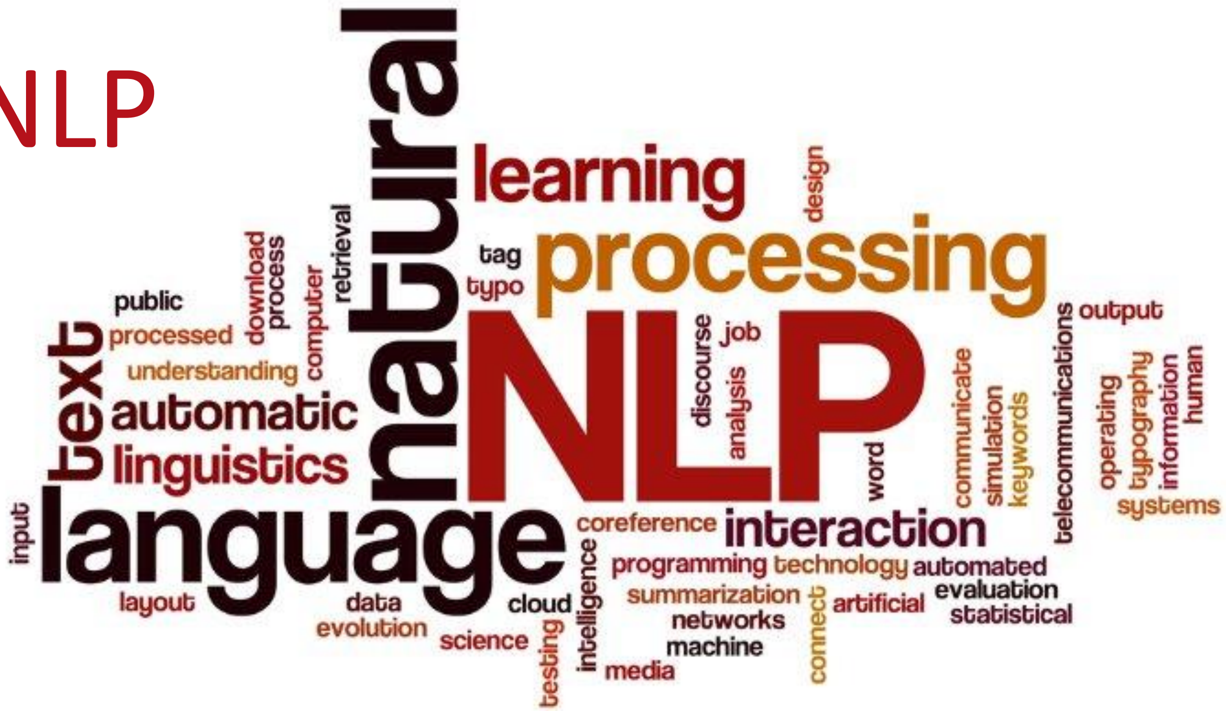
Text Vectorization in NLP

Dr. Ignatius Ezeani
Monday, 08 March 2021



Lecture Outline

This lecture is structured in three parts:

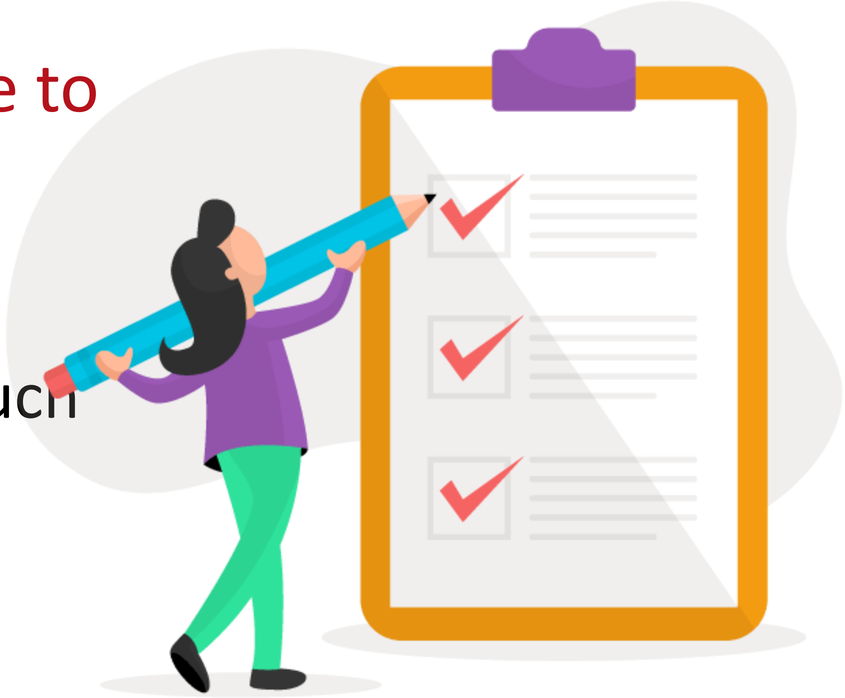
- **Part 1:**
  - Review of last Lecture + Lab
- **Part 2**
  - Overview of Text Preprocessing
  - Bag of Words (BoW) Models
  - TF.IDF
- **Part 3**
  - Neural Networks for Texts
  - Embedding Models

Intended Learning Outcomes

Lancaster University

At the end of this lecture, you should be able to

- perform basic pre-processing on text
- explain the concept of text vectorisation
- explain classical feature extraction methods such as BoW and TF.IDF
- discuss the fundamentals of word embedding



Summary of Week 18

In Week 18, we covered the following

- What is NLP and why do we need it?
- Why is NLP hard?
- Common Tasks in NLP
- Applying Machine Learning to NLP Tasks
- Basic Text Feature Extraction
- Lab Exercise: Simple Classification Tasks
    - Gender Identification and Movie Reviews

What is NLP and why do we need it?

- **What is NLP?**
  - The study of the computational processing of human language which creates systems that understand and generate human language
- **Why NLP?**
  - Zettabytes of human language data (~44Zb) (~44ZettaBytes) on the internet
  - Need for efficient tools and techniques to make sense of them as well as generate actionable outputs

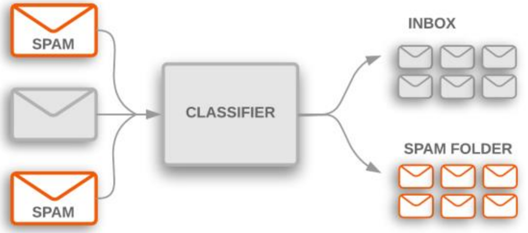
Why is NLP hard?

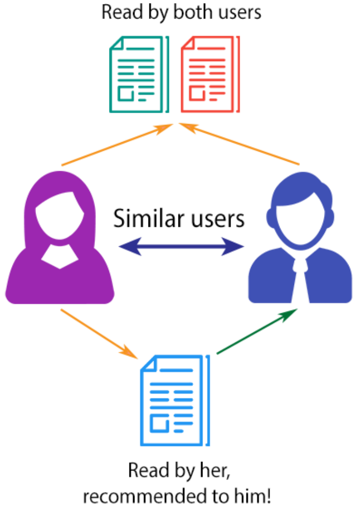
- Human language is ambiguous
  - *Iraqi Head Seeks Arms* (*head* and *arms* are ambiguous)
- Language is subtle
- Language representations are unique to their domains
- Language evolves – changes over time
- ~ 7000 human (living) languages

Common Tasks in NLP

Lancaster University

- Parts-of-Speech Tagging
- Words Sense Disambiguation
- Text Classification
- Machine Translation
- Named-Entity Recognition
- Information Extraction
- “Spoken” Language Systems






"Il est impossible aux journalistes de rentrer dans les régions tibétaines"

Bruno Philip, correspondant du "Monde" en Chine, estime que les journalistes de l'AFP qui ont été expulsés de la province tibétaine du Qinghai "n'étaient pas dans l'illégalité".

Les faits Le dalaï-lama dénonce l'"enfer" imposé au Tibet depuis sa fuite, en 1959

Vidéo Anniversaire de la rébellion


"It is impossible for journalists to enter Tibetan areas"

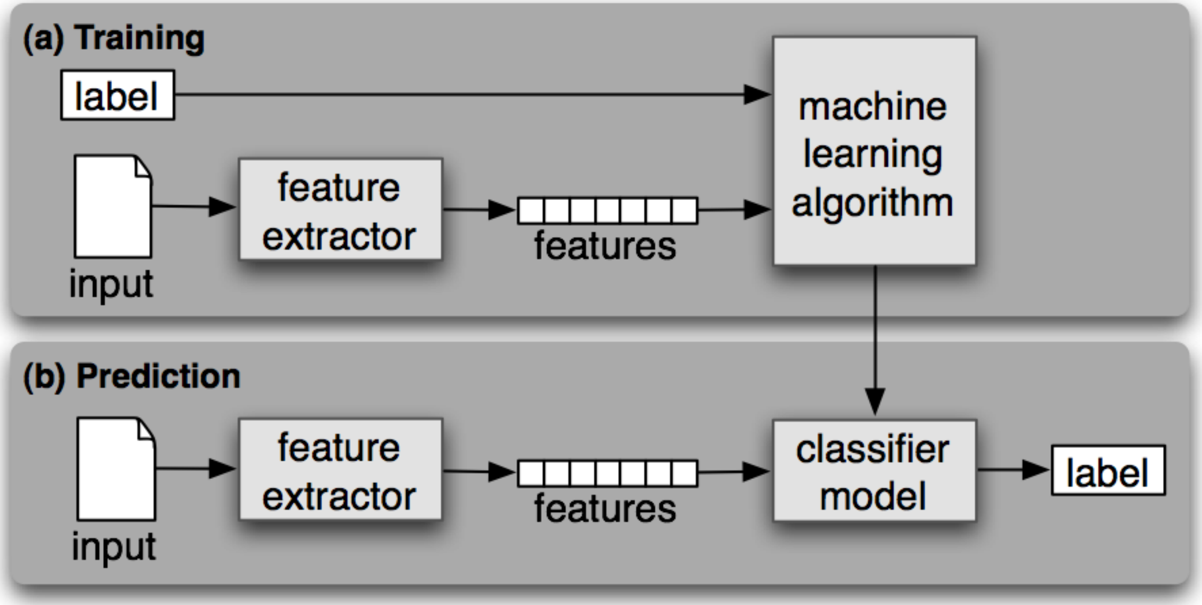
Philip Bruno, correspondent for "World" in China, said that journalists of the AFP who have been deported from the Tibetan province of Qinghai "were not illegal."

Facts The Dalai Lama denounces the "hell" imposed since he fled Tibet in 1959

Video Anniversary of the Tibetan rebellion: China on guard

Supervised ML for Text

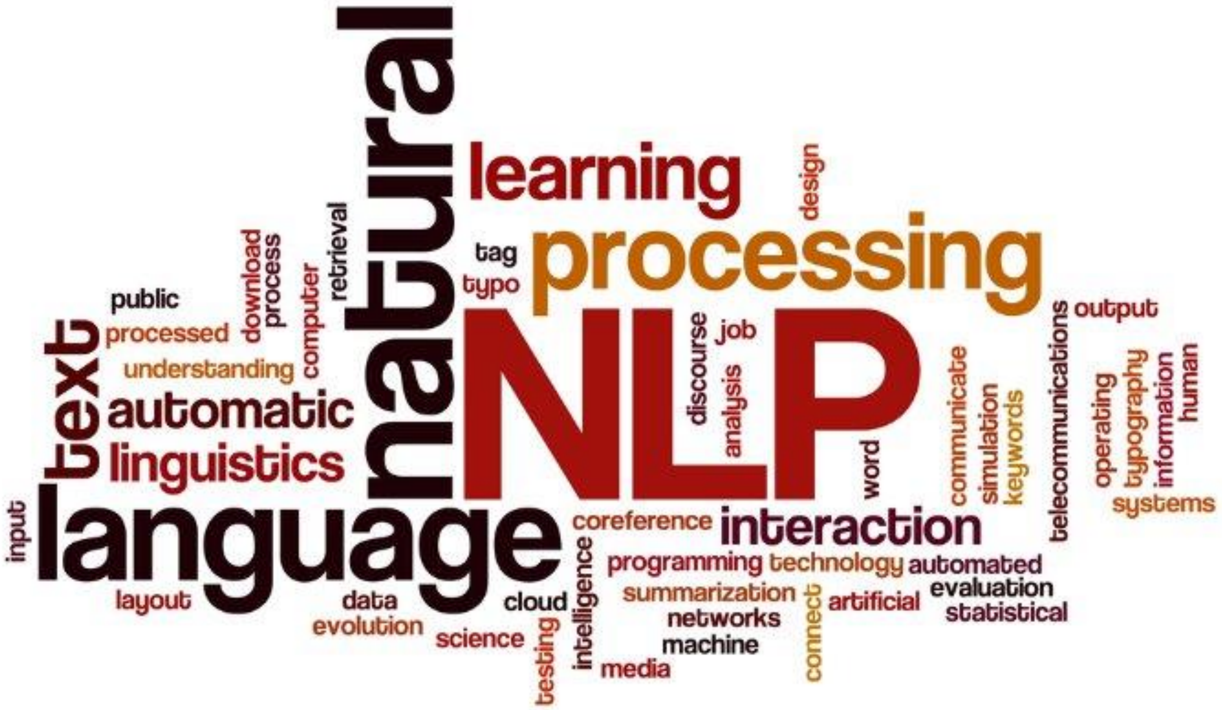
- *A computer **learns** from experience **E** with respect to task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improved with experience **E**"*
- Key components
  - input (**training**) data (instances)
  - Correct **labels**
  - **Feature extractor**
  - Machine **learning algorithm**
  - Classifier **model**



Feature Extraction

- **Features** are the key ingredients for creating data instances for training machine learning models
- **Feature extraction** in NLP involves detecting patterns in text that can help in building an accurate model.
    - **POS-tagging:** Words ending in *-ed* tend to be past tense verbs.
    - **Document classification:** Frequent use of *will* is indicative of news text
- A **feature extractor** is a function that converts each input value to a *feature set.*

Preprocessing, BoW and TF.IDF



See an extended list of NLP tasks: https://nlpprogress.com/

Focusing on Text Classification…

- Example: **sentiment analysis**
- **Input**:
  - text of review
- **Output:**
  - class of sentiment
- **Classes**
  - **Binary** (i.e. 2 classes)**:** positive vs negative
  - **Fine-grained:** positive, somewhat positive, neutral, somewhat negative, negative

Focusing on Text Classification…

- Positive example:
  - The hotel is really beautiful. Very nice and helpful service at the front desk.
- Negative example:
  - We had problems to get the Wi-Fi working. The pool area was occupied with young party animals. So the area wasn’t fun for us.

What is text?

- Text, in this context, could be taken as a sequence of
- characters
- words
- phrases and named-entities
- sentences
- paragraphs
- etc

What is a word?

- A word is a “meaningful” sequence of characters
- In English, we can split a sentence by spaces of punctuations

Input: Friends, Romans, Countrymen, lend me your ears;

Output: Friends | Romans | Countrymen | lend | me | your | ears

- In German, compound words are written without space:
  - *Rechtsschutzversicherungsgesellschaften* means
  - “insurance companies that provide legal protection”
- Japanese has no spaces at all!
  - Butyoucanstillreaditright?
  - しかしあなたはまだそれを正しく読むことができますか？

Tokenization: `WhitespaceTokenizer`

- One major pre-processing task is **tokenisation** i.e. splitting an input sequence into **tokens**.
- A token is a useful unit for semantic processing, and can be words, sentences, paragraphs, etc
- Example: **`nltk.tokenize.WhitespaceTokenizer`**
  - Input text: *"This is Andrew's text, isn't it?"*
  - Output:

This	is	Andrew's	text,	isn't	it?

- Problem:
  - Punctuations attached to words, inflates the dictionary
  - e.g. **"it"** == **"it?"**

Tokenization: `WordPunctTokenizer`

- Using different tokenizers gives different results
- Example: **`nltk.tokenize.WordPunctTokenizer`**
  - Separates the punctuations
  - Input text: "*This is Andrew's text, isn't it?*"
  - Output:

This	is	Andrew	'	s	text	,	isn	'	t	it	?

- Problem:
  - Creates strange words: **"s", "isn", "t"** are not very meaningful

Tokenization: `TreebankWordTokenizer`

- Example: **`nltk.tokenize.TreebankWordTokenizer`**
  - Separates the punctuations
  - Input text: “*This is Andrew’s text, isn’t it?*”
  - Output: This | is | Andrew | ’s | text | , | is | n’t | it | ?

- Output looks better
  - **“s”**, and **“n’t”** seem more meaningful

Tokenization: Python example

```
import nltk
text = "This is Andrew's text, isn't it?"
```

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokenizer.tokenize(text)
```

```
['This', 'is', "Andrew's", 'text,', "isn't", 'it?']
```

```
tokenizer = nltk.tokenize.TreebankWordTokenizer()
tokenizer.tokenize(text)
```

```
['This', 'is', 'Andrew', "'s", 'text', ',', 'is', "n't",
'it', '?']
```

Token Normalisation

- This is the process of transforming **variants** of a token into **a common form** that retains its meaning
- It is an essential pre-processing step that reduces variations in word forms to common that mean the same thing
  - US, U.S.A → USA
  - wolf, wolves → wolf
  - talk, talks, talked→ talk
- Two common normalisation method:
  - **stemming** and **lemmatization**

Stemming

- Stemming refers to the process of removing and replacing suffixes to get to the **stem** (i.e. root form) of the word
- Sequentially applied heuristics that chop off suffixes
- Example: **`nltk.stem.PorterStemmer`**
  - feet → feet cats → cat
  - wolves → wolv talked → talk
- Problem:
  - fails on irregular forms
  - Produces non-words

Rule	Example
SSES → SS	caresses → caress
IES → I	Ponies → poni
SS → SS	caress → caress
S →	cats → cat

Lemmatization

- Applies vocabulary and morphological analysis to a word to return the **lemma** i.e. base or dictionary form of a word
- Example – **`nltk.stem.WordNetLemmatizer`**
  - feet → foot　　　cats → cat
  - wolves → wolf　　　talked → talked
- Problems:
  - Not all forms are reduced
- It may worth trying both to see which is better for the task.
- Normalizations on *capitalization* or *acronyms* are also common

Stemming: Python Example

```python
import nltk
text = "feet cats wolves talked"
tokenizer = nltk.tokenize.TreebankWordTokenizer()
tokens = tokenizer.tokenize(text)
```

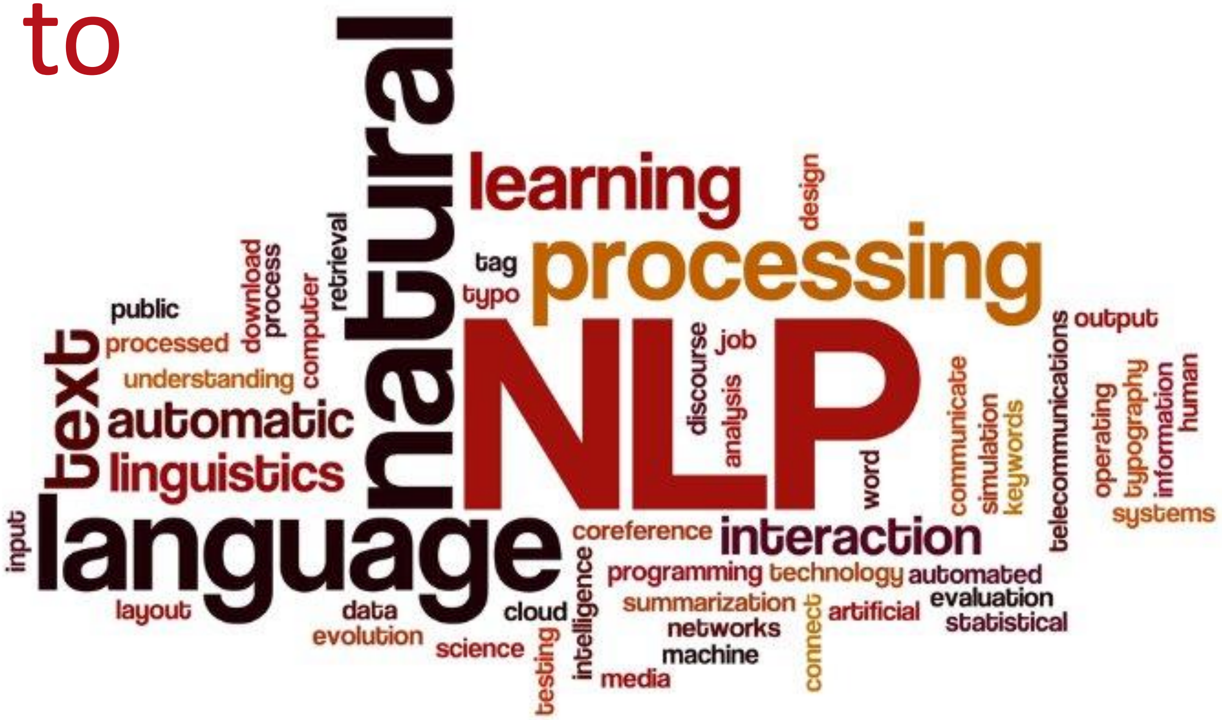
```python
stemmer = nltk.stem.PorterStemmer()
" ".join(stemmer.stem(token) for token in tokens)
```

```
u'feet cat wolv talk'
```

```python
stemmer = nltk.stem.WordNetLemmatizer()
" ".join(stemmer.lemmatize(token) for token in tokens)
```

```
u'foot cat wolf talked'
```

Transforming Tokens to Features



[Machine Learning RoadMap YouTube]

Recall - Feature Extraction

- **Features** are the key ingredients for creating data instances for training machine learning models
- **Text feature extraction** refers to detecting patterns in text that can help in building an accurate model.
    - **POS-tagging:** Words ending in *-ed* tend to be past tense verbs.
    - **Document classification:** Frequent use of *will* is indicative of news text
- We can achieve this with **text vectorization**

Meaning in Context

- One of the most successful ideas in modern NLP is that the meaning of a word is determined by it’s context.
- The foundational assumption was proposed by the linguist John Rupert Firth in 1957
- “You shall know a word by the company it keeps”
  – *Firth J.R. 1957:11*
- We gain a lot by representing a word in terms of its neighbours

Text Vectorisation

- Machine learning algorithms most often take *numeric feature* vectors as input.
- Working with text documents, we convert each document into a numeric vector.
- This representation format is known as **text vectorization**.
- Text vectorization aims to capture the semantic relationship between a word and other words in similar context

Approaches to Text Vectorization

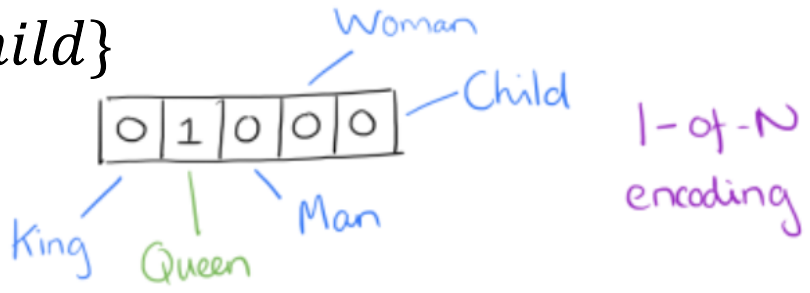
- Localist approach
  - Assign *token_ids*
  - Use 1-Hot vector
- Bag of Words (BoW)
- Term Frequency – Inverse Document Frequency (TF-IDF)
  - Word Embedding
    - Word2Vec
    - Glove
- Similarity (distance) measure
  - Cosine distance
  - Euclidean distance

Assign token_ids to words

- Vocabulary, $V = \{king, queen, man, woman, child, \ldots, horse\}$
- Assign a specific **`token_id`** each word encountered
  - $\{king = 0;\ queen = 1, man = 2, woman = 3, child = 4, \ldots, horse = 100000\}$
- Problems
  - Token_ids refers to position
  - Captures to meaning or relationship
  - Not properly normalised

1 – Hot Vector

- Vocabulary, $V = \{king, queen, man, woman, child\}$
- Each word is assigned a vector
  - 1: in the position of word
  - 0: everywhere else
- Problems
  - $similarity(\text{'}king\text{'}, \text{'}queen\text{'}) = 0$
  - Orthogonal vectors
  - No notion of similarity
  - Sparse, shape = $|V| \times |V|$
  - $|V|$ can be 10s or 100s of thousands



word	features				
king	1	0	0	0	0
queen	0	1	0	0	0
man	0	0	1	0	0
woman	0	0	0	1	0
child	0	0	0	0	1

Bag of Words (BoW)

- Bag of words represents the counts of occurrences of particular tokens in the text
- Example, in the movie review text

good movie
not a good movie
did not like

good	movie	not	a	did	like
1	1	0	0	0	0
1	1	1	1	0	0
0	0	1	0	1	1

- Problems:
  - The order of words are not preserved, hence bag of words
  - counters are not normalized

Bag of Words (BoW)

- To preserve some ordering, we can count **n-grams**
  - e.g word pairs (bigrams) or triples (trigrams)
- Problem
  - Too many features

good movie
not a good movie
did not like

good movie	movie	did not	a	…
1	1	0	0	…
1	1	0	1	…
0	0	1	0	…

Reducing the number of features

- Remove some n-grams from features based on their occurrence frequency in documents of our corpus
- **High frequency n-grams:**
  - stopwords: Articles, prepositions, etc. (e.g: and, a, the)
  - They won’t help us to discriminate texts à remove them
- **Low frequency n-grams:**
  - Typos, rare n-grams
  - We don’t need them either; will likely overfit
- **Medium frequency n-grams:**
  - We need to focus on these n-grams

Reducing the number of features

- It proved to be useful to look at n-gram frequency in our corpus for filtering out bad n-grams
- What if we use it for ranking of medium frequency n-grams?
- **Hint**: n-gram with smaller frequency can be more discriminating because it can capture a specific issue in the review

Term Frequency (TF)

- Term frequency $\boldsymbol{tf(t, d)}$
  - frequency of term (or n-gram) $\boldsymbol{t}$ in document $\boldsymbol{d}$
- Variants
  - binary
  - raw count
  - term frequency
  - log normalization

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$

Inverse Document Frequency (IDF)

- $\boldsymbol{N} = |\boldsymbol{D}|$ = total number of document in corpus
- $|\{\boldsymbol{d} \in \boldsymbol{D} : \boldsymbol{t} \in \boldsymbol{d}\}|$ = number of documents where the term $\boldsymbol{t}$ appears
- $\boldsymbol{idf}(\boldsymbol{t}, \boldsymbol{D}) = \log \frac{\boldsymbol{N}}{|\{\boldsymbol{d} \in \boldsymbol{D} : \boldsymbol{t} \in \boldsymbol{d}\}|}$

TF-IDF

- $\boldsymbol{N} = |\boldsymbol{D}|$ = total number of document in corpus
- $|\{\boldsymbol{d} \in \boldsymbol{D}: \boldsymbol{t} \in \boldsymbol{d}\}|$ = number of documents where the term $\boldsymbol{t}$ appears
- $\boldsymbol{idf}(\boldsymbol{t}, \boldsymbol{D}) = \log \frac{\boldsymbol{N}}{|\{\boldsymbol{d} \in \boldsymbol{D}: \boldsymbol{t} \in \boldsymbol{d}\}|}$

- $\boldsymbol{tfidf}(\boldsymbol{t}, \boldsymbol{d}, \boldsymbol{D}) = \boldsymbol{tf}(\boldsymbol{t}, \boldsymbol{d}) \cdot \boldsymbol{idf}(\boldsymbol{t}, \boldsymbol{D})$
- High weight TF-IDF = high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents

Improving the Bag of Word model

- Replace counters with the TF-IDF
- Normalise the result row-wise (divide by L_2-norm)
  - L_2-norm is the square root of the sum of the squared vector values

good movie
not a good movie
did not like

→

good movie	movie	did not	…
0.17	**0.17**	0	…
0.17	**0.17**	0	…
0	0	**0.47**	…

TF-IDF Python example

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
texts = [
    "good movie", "not a good movie", "did not like",
    "i like it", "good one"
]
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(texts)
pd.DataFrame(
    features.todense(),
    columns=tfidf.get_feature_names()
)
```

	good movie	like	movie	not
0	0.707107	0.000000	0.707107	0.000000
1	0.577350	0.000000	0.577350	0.577350
2	0.000000	0.707107	0.000000	0.707107
3	0.000000	1.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000

Limitations of Bag of Words model

- Meaning:
  - Discarding word order ignores the context, and in turn meaning of words in the document (semantics)
- Vocabulary:
  - The vocabulary needs careful design to manage the size, which impacts the sparsity of the document representations.
- Sparsity:
  - Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons,

Lancaster University

5-10 minutes break & Question Time

Next: Word Embedding Models

Dense Vector Representation

Word Embedding Models

- An alternative to BoW models is the dense vector representations known as **word embedding models** e.g. *Word2Vec*, *Glove*
- Real-values vector representation is **learned** from a corpus of text
- A **word** is a represented by a low dimensional vector
- Similar words have similar vectors
- Models capture semantic and syntactic details
- Can be used as input to machine learning

Word2Vec Architecture

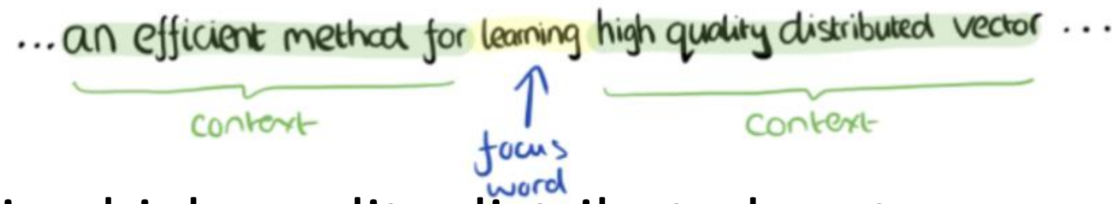
- Uses a neural network model to learn word associations from a large corpus of text
- It can detect synonymous words or suggest additional words for a partial sentence.
- Represents each distinct word with a particular list of numbers called a vector.
- **Cosine similarity** between the vectors indicates the level of semantic similarity between the words represented by the those vectors

Word2Vec - CBOW

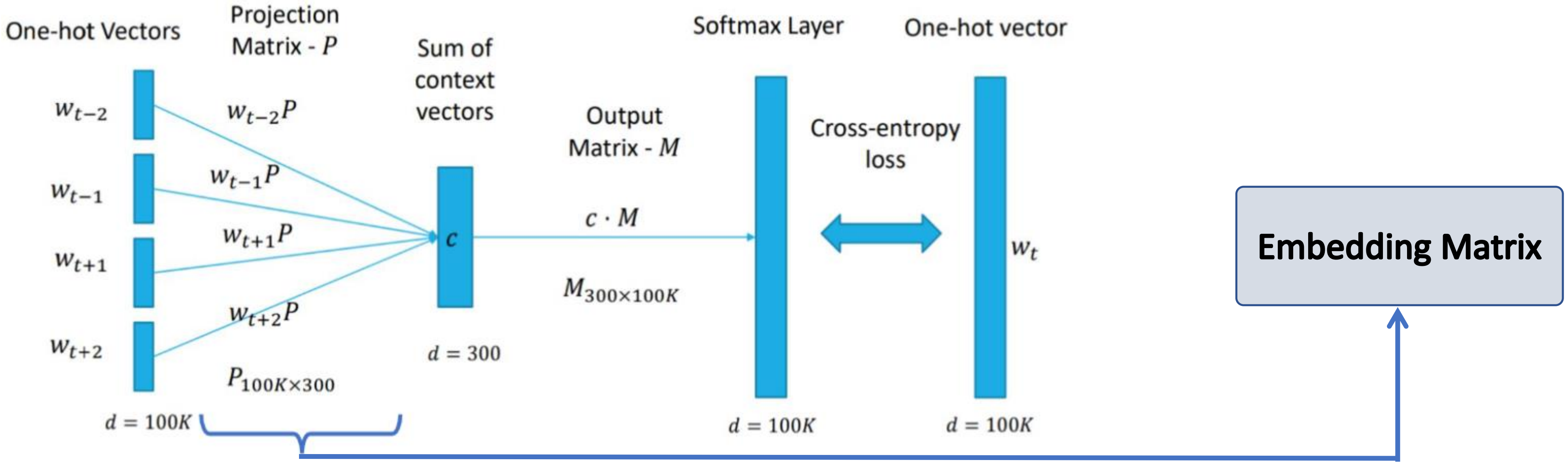
- An efficient method for learning high-quality distributed vectors
- **Goal**: Predict word given the context

Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - CBOW

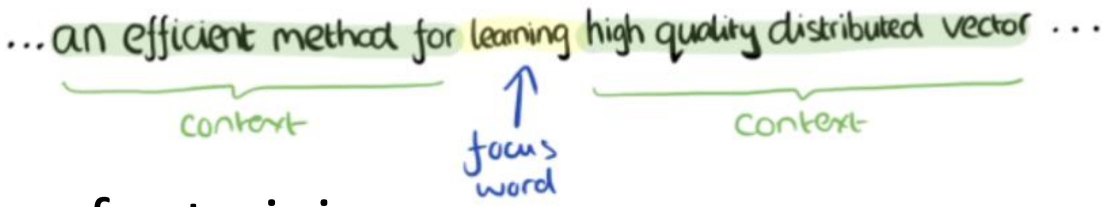


- An efficient method for learning high-quality distributed vectors
- **Goal**: *Predict **word** given **context***

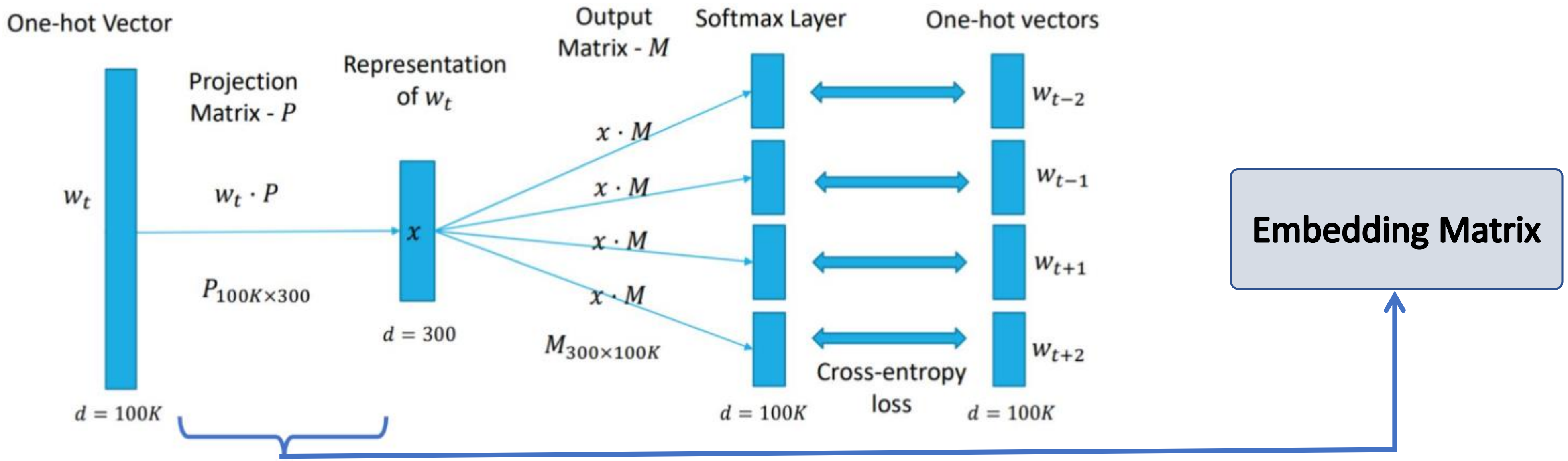


Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf

Word2Vec – Skip-Gram



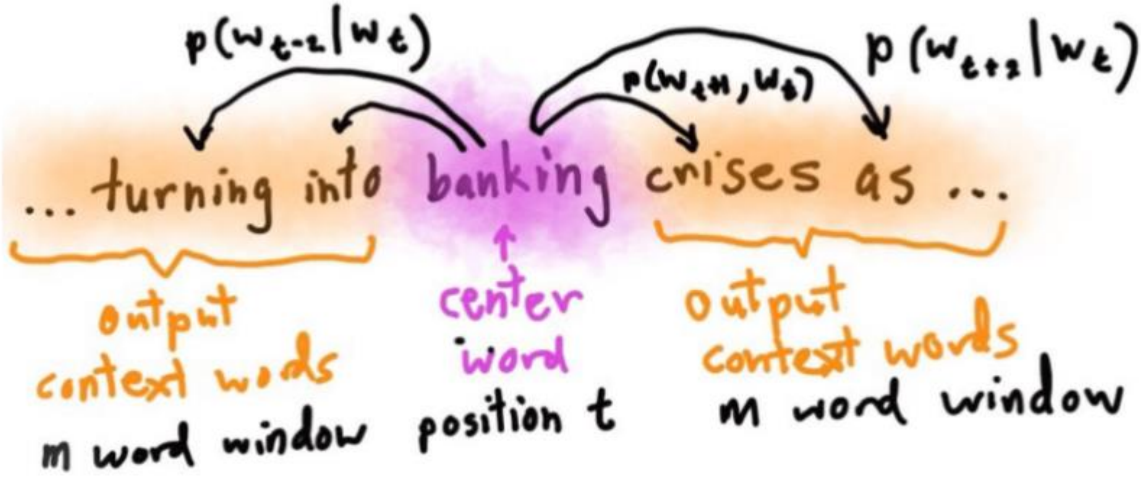
- Skip-gram: another architecture for training
- **Goal**: *Predict **context** given **word***



Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf

Word2Vec Definition

- Given a sequence of words: $w_1, w_2 \ldots w_T$



$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- $p(w_{t+j}|w_t)$ is defined with softmax as:

$$p(w_{t+j}|w_t) = \frac{\exp(v'_{w_{t+j}} v_{w_t})}{\sum_{i=1}^{T}\exp(v'_{w_i} v_{w_t})}$$

v - input vector representations

v' - output vector representations

- Implementations applies techniques like [negative sampling] and [hierarchical softmax] to achieve better results and training efficiency (see links for details)

Word2Vec - Evaluation

- Two intrinsic evaluation categories: **semantic** and **syntactic**
- 5 Semantic types
  - 8869 questions
- 9 Syntactic types
  - 10675 questions

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - Evaluation

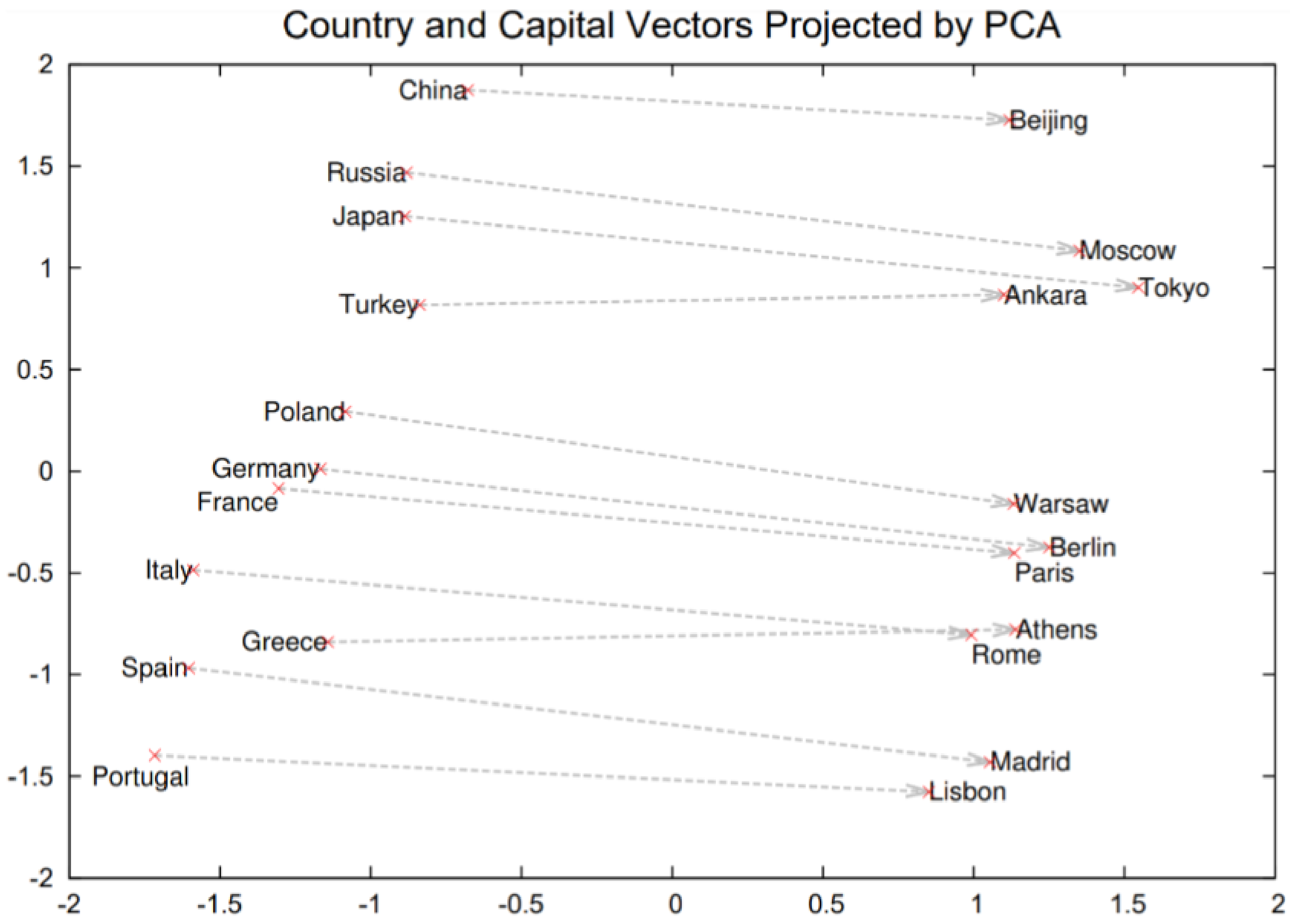
- Learned relationships
  - Word-pair

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - Evaluation

- Learned relationships
  - Word-pair
  - Countries and their capitals



https://arxiv.org/pdf/1301.3781.pdf

Training Parameters – Window size

- Walk

Window size = 3

Word	Cosine distance
go	0.488083
snipe	0.464912
shoot	0.456677
fly	0.449722
sit	0.449678
pass	0.442459
climbs	0.440931
walked	0.436502
ride	0.434034
stumble	0.426750
bounce	0.425577
travelling	0.419419
walking	0.412107
walks	0.410949
trot	0.410418
leaping	0.406744

Window size = 30

Word	Cosine distance
walking	0.486317
walked	0.430764
walks	0.406772
stairs	0.401518
go	0.399274
sidewalk	0.385786
stand	0.380480
cortege	0.371033
wheelchair	0.362877
strapped	0.360179
hollywood	0.356544
carousel	0.356187
grabs	0.356007
swim	0.355027
breathe	0.354314
tripped	0.352899

Training Parameters – Epochs (iterations)

- Walk

No. of iterations = 1

Word	Cosine distance
walking	0.851438
walks	0.846485
bat	0.843796
ride	0.830734
crowd	0.821692
quiet	0.812538
spot	0.802777
steal	0.787917
door	0.787571
doors	0.786485
bed	0.773686
dinner	0.772160
shadow	0.769573

No. of iterations = 100

Word	Cosine distance
walked	0.483473
ride	0.470925
walks	0.470889
stand	0.449993
walking	0.449071
go	0.430172
shoot	0.421110
get	0.404258
move	0.403757
live	0.403347
fly	0.400929
climbs	0.396346
throw	0.391768

Training Parameters – dimensions

- Walk

No. of dimensions = 5

Word	Cosine distance
catcher	0.998074
shirt	0.996589
lechuck	0.995313
bullseye	0.994644
bowler	0.994381
punter	0.993154
lovell	0.992815
heels	0.992255
whip	0.992085
outfit	0.992047
tore	0.991924
steals	0.991524

No. of dimensions = 1000

Word	Cosine distance
walks	0.304954
walked	0.303322
snipe	0.287221
walking	0.272690
ride	0.266770
canter	0.251025
bandleaders	0.246454
climbs	0.233725
catapulted	0.230075
climb	0.229263
trot	0.228362
shouted	0.227306

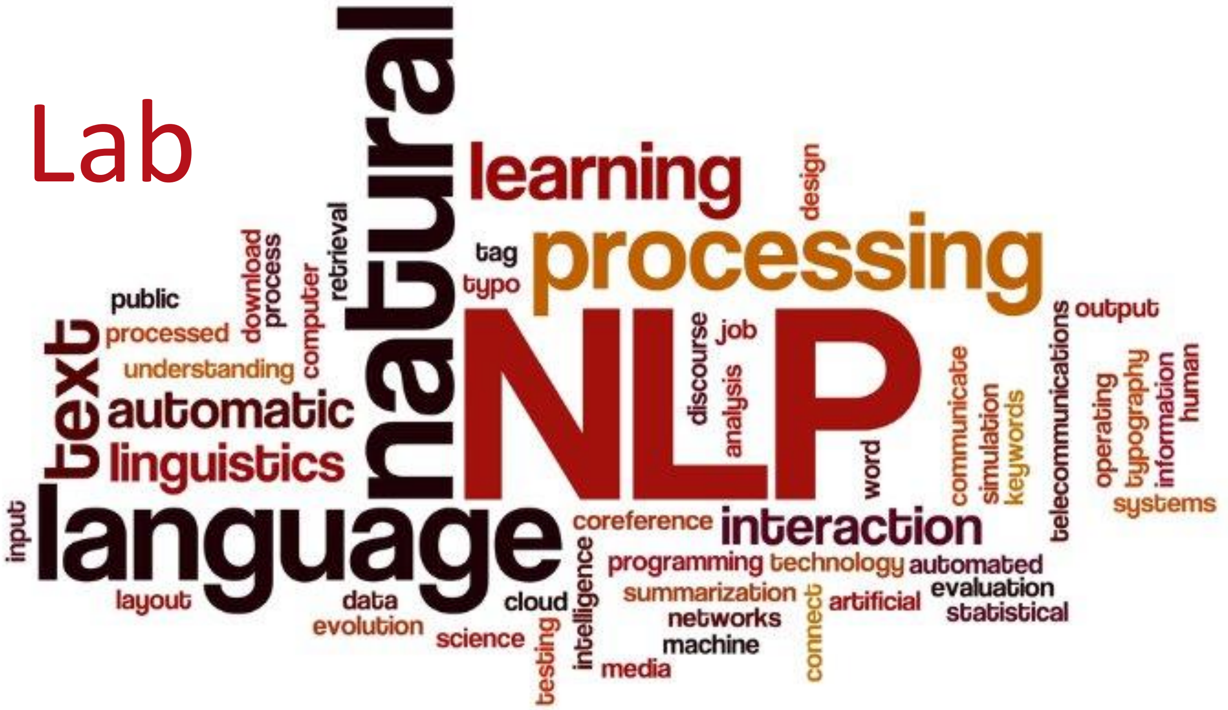
Other Type of Embedding Models

- Character models - Karpathy(2015):
  - The Unreasonable Effectiveness of Recurrent Neural Networks
- Subword models - Bojanowski(2017):
  - Enriching Word Vectors with Subword Information -ACL
- Contextualised models:
  - BERT (Bidirectional Encoder Representations from Transformers)
  - ELMO (Embeddings from Language Model)

Other Type of Embedding Models

- Sentence/Paragraph/Document models:
  - Le & Mikolov(2014): Distributed Representations of Sentences and Documents
  - Logeswaran & Lee (2018): An efficient framework for learning sentence representations
- Thomas Wolf (2018):
  - The Current Best of Universal Word Embeddings and Sentence Embeddings

Review of Last Week Lab



Coming next…

- **Labs** (Wednesday and Thursday):
  - **Support for course work** – use the lab sessions to ask all your questions on the coursework
- **Coursework deadline:**
  - Endeavour to hand-in your coursework before the deadline

Thank you for attending, any questions?