SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Word embeddings

PROF. D. HERREMANS

50.038 Computational Data Science

Intro

REPRESENTING TEXT

How to represent meaning of text

- Meaning:

  What is meant by a word, text, concept, or action.
  - *Implied or explicit significance.*
  - *Important or worthwhile quality; purpose.*

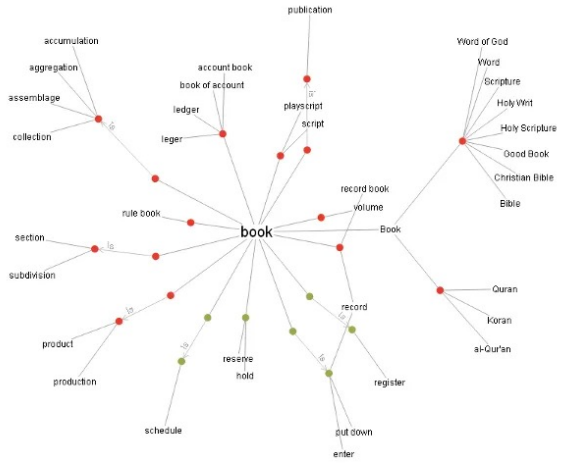
  -- Oxford dictionary

- → Natural Language Processing (NLP)

How to represent meaning

- **WordNet** – first release: Princeton, mid 1980s
- Has a taxonomy that includes synonyms, hypernyms ('is … a' relationship), etc.

```
[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')]
```




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

How to represent meaning

- Common answer: **WordNet**
- Has a taxonomy that includes synonyms and 'is … a' relationships, etc.

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
…
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good

Synonyms of 'good'

Problems with this discrete representation

Great as resource, BUT:

- Missing **nuances**, e.g. synonyms:

  adept, expert, good, practiced, proficient, skillful?

- Missing **new words** (impossible to keep up to date):

  wicked, badass, nifty, crack, ace, wizard, genius

  → Subjective

Problems with this discrete representation

- Requires human labour to create and adapt
- Hard to compute accurate word similarity
- The vast majority of rule-based and statistical NLP work regards words as atomic symbols

One-hot representation

- Dimensionality:
  - 20K (speech)
  - 500K (big vocab)
  - 13M (Google Web 1T)

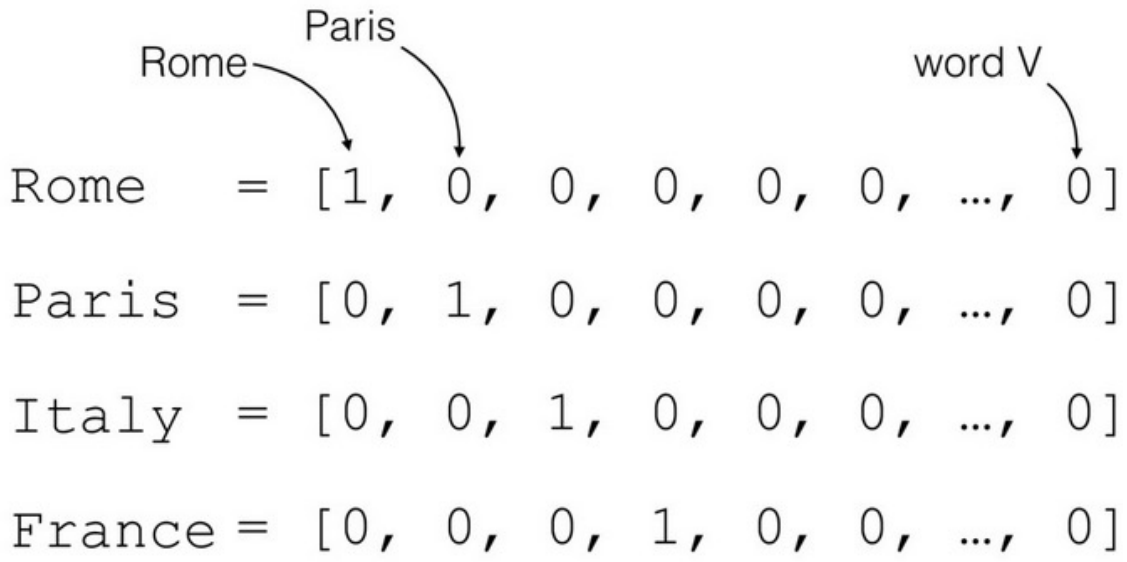


Image: https://medium.com/@athif.shaffy

One-hot representation

- Simplest method to implement
- Unordered, therefore the context of words are lost.
- The vector representation is in binary form, therefore no frequency information is taken into account.

Bag of words (BOW)

- D1: I went to the movies yesterday
- D2: Listening to music is great
- D3: Yesterday I went swimming and I ran

	I	Went	to	the	movies	yesterday	it	listening	to	music	is	great	swimming	and	ran
D1	1	1	1	1	1	1									
D2								1	1	1	1	1			
D3	**2**	1				1							1	1	1

Bag of words (BOW)

- High dimensional and very sparse!
- Can’t capture word order:

  → “good but expensive” and “expensive but good”: same representation!
- Can’t capture similarities:

  boy, girl, giraffe… BOW can’t say who are more similar

TF-IDF

- TD-IDF: Term Frequency - Inverse Document Frequency
- Motivation: words that appear in a large number of documents are not significant, so they get a smaller weight
- Rare words carry more meaning:
  - aardvark, cryogenic, … - topical content
  - The, a, big,… - linguistic glue

	Mary	Loves	Movies	Cinema	Art	John	Went	to	the	Delicatessen	Robert	Football	Game	and	for
d1	0.3779	0.3779	0.3779	0.3779	0.3779									0.0001	
d2							0.4402	0.001	0.02			0.4558	0.458		
d3			0.001				0.01		0.01		0.458				0.0001

Table based on slideshare.net/hadyelsahar

TF-IDF

- **Term frequency (TF):**

  Number of times the term appears in the doc/total number of words in the document

- **Inverse Document Frequency (IDF):**

  log(number of docs/number docs the term appears in)

- TFI-DF = TF *IDF

- The higher the TF-IDF score, the rarer the term is and vice-versa.

- Note: variations of TF definition exist, for instance, using logarithmic count

TF-IDF: Example

- Say that you want to know the TD-IDF weight (W_{cat}) of the word 'cat'. In your current 100-word document, 'cat' appears 12 times.
- In the total corpus you have (10 million documents), the word 'cat' appears in 0.3 million of those.

- $TF_{cat} = 12/100 = 0.12$
- IDF (cat) = log (10,000,000/300,000) = 1.52

- => $W_{cat} = (TF*IDF)_{cat} = 0.12 * 1.52 = 0.182$

Dense vector embeddings

SINGULAR VALUE DECOMPOSITION ETC.

Word vector representations

- Also called word embeddings
- Distributional semantics hypothesis:

You shall know a word by the company it keeps
– John Rupert Firth 1957

➔ Words that occur in a similar context tend to have similar meaning (Harris, 1954)

Distributional semantics hypothesis

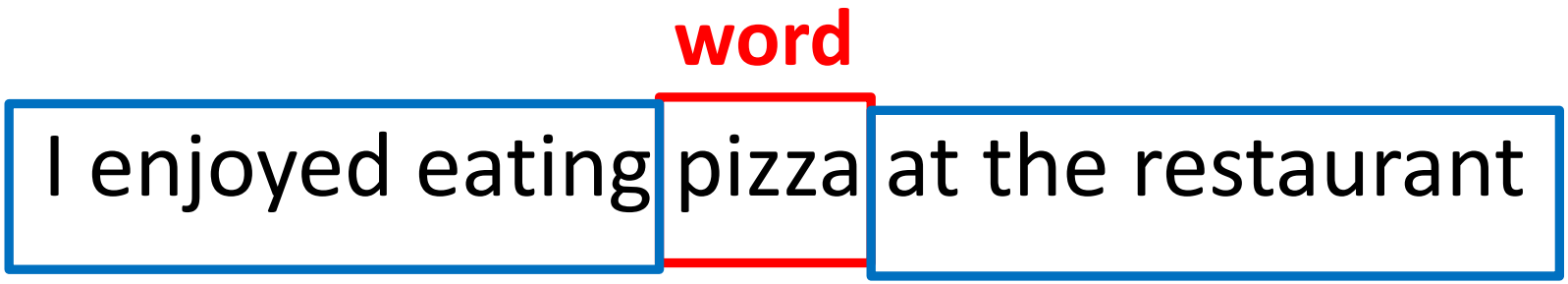
I enjoyed eating pizza at the restaurant

Distributional semantics hypothesis

word

I enjoyed eating pizza at the restaurant

Distributional semantics hypothesis



The company it keeps

Distributional semantics hypothesis

I enjoyed eating pizza at the restaurant

I enjoyed eating pineapple at the restaurant

Same context, same meaning?

How to integrate neighbours when representing words

- Answer: With a co-occurrence matrix X
- 2 Options: full document vs windows
  1. Word - document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
  2. Instead: Window around each word → captures both syntactic (POS) and semantic information

Window-based co-occurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window-based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with co-occurrence models

- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues

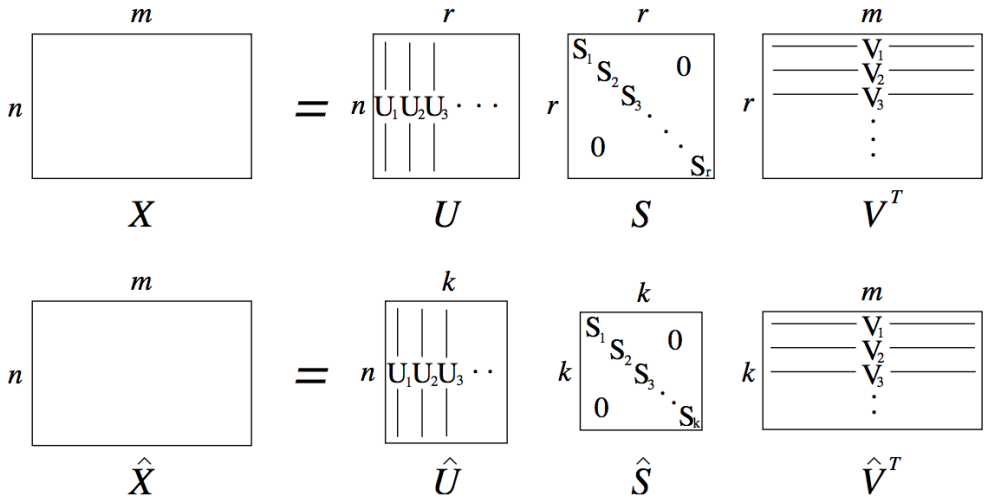
→Models are less robust

Solution: low dimensional vectors!

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually around 25 – 1000 dimensions
- How to reduce the dimensionality?

Method 1: dimensionality reduction on X

- Singular value decomposition of co-occurrence matrix X.
- U is used to represent the words.



$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

Simple SVD word vectors in Python

- Corpus: I like deep learning. I like NLP. I enjoy flying.

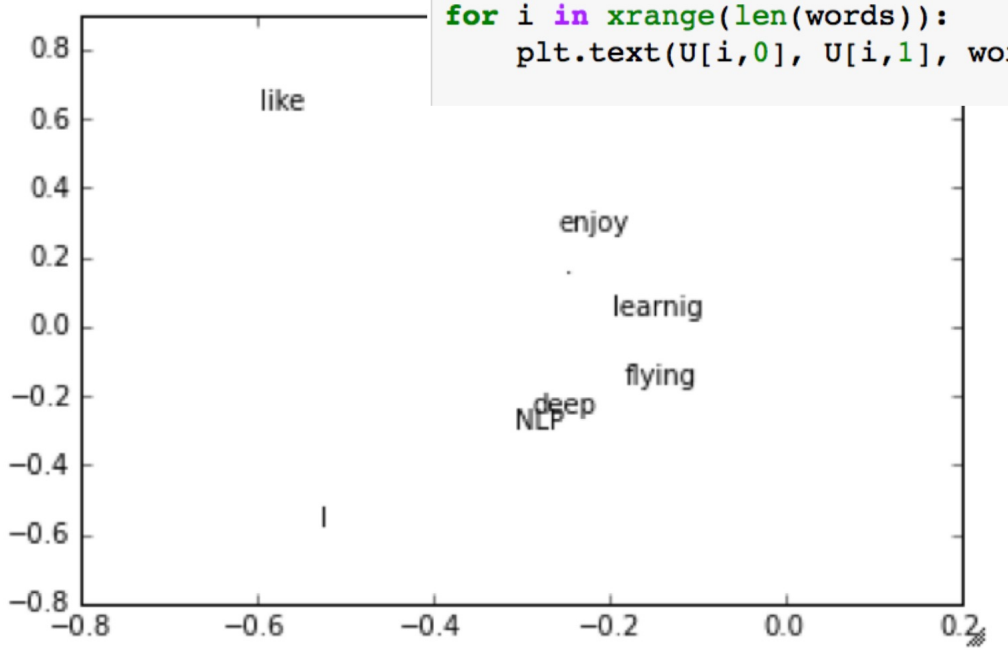
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

- Corpus: I like deep learning. I like NLP. I enjoy flying.
  Printing first two columns of U corresponding to the 2 biggest, most important components of the original matrix (largest singular values).

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Word meaning ~ word vectors

- In the models that follow, a word is always represented as a dense vector

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

t-SNE

- How to represent multi-dimensional vectors in 2-D space?
- t-Distributed Stochastic Neighbor Embedding (t-SNE) (van der Maaten & Hinton, 2008)
- Very well suited for the visualization of high-dimensional datasets
- How does it work?

1. t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked.
2. t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map.

   ➔ optimizes such that similar points in high-dimensional space are still closeby in low-dimensional space, and vice-versa.

KL Divergence

The formula for KL divergence between two probability distributions $P(x)$ and $Q(x)$ over the same probability space is given by:

$$D_{\mathrm{KL}}(P||Q) = \sum_{x \in \mathrm{Support}(P)} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

In the case of continuous distributions, the summation is replaced by integration:

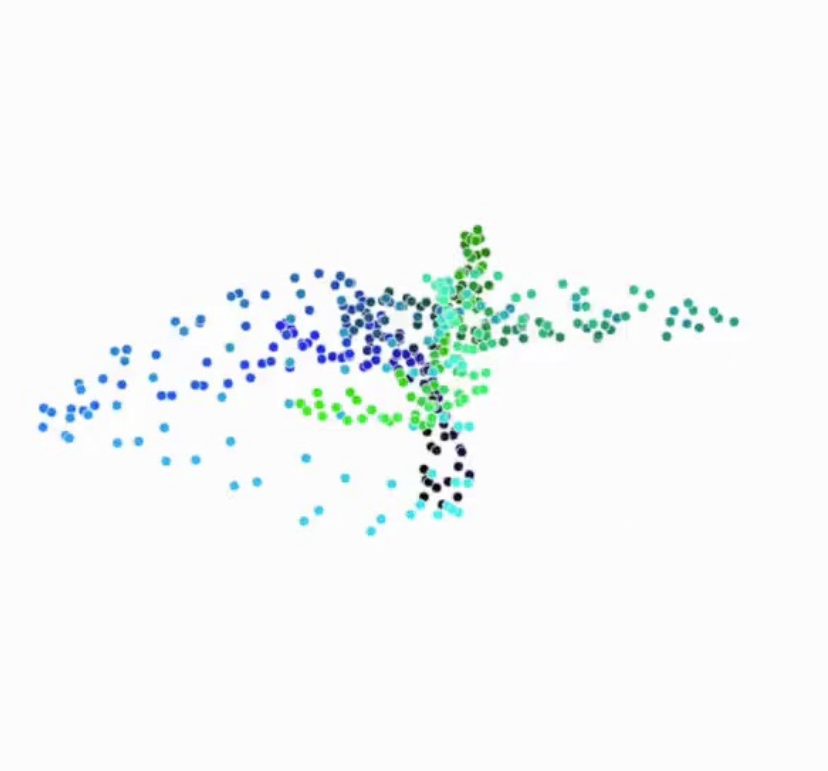
$$D_{\mathrm{KL}}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Where:

- $D_{\mathrm{KL}}(P||Q)$ represents the KL divergence from distribution Q to distribution P.
- $P(x)$ and $Q(x)$ are the probability density functions (PDFs) of distributions P and Q, respectively.
- $\mathrm{Support}(P)$ denotes the support of the probability distribution $P(x)$, which is the set of all possible values of x where $P(x) > 0$.
- $p(x)$ and $q(x)$ are probability density functions in the case of continuous distributions.

The KL divergence is always non-negative, with $D_{\mathrm{KL}}(P||Q) = 0$ if and only if $P(x) = Q(x)$ for all x in the support of P. It measures the extra amount of information needed to encode samples from one distribution using a code optimized for another distribution.

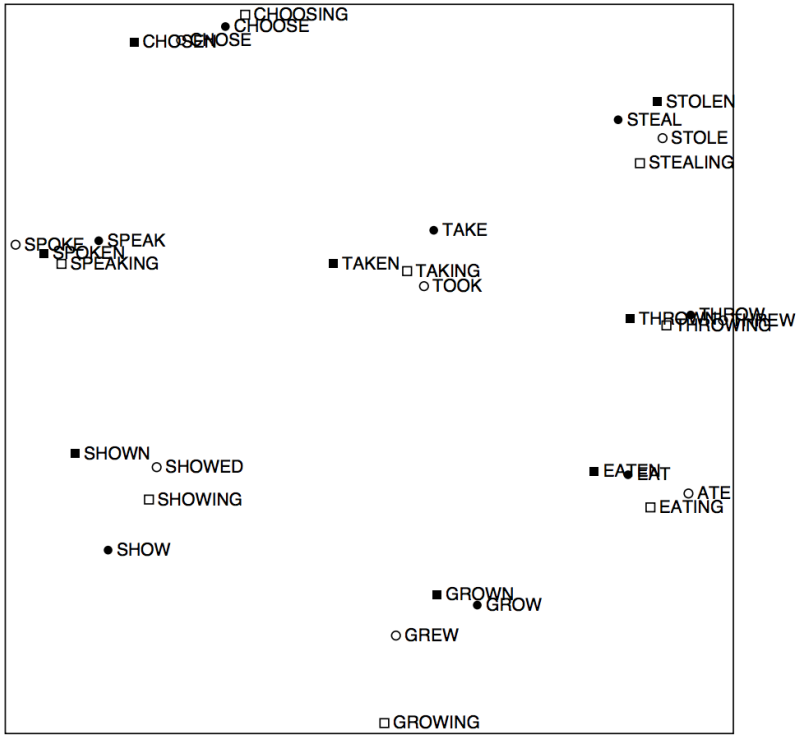
t-SNE



```
tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
new_values = tsne_model.fit_transform(tokens)
```

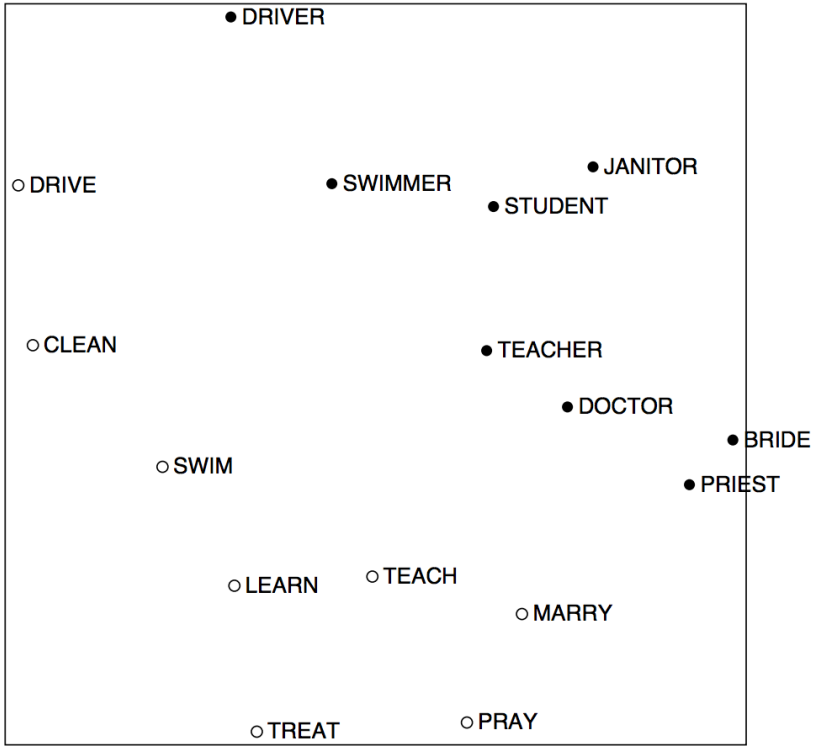
Interesting patterns emerge in word vectors

- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)



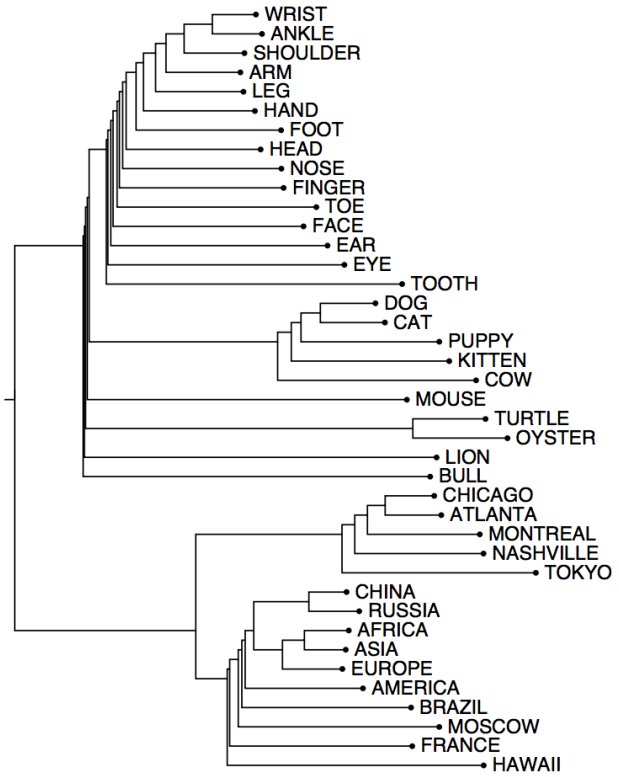
Interesting patterns emerge in word vectors

- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)



Interesting patterns emerge in word vectors

- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)



Problem with SVD

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ (when n > m)

  → Bad when dealing with millions of words / documents

- Hard to incorporate new words or documents
- Different learning regime than other DL models

  ➔ Let's directly learn low-dimensional word vectors!

Word2vec

AN EFFICIENT EMBEDDING MODEL

Word representations

Traditional Method - Bag of Words Model	Word Embeddings
• **one hot** encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . 0 0 0 0 • Context information is not utilized	• Each word is a **point in *n*-dimensional space**, represented by a **vector of length *n***, (*n* ~ 100 - 300) • Trained by using big text dataset • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Word embeddings

- First usage: 2003 (Bengio et al.):

  Proposed this new model for distributed representations of words using neural networks.

- 2008, Collobert and Weston made ‘Word Embeddings’ approach mainstream by proposing ‘A Unified Architecture for Natural Language Processing’.

- 2013, Mikolov published a paper called Word2Vec which basically popularized the use of word embedding and especially pre-trained word embeddings in many NLP tasks.

Word2vec

- Instead of capturing co-occurrence counts directly

==> Predict surrounding words of every word

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

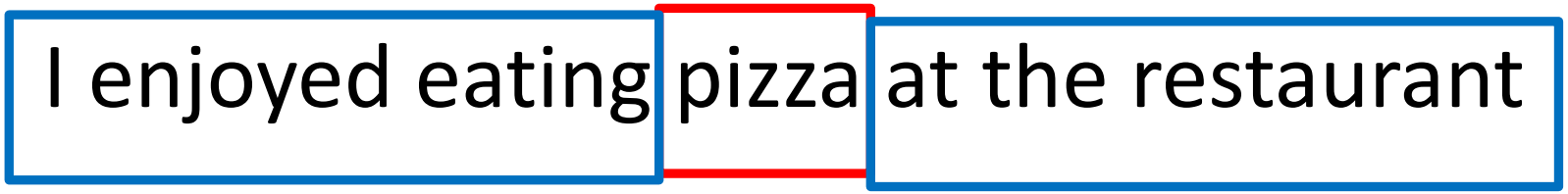
Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Word2vec

- Training objective:

  Maximize the likelihood of the **context**, given the **focus word**

  - P (I | pizza)
  - P (enjoyed | pizza)
  - P (restaurant | pizza)



Example

- P (I | pizza)
- P (enjoyed | pizza)
- P (eating | pizza)
- P (at| pizza)
- P (the| pizza)
- P (restaurant| pizza)

I enjoyed eating pizza at the restaurant

Example

- Move to the next word and repeat
- P (I | at)
- P (enjoyed | at)
- P (eating | at)
- P (pizza| at)
- P (the| at)
- P (restaurant| at)

I enjoyed eating pizza at the restaurant

Example

- P (eating | pizza

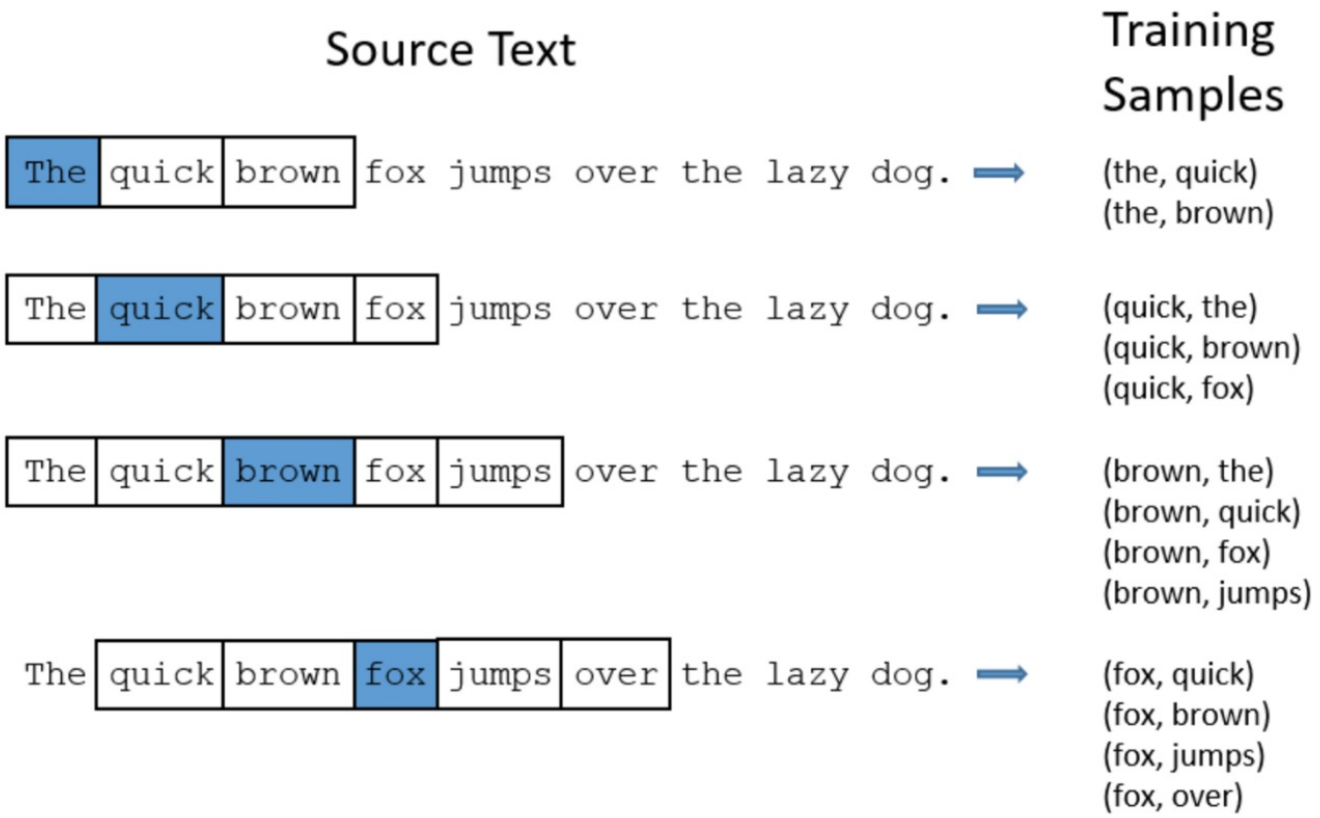
  Output word - **Input word**

- P (v_{out} | v_{in})

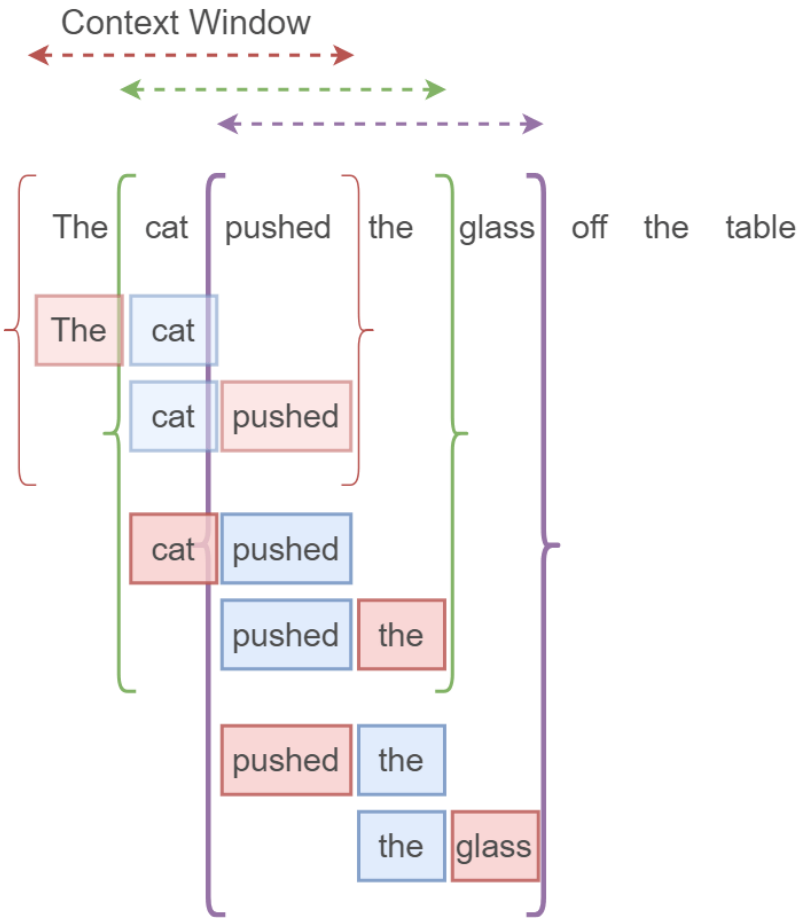
  Calculated through the input of training samples (in, out)

I enjoyed eating pizza at the restaurant

Skip-gram – another example

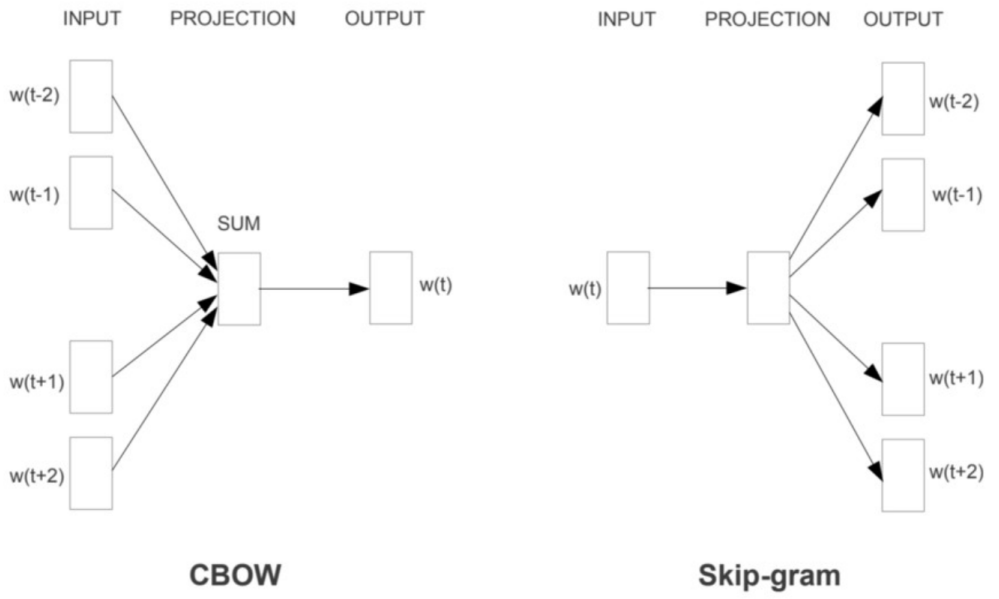


Context window



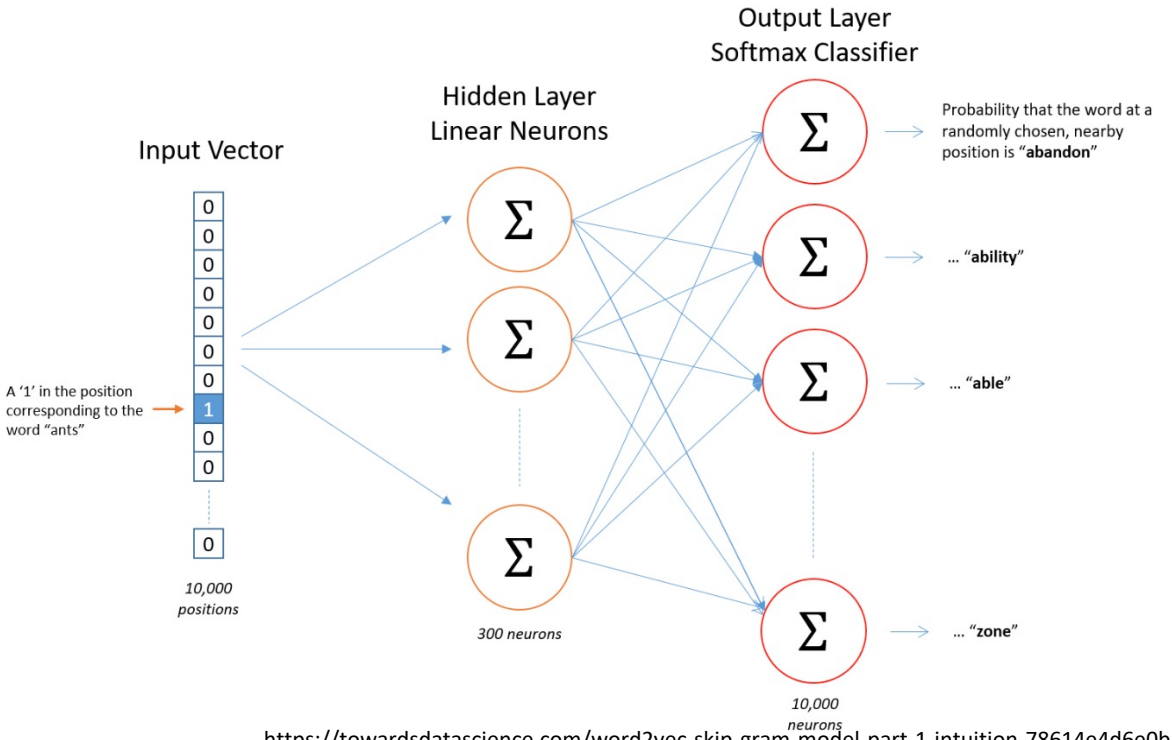
Word2vec

- 2 architecture options (both 1 layer): CBOW and Skip-gram (the latter more popular)
  - Skip-grams(SG) Predicting surrounding context words given a center word.
  - Continuous Bag of Words (CBOW) Predicting a center word form the surrounding context.



Skip-gram

- Feed-forward neural network – 1 hidden layer



https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

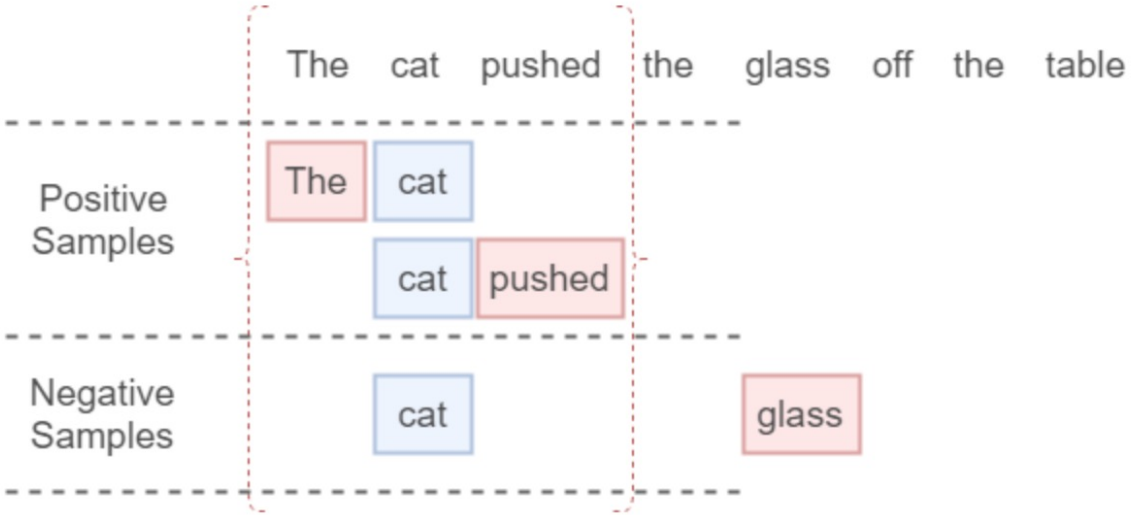
Word2vec

- To train this neural network (i.e., set the weights), we need:
  - Decide window size – dependent on your dataset size / quality
  - Decide the embedding size (i.e., number of nodes in the network)
  - Size of the vocabulary: typically words that occur only once or words like ‘the’ are omitted.

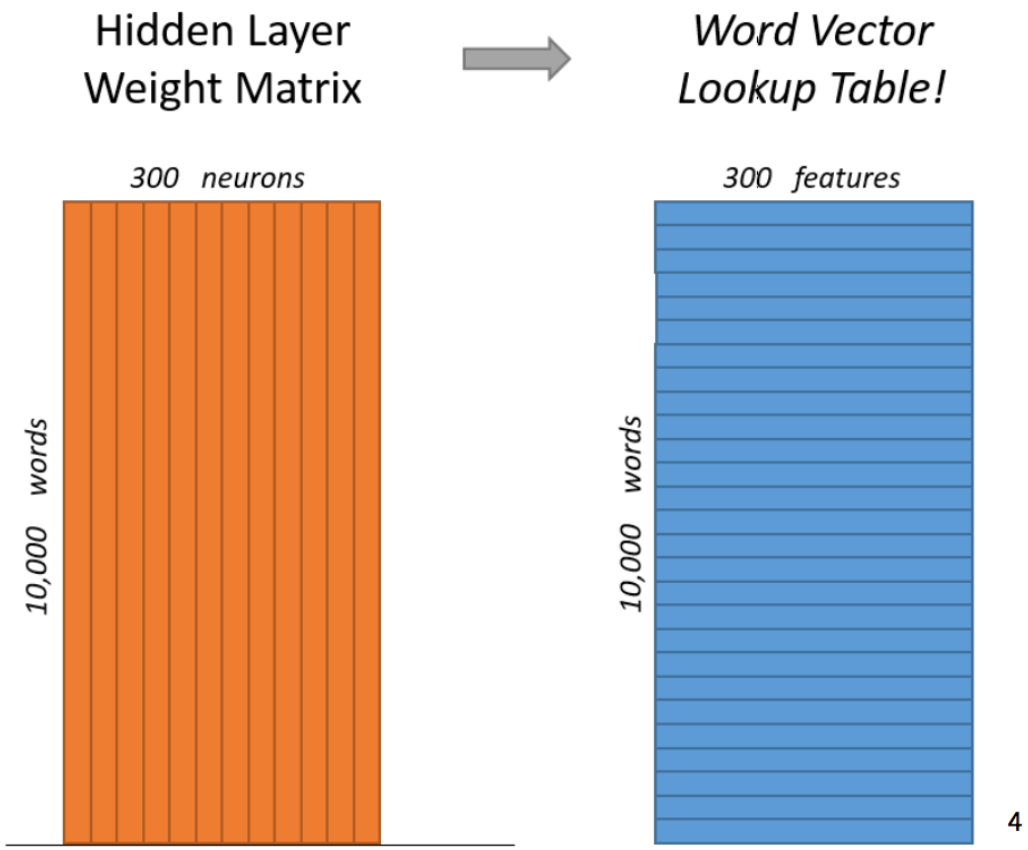
  - Training objective? Softmax cross-entropy? → hard to compute
  - Better solution: negative sampling

Negative sampling as objective

- The original training objective is thus approximated by a new, more efficient, formulation that implements a binary logistic regression to classify between data and noise samples.



Resulting embeddings



[4] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Results

- Fast and accurate predictive model
- Captures semantic content
- Very popular
- Pretrained models available
- Super fast to train
- Easy to use with Python Gensim library

```
model = gensim.models.Word2Vec (documents, size=150, window=5, min_count=2, workers=4)
model.train(documents,total_examples=len(documents),epochs=10)
```

GloVe

AN EFFICIENT COUNT-BASED MODEL

GloVe

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Abstract

Recent methods for learning vector space representations of words have succeeded in capturing fine-grained semantic and syntactic regularities using vector arithmetic, but the origin of these regularities has remained opaque. We analyze and

the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation *king* − *queen* = *man* − *woman*. This evaluation scheme

GloVe – Global Vectors (for word representation)

- **Combines** elements from the two main word embedding models which existed when GloVe was proposed:
  - global matrix factorization and local context window methods.
- *Context window-based methods* suffer from the disadvantage of not learning from the global corpus statistics. → repetition and large-scale patterns may not be learned as well with these models as they are with global matrix factorization.
- Performing *matrix factorization* gives us a low rank approximation of the whole of the data contained in the original matrix.
- Instead of learning the raw co-occurrence probabilities
  → **learn ratios of these co-occurrence probabilities**

GloVe – word frequencies

- Suppose we wish to study the relationship between two words, ***i = ice*** and ***j = steam.*** We'll do this by examining the co-occurrence probabilities of these words with various "probe" words. We define the co-occurrence probability of an arbitrary word ***i*** with an arbitrary word ***j*** to be the probability that word ***j*** appears in the context of word ***i***. This is represented by:

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\sum_k X_{ik}},$$

$$X_{ij} = \text{number of times word } j \text{ occurs in the context of word } i.$$

GloVe – word frequencies

- Since we are trying to determine information about the relationship *between* the words ***ice*** and ***steam***, ***water*** doesn't give us a lot of useful information.
- For discriminative purposes, it doesn't give us a good idea of how "far apart" ***steam*** is from ***ice***, and the information that ***steam***, ***ice***, and ***water*** are all related is already captured in the discriminative information between ***ice*** and ***water***, and ***steam*** and ***water***.
- Words that don't help us distinguish between ***i*** and ***j*** are referred to as **noise points**, and it is the use of the *ratio* between co-occurrence probabilities helps filter out these noise points.

https://nlp.stanford.edu/pubs/glove.pdf

GloVe – word frequencies

- Words that don't help us distinguish between ***i*** and ***j*** are referred to as noise points, and it is the use of the ***ratio*** between co-occurrence probabilities helps filter out these noise points.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

https://nlp.stanford.edu/pubs/glove.pdf

GloVe

- The final function for the GloVe model is considerably more **complex**
  - **Least squares objective** J that directly aims to reduce the difference **between the dot product of the vectors** of two words and the **logarithm of their number of co-occurrences**:

$$J = \sum_{i,j=1}^{V} f(X_{ij})\,(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

  - w_i and b_i are the word vector and bias respectively of word I
  - $\tilde{w}_j$ and b_j are the context word vector and bias respectively of word j
  - Xij is the number of times word i occurs in the context of word j
  - f is a **weighting function** that assigns relatively lower weight to rare and frequent co-occurrences.

=> Gradient descent or or specialized solvers for large-scale linear systems.

- Directly optimizes the objective function.

Conclusion:

- fast training
- Scalable to huge corpora, good performance
- Easier to parallelize than word2vec, but comparable accuracy
- Ratio of co-occurrence probabilities can encode meaning

Evaluation of word vectors

1. Intrinsic evaluation: it encodes semantic information, e.g. similarity
2. Extrinsic evaluation: is it useful for other NLP tasks

Word similarity task

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10.00
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

```
# similarity between two similar words
model.wv.similarity(w1="pretty",w2="beautiful")
```

```
0.58568496
```

Word similarity task

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

Results from : GloVe: Global Vectors for Word Representation, Pennington et al 2014.

GloVe – word similarity

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

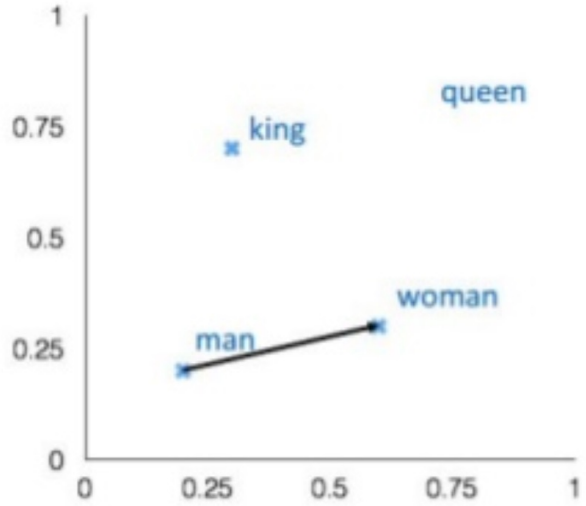
eleutherodactylus

Word analogy task

$$x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$$

man:woman :: king:?

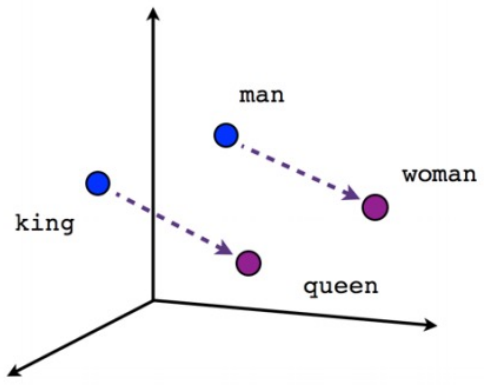
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



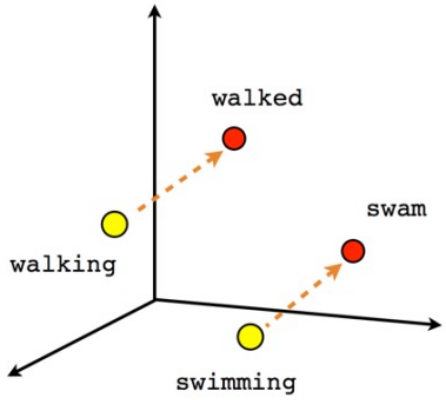
word2vec

Analogies

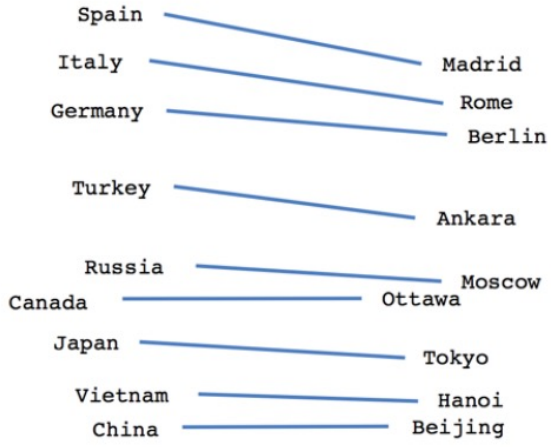
- vector[Queen] = vector[King] - vector[Man] + vector[Woman]



Male-Female

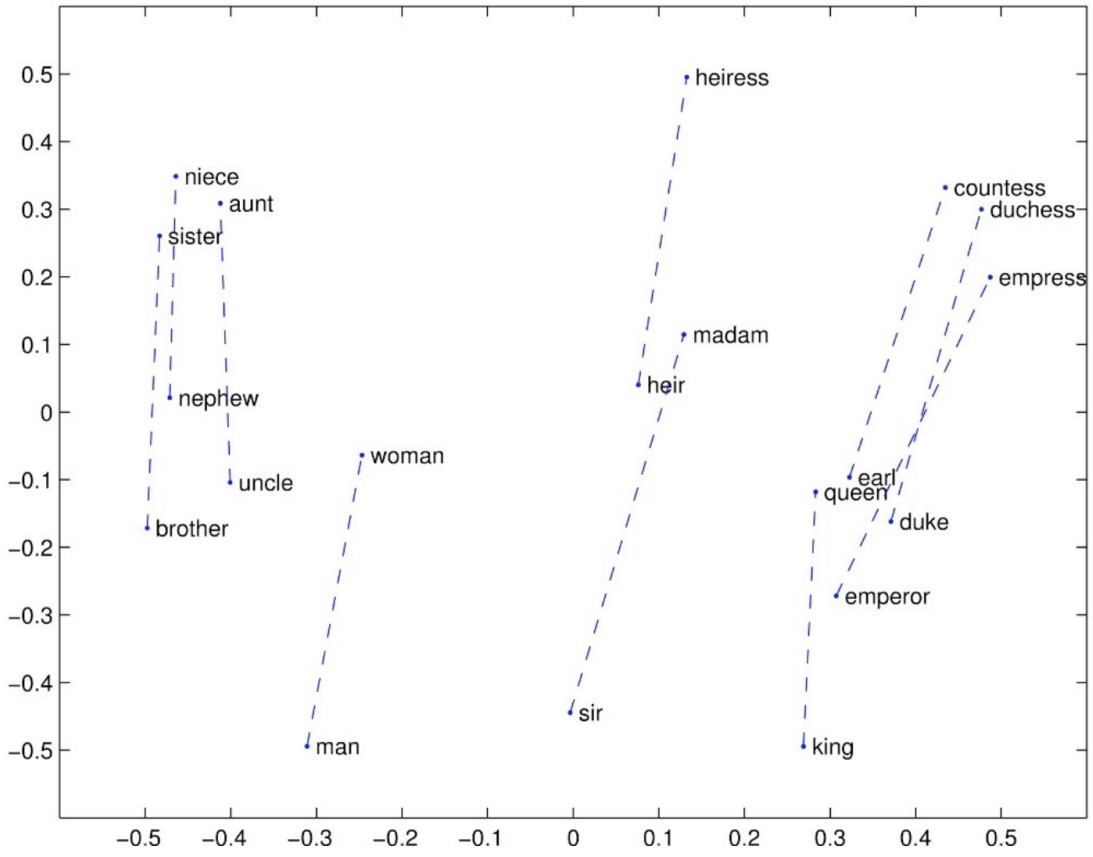


Verb tense

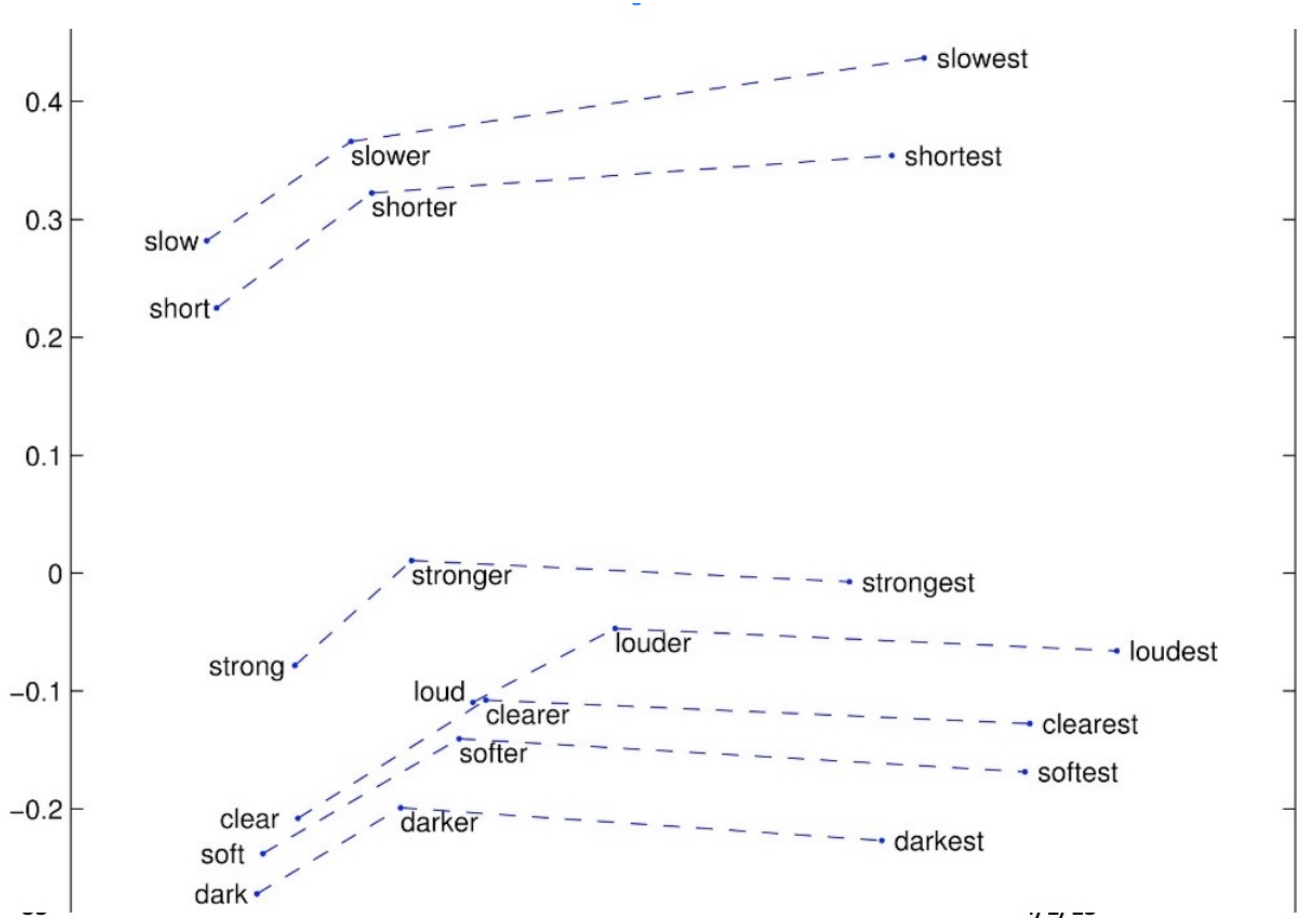


Country-Capital

Analogies - GloVe



GloVe - superlatives



Extrinsic evaluation

Part of Speech Tagging :

input :	Word	Embeddings	are	cool
output:	Noun	Noun	Verb	Adjective

Named Entity recognition :

input :	Nous	sommes	charlie	hebdo
output:	Out	Out	Person	Person

Applications for data science

- Input to other, hybrid, models (lab 10b)
- Sentiment analysis, example: sentiment analysis

Sentiment analysis

☺☹

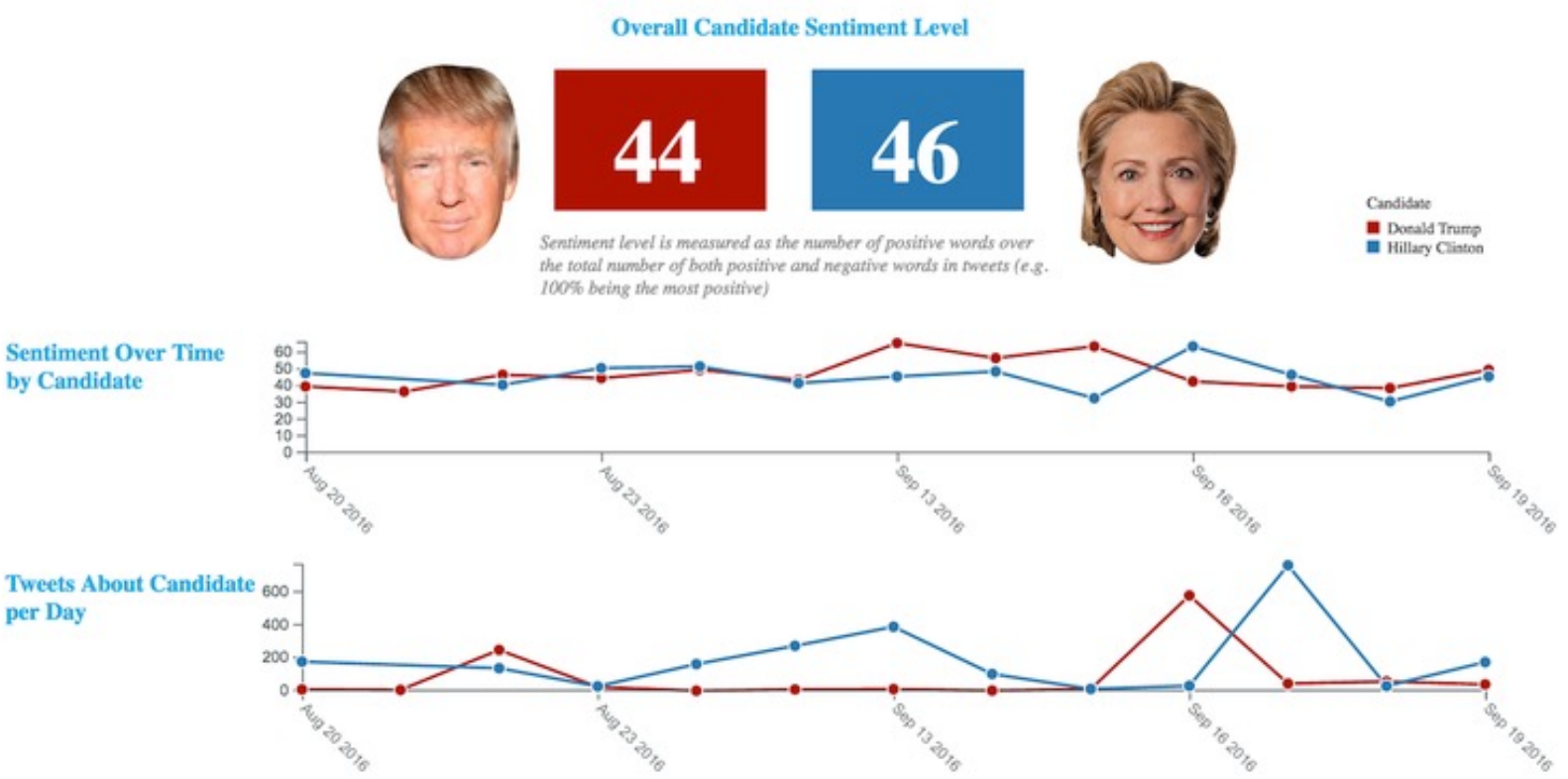
Sentiment analysis

- Sentiments are feelings, opinions, emotions, likes/dislikes, good/bad
- Sentiment analysis -> NLP and information extraction tasks that aims to obtain the writer's feelings expressed in text.
- Also known as opinion mining
- Useful for stock market prediction, etc.

Types of sentiment

- It's a great movie (positive statement)
- I did not like this (negative statement)
- I'm going for a walk (neutral statement)

Example


Overall Candidate Sentiment Level
44
46
Sentiment level is measured as the number of positive words over the total number of both positive and negative words in tweets (e.g. 100% being the most positive)
Candidate
Donald Trump
Hillary Clinton
Sentiment Over Time by Candidate
60
50
40
30
20
10
0
Aug 20 2016
Aug 23 2016
Sep 13 2016
Sep 16 2016
Sep 19 2016
Tweets About Candidate per Day
600
400
200
0
Aug 20 2016
Aug 23 2016
Sep 13 2016
Sep 16 2016
Sep 19 2016

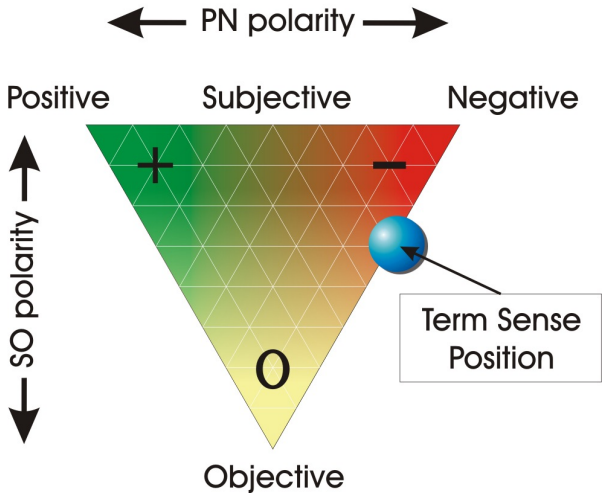
Other examples

- Businesses & organizations:
  - Brand analysis
  - New product perception
  - Produce and service benchmarking
- Individuals:
  - When purchasing a product
  - When finding opinions on political topics, movies, etc.
- Social Media:
  - General opinion about recent hot topics in town?
- Ad placements
  - Place an add when you praise a product
  - Place a competitor product add when you criticize a product

Approaches

- Traditional:
  - Use semantics to understand the language
  - SentiWordNet: extension of WordNet that adds 3 polarities
  - Based on grammar, synonyms, etc.
- Word Embedding based

→ Let's explore the latter!



Sentiment prediction

- In essence: classification problem.
  - Input: Short texts (e.g. tweets) with sentiment label (could be smiley!)
  - Classes: Positive – Neutral – Negative
- We can use any classification algorithm, but we should be clever about the input:
  - One-hot-encoding?
  - Bag of words?
  - Word vectors?

Word vectors for text classification

1. Preprocessing: remove URLs, non-English words, weird characters, code tags, etc.
2. Tokenize your tweet, i.e., split text into tokens such as words, punctuation, and numbers.
3. Now our input is a list of words per tweet. We can get the embeddings for the words. But…. We have multiple words for each tweet, how do we feed this in the classifier? Variable length input?

   Possible solution: aggregation, e.g. average the vectors.

4. This can be solved with simple classification model
   → problem: does not work super well…

Tokenizing and connecting models in Keras

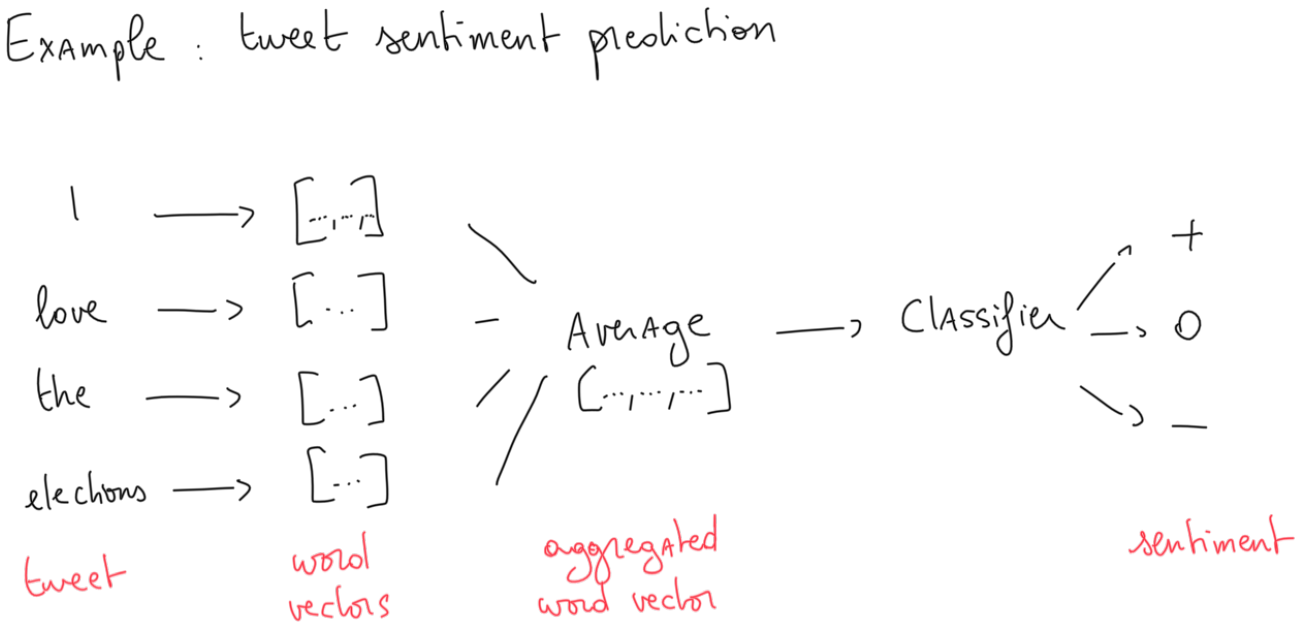
```
import nltk.data
nltk.download('punkt')

def w2v_tokenize_text(text):
    # create tokens, a list of words, for each post. This function will do some cleaning based on
English language
    tokens = []
    for sent in nltk.sent_tokenize(text, language='english'):
        for word in nltk.word_tokenize(sent, language='english'):
            if len(word) < 2:
                continue
            tokens.append(word)
    return tokens
```

```
train, test = train_test_split(df, test_size=0.3, random_state = 42)

test_tokenized = test.apply(lambda r: w2v_tokenize_text(r['post']), axis=1).values
train_tokenized = train.apply(lambda r: w2v_tokenize_text(r['post']), axis=1).values
```

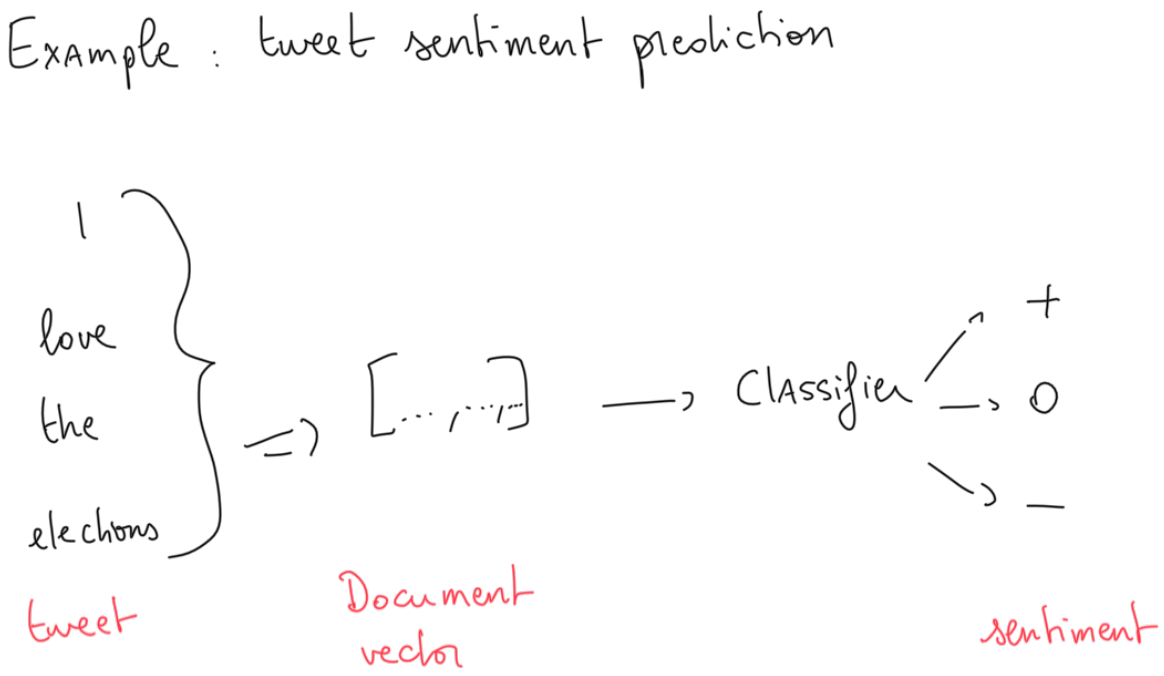
Word vectors for text classification



→ problem: does not work super well…

```
# use gensim's method to calculate the mean of all the words appended to mean list
mean = gensim.matutils.unitvec(np.array(mean).mean(axis=0)).astype(np.float32)
return mean
```

Better solution



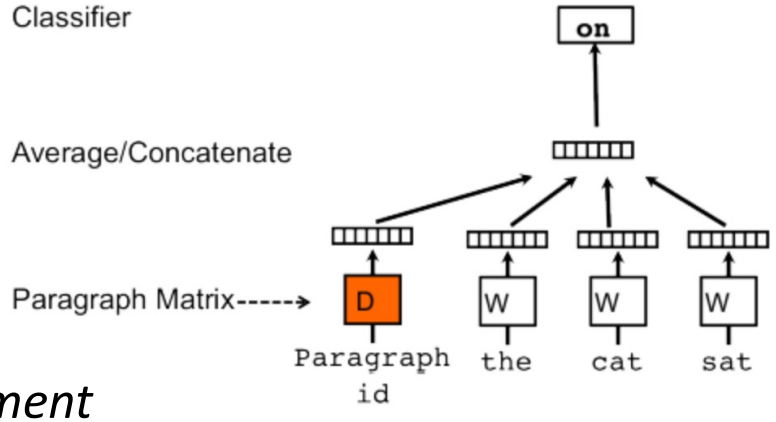
→ Doc2vec

Doc2vec

- Generalizes word2vec to whole documents (phrases, sentences, tweets, etc.) using:

  -> *Distributed Memory version of Paragraph Vector* (PV-DM)

- Provides fixed-length vector
- Use the word2vec model, and add another vector (Paragraph ID below)
- We treat the paragraph as an extra word. Then it is concatenated/averaged with local context word vectors when making predictions.



-> Result: one vector for your document

Doc2vec in Keras

```
def label_sentences(corpus, label_type):
    """
    Gensim's Doc2Vec implementation requires each document/paragraph to have a label associated wi
th it.
    We do this by using the TaggedDocument method. The format will be "TRAIN_i" or "TEST_i" where
"i" is
    a dummy index of the post.
    """
    labeled = []
    for i, v in enumerate(corpus):
        label = label_type + '_' + str(i)
        labeled.append(doc2vec.TaggedDocument(v.split(), [label]))
    return labeled
```

```
X_train, X_test, y_train, y_test = train_test_split(df.post, df.tags, random_state=0, test_size=0.
3)
X_train = label_sentences(X_train, 'Train')
X_test = label_sentences(X_test, 'Test')
all_data = X_train + X_test
```

Doc2vec in Keras

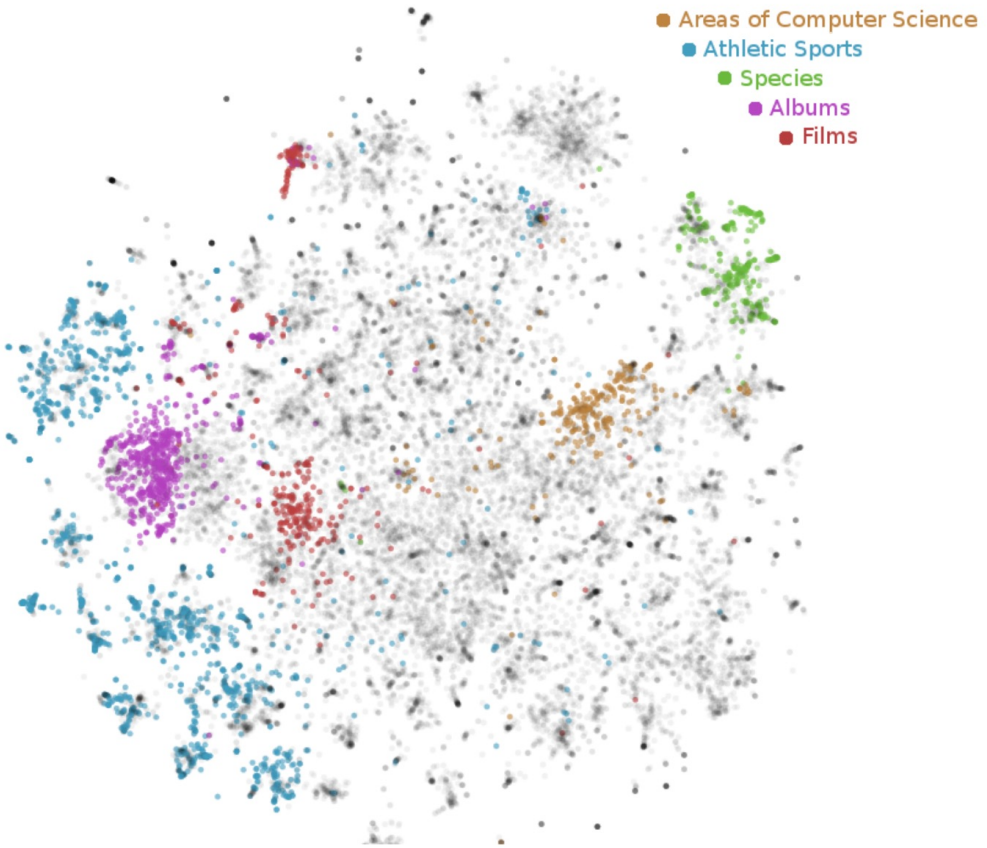
```
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, min_count=1, alpha=0.065, min_alpha=0.065)
model_dbow.build_vocab([x for x in tqdm(all_data)])
```

```
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(all_data)]), total_examples=len(all_data), epo
chs=1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

```
train_vectors_dbow = get_vectors(model_dbow, len(X_train), 300, 'Train')
test_vectors_dbow = get_vectors(model_dbow, len(X_test), 300, 'Test')
```

Doc2vec

- Wikipedia pages grouped per area



(Andrew et al, 2015)

Doc2vec

- Nearest neighbors to Lady Gaga

Article	Cosine Similarity
Christina Aguilera	0.674
Beyonce	0.645
Madonna (entertainer)	0.643
Artpop	0.640
Britney Spears	0.640
Cyndi Lauper	0.632
Rihanna	0.631
Pink (singer)	0.628
Born This Way	0.627
The Monster Ball Tour	0.620

(Andrew et al, 2015)

Doc2vec

- Faster and consumes less memory then word2vec
- Typically obtains better results.
- We will explore this in the lab.

Sentiment prediction libraries

- This was an example how you can do text-classification
- The specific topic of sentiment analysis has been extensively researched and tools are available already:
  - NLTK: natural language processing toolkit
  - Google Cloud NL API
  - Vader
  - …

Pretrained word embeddings ready for use

- Word2vec: https://code.google.com/archive/p/word2vec/
- Glove: https://nlp.stanford.edu/projects/glove/
- Dependency (based on skip-gram): https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/

  => Don't reinvent the wheel ☺

```
from gensim.models import Word2Vec

wv = gensim.models.KeyedVectors.load_word2vec_format("data/GoogleNews-vectors-negative300.bin.gz",
binary=True)
wv.init_sims(replace=True)
print('Model loaded')
```

Word2vec for music

BONUS APPLICATION

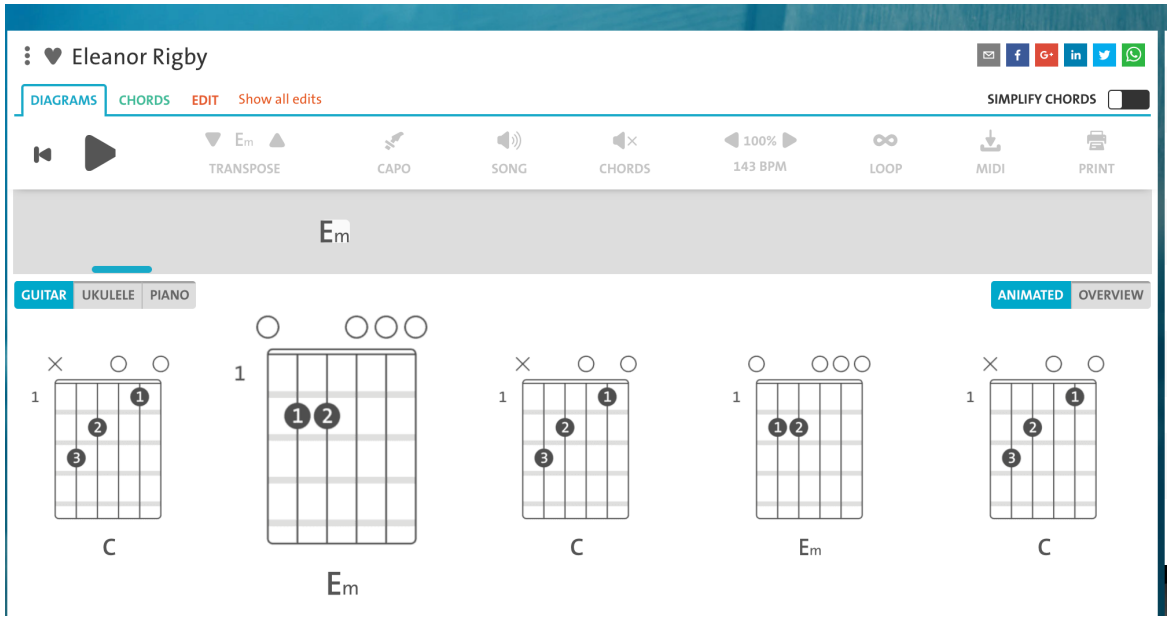
Similarity between language and music

Besson and Schön (2001) :

- Common ancestor of music and language: expression of **emotive meaning**
  - e.g. emotional excitement: fast, accelerating, and high-pitched sound patterns
- **Sequential elements** that unfold in time:
  - Rhythm
  - Temporal ratios (notes/phonemes, chords/words)
- **Grammar**: musical grammar can include contour, cadence for closing, etc (more flexible)
- Ability to generate strong **expectancies**: both unexpected (incongruent) words and notes generate peaks measurable in brain potentials (N400 & P600)

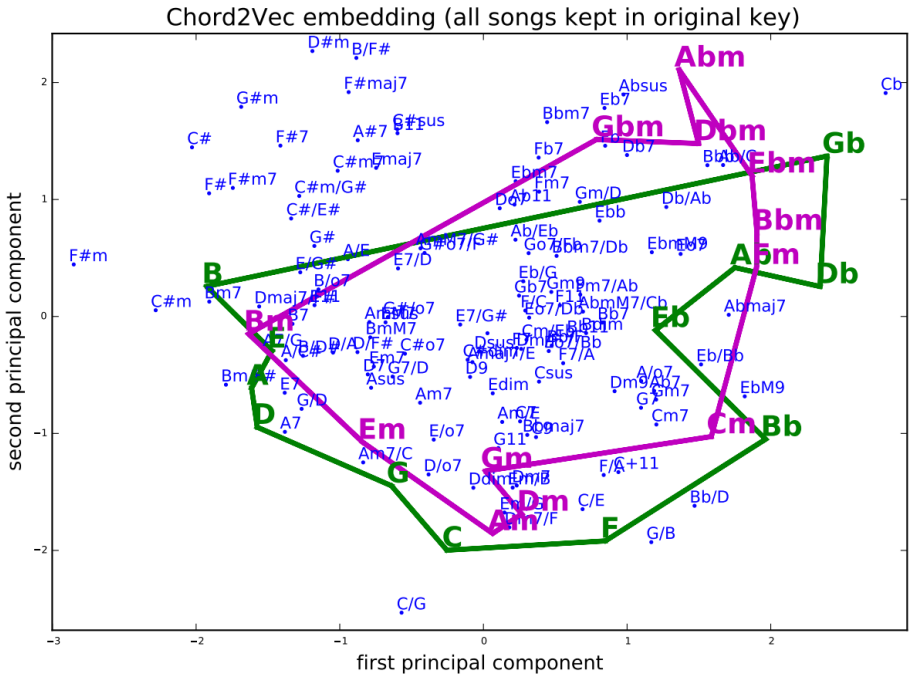
Multidisciplinary

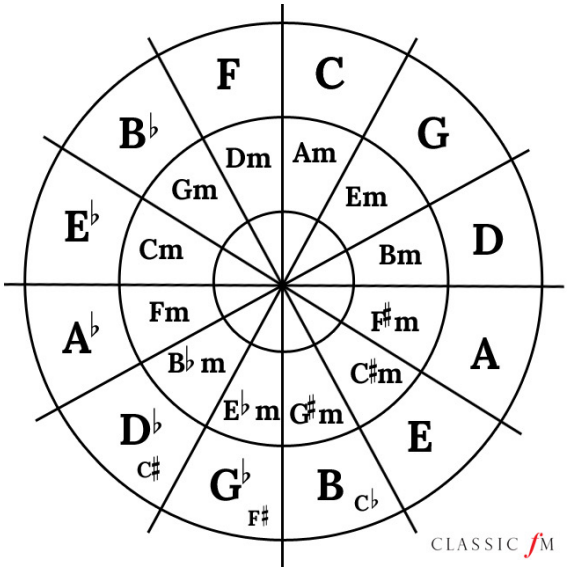
- Word2vec models can be used for other things than just text.
- Chords? (Huang et al., 2016)
- Polyphonic music? (Herremans & Chuan, 2017)



Chords

- t-SNE projection: Huang et al. (2016)'s research discovers circle-of-fifths





Word2vec for polyphonic music

- Skip-gram approach
- Complex polyphonic music:
  - Musical slices ~ words
- Dataset:
  - 130,000 pieces
  - 8 genres

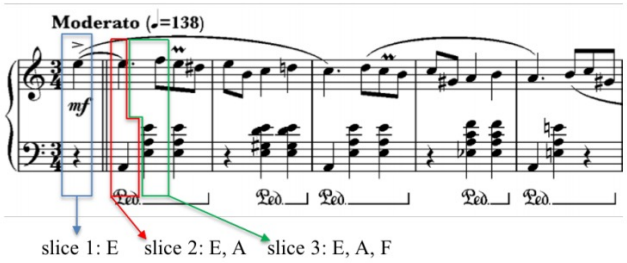


Fig. 2: Representing polyphonic music as a sequence of slices.

From Context to Concept: Exploring Semantic Relationships in Music with Word2Vec

Ching-Hua Chuan · Kat Agres · Dorien Herremans

Received: 22/06/2018 / Accepted: 26/11/2018

Abstract We explore the potential of a popular distributional semantics vector space model, *word2vec*, for capturing meaningful relationships in ecological (complex polyphonic) music. More precisely, the skip-gram version of word2vec is used to model slices of music from a large corpus spanning eight musical genres. In this newly learned vector space, a metric based on cosine distance is able to distinguish between functional chord relationships, as well as harmonic associations in the music. Evidence, based on cosine distance between chord-pair vectors, suggests that an implicit circle of fifths exists in the vector space. In addition, a comparison between

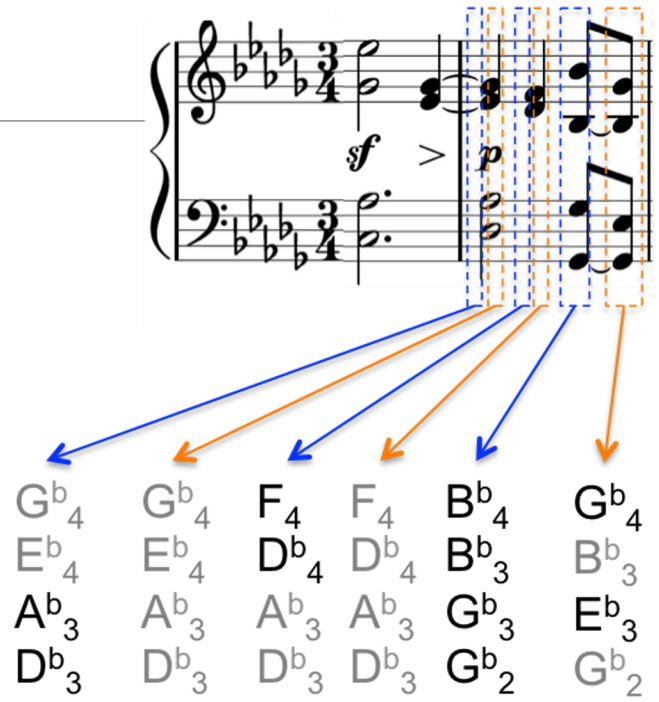
Chuan C.-H., Agres K., Herremans D.. 2018. From Context to Concept: Exploring Semantic Relationships in Music with Word2Vec. Neural Computing and Applications. https://arxiv.org/abs/1811.12408

six unique slices
(unit = 8th note)

Music as words

Segmenting polyphonic music into slices in a generic way

- **Time unit** of the slice is selected as the smallest duration between adjacent onsets that occur more than 5% of the duration distribution.
- **All notes** including chord tones, non-chord tones, ornaments are all recorded.
- Only the **global key** is used to transpose the piece into C major or A minor.



transpose to C major

F4 D4 G3 C3
F4 D4 G3 C3
E4 C4 G3 C3
E4 C4 G3 C3
A4 A3 F3 F2
G4 A3 D3 F2

Cosine similarity

- Similarity metric for two non-zero vectors A and B in the n-dimensional vector space, with angle θ:

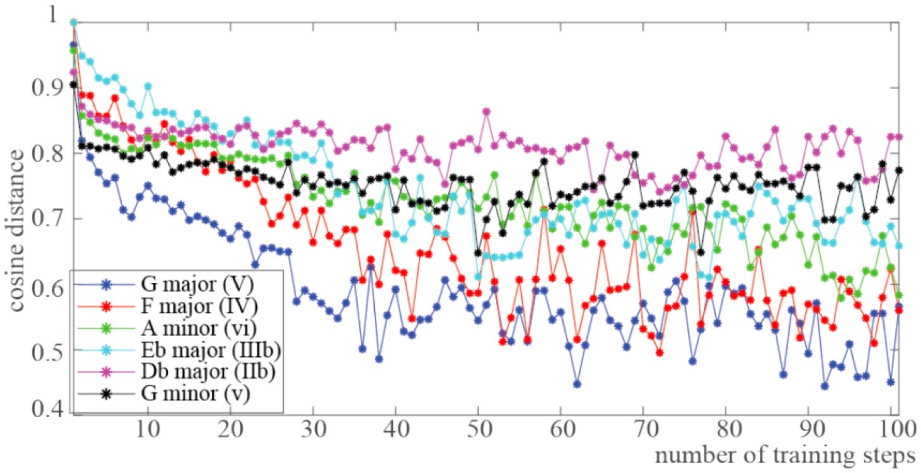
$$\text{Similarity}(\text{A}, \text{B}) = cos(\theta) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

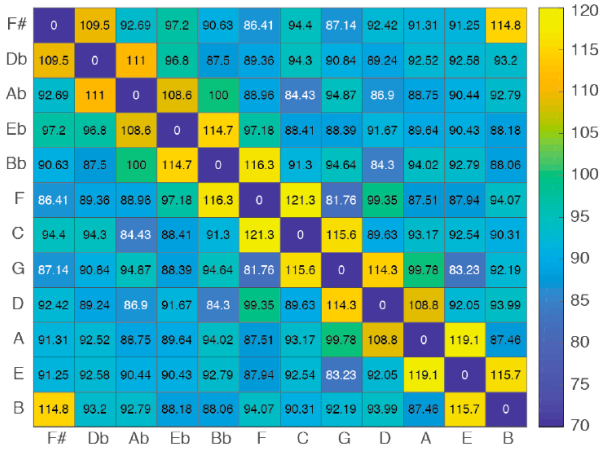
=> Distance matrix: allows us to easily identify words that occur in the same context

Word2vec for polyphonic music

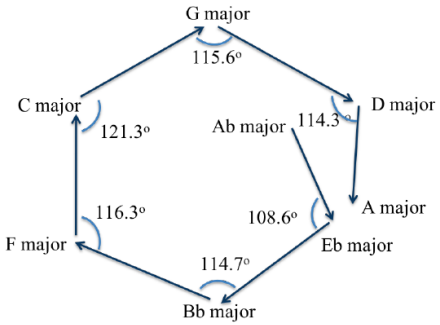
- Without knowing what is in which slice:
  - Tonally close chords have low distance
  - I-V chord-vector angle reflects
    circle-of-fifths between keys

Promising approach to model music!





(a)



(b)

Natural Language Processing Methods for Symbolic Music Generation and Information Retrieval: a Survey

DINH-VIET-TOAN LE*, Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
LOUIS BIGO, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France
MIKAELA KELLER, Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
DORIEN HERREMANS, Singapore University of Technology and Design, Singapore

Abstract – Several adaptations of Transformers models have been developed in various domains since its breakthrough in Natural Language Processing (NLP). This trend has spread into the field of Music Information Retrieval (MIR), including studies processing music data. However, the practice of leveraging NLP tools for symbolic music data is not novel in MIR. Music has been frequently compared to language, as they share several similarities, including sequential representations of text and music. These analogies are also reflected through similar tasks in MIR and NLP.

This survey reviews NLP methods applied to symbolic music generation and information retrieval studies following two axes. We first propose an overview of representations of symbolic music adapted from natural language sequential representations. Such representations are designed by considering the specificities of symbolic music. These representations are then processed by models. Such models, possibly originally developed for text and adapted for symbolic music, are trained on various tasks. We describe these models, in particular deep learning models, through different prisms, highlighting music-specialized mechanisms. We finally present a discussion surrounding the effective use of NLP tools for symbolic music data. This includes technical issues regarding NLP methods and fundamental differences between text and music, which may open several doors for further research into more effectively adapting NLP tools to symbolic MIR.

https://arxiv.org/pdf/2402.17467.pdf

Back to language

ELMo (2018)

- **E**mbeddings from **L**anguage **Mo**del
- ELMo: deep contextualised word representation (Peeters et al., 2018)
- BiDirectional LSTM
- "Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding."

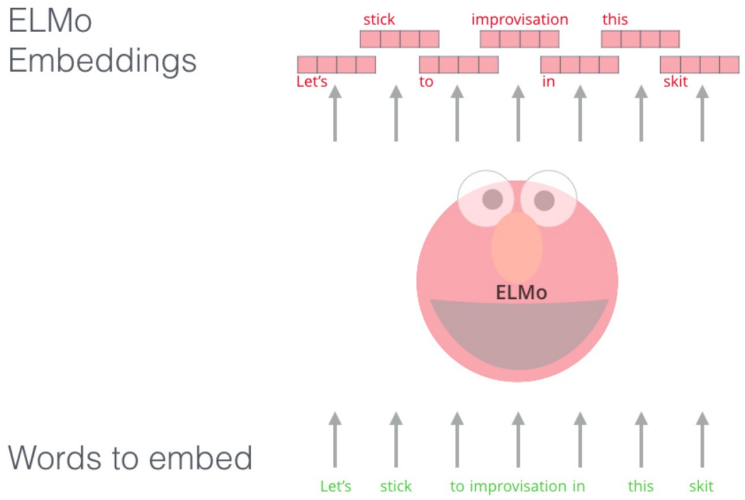


Image from http://jalammar.github.io/illustrated-bert/

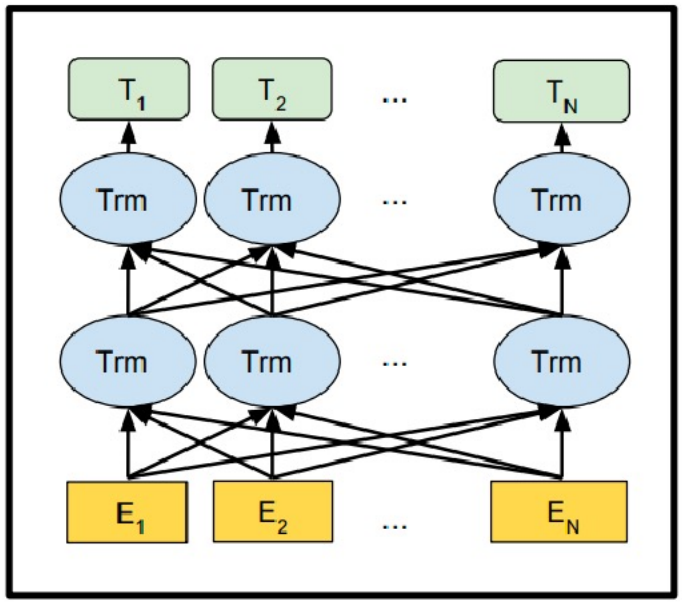
ELMO

1. **Contextual Embeddings**: ELMo captures the meaning of words in context by considering the surrounding words in a sentence or text sequence. This allows it to generate different embeddings for the same word depending on its context, enabling better representation of polysemy and capturing nuances in meaning.
2. **Deep Bidirectional Architecture**: ELMo employs a deep bidirectional LSTM architecture, which enables it to model complex patterns and dependencies in language data. By processing input text in both forward and backward directions, ELMo captures long-range dependencies and syntactic structures effectively.
3. **Layered Representations**: ELMo produces word embeddings by combining representations from multiple layers of the bidirectional LSTM. Each layer captures different aspects of the input text, ranging from low-level features (e.g., character-level information) to high-level semantic features. By aggregating information from multiple layers, ELMo generates rich and informative embeddings.

BERT

- Leveraging huge unlabeled and high quality data: 7000 books + Wikipedia (together 3300M words) (Devlin et al., 2018)

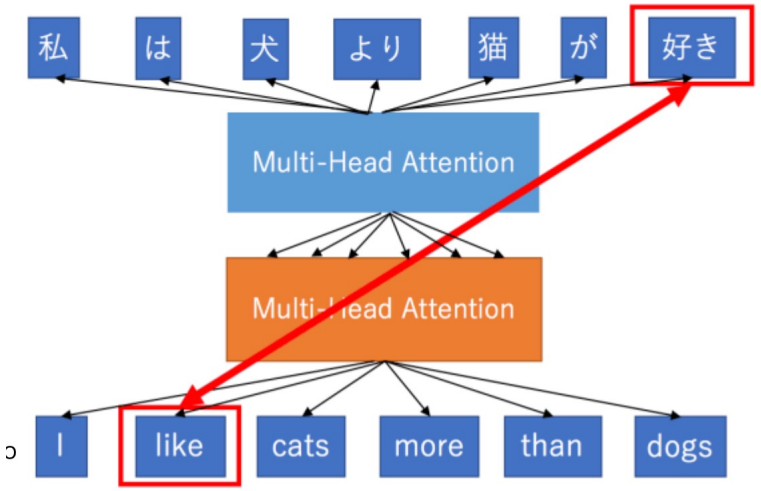
- BERT: Bi-directional **transformer**
- Pre-training BERT uses two unsupervised tasks, that are Masked LM and Next Sentence Prediction (NSP) to train.
- Multi head self-attention blocks in Transformer:
  - modelling the intra and extra word word relations
  - parallelable within instance and thus efficient



12-layers BERT model a token will have
12 intermediate embedding representations

BERT

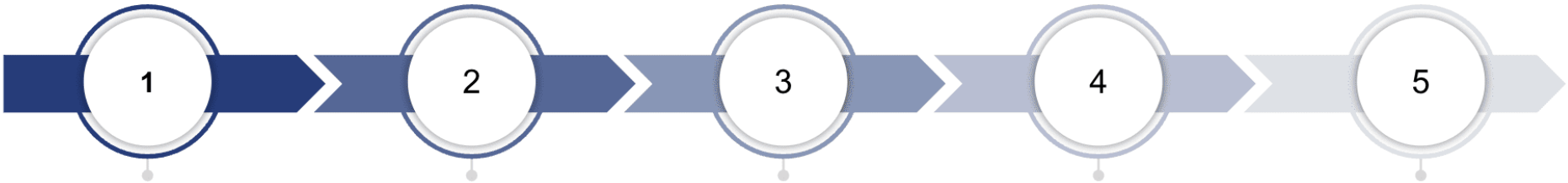
- Transformer/attention network learns which surrounding words are more/less important
- Developed for sequence to sequence (n to n) tasks:
  - Translating sentences
  - How similar are two sentence?
  - Recently also adopted for n to 1
- Sentence BERT



BERT Base	Original BERT model with 12 transformer layers and 110 million parameters.	2018
BERT Large	Larger version of BERT with 24 transformer layers and 340 million parameters.	2018
RoBERTa	Variant of BERT developed by Facebook AI, incorporating improvements such as dynamic masking and longer training duration.	2019
DistilBERT	A smaller and faster version of BERT developed by Hugging Face, retaining much of BERT's performance while reducing model size and computational requirements.	2019
ALBERT	Focuses on parameter reduction and sharing techniques to improve efficiency without sacrificing performance.	2019
T5	Text-To-Text Transfer Transformer developed by Google, capable of performing various NLP tasks using a unified framework where input and output are both in text format.	2019
XLNet	Uses a permutation language modeling objective to capture bidirectional context effectively.	2019
GPT (Generative Pre-trained Transformer)	Developed by OpenAI, uses an autoregressive language modeling objective and is trained to generate text sequentially.	2018

GPT

- Generative Pretrained Model
- Has embeddings too
- Large transformer (decoder)-based language model, with generative pre-training of a language model on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task.
- Key innovation in the model is that the structure of its response can be tuned by prompting
- GPT-4: multimodal (text-image inputs)



GPT-1

- Released: 2018
- 117 million parameters
- Trained using unsupervised learning
- Gave state of the art performance for LAMBADA.
- Gave Competitive performance for GLUE and SQuAD

GPT-2

- Released: 2019
- Improved version of GPT-1.(1.5 billion Params)
- Trained on larger dataset.
- Additional improvements like including a modified training objective, and a more efficient sampling algorithm for generating text.

GPT-3

- Released: 2020
- Improved version of GPT-2 (175 billion Params)
- Trained on larger dataset.
- Additional improvements like better training methodology (Gshard) and Few shot learning Capability.

GPT-3.5

- Released:2022
- Improved version of GPT-3
- Three variants , 1.3 B, 6B and 175 B params
- The main element is to eliminate toxic outputs to certain extent using RLHF (reinforcement learning with human feedback).

GPT-4

- Released: 2023
- Not sure if its a fine tuned version of previous model.
- 1 trillion Parameters
- It can take text and image as inputs and produces text outputs.
- RLHF is used in the training.

Read more on GPT

- https://medium.com/@gauravghati/comparison-between-bert-gpt-2-and-elmo-9ad140cd1cda
- https://www.eastagile.com/blogs/generative-pre-training-gpt-2-vs-gpt-3
- https://beebom.com/openai-gpt-4-new-features-image-input-how-to-use/
- https://keras.io/examples/generative/text_generation_gpt/

References

- https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-understanding-word2vec-e0128a460f0f
- https://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf
- https://towardsdatascience.com/mapping-word-embeddings-with-word2vec-99a799dc9695
- https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-understanding-word2vec-e0128a460f0f
- https://www.youtube.com/watch?v=lXomAQ0xWnk
- http://www.sfs.uni-tuebingen.de/~ddekok/dl4nlp/glove-presentation.pdf
- https://towardsdatascience.com/emnlp-what-is-glove-part-iii-c6090bed114