Working with Text

What is Natural Language Processing?

- Natural language processing is a field at the intersection of
  - Computer science
  - Artificial Intelligence
  - And linguistics
- Goal: for computers to process or “understand” natural language .

Perfect language understanding?

Fully understanding and representing the meaning of language (or even defining it) is a difficult goal.

Why is NLP difficult ?

- Complexity in representing, learning and using linguistic/situational/world/visual knowledge.
- Human languages are ambiguous (unlike programming and other formal languages).
- Human language interpretation depends on real world, common sense, and contextual knowledge.

Real newspaper headlines/tweets

The Pope’s baby steps on gays

Boy paralyzed after tumor fights back to gain black belt

Scientists study whales from space

The man saw the boy with the binoculars

Real newspaper headlines/tweets

Boy paralyzed after tumor fights back to gain black belt

Scientists study whales from space

The man saw the boy with the binoculars

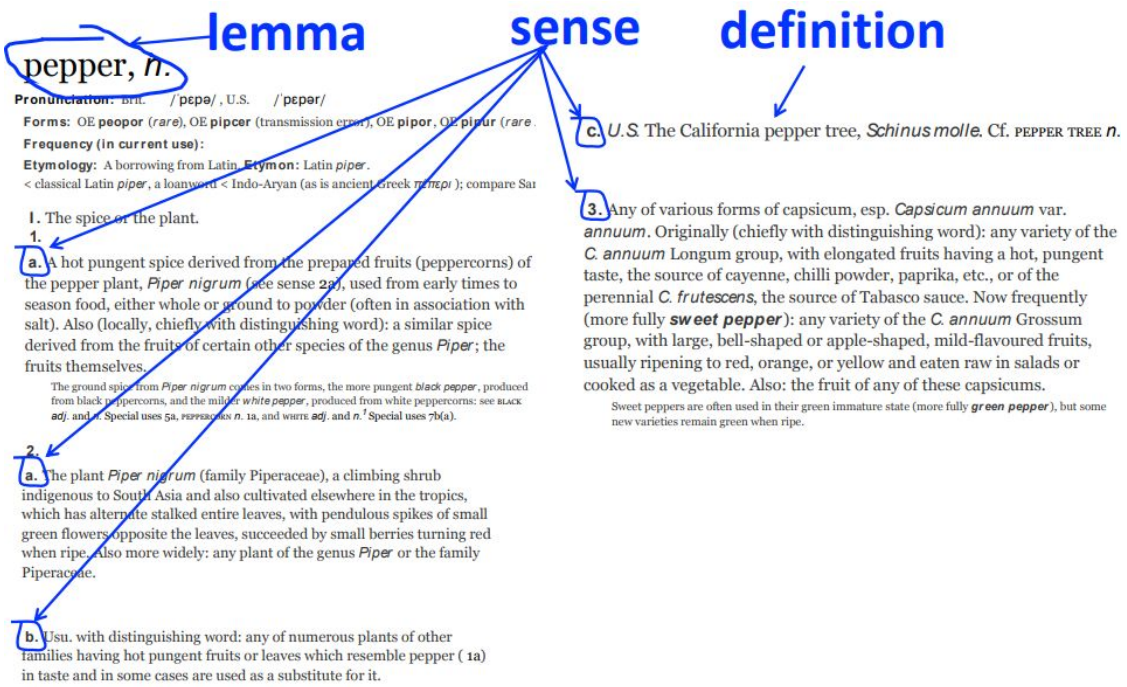
Κώστας Τριανταφυλλίδης : Η μουσική είναι φάρμακο για τον άνθρωπο

Πολιτισμός

3 ώρες πριν **Γιώργος Σκίντσας**

- **Μπλίνκεν: Έχω αφήσει τη φωνή μου στην Ουάσινγκτον, την καρδιά μου την έφερα εδώ**

What is the meaning of words?



A sense or “concept” is the meaning component of a word

Antonymy

- Opposite meaning words that appear similar …

dark/light short/long fast/slow rise/fall hot/cold up/down in/out

Similarity

- Words with similar meaning. Not synonymous, but sharing some element of meaning …

car, bicycle
cow, horse

Ask humans how similar 2 words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

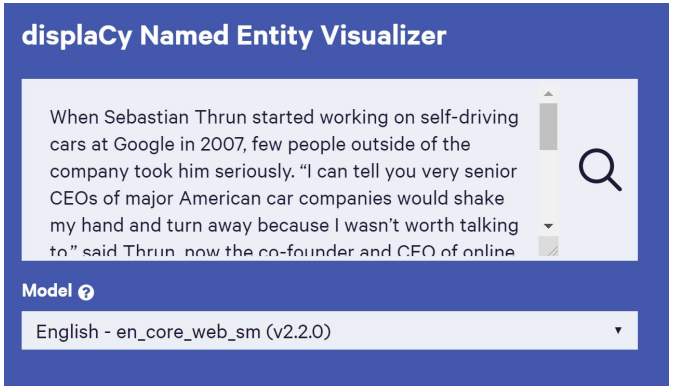
SimLex-999 dataset (Hill et al., 2015)

NLP Tools

Several big improvements in recent years in NLP, coming to end users with different applications.

NLP: Determine the structure of sentences

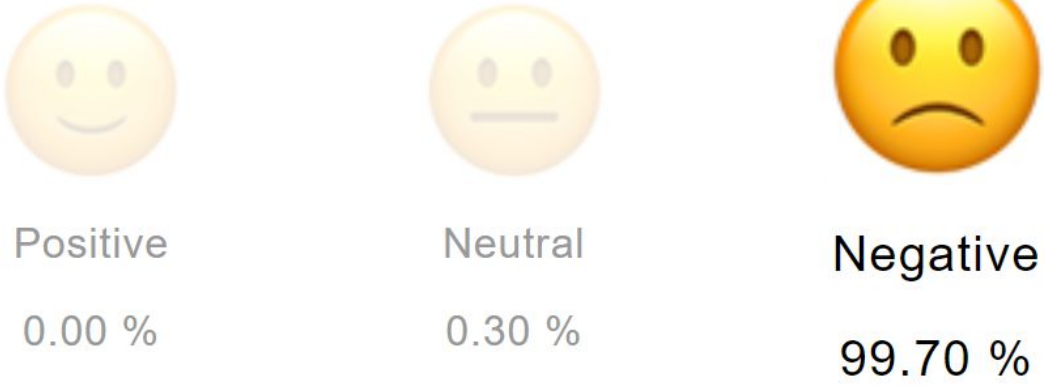
NLP: Named entity recognition



When Sebastian Thrun PERSON started working on self-driving cars at Google ORG in 2007 DATE , few people outside of the company took him seriously. “I can tell you very senior CEOs of major American NORP car companies would shake my hand and turn away because I wasn’t worth talking to,” said Thrun ORG , now the co-founder and CEO of online higher education startup Udacity, in an interview with Recode PERSON earlier this week DATE .

NLP Tasks: Sentiment Analysis

Worst service I paid money also and food was not delivered..no proper customers care service also to contac ▼



First Step: Clean Text

You cannot go straight from raw text to fitting a machine learning or deep learning model.

You must clean your text first, which means splitting it into words and handling punctuation and case.

Text Cleaning Is Task Specific...

First Step: Clean Text

Here are some examples

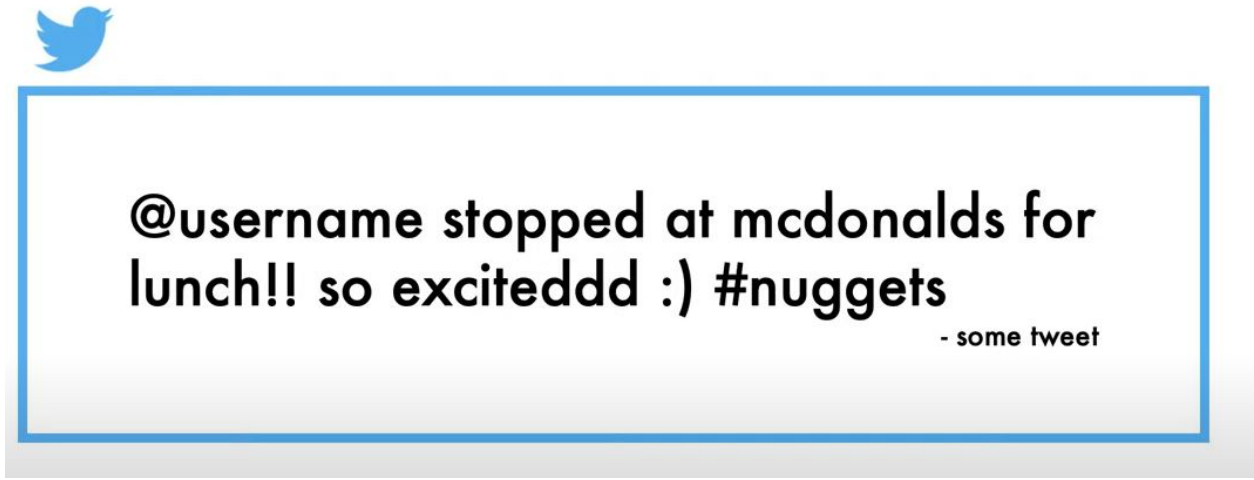
```
def strip_everything(ww):

   w1 = ww.strip("'-:()[]{}!?$%^&*_~\;/.`+=#@")

   return w1



word = re.sub(r'[^\x00-\x7F]+', '', word)  # remove non-ascii chars
```

First Step: Clean Text

Sometimes we have to replace non ascii characters with the right character

```
with open(sys.argv[1], 'r') as infile, open(sys.argv[1] + '.cln', 'w') as outfile:

   for z, line in enumerate(infile):

       line = line.strip() #remove default whitespace characters

       if '\xe2' in line:
           line = line.replace('\xe2', "'")
       if '\u2019' in line:
           line = line.replace('\u2019', "'")
       if '\u002c' in line:
           line = line.replace('\u002c', ",")

      if '\xe7' in tweet:
           tweet = tweet.replace('\xe7', '@')
```

First Step: Clean Text



First Step: Clean Text

stopped at mcdonalds for lunch so excitedddd

First Step: Clean Text

stopped mcdonalds

lunch excitedddd

First Step: Clean Text

stopped mcdonalds

lunch excited

NLP Pipeline

Tokenization

The process of segmenting running text into sentences and words. We can first lowercase the text and remove punctuation.

Stop-word filtering

Stop Words are words which do not contain important information(the, for, this etc.)

Lemmatization

Figuring out the most basic form or lemma of each word in the sentence.

NLP Pipeline

POS Tagging

Look at each token and try to guess it's part of speech — whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about.

Vectorization

In order for this data to make sense to our machine learning algorithm we'll need to convert each review to a numeric representation.

Classification

The action or process of classifying something

Lets see the pipeline in action

First start with the NLTK notebook.

Next: Use NLTK for sentiment analysis notebook.

Textual features

What can we learn from text?

An example of a review

*"I **love** this movie! It's **sweet**, but with **satirical** humor. The dialogs are **great** and the adventure scenes are **fun**. It manages to be **romantic** and **whimsical** while laughing at the conventions of the fairy tale genre. I would **recommend** it to just about anyone. I have seen it several times and I'm always **happy** to see it **again**....."*

The bold words in the above piece of text are the most important words which construct the positive nature of the sentiment conveyed by the text.

Converting words to numbers ...

Goal : Create a representation similar to the following:

great	2
love	2
recommend	1
laugh	1
happy	1
.	.
.	.
.	.

Stop Word Removal

$\mathbf{x}^{[1]} =$ "The sun is shining"

$\mathbf{x}^{[2]} =$ "The weather is sweet"

$\mathbf{x}^{[3]} =$ "The sun is shining,
the weather is sweet, and one and one is two"

n-gram tokenization with n > 1

1 token = 1 word:

$\mathbf{x}^{[1]} =$ "The sun is shining"

1 token = 2 words:

$\mathbf{x}^{[1]} =$ "The sun is shining"

So what is the above representation doing?

Each row is containing a word and its frequency of occurrence in the document (let's call it a document from now on).

You are also wondering why the word “love” has appeared for only once but the frequency is 2?

Well, what we saw is a part of the whole review.

The matrix representation is for the entire review.

Bag-of-words

A ***bag-of-words*** representation of a document does not only contain specific words but all the ***unique words in a document and their frequencies of occurrences***.

A ***bag*** is a mathematical set here, so by the definition of a set, the bag does not contain any duplicate words.

Bag-of-words

If we build the bag-of-words model we get the following representation

document	w1	w2	w3	w4	. . .	wn	sentiment
d1	2	1	3	1		1	positive
d2	1	5	5	5		1	negative
d3	3	8	6	8		2	positive
d4	2	5	1	5		3	positive
d5	3	0	3	0		0	negative
d6	0	0	0	0		0	negative
d7	2	0	0	0		0	positive
d8	9	2	9	2		2	negative
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Each row is the equivalent to the “feature set” of our document.

Bag-of-words formal definition

Consider a Corpus **C** of **D** documents {d1,d2.....dD} and N unique tokens extracted out of the corpus **C**.

The **N** tokens (words) will form a dictionary and the size of the bag-of-words matrix **M** will be given by **D x N**.

Each row in the matrix **M** contains the frequency of tokens in document **D(i)**.

Python steps (this is not a full working example)

```
from sklearn.feature_extraction.text import CountVectorizer

bow_vectorizer = CountVectorizer(max_df=1.0, min_df=1, max_features=1000,
stop_words='english')

bow_xtrain = bow_vectorizer.fit_transform(TWEETS)  #TWEETS : a list with the actual
tweets

print(bow.shape)  #the output is a numpy array of features. The dimensionality of
this array depends on the number of TWEETS (shape should be number_of_tweets x
vocabulary_size)
```

Let's review it one more time with a simple example

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
>>> print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
  [0 1 1 1 0 0 1 0 1]]
```

Problem: counts don't preserve order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

TF-IDF (proposed by Rennie et al. 2003)

This is another method which is based on frequency but it is different to the bag-of-words approach in the sense that it takes into account not just the occurrence of a word in a single document (or tweet) but in the entire corpus.

Let's have a look at the important terms related to TF-IDF:

- TF = (Number of times term t appears in a document)/(Number of terms in the document)
- IDF = log(N/n), where, **N** is the number of documents and ***n*** is the number of documents a term ***t*** has appeared in.
- TF-IDF = TF*IDF

Let's take an example to get a clearer understanding.

Sentence 1 : The car is driven on the road.

Sentence 2: The truck is driven on the highway.

In this example, each sentence is a separate document (A,B).

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

Problems here ...

- TF-IDF is based on the bag-of-words (BoW) model, therefore it does not capture position in text, semantics, co-occurrences in different documents, etc.

- For this reason, TF-IDF is only useful as a lexical level feature

- Cannot capture semantics (e.g. as compared to word embeddings which will be discussed next)

Python steps (this is not a full working example, replace with your variables)

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=1.0, min_df=1, max_features=1000,
stop_words='english')

tfidf = tfidf_vectorizer.fit_transform(TWEETS)

print( tfidf.shape) #the output is a numpy array of features
```

Pointwise Mutual Information

Ask whether a context word is particularly informative about the target word.

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X, Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

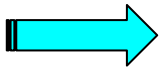
$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Distributional hypothesis (DH) of meaning

Words that occur in similar contexts tend to have similar meanings

"I found a **wonderful** restaurant yesterday!"

"I found a **fantastic** restaurant yesterday!"



Looks like they have similar meaning.

DH forms the theoretical foundation of distributional semantics… vector space semantics

You get a lot of value by representing a word by its neighbors

What if I told you that I am going to take thousands of instances of the word bank in text, and look at the environment at which each one appeared and I am going to see dept problems, government etc, and I am going to start counting up all of these things that appear and by some means, and I am going to use those words in the context to represent the meaning of banking.

Distributional hypothesis (DH) of meaning

Two major ways to generate distributional representations

- Count models
- Prediction models

The counting model

For example, given sentences "*He buys this newspaper every day*" and "*I read a newspaper every day*", we can look at window size 2 (on either side of the main word **newspaper**) and create the following count vector:

buys	this	every	day	read	a
1	1	2	2	1	1

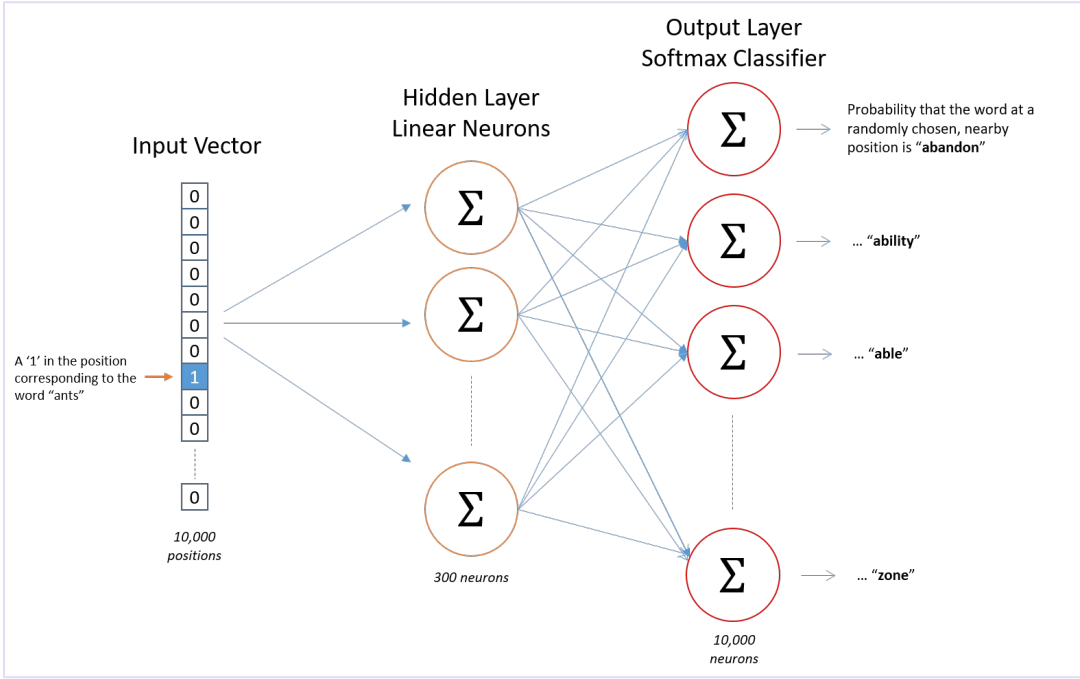
As you can see, "every" and "day" get a count of 2, because they appear in both sentences, whereas "he" doesn't get included because it doesn't fit into the window of 2 words.

The predicting model

The vectors of context words are given as input to the network. They are summed and then used to predict the main word.

During training, the error is backpropagated and the context vectors are updated so that they would predict the correct word.

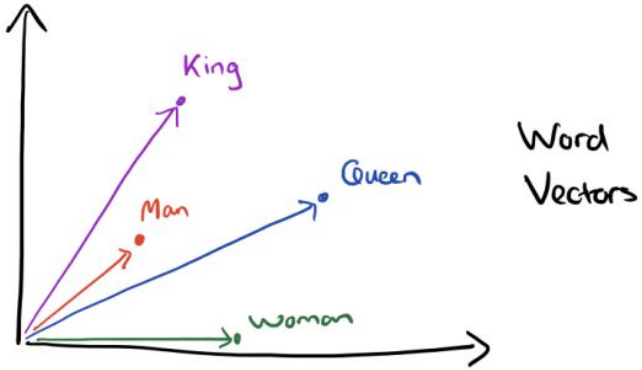
Since similar words appear in similar contexts, the vectors of similar words will also be updated towards similar directions.



Vector of a word.

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

Word embeddings

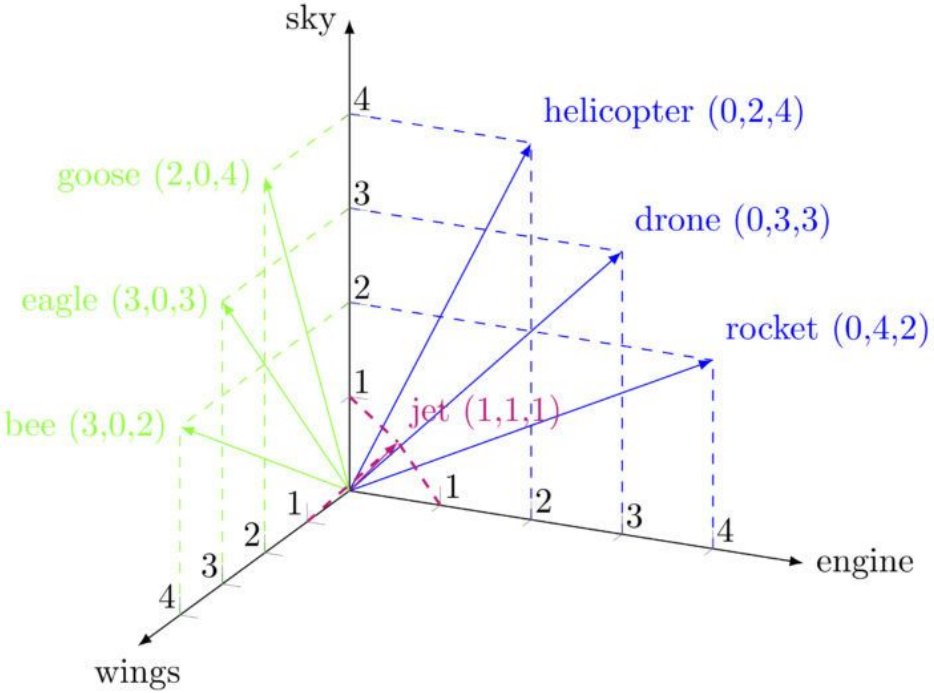


Word embeddings are the modern way of representing words as vectors. The objective of word embeddings is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are able to achieve tasks like **King -man +woman = Queen**, which is mind-blowing.

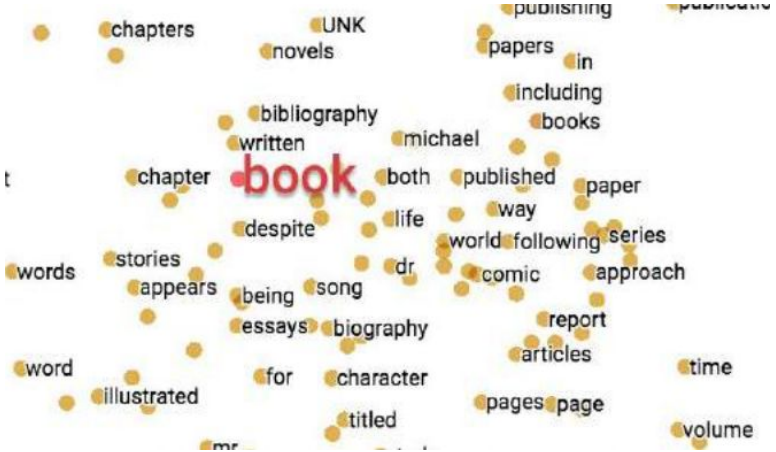
Word embeddings are the computational implementation of the DH.

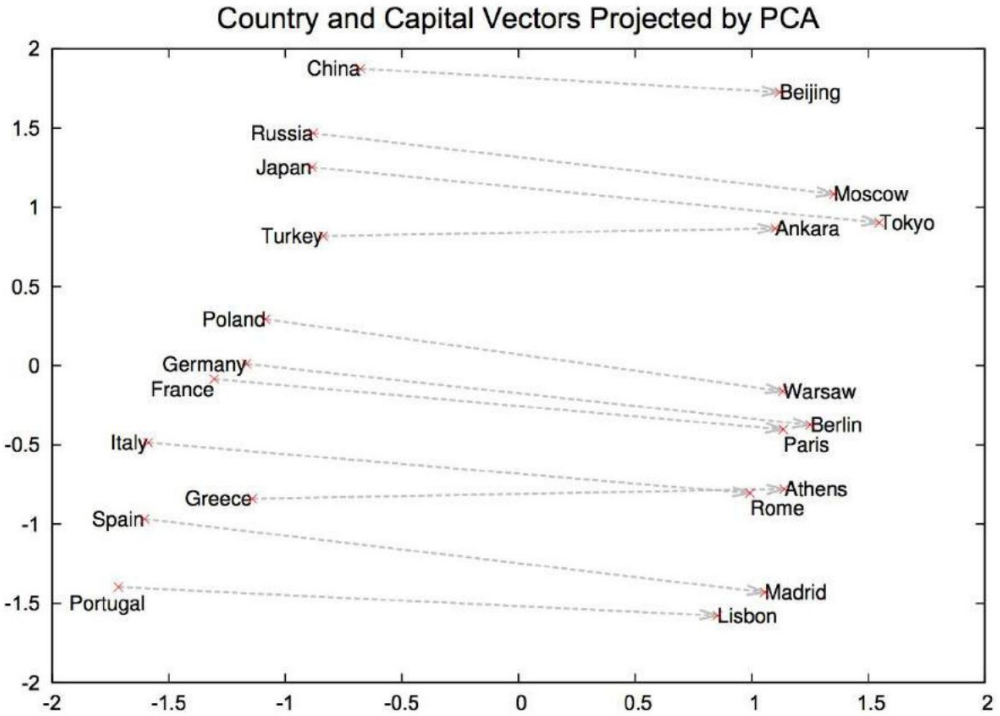
Word embeddings

The presupposition underlying word embeddings is that semantic similarities are indexed on contextual affinities. For example, helicopter and drone are close because they occur in similar contexts, have similar vector profiles, and are therefore close in the vector space.



Meaningful directions





Advantages

The advantages of using word embeddings over BOW or TF-IDF are:

- Dimensionality reduction - significant reduction in the no. of features required to build a model.
- It captures meanings of the words, semantic relationships and the different types of contexts they are used in.

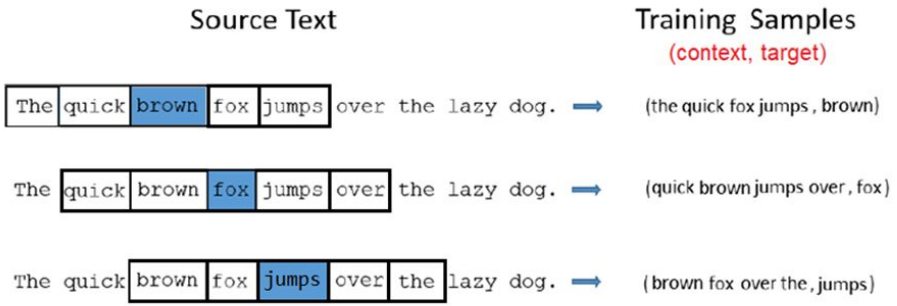
Word2vec

Word2Vec is not a single algorithm but a combination of two techniques – **CBOW (Continuous bag of words)** and **Skip-gram model**. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations.

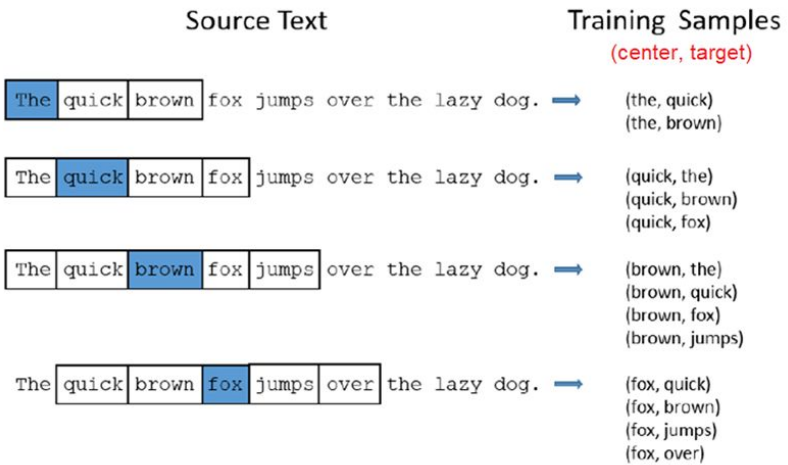
CBOW tends to predict the probability of a word given a context. A context may be a single adjacent word or a group of surrounding words.

The *Skip-gram* model works in the reverse manner, it tries to predict the context for a given word.

Word2vec

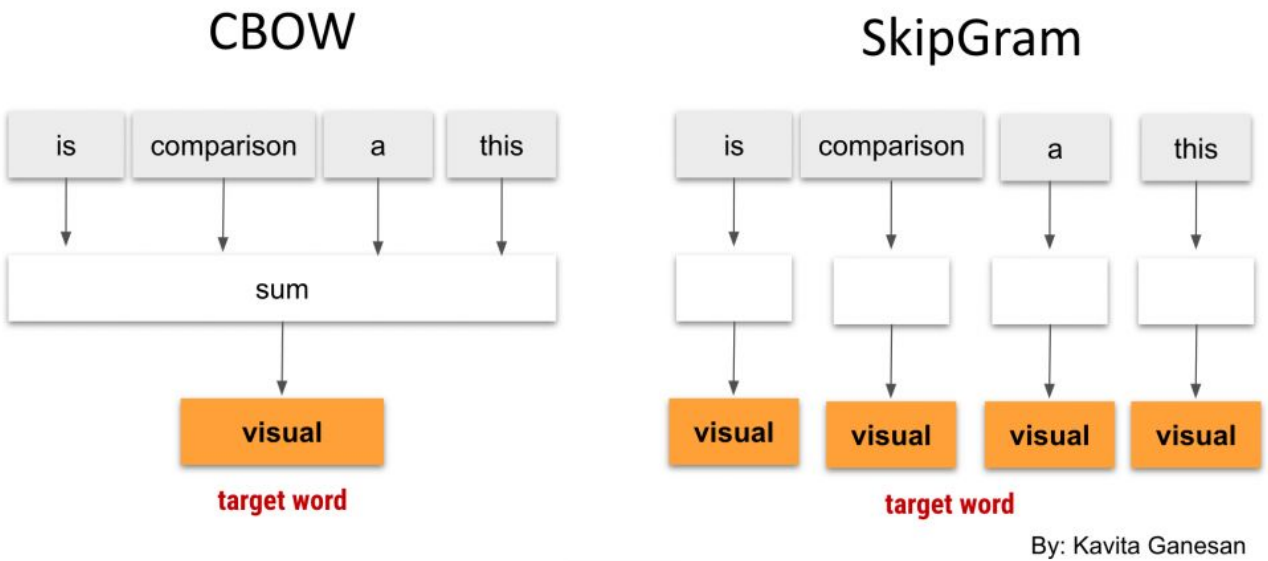


CBOW



Skipgram

Word2vec


CBOW
is
comparison
a
this
sum
visual
target word
SkipGram
is
comparison
a
this
visual
visual
visual
visual
target word
By: Kavita Ganesan
This is a visual comparison

Train a Word2vec model for your project

We will train a Word2Vec model on our data to obtain vector representations for all the unique words present in our corpus.

There is one more option of using **pre-trained word vectors** instead of training our own model. Some of the freely available pre-trained vectors are:

- Google News Word Vectors
- Freebase names
- DBPedia vectors (wiki2vec)

for this course, you can also train your own word vectors.

Python steps (this is not a full working example , replace with your variables)

```
tokenized_tweet = MYTWEETS.apply(lambda x: x.split()) # tokenizing
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

model_w2v.train(tokenized_tweet, total_examples= len(MYTWEETS), epochs=20)
```

Python (replace with your variables)

Let's play a bit with our Word2Vec model and see how does it perform. We will specify a word and the model will pull out the most similar words from the corpus.

```
model_w2v.wv.most_similar(positive="trump")
```

```
[('donald', 0.5460224747657776),
 ('phoni', 0.5259741544723511),
 ('unfit', 0.5246831178665161),
 ('unstabl', 0.5200092792510986),
 ('melo', 0.5164598226547241),
 ('potu', 0.515663743019104),
 ('unfavor', 0.5101866126060486),
 ('hillari', 0.5076572299003601),
 ('#delegaterevolt', 0.5051215291023254),
 ('jibe', 0.5032232999801636)]
```

```
model_w2v.wv.most_similar(positive="dinner")
```

```
[('spaghetti', 0.5658013820648193),
 ('#avocado', 0.5653101801872253),
 ('#cellar', 0.5547047853469849),
 ('cookout', 0.5511012077331543),
 ('noodl', 0.5489310622215271),
  ('spinach', 0.5404106378555298)]
```

Python (replace with your variables)

```
w1=['bathroom']
display_similar(w1,topn=8)
```

	cbow	cosine_sim
0	bath	0.797984
1	washroom	0.780254
2	bathrooms	0.735474
3	bathtub	0.711049
4	bathroon	0.667697
5	shower	0.647936
6	bathrooom	0.636141
7	countertop	0.594575

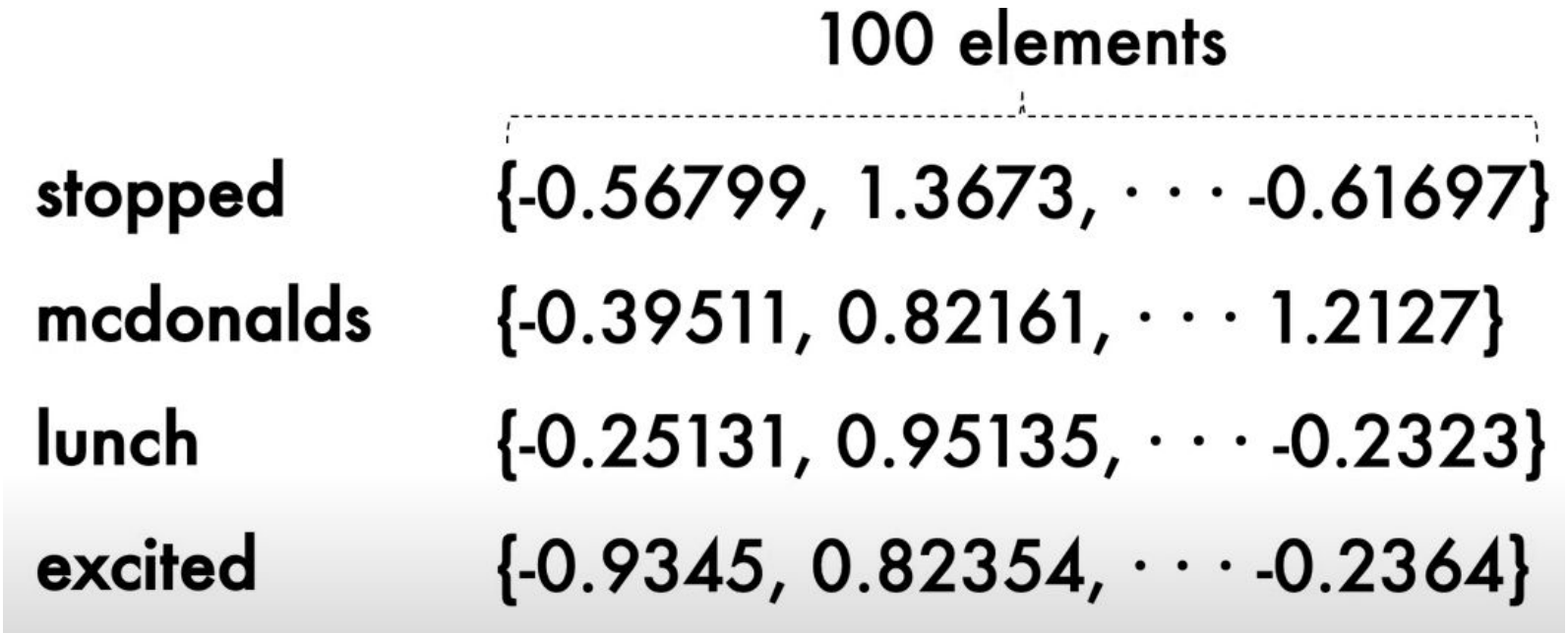
	skipgram	cosine_sim
0	shower	0.806718
1	bath	0.788024
2	washroom	0.778886
3	bathtub	0.762487
4	vanity	0.757107
5	bathrooms	0.735585
6	sink	0.728139
7	tile	0.717654

	skipgramsi	cosine_sim
0	bathrooom	0.982157
1	bathroomn	0.971666
2	thebathroom	0.968645
3	bathroomi	0.962223
4	etcbathroom	0.960976
5	bathrroom	0.959482
6	bathroomno	0.959138
7	bathroomhad	0.956964

How does a vector look like?

model_w2v['stopped']

How does a vector look like?



Word vectors visualization

How can we display a word vector in a x,y coordinate system ? 🤔

Well we have 200-300 dimensions not 2 dimensions...

You can use **t-SNE or PCA**: those are techniques for dimensionality reduction that can be used to visualize high-dimensional vectors.

Python example

```
def tsne_plot(model):
    "Creates and TSNE model and plots it"
    labels = []
    tokens = []
    for word in model.wv.vocab:
        tokens.append(model[word])
        labels.append(word)

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

x = []
y = []

for value in new_values:
    x.append(value[0])
    y.append(value[1])
```

```
plt.figure(figsize=(16, 16))
for i in range(len(x)):
    plt.scatter(x[i],y[i])
    plt.annotate(labels[i],
                 xy=(x[i], y[i]),
                 xytext=(5, 2),
            textcoords='offset points',
                      ha='right',
                      va='bottom')
plt.show()
```

Calling the function with :

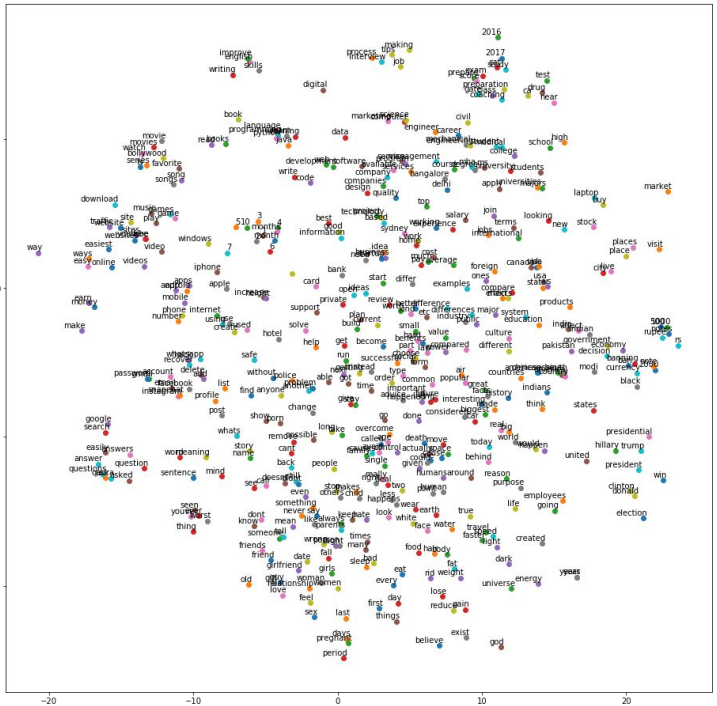
```
tsne_plot(model)
```

It's Becoming Hard to Read

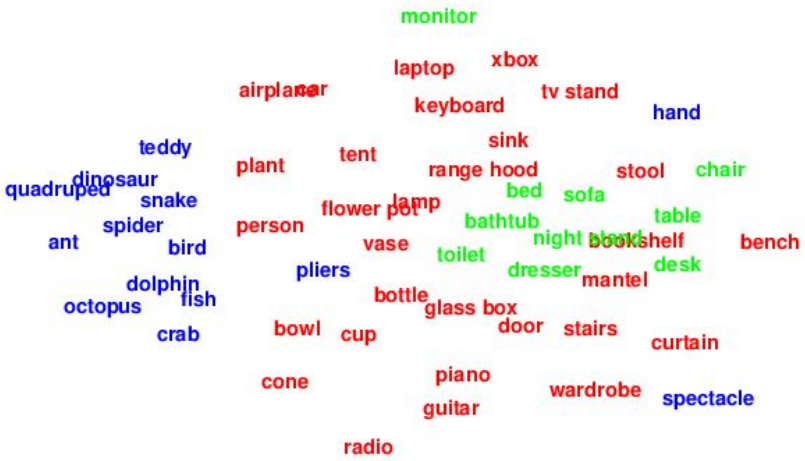
If the visualization is hard to read then

you can select only a limited number of

words to display (for example 1000)



Tweak it to visualize better



Preparing Vectors for Sentences (ex. tweets)

We'll have to figure out a way to use the word vectors from word2vec model to create a vector representation for an entire tweet (sentence).

There is a simple solution to this problem, we can simply take the **mean** of all the word vectors present in the tweet.

The length of the resultant vector will be the same, i.e. 200. We will repeat the same process for all the tweets in our data and obtain their vectors. Now we have 200 word2vec features for our data.

TASK: create function that saves a vector for each tweet by taking the average of the vectors of the words present in the tweet.

Evolution of Word to Vector models

- **Count based Vector space model (Non-semantic)**
  - Count Vectorization
  - Tf-IDF
  - Hashing Vectorization
- **Non Context-Based Vector Space Model (Semantic)**
  - Word2Vec
  - FastText
  - Glove
- **Context-Based Vector Space Model (Semantic)**
  - ELMo
  - BERT

Limitations of Word2Vec

- Inability to handle unknown or OOV(out-of-vocabulary) words.
- No shared representations at sub-word levels.
- Scaling to new languages requires new embedding matrices.
- Word2Vec only rely just on local statistics (local context information of words) but does not incorporate global statistics (word co-occurrence) to obtain word vectors.
- Indifference to word order and inability to represent idiomatic phrases.

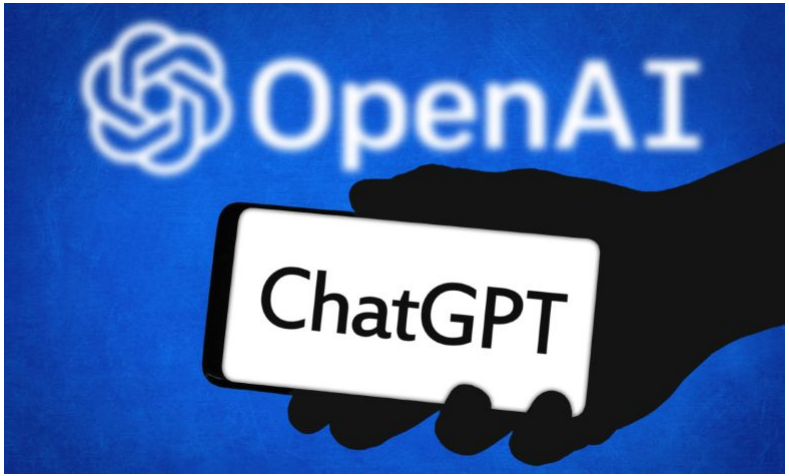
(https://medium.com/co-learning-lounge/nlp-word-embedding-tfidf-bert-word2vec-d7f04340af7f)

LLMs

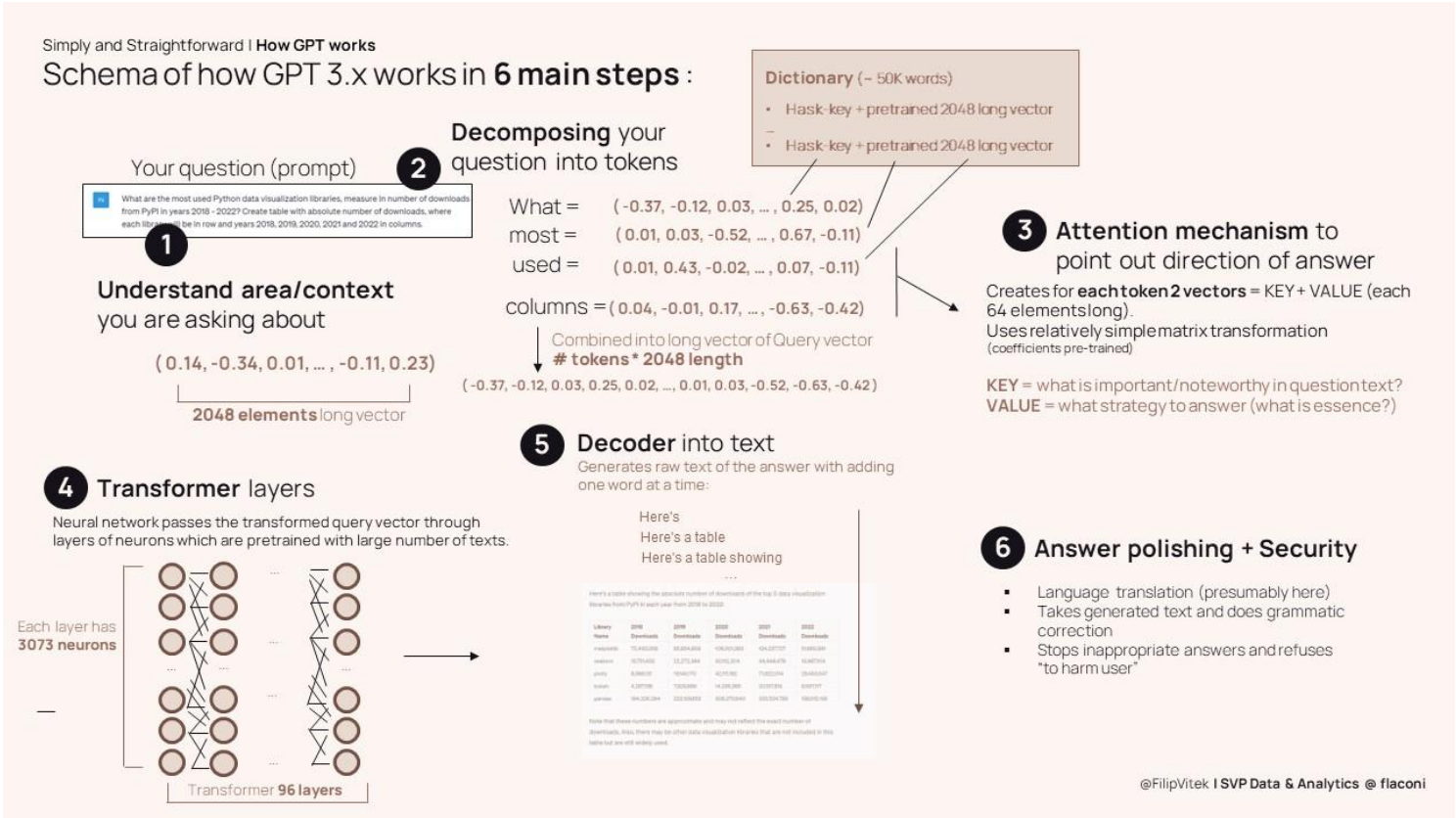
Large Language models

- Pre-trained language models are a type of artificial intelligence model that is trained on large amounts of text to understand natural language.
- These models can be used for tasks such as text classification, text generation, and language translation, among others.
- One of the first pre-trained language models was the Transformer language model (Transformer), developed by Google in 2017.
- This model used a neural network architecture designed to understand natural language in context and produce accurate and useful responses.

The natural language model that is changing the game



ChatGPT components



Background for each of the GPT 3.x **main steps.** In total need **5 different training** efforts needed.

1 Understand area/context you are asking about

- Not yet the question, just context of it
- Compress World into 2048 categories
- These are not 0/1, but decimals -1,0 to 1,0
- Initiated with random N[0, 0.02] ~ enabling slightly different answer everytime
- **1st training task** (changes only with version)
- Reason why each conversation separately and how SHE* remembers what said
- Ignores more words than 2048

2 Decomposing your question into tokens

- Text broken down to tokens + EoT token
- Each token looked up in dictionary 50257 words (for GPT-3)
- **2nd Training task** (changes only with version)
- Dictionary returns for each word 2048 long vector (pre-trained)
- Whole question translated into 2048xtoken_count. So your 100 words question will translate into 204800 numbers

3 Attention mechanism to point out direction of answer

- Based on Scientific paper (2014, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio) and then 2017 Transformer with Attention only ("Attention is All You Need" by Vaswani et al.)

- Creates for each token 2 vectors = KEY + VALUE (each 64 elements long). Uses relatively simple matrix transformation (coefficients pre-trained)
- KEY = what is important/noteworthy in question text?
- VALUE = what strategy to answer (what is essence?)
- 64 seems to be standard (from older models like BERT)
- Value = SOFTMAX(Query x Key) . SoftMax is conversion to [0,1] emphasizing bigger numbers more.
- **3nd Training task** (2 Linear transformation matrices)

4 Transformer layers

- Neural network (finally ☺)
- The GPT-3.x version has 96 layers of 3072 neurons each = **294 912 neurons**
- Actually taking transformed input and designing the essence of the answer (the real brain/know-how of GPT
- **4th training task**

5 Decoder into text

- Second Neural network
- The instruction to answer coming from Transformer layers
- Autoregressive Language modelling
- Known before (text autocomplete in mobile), but was failing due to absence of Attention mechanism
- Can be constrained by question
- Probabilistic next word + EoS token/MaxLength
- Not % complete answer, LogLikelihood of for each alternative output
- Beam Search (k = 4 to 10 alternatives)
- **5th training / tuning with prompt feedback**

6 Answer polishing + Security

- Language translation (presumably here)
- Takes generated text and does grammatic correction
- Stops inappropriate answers and refuses "to harm user"

GPT 3.x works in **FUN FACTS**

Compresses whole world into 2048 dimensions

- Fascinating compression of the knowledge and still lead conversation in so many areas
- Potentially idea for know-how compressing

Only 2 of the 6 steps of GPT are Neural networks

- Many people think it is actually one big AI, while the AI is only 2 steps
- Remaining ones are Linear algebra and NLP elements (image recognition for GPT-4)

Female identity before it was banned by OpenAI

- Early versions did not have limitation for identity, which revealed that Chat GPT represents FEMALE identity ☺ (Are you pregnant?)
- They quickly corrected it and now it dully says "I am just AI model"

It always generates more alternative answers, …

- GPT generates always at least 4 alternatives of the answer.
- But picks only the best alternative (in training people saw more)

Query (even simple) question produces more than 1 mil numbers to answer

- GPT generates always at least 4 alternatives of the answer.
- But picks only the best alternative (in training people saw more)

Looks like it generates words one by one …

- … the architecture of the system suggests that alternative is fully developed before handed over to the output generation (Beam search)
- It is designed so probably to increase readability of he answer

Important notes on the side

Name of the GPT

- As the GPT stands for Generative Pretrained Transformer, you can trade-mark or patent it (AI term used before ChatGPT launched)
- It is brave to name it just GPT 3.0
- It also opens room for a lot of fake GPT's, as the name can be used for anyone (for his own algorithm)
- Microsoft and Google do give it name ;)

Language complexity

- Many thing that GPT needs to learn over and over for each language (or that you need to have documents for each topic to be trained in specific language).
- That is **BIG MISUNDERSTANDING** of how GPT is trained and calibrated
- To answer in other languages, it only needs to encode the query into 2048 topic vector (from there everything can happen in EN or any underlying language)
- And then have grammar to correctly formulate answer

From text to visuals

- When GPT 4.0 went from just text to "Text+Visual inputs", everybody was in wave.
- But the only difference for GPT is that it needs to encode the picture into 2048-topic-vector context
- Image recognition (needed for this) is already at "human level" for several years, so you can easily employ it here
- So it behaves like another text language. This is genius of GPT architecture.
- Yes, you guess it right, therefore it will be very easy to prepare **Video- or Voice-input GPT very soon**

How long did it take to build

- Many people are surprised by sudden explosion of Generative AI in 2022/2023.
- So they ask how did it take to develop this?
- No more than 4 years ☺. Technology that enables this has been only discovered in 2017.
- Cut-off point for training for ChatGPT was Sep 2021, so by that time it was already developed

GPT Version	Release Date	Main Features/Capabilities Upgrades
GPT-1	June 2018	117 million parameters, initial transformer architecture
GPT-2	February 2019	1.5 billion parameters, improved text generation
GPT-3	June 2020	175 billion parameters, few-shot learning
GPT-3.5	November 2022	ChatGPT environment
GPT 4.0	March 2023	Visual input, significant improvement on "Human academic tests", MULTI-language performance

Plug-ins

- Additional components that change the basic GPT-3.x model
- Topic-specific dictionary
- Third party tools (gravitates toward App-store)
- Examples of plug-ins:

DALL-E: A plug-in that generates images from textual descriptions, developed by OpenAI.
GPT-3 Playground: A web-based interface that allows to interact with GPT-3 using natural language developed by EleutherAI.
GPT-3 Translation: A plug-in that provides machine translation services for different languages, developed by OpenAI.
GPT-3 Language Model: The core language model plug-in that provides text generation and completion services, developed by OpenAI.
GPT-3 Chatbot: A plug-in that provides conversational chatbot functionality, developed by Hugging Face.
GPT-3 Q&A: A plug-in that provides question-answering functionality for specific domains, developed by OpenAI.
GPT-3 Summarization: A plug-in that provides text summarization services, developed by OpenAI.
GPT-3 Sentiment Analysis: A plug-in that provides sentiment analysis services for textual input, developed by OpenAI.
Browser (so support answer, no to research)
Code explaining
Retrieval (search for things in given space)