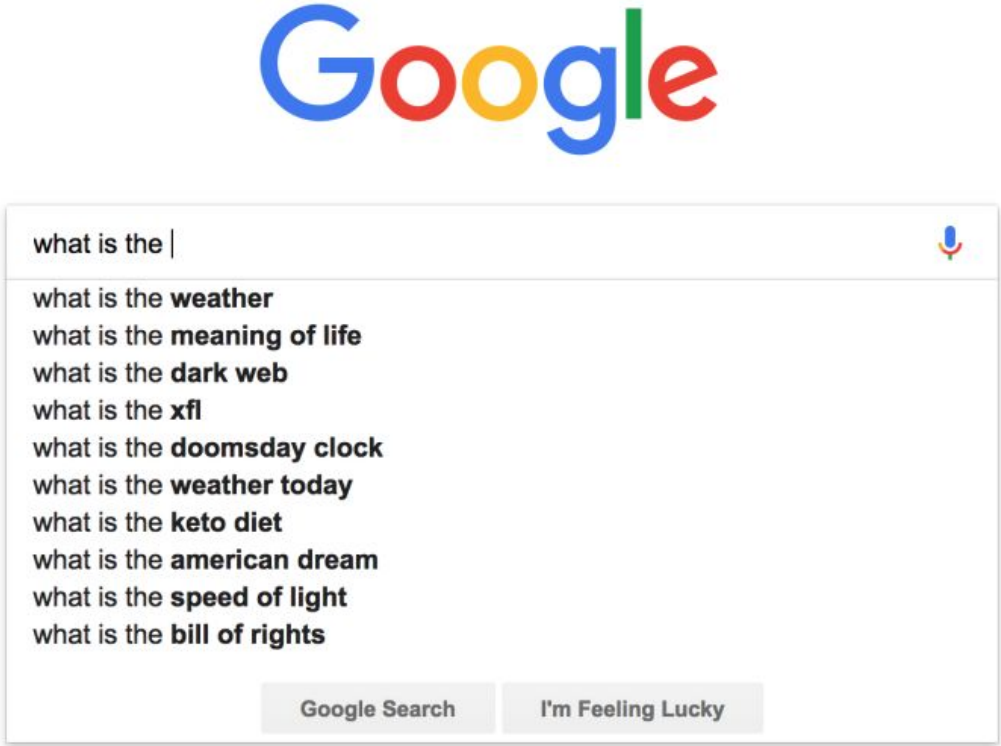
Language Modeling

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

Very important concept to understand. We will use this very often later when talking about generative models, e.g., ChatGPT.

Language Modeling





Language Modeling

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

More formally: given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Language Modeling

- You can also think of a Language Model as a system that **assigns a probability to a piece of text**

For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Importance of Language Modeling

- A benchmark task to track our progress on understanding language
- An important component of many NLP tasks, especially those involving generating text or estimating the probability of a text

  - Speech recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Evaluating Language Models

Source: stanford 224n

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Cross-entropy loss

Lower perplexity is better!

N-Gram Language Model

- Definition: An n-gram is a chunk of n consecutive words.
  - **unigrams**: “the”, “students”, “opened”, ”their”
  - **bigrams**: “the students”, “students opened”, “opened their”
  - **trigrams**: “the students opened”, “students opened their”
  - **four-grams**: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-Gram Language Model

Source: stanford 224n

First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

N-Gram Language Model

Source: stanford 224n

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Sparsity Problem with N-Grams

Source: stanford 224n

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then *w* has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any w*!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

Storage Problem with N-Grams

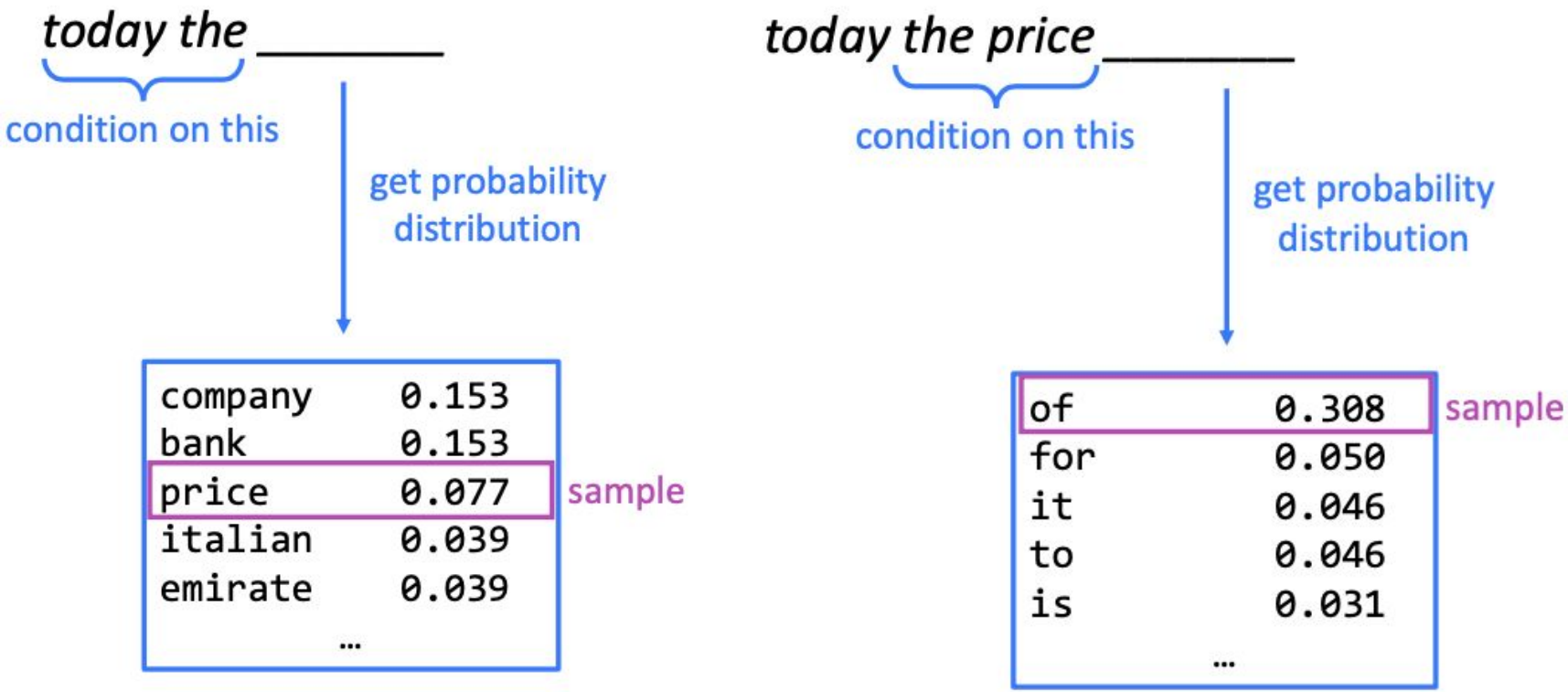
Source: stanford 224n

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

Increasing *n* or increasing corpus increases model size!

Generating Text with N-Gram LMs



Generating Text with N-Gram LMs

Source: stanford 224n

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

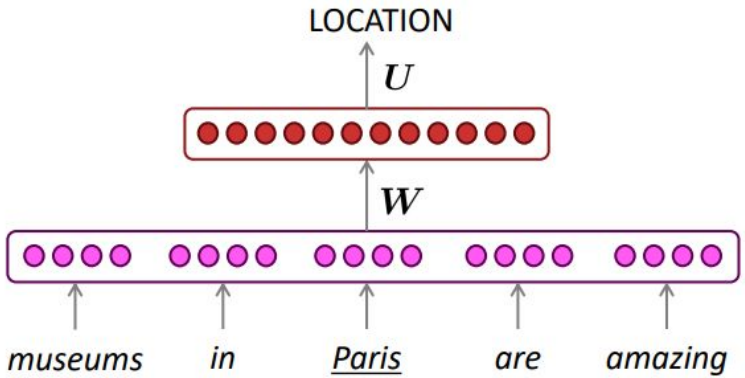
But increasing *n* worsens sparsity problem, and increases model size...

Neural Language Model

- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob. distribution of the next word $P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)})$
- A fixed-window neural language model
  - Y. Bengio, et al. (2003): A Neural Probabilistic Language Model

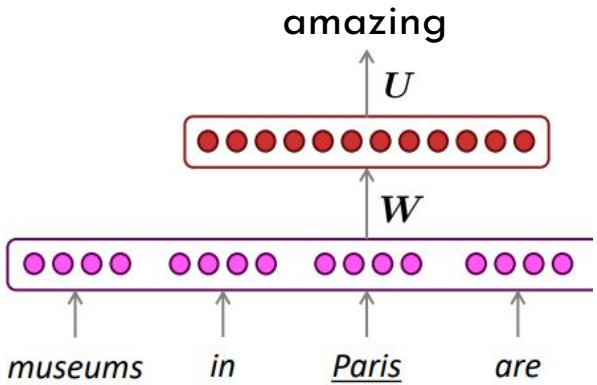
Neural Language Model

- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob. distribution of the next word $P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)})$
- Recall a fixed-window NER model: predict the class for "Paris"



Neural Language Model

- Recall the Language Modeling task:
  - Input: sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  - Output: prob. distribution of the next word $P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)})$
- Instead of predicting the class for the center word, **predict the next word**



Neural Language Model

Source: stanford 224n

$$\hat{y}_i = \frac{\exp(u_i h + b_i)}{\sum_j^{|V|} \exp(u_j h + b_j)}$$

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

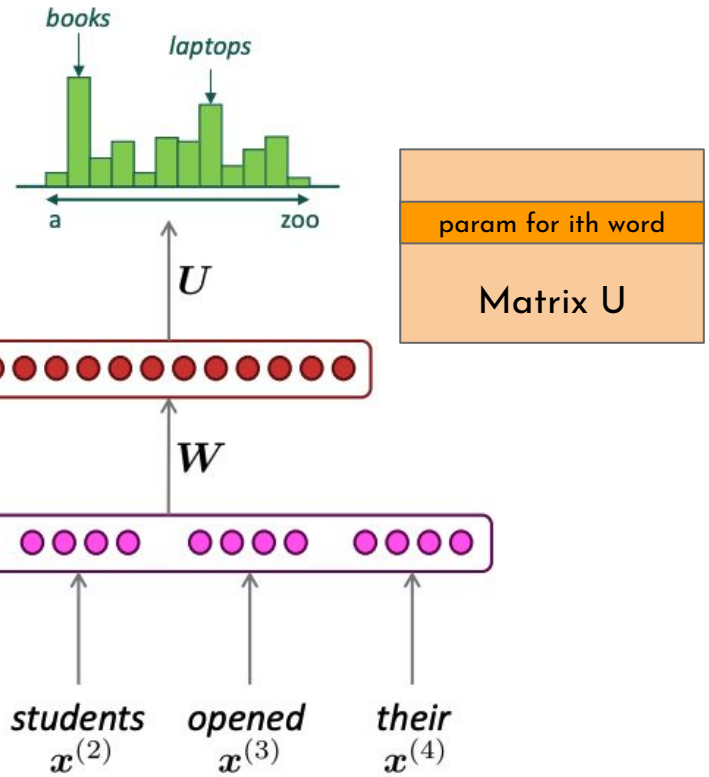
(Or average)

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Neural Language Model

Source: stanford 224n

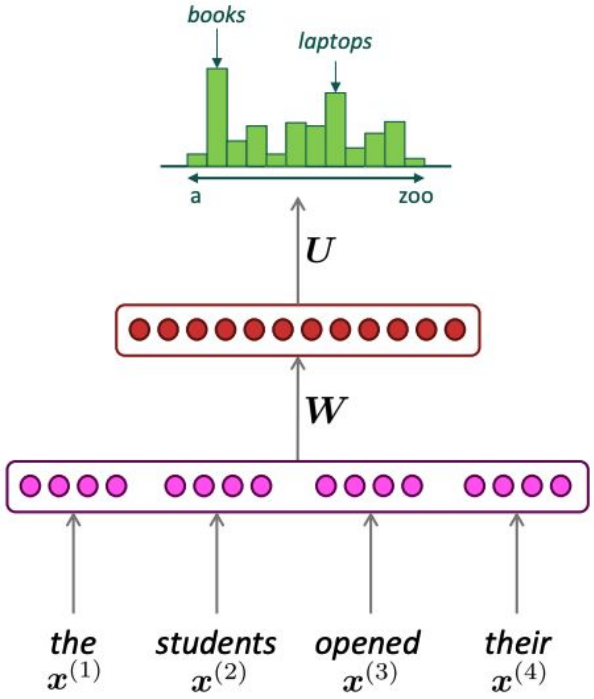
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Language Model vs Word2Vec

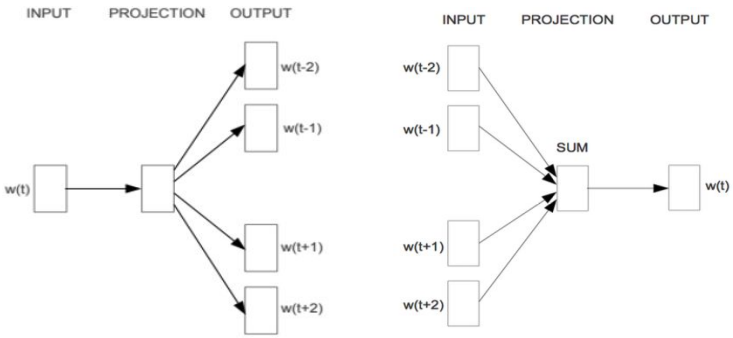
- **2003** (1st Neural LM) vs **2013** (Word2vec)
- Both capable of learning **word embeddings**
- Learning objectives:
  - **Neural LM:** $J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})$
  - **Word2vec:** $J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$
- No hidden layer in word2vec, more computational efficient
- Word2vec cannot be used for language modeling

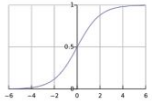
More to come later.....

Summary

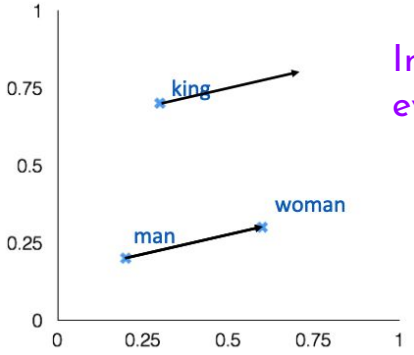
Source: stanford 224n

word2vec





Extrinsic evaluation



Intrinsic evaluation

Neural language model