Gensim:
Topic Modelling for Humans

Radim Řehůřek, Ivan Menshikh, Shiva Manne

Setup instructions: https://goo.gl/EaU8Gm

Workshop goals

- Introduce Gensim capabilities and API
- Understand text embeddings: Word2vec, FastText & co
- Learn to apply Gensim to NLP tasks
- Have a good time :)

Let’s dive to NLP with Gensim :)

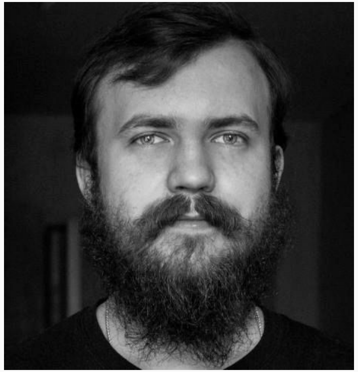
Introductions & agenda

Radim Řehůřek
Founder RARE Technologies

@RadimRehurek
piskvorky

Shiva Manne
R&D

@shivamanne_
manneshiva



Ivan Menshikh
Maintainer of Gensim

@menshikh_iv
menshikh-iv

You?

Gensim: academia & industry

Facts:

- LGPL 2.1 since 2011
- 270+ code contributors
- 6500+ stars, 2500+ forks
- 900+ academic citations
- Thousands of industry adopters

Gensim: Sphere of applications

Gensim (**gen**erate **sim**ilar) is all about **text embeddings** (of words, sentences, documents) and **finding similar texts** using vector representations.

Unsupervised text analysis is central to many business tasks:

- Content clustering and theme discovery: customer support
- Semantic search engines, “concept search”: contracts, digital libraries
- Recommendation systems: suggested apps on Google Play
- Sentiment analysis: customer feedback
- Summarization: newsletters

Gensim capabilities

- Data streaming: you can use datasets much larger than RAM
- Fast training using BLAS, multi-core & distributed mode
- Robust implementations of popular models: LDA, LSI, Word2Vec, …
- Gensim-data repository with pre-trained NLP models & datasets
- Community & support: Twitter, Gitter, mailing list
- Student incubator + GSoC mentoring organization
- `pip install -U gensim`

Gensim: what we will solve today (spoiler)

Semantic search engine

Text Classification

Gensim walkthrough: bag of words

Text

The quick brown fox jumps over the lazy dog. A cat says meow, a dog says woof.

Document x Term matrix

	cat	brown	woof	meow	the	quick	dog	says	fox	a	lazy	over	jumps
#1		1			2	1	1		1		1	1	1
#2	1		1	1			1	2		2			

Gensim walkthrough: bag of words

```python
from gensim.parsing import preprocess_string
from gensim.corpora import Dictionary, MmCorpus

text = """
The quick brown fox jumps over the lazy dog.
A cat says meow, a dog says woof.
"""
documents = [
    preprocess_string(doc.strip())
    for doc in text.split(".")
    if len(doc.strip())
]
print(documents)
# [[u'quick', u'brown', u'fox', ...], [u'cat', u'sai', ...]]

dct = Dictionary(documents)
corpus = [dct.doc2bow(doc) for doc in documents]
print(corpus)
# [[(0, 1), (1, 1), (2, 1), ...], [(1, 1), (6, 1), ...]]

```

Gensim walkthrough: TF-IDF

Q: Bag of words is a “crude” representation (only a counter), can we make it better?
A: Of course, let’s try TF-IDF.

```
from gensim.models import TfidfModel

tfidf = TfidfModel(corpus, id2word=dct)

for doc in tfidf[corpus]:
    print(doc)

# [(0, 0.4472135954999579), (2, 0.4472135954999579), (3, 0.4472135954999579), ... ]
# [(6, 0.3779644730092272), (7, 0.3779644730092272), (8, 0.7559289460184544), ... ]
```

Gensim walkthrough: limits of sparse representations

Q: How good is this sparse TF-IDF representation?
A: Surprisingly good baseline in IR, but there are disadvantages:

- Large dimensionality (typical size of Dictionary in business tasks: 0.1-1M words)
- Almost impossible to apply non-linear methods
- Bad at capturing context (can be partially fixed by word n-grams)

Q: How to solve this problems (at least partially)?
A: Use dimensionality reduction (compression) techniques that produce a dense representation, aka **embeddings**. Practical algorithms such as LSI or word2vec can often be expressed in terms on **Matrix Factorization**.

Gensim walkthrough: matrix factorization

Matrix Factorization - decompose input matrix into a product of matrices.

DOCUMENTS

WORDS

	#1	#2	#3	#4	#5	#6	#7
cat	1	0	0	0	3	0	0
dog	2	1	0	0	0	1	0
woof	0	0	0	2	0	0	0
meow	0	0	1	0	1	0	0
say	0	1	0	1	0	4	0

$\approx$

EMBEDDING

WORDS

	x_1	x_2
cat		
dog		
woof		
meow		
say		

DOCUMENTS

EMBEDDING

	#1	#2	#3	#4	#5	#6	#7
y_1							
y_2							

Gensim walkthrough: LSI (truncated SVD)

DOCUMENTS

WORDS	#1	#2	#3	#4	#5
cat	1	0	0	0	3
dog	2	1	0	0	0
woof	0	0	0	2	0
meow	0	0	1	0	1
say	0	1	0	1	0

≈

L singular vectors

WORDS	x_1	x_2
cat		
dog		
woof		
meow		
say		

σ_1	0
0	σ_2

singular values

DOCUMENTS

R singular vectors

	#1	#2	#3	#4	#5
y_1					
y_2					

$$C \approx U \Sigma V^T$$

Gensim walkthrough: LSI (truncated SVD)

```
import gensim.downloader as api
from gensim.models import LsiModel
from gensim.corpora import Dictionary


data = api.load("text8")  # load `text8` dataset using gensim-data api

dct = Dictionary(data)
print(len(dct))  # big dictionary: 253854 tokens
dct.filter_extremes(no_below=7, no_above=0.2)
print(len(dct))  # now size is OK: 42839 tokens

corpus = [dct.doc2bow(doc) for doc in data]  # convert corpus to BoW format
model = LsiModel(corpus, id2word=dct, num_topics=300)  # train LSI

model.projection.u  # U - left singular vectors (word-embeddings)
model.projection.s  # Σ - singular-values

vt = model[corpus]  # Vt - right singular vectors (document-embeddings)
```

Gensim walkthrough: LSI (truncated SVD)

```
for topic_id, representation in model.show_topics(15, num_words=5):
    print("#{}: {}".format(topic_id, representation))


#0: 0.727*"import" + 0.156*"info" + 0.150*"duplicate" + 0.146*"jargon" + 0.101*"rfc"
#1: 0.573*"import" + 0.118*"duplicate" + 0.113*"info" + 0.111*"jargon" + -0.102*"est"
#2: -0.945*"bwv" + -0.132*"fugue" + -0.100*"gott" + -0.097*"prelude" + -0.086*"concerto"
#3: 0.681*"kong" + 0.660*"hong" + 0.096*"est" + 0.071*"prc" + 0.063*"macau"
#4: 0.410*"est" + -0.385*"apple" + -0.239*"mac" + -0.220*"os" + -0.183*"microsoft"
#5: -0.462*"apple" + -0.331*"est" + -0.286*"mac" + -0.278*"os" + -0.205*"macintosh"
#6: 0.440*"lebanon" + 0.335*"inducted" + 0.192*"lebanese" + 0.188*"israeli" + 0.158*"batman"
#7: -0.469*"inducted" + 0.412*"lebanon" + -0.395*"finalist" + 0.181*"lebanese" + -0.177*"champion"
#8: 0.651*"finalist" + -0.582*"inducted" + 0.277*"champion" + 0.184*"wimbledon" + 0.106*"lebanon"
#9: -0.424*"inducted" + 0.389*"batman" + -0.329*"apollo" + -0.208*"finalist" + -0.172*"lebanon"
#10: 0.670*"apollo" + -0.303*"inducted" + 0.291*"lunar" + -0.182*"finalist" + 0.147*"module"
#11: 0.644*"batman" + -0.348*"lincoln" + 0.189*"comics" + 0.129*"wayne" + 0.122*"bruce"
#12: 0.549*"microsoft" + -0.276*"apple" + 0.203*"wikipedia" + 0.203*"linux" + -0.158*"nfl"
#13: -0.587*"lincoln" + 0.272*"nfl" + -0.223*"batman" + -0.183*"ford" + -0.145*"aristotle"
#14: 0.398*"bass" + -0.265*"microsoft" + 0.222*"guitar" + 0.212*"ford" + 0.178*"blues"
```

LSI Demo on EN Wikipedia:

https://goo.gl/6w4th7

Gensim walkthrough: semantic search engine

We already know about BoW and LSI, it’s time to apply this in practice

Let’s make a **semantic search engine**

PragueML-walkthrough.ipynb

Let’s dive to notebook →

Gensim walkthrough: LDA

Q: Great, but what if we want to have interpretable topics? “topics with names”?
A: Latent Dirichlet Allocation is your choice!

Which topic will I talk about?
Throw small dice to decide (~100 faces)
(topics in a document are modeled as a Dirichlet probability distribution)

Which word will I say from the topic **X**?
Throw big dice to decide (~500,000 faces)
(words in a topic are modeled as another Dirichlet probability distribution)

Gensim walkthrough: LDA

Example: we have 2 topics"**cat**" and "**dog**" (both equiprobable)

Cat-related words: meow, cat, wooly, kitten, cute (all equiprobable)
Dog-related words: woof, wooly, dog, bark, angry (all equiprobable)

Generated text: meow cute woof meow cat kitten bark wooly cat wooly wooly meow wooly angry angry woof cat cat meow kitten woof meow cute meow woof wooly cat angry wooly dog wooly wooly meow bark dog wooly wooly dog cute bark wooly kitten wooly meow meow wooly cute bark cute angry ...

Gensim walkthrough: LDA

DOCUMENTS

WORDS

	#1	#2	#3	#4	#5	#6	#7
cat	1	0	0	0	3	0	0
dog	2	1	0	0	0	1	0
woof	0	0	0	2	0	0	0
meow	0	0	1	0	1	0	0
say	0	1	0	1	0	4	0

$\approx$

TOPICS

WORDS

	x_1	x_2
cat		
dog		
woof		
meow		
say		

DOCUMENTS

TOPICS

	#1	#2	#3	#4	#5	#6	#7
y_1							
y_2							

$$C \approx \Phi \quad \Theta$$

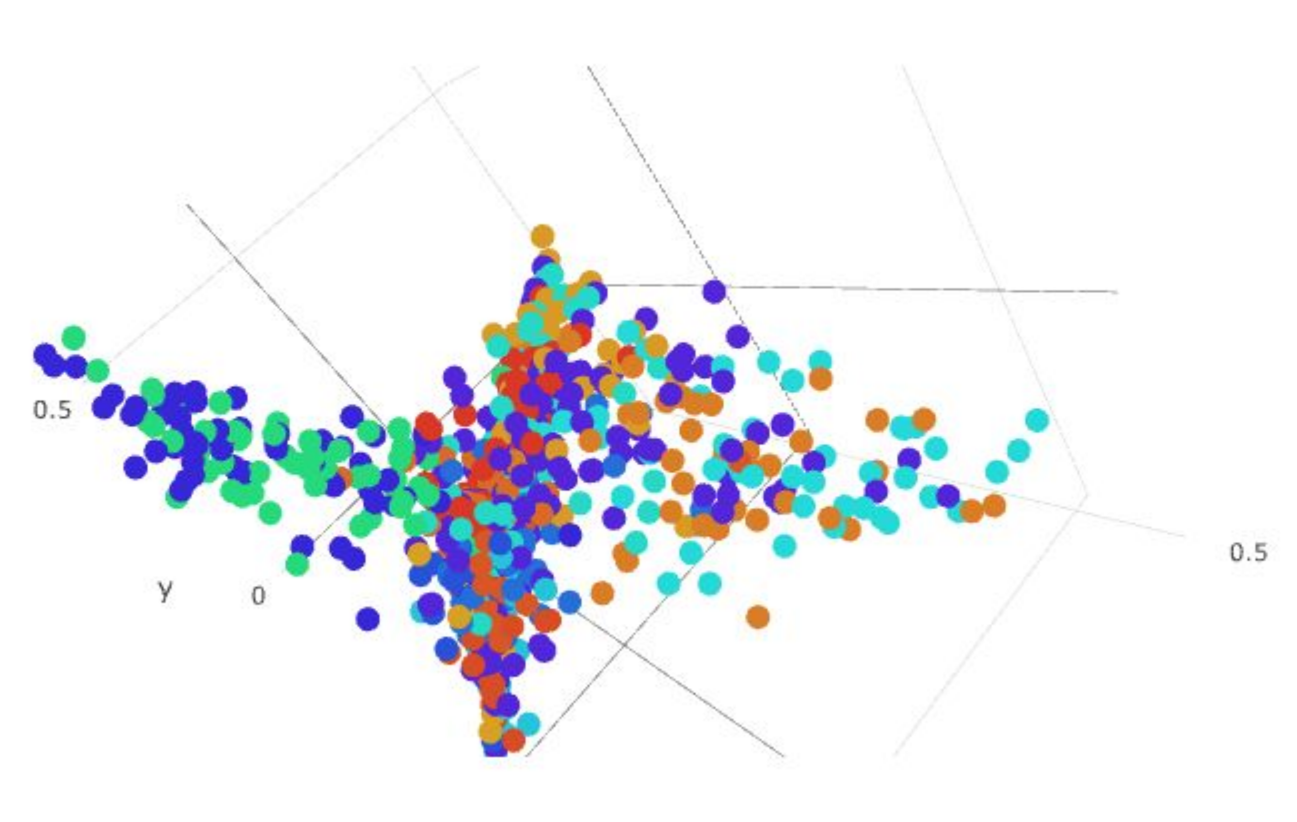
Gensim walkthrough: LDA

```python
import logging
import gensim.downloader as api
from gensim.parsing import preprocess_string
from gensim.models import LdaModel
from gensim.corpora import Dictionary

logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(message)s')

data = api.load("20-newsgroups")
dataset = [preprocess_string(doc["data"]) for doc in data]

dct = Dictionary(dataset)
dct.filter_extremes(no_below=5, no_above=0.15)
corpus = [dct.doc2bow(doc) for doc in dataset]

model = LdaModel(corpus, id2word=dct, num_topics=30,
                 passes=10, random_state=42)
```

Gensim walkthrough: LDA

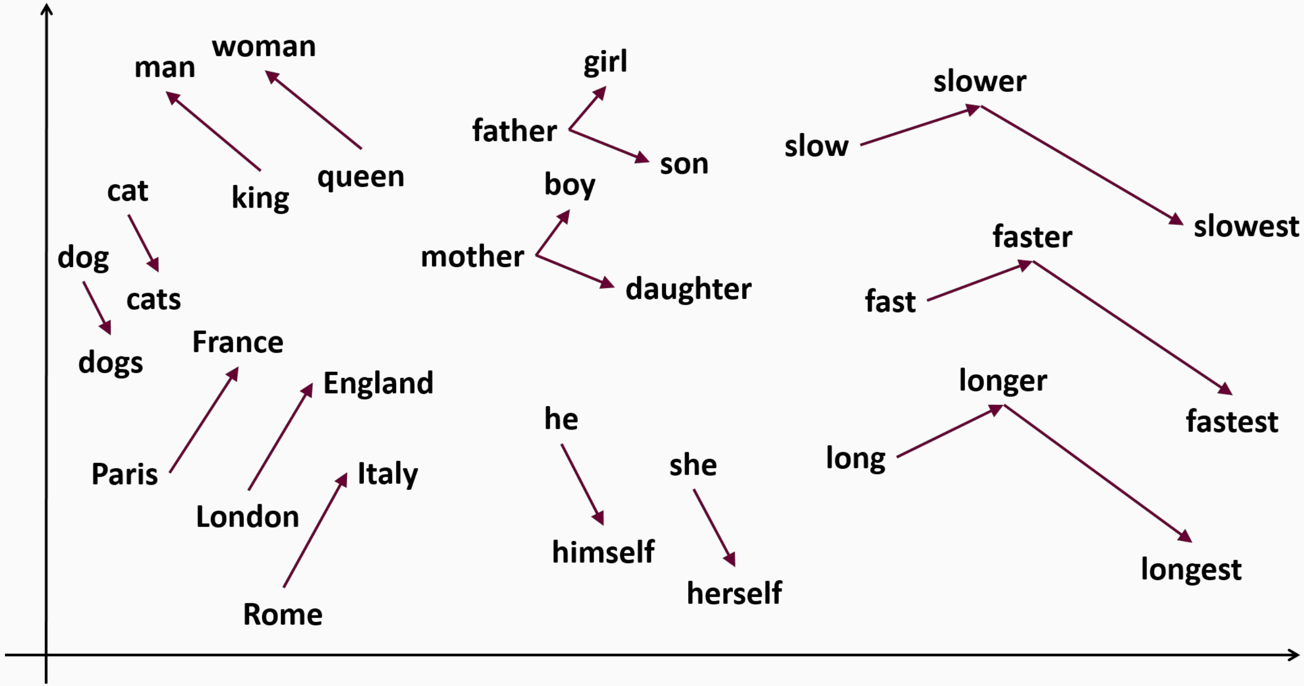
```
for topic_id, representation in model.show_topics(20, num_words=7):
    print("#{}: {}".format(topic_id, representation))

#6: 0.020*"govern" + 0.014*"kei" + 0.013*"presid" + 0.010*"law" + 0.008*"secur" + 0.008*"clipper" + 0.008*"clinton"
#0: 0.021*"drive" + 0.015*"card" + 0.012*"window" + 0.010*"driver" + 0.009*"help" + 0.009*"version" + 0.009*"disk"
#24: 0.010*"jame" + 0.009*"amend" + 0.009*"liver" + 0.007*"ear" + 0.007*"phil" + 0.007*"pgf" + 0.007*"bear"
#17: 0.013*"power" + 0.009*"water" + 0.007*"ground" + 0.007*"air" + 0.006*"chang" + 0.006*"heat" + 0.006*"batteri"
#15: 0.025*"bit" + 0.024*"scsi" + 0.022*"encrypt" + 0.021*"chip" + 0.017*"data" + 0.015*"isra" + 0.013*"cramer"
#22: 0.028*"gun" + 0.019*"drug" + 0.011*"david" + 0.010*"washington" + 0.009*"control" + 0.008*"kill" + 0.008*"crime"
#29: 0.190*"max" + 0.032*"photographi" + 0.019*"princeton" + 0.018*"astro" + 0.015*"enet" + 0.014*"giz" + 0.014*"bhj"
#26: 0.027*"file" + 0.019*"window" + 0.016*"imag" + 0.014*"program" + 0.010*"ftp" + 0.008*"server" + 0.008*"version"
#19: 0.028*"car" + 0.008*"bui" + 0.008*"price" + 0.008*"bike" + 0.008*"engin" + 0.007*"sell" + 0.007*"light"
#23: 0.019*"religion" + 0.017*"god" + 0.013*"believ" + 0.013*"exist" + 0.012*"belief" + 0.012*"atheist" + 0.011*"christian"
#18: 0.030*"brian" + 0.022*"score" + 0.020*"mit" + 0.019*"goal" + 0.016*"period" + 0.015*"pit" + 0.015*"arizona"
#3: 0.021*"purdu" + 0.019*"georg" + 0.016*"usr" + 0.014*"portal" + 0.013*"duke" + 0.013*"miller" + 0.011*"jake"
#4: 0.027*"netcom" + 0.026*"ibm" + 0.015*"monei" + 0.014*"servic" + 0.014*"cost" + 0.013*"insur" + 0.013*"austin"
#12: 0.015*"said" + 0.013*"dai" + 0.009*"start" + 0.008*"food" + 0.008*"go" + 0.008*"got" + 0.007*"happen"
#13: 0.019*"armenian" + 0.015*"israel" + 0.013*"war" + 0.011*"muslim" + 0.010*"jew" + 0.009*"turkish" + 0.007*"nation"
#2: 0.025*"sun" + 0.016*"blue" + 0.013*"colorado" + 0.013*"leagu" + 0.012*"green" + 0.012*"pitt" + 0.011*"san"
#11: 0.045*"infect" + 0.026*"hiv" + 0.020*"soviet" + 0.020*"jon" + 0.018*"leaf" + 0.017*"appear" + 0.014*"sgi"
#21: 0.020*"planet" + 0.015*"earth" + 0.012*"rai" + 0.012*"atmospher" + 0.012*"temperatur" + 0.010*"venu" + 0.010*"caltech"
#14: 0.037*"hst" + 0.023*"berkelei" + 0.019*"gamma" + 0.016*"umd" + 0.015*"eng" + 0.011*"demon" + 0.010*"thoma"
#1: 0.012*"program" + 0.011*"spacecraft" + 0.011*"april" + 0.010*"book" + 0.009*"inform" + 0.009*"number" + 0.008*"solar"
```

Gensim walkthrough: LDA



Gensim walkthrough: The BoW is dead. Long live the Word2Vec!

Q: LSI is nice but old, how about some **neural network** approaches?
A: Yes, right now most popular approaches (that actually work) are “Word2Vec-based”



Gensim walkthrough: Word2Vec Intro

And again, idea based on **matrix factorization**, using **word co-occurrence** matrix as the algorithm input.

WORDS (columns) × **WORDS** (rows)

	cat	dog	woof	meow	say
cat	0	2	0	0	3
dog	2	0	0	0	0
woof	0	0	0	1	0
meow	0	0	1	0	1
say	3	0	0	1	0

$\approx$

WORDS (rows) × **EMBEDDING** (columns)

	x_1	x_2
cat		
dog		
woof		
meow		
say		

EMBEDDING (rows) × **WORDS** (columns)

	cat	dog	woof	meow	say
y_1					
y_2					

Word Embeddings: Introduction

- NLP Tasks -> textual data

- Basic units of data -> words!

- Models are incapable of dealing with raw words/plain text. They only process numbers/vectors/matrices.

  Word Embeddings = mapping each word to a vector (in an N-dimensional space)

- One-hot encoding: Vocabulary = ['cat', 'dog', 'laugh']
  - cat = [1, 0, 0]
  - dog = [0, 1, 0]
  - laugh = [0, 0, 1]

- Problems: high dimensionality, sparse, no semantics preserved.

Word2Vec: How to come up with an embedding?

"You shall know a word by the company it keeps"

-J. R. Firth 1957

Distributional Hypothesis:
Words that occurs in the same context tend to have similar meaning.

- He handed her a glass of **bardiwac**.
- Beef dishes are made to complement the **bardiwacs**.
- Nigel staggered to his feet, face flushed from too much **bardiwac**.
- Malbec, one of the lesser-known **bardiwac** grapes, responds well to Australia's sunshine.
- I dined off bread and cheese and this excellent **bardiwac**.
- The drinks were delicious: blood-red **bardiwac** as well as light, sweet Rhenish.

What does **bardiwac** mean?

Word Embeddings:

“**Bardiwac**” is a heavy red alcoholic beverage made from grapes.

Word2vec: Introduction

Objective: Come up with dense, semantic preserving vector representations for words

Input: A large corpus of text (e.g. Wikipedia dump).

Fake Task:

- The idea is to train a neural network to predict the context word given an input word.

- Input: one-hot vector for input word.
- Output: probability for each word (being the context) in the vocabulary

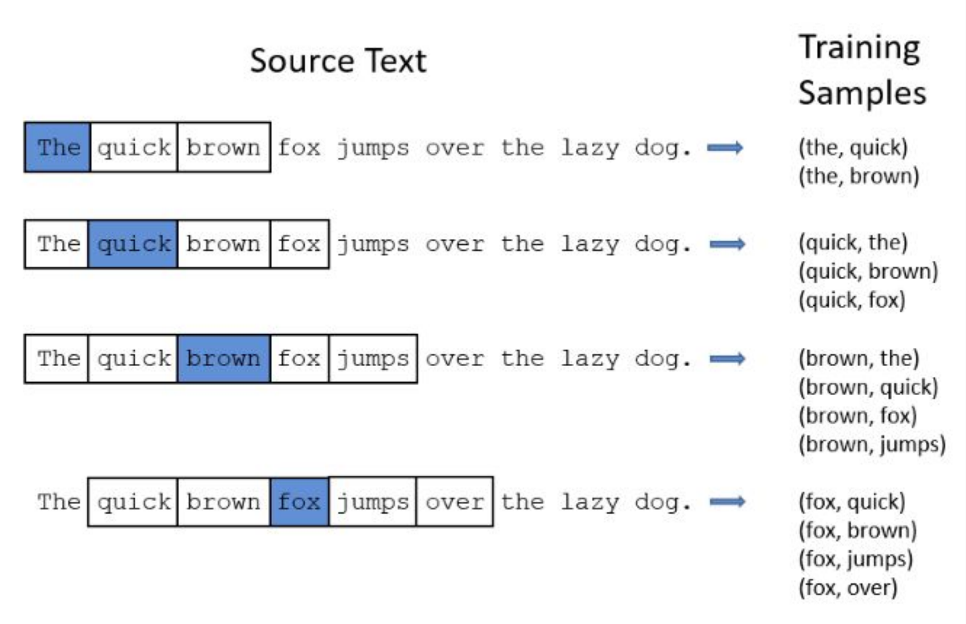


Image courtesy: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2vec: Architecture & Loss function

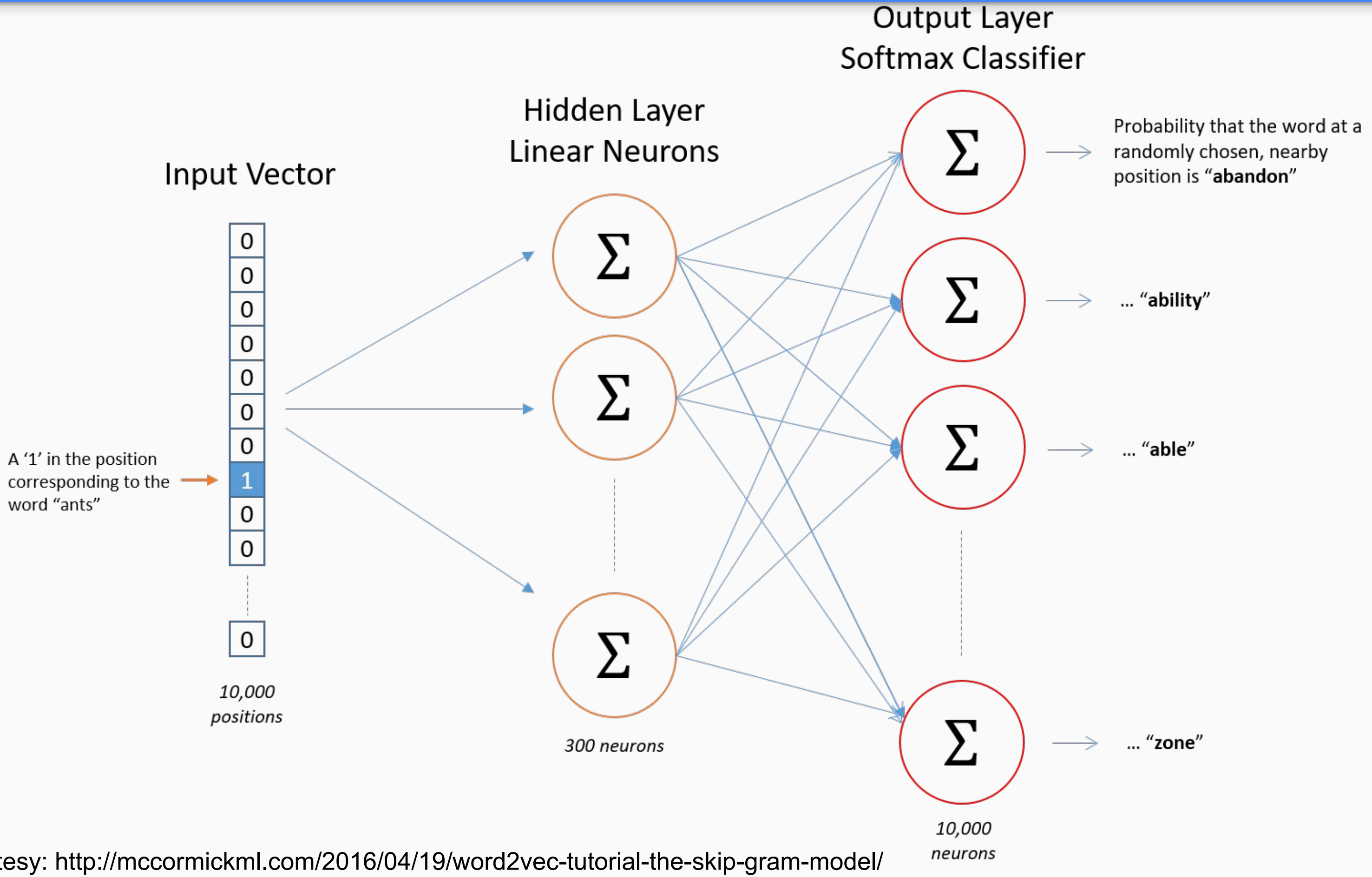


Image courtesy: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2vec: Mathematics - Loss function

- Model the probability of **context word** given a word.
- Vector for input word: v_{in}
  Vector for context word: v_{out}
- $P(v_{out} | V_{in})$

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

- Cosine similarity: (-1, 1) -> Probability (0,1)

$$softmax = \frac{exp(v_{in} \cdot v_{out})}{\sum_{k \in V} exp(v_{in} \cdot v_k)} = P(v_{out} | v_{in})$$

- Maximize the log probability of any context word given centre word

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log p(w_{t+j} | w_t)$$

Θ represents all variables we want to optimize

“The fox jumped **over** the lazy dog”

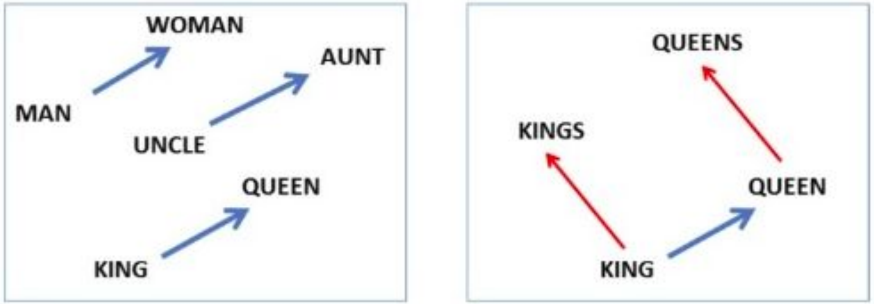
P(the|over)
P(fox|over)
P(jumped|over)
P(the|over)
P(lazy|over)
P(dog|over)

Image courtesy:
http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf
http://www.slideshare.net/ChristopherMoody3/word2vec-lda-and-introducing-a-new-hybrid-algorithm-lda2vec

Word2Vec: Results

Unintended effect: Arithmetics make sense!!

vec("man") – vec("king") + vec("woman") = vec("queen")

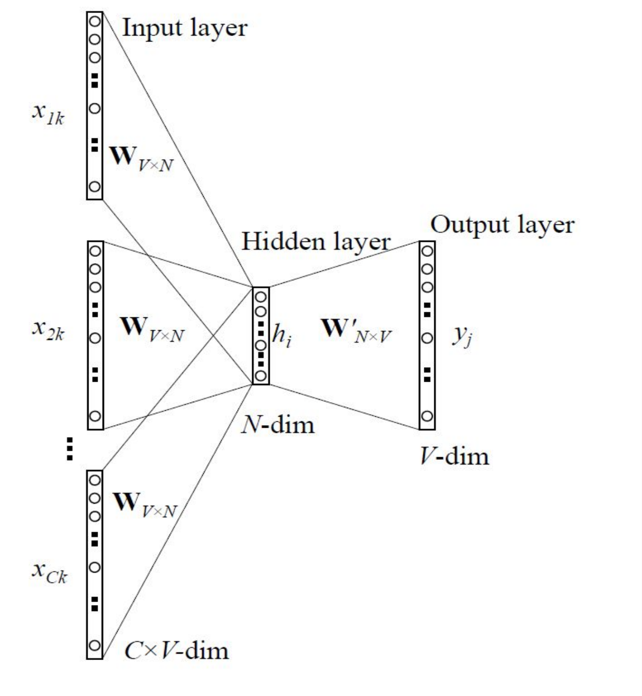


Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Image courtesy: https://www.slideshare.net/mlprague/tom-mikolov-distributed-representations-for-nlp
https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Word2Vec: 2 Flavours

Continuous Bag of Words (CBOW)



Skipgram

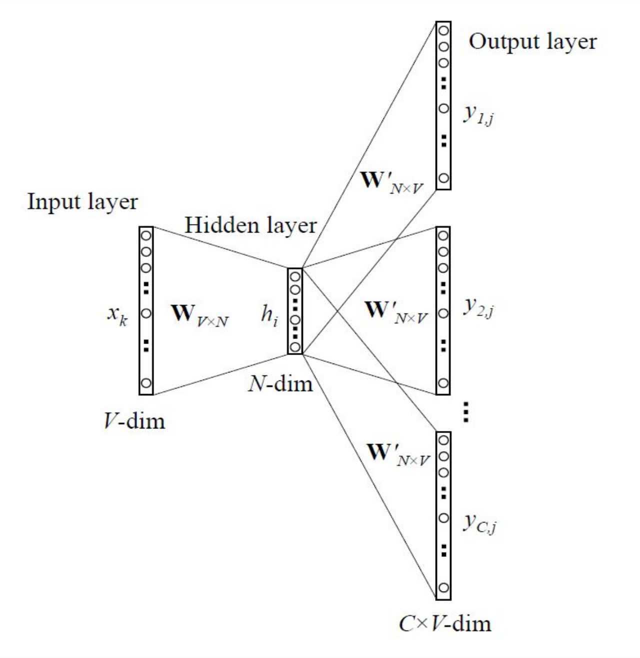


Image courtesy: Xin Rong - “word2vec Parameter Learning Explained” (https://arxiv.org/pdf/1411.2738.pdf)

Word2Vec: Subsampling and rare word pruning

- Word2Vec model has two additional parameters:
  - `min_count`: words occurring less than `min_count` number of times in the entire corpus are discarded.
  - `sample`: each word 'w_i' is discarded with a probability given by:

$$P(w_i) = 1 - \sqrt{\frac{sample}{f(w_i)}}$$

- Frequently occuring words such "the", "of", "and" do not provide much information.
- Effective window size increased.
- Improvement in quality of embeddings.

Word2Vec: Computational Optimizations

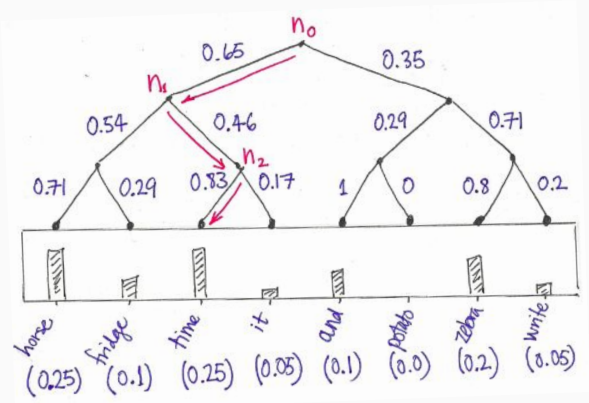
Having to update every output word vector for every word in a training instance is very expensive

“To solve this problem, an intuition is to limit the number of output vectors that must be updated per training instance. One elegant approach to achieving this is hierarchical softmax; another approach is through sampling.”

$$softmax = \frac{exp(v_{in} \cdot v_{out})}{\sum_{k \in V} exp(v_{in} \cdot v_k)} = P(v_{out}|v_{in})$$

- **Negative Sampling:** only update a sample of output words per iteration.
  - Randomly select just a small number of “negative” words (say 5) and only update weights to update the weights for.

- **Hierarchical Softmax:**
  - Efficient way of computing softmax
  - Create a binary tree to represent all words in the vocabulary
  - Words are leaves of this tree
  - Log(|Vocabulary|) updates required instead of |Vocabulary|



Word2Vec: Application

Let us play a little with “Word2Vec” and apply on a real task - Multiclass text classification

FastText: “Enriching Word Vectors with Subword Information”

- Library by Facebook for supervised text classification and learning unsupervised word embeddings
- **Idea:** Represent words as sum of its character n-ngrams
  - where = whe + her + ere + wher + here + where
- Useful for morphologically rich languages - Turkish, Arabic, Hebrew.

	Singular	**Plural**
Nominative	mensa	mensae
Vocative	mensa	mensae
Accusative	mensam	mensās
Genitive	mensae	mensārum
Dative	mensae	mensīs
Ablative	mensā	mensīs

amazing : amazingly :: calm : ?(calmly)

embedding(amazingly) – embedding(amazing) = embedding(calmly) – embedding(calm)

Image courtesy: http://www.livelatin.org/declensions/

FastText: Model

- As in skip-gram: model probability of a context word given a word

classifier for word c: v_c

feature for word w: h_w

$$p(c|w) = \frac{e^{h_w^\top v_c}}{\sum_{k=1}^{K} e^{h_w^\top v_k}}$$

- Feature of a word computed using n-grams:

$$h_w = \sum_{g \in w} x_g$$

mang erai ange
man ang gera
era
nge ger rai nger

+

mangerai

Character n-grams

Word itself

- As for the previous model, use hashing for n-grams

Credits: Slide picked up from Piotr Bojanowski's talk on FastText (https://www.youtube.com/watch?v=CHcExDsDeHU)

FastText: Out of Vocabulary (OOV) word embeddings

- Possible to build vectors for unseen words!

$$h_w = \sum_{g \in w} x_g$$

Credits: Slide picked up from Piotr Bojanowski's talk on FastText (https://www.youtube.com/watch?v=CHcExDsDeHU)

FastText

Continued in Notebook...

More *2Vecs: Word2Vec extensions

- Emoji2Vec:
  - Why? - useful for analysing social media data
  - Emojis vectors in the same space as word vectors
  - Train task: predict emoji given it’s description

- App2Vec:
  - Embeddings for mobile applications
  - Why? App recommendation, App clustering
  - Words = app_ids
  - “app sessions within short time gaps to the target app should contribute more in predicting the target app” => weighted(by recency) context average

More *2Vecs: Word2Vec extensions

- Item2Vec
  - Item embedding for collaborative filtering
  - Word = item, context = other items for same user / in same order
  - User1 = {item1, item2, item3} or Order1= {item4, item5, item6}

- Sense2Vec
  - Drawback of word2vec: can’t model polysemic words (multiple meaning)
  - Duck(Verb), Duck(Noun)
  - Append POS Tags, Named Entities
  - Duck_Verb, Duck_Noun

More *2Vecs: Word2Vec extensions

- CV2Vec
  - “find candidates that are most similar to a reference person or the job ad itself”
  - Not open source yet

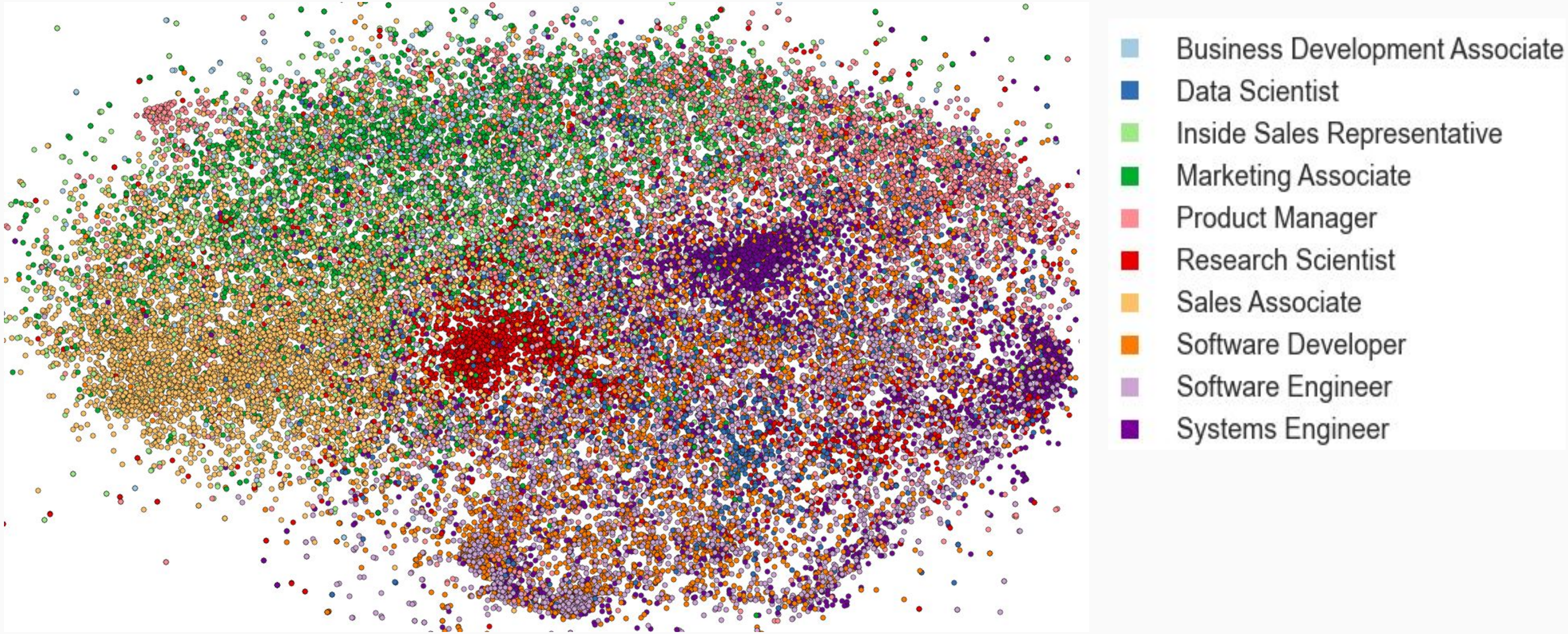


Image: https://medium.com/talla-inc/introducing-cv2vec-a-neural-model-for-candidate-similarity-e215b1b12472

More *2Vecs: Sent2Vec

- "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features" - *Matteo Pagliardini, Prakhar Gupta, Martin Jaggi*

- Unsupervised sentence embeddings learning
- Extension of FastText and Word2Vec(CBOW)

- Task:
  - Learn word embeddings
  - Predict missing words from sentences given the sentence
  - $P(V_{\text{missing word}} \mid V_{\text{sentence}})$
  - V_{sentence} = average of word n-gram embeddings

- To get a sentence vector for a new word, simply average all words embeddings in the new sentence

Thanks for listening!

No time to cover more slides.

Doc2Vec: Extending Word2Vec to get Document/Paragraph Vectors

- What does it do?
  Dense, fixed low-dimensional vectors for variable sized text fragments - sentences, paragraphs or documents.

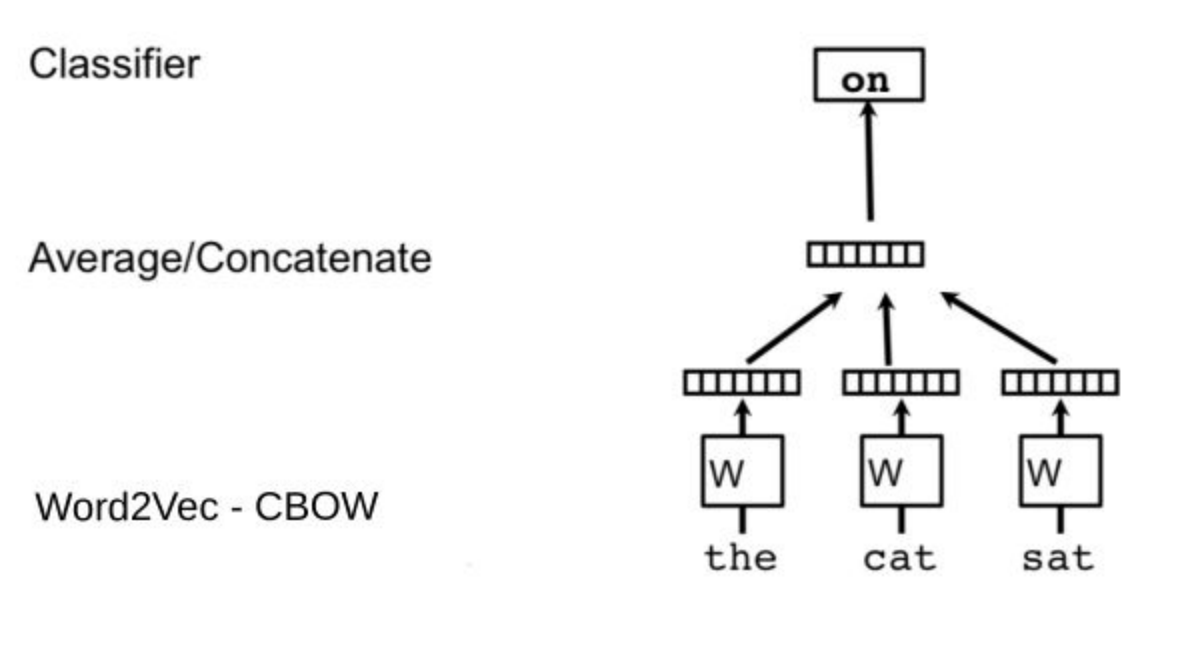
- Biggest benefits over other models that calculate document vectors
  - Learns from unlabelled data
  - Don’t have to deal with high dimensional and sparse data (like Bag of n-grams))
  - Outperforms other models on a number of text classification and sentiment analysis tasks

- Same underlying architecture as Word2Vec

- Recall the fake task/training in Word2Vec:
  - Randomly initialize weights
  - Predict context words from a target word (Skipgram) or predict target word from context words(CBOW)
  - Minimize the loss, applying Stochastic Gradient Descent (SGD)

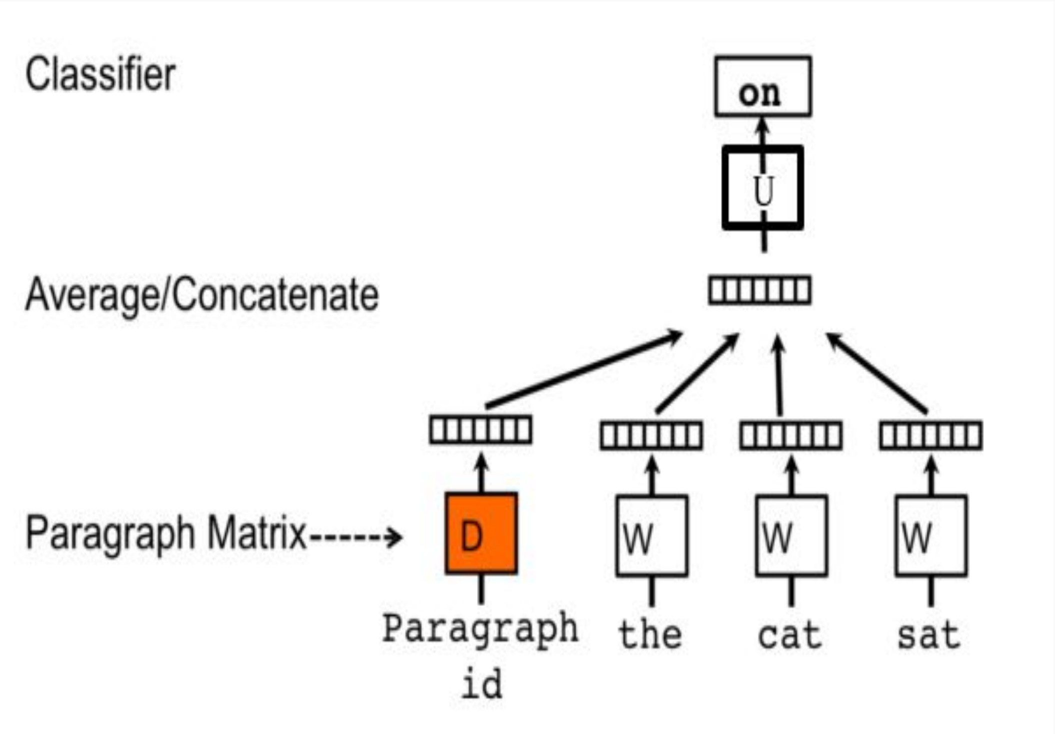
Doc2Vec: Architecture

Recollect Word2Vec CBOW Model



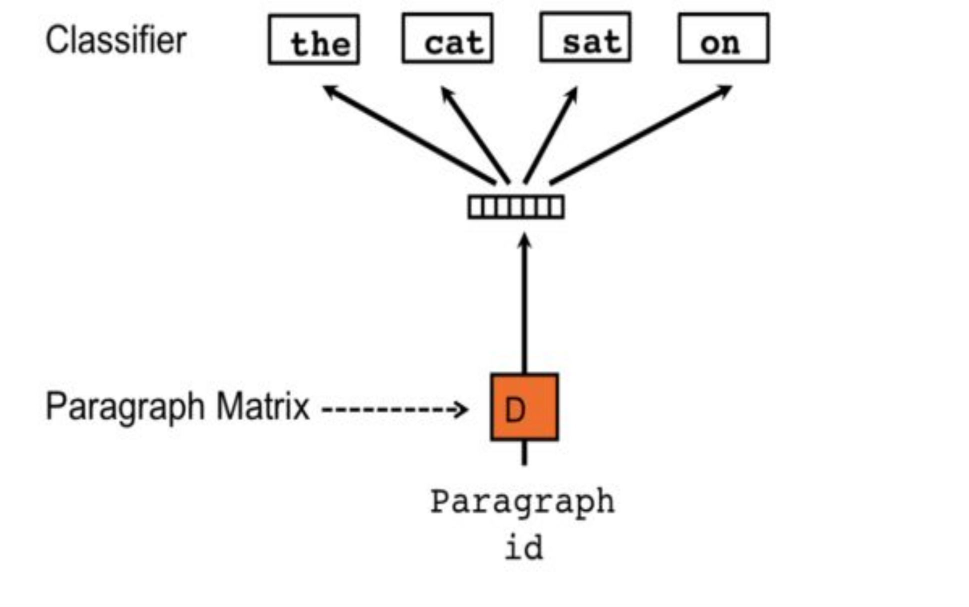
Doc2Vec: Architecture - Distributed Memory Model of Paragraph Vectors (PV-DM)

- Words mapped to vectors - Matrix **W**
- Paragraphs mapped to vectors - Matrix **D**
- Fake task:
  Predict a target word given it's context words and the paragraph ID from which the context is sampled.
- The paragraph vectors are also asked to contribute to the prediction task



Doc2Vec: Architecture - Distributed Bag of Words version (PV-DBOW)

- Analogous to Word2Vec Skipgram model
- Predict words randomly sampled from the paragraph in the output
- PV-DM performs better than PV-DBOW, but combining both PV-DM and PV-DBOW can give better results.



Doc2Vec: Inference Stage

To get vectors for new/unseen documents:

- Add more columns to Matrix **D**
- Train the model through gradient descent, holding matrices **W, U** fixed
- Only train on new data/documents/paragraphs