ADVANCED DATA MANAGEMENT

Gianluca Oldani – Tutoring 6

What will be discussed today

- Litte excursus on why symbol frequency is important
- NLP – Semantic

Recap of previous lesson

- We understood that the frequency of a term carries information about it
- TF-IDF is based on this very idea as well as the concept of stop-words
- But is there some real world evidence of this value?

Introducing ciphers

Ciphers, also called encryption algorithms, are systems for encrypting and decrypting data. A cipher converts the original message, called plaintext, into ciphertext using a key to determine how it is done.

Caesar Cipher

- One of the earliest known ciphers
- Also known as ROTX, where X is the number of places a letter is shifted (e.g., ROT13)

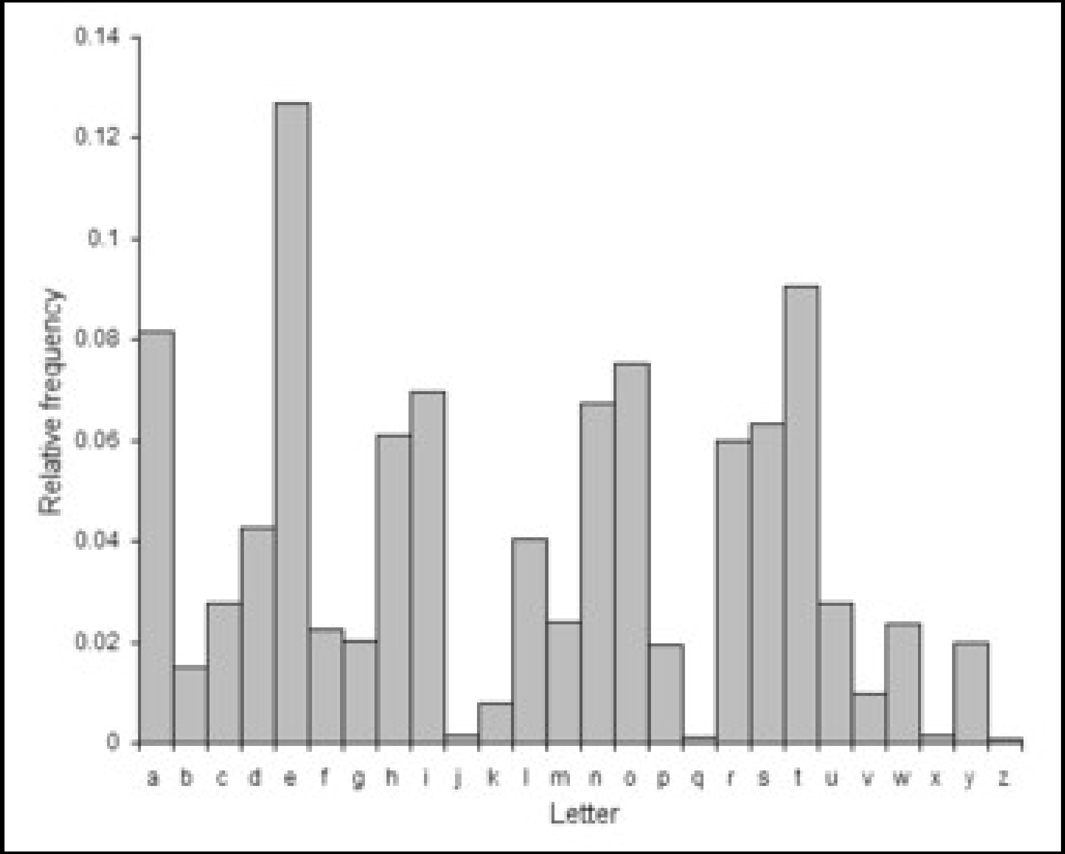
Caesar Cipher

- The main problem with this approach is that the **secret key** is not big (same size of the language alphabet)
- In what is called a **white box** scenario, this algorithm is weak to brute force attacks
- What about **black box**?

Caesar Cipher

- Even in this case, there is not a lot to do: the algorithms perfectly preserves the frequency of the letters in a used language
- Through frequency analysis it is possible to clearly see that this cipher has been employed and retrieve the key to obtain the plaintext

Caesar Cipher

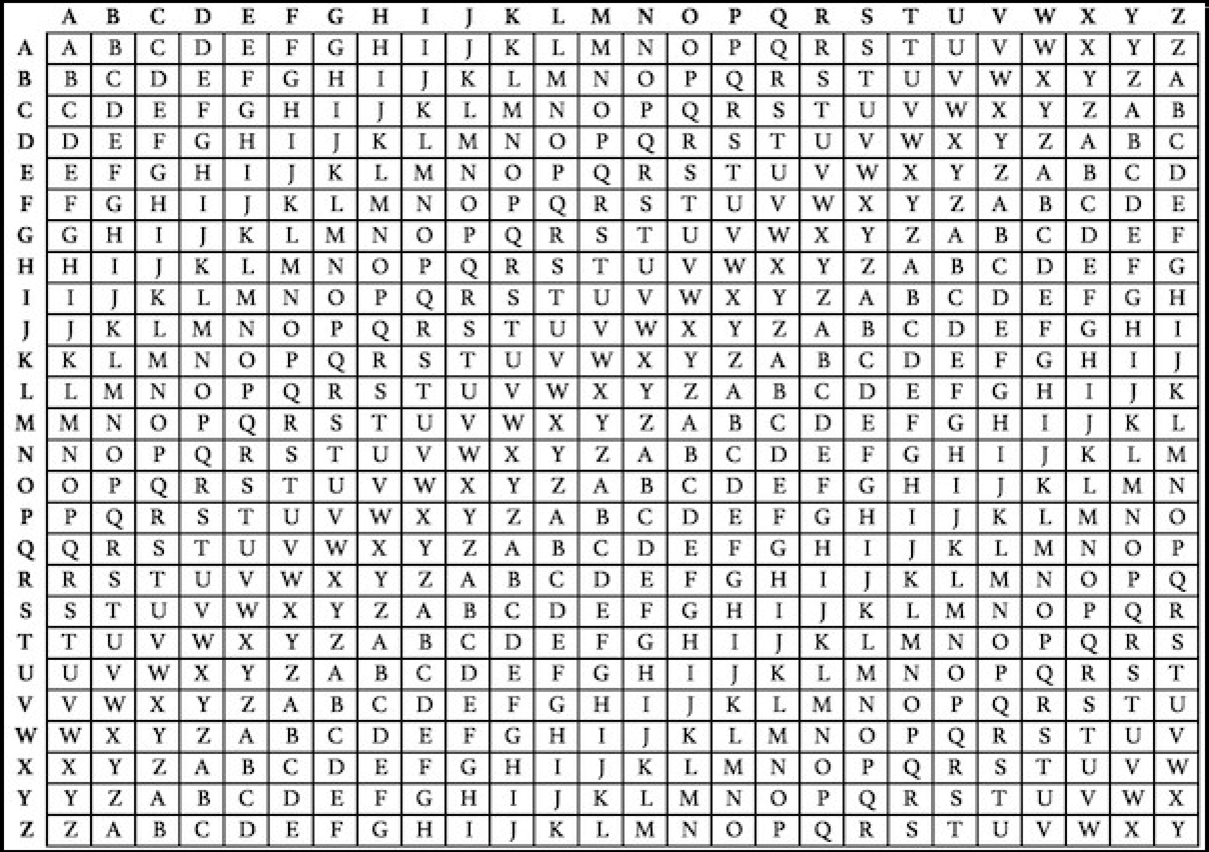


Brutus: let's go to the senate

Caesar: wait what's that knife

Brutus:

Vigenère Cipher



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

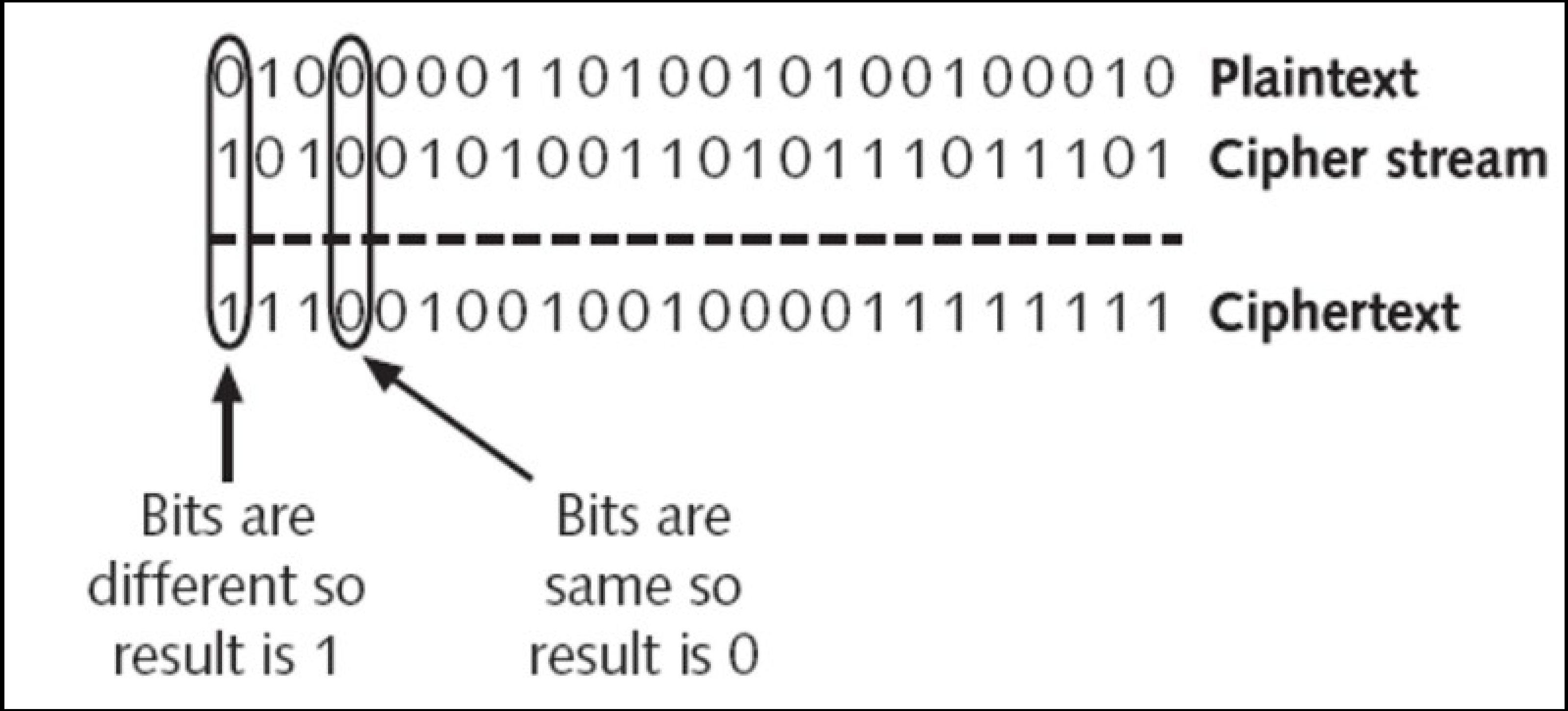
Vigenère Cipher

- The idea to break it similar to Caesar but with one more step: it is necessary to retrieve the length of the **key**
- This can be done by retrieving repeated sequences in the ciphertext, since, if the repetition is smaller than the secret key, repeated **sequences** in the ciphertext correspond to repeated **sequences** in the plain text
- One way to automate this procedure is the following:
  - Create letter groups of different sizes, these sizes are guess of the actual keysize
  - For each size, measure the syntatic distance between some of the groups (Hamming or Edit, or Hamming using bytes)
- Once the keysize is known, we simply return to the case of multiple Caesar ciphers: group the symbols that have been shifted by the same letter, check their frequency and retrieve the actual letter; alternatively, brute force the single letter of the key and check that the obtained plaintext group has the letter frequency of English

Classify if a piece of text is English

- **ETAOIN SHRDLU:** it is the approximate order of frequency of the 12 most commonly used letters in the English language
- In the past, it appeared on discarded lines of an articles, since the letters on type-casting machine keyboards were arranged by descending letter frequency to speed up the mechanical operation of the machine
- If used to distinguish between plaintext and ciphertext, also the number of spaces is a great indicator of text that is actually written in English and not random bytes

Introducing XOR ciphers


0100000110100101001000010 Plaintext
1010010100110101110111101 Cipher stream
1110010010010000111111111 Ciphertext
Bits are different so result is 1
Bits are same so result is 0

Mapping between XOR and letters

- Caesar cipher: Single-byte XOR
  - 1 Byte is used to XOR each Byte in the plaintext
- Vigenère cipher: Repeating-key XOR
  - A sequence of <KEYSIZE> Bytes is used to XOR <KEYSIZE> Bytes of plaintext at each iteration

Even XOR can be broken

- Caesar cipher: Single-byte XOR
  - Brute force can be used: just try all the 256 possible Bytes
  - End the brute force procedure when something is English (use **ETAOIN SHRDLU**)
- Vigenère cipher: Repeating-key XOR
  - Find the KEYSIZE using the Hamming distance of Bytes
- Break each Byte in the secret key like it is a Single-Byte XOR (**ETAOIN SHRDLU**)

Back to the main track: NLP

- All the exposed techniques do not take into account the semantic aspects of a term
- For this kind of approaches the sentences
  - The apple is red
  - The carpet is covered in red stains

  are more similar than
  - The cup is full of orange juice
  - The glass is filled with a liquid made of oranges

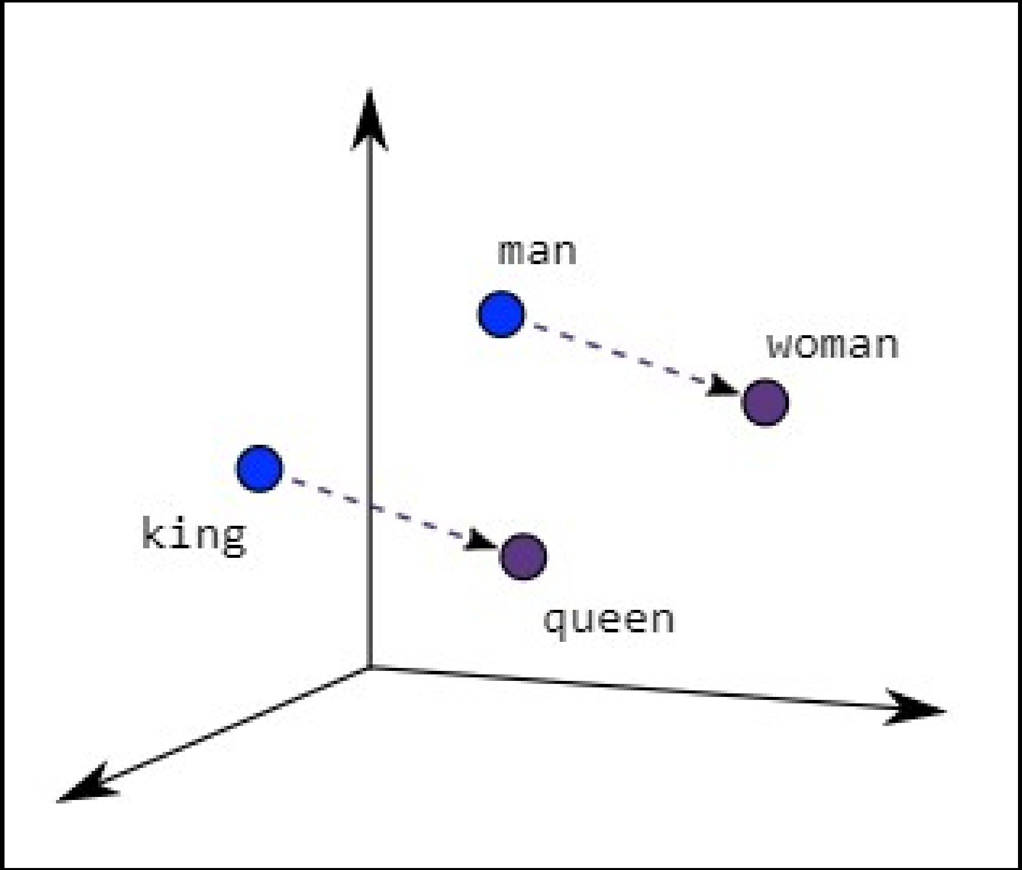
The basics: Word Embeddings

- **Word embeddings**: a technique where individual words are transformed into a numerical representation of the word (a vector)
- The vector of a word tries to capture characteristics of that term, such as definition and context
- **BoW** is a Word Embedding technique, but with several limitations:
  - Does not convey actual information about the context
  - Becomes larger depending on the size of the analyzed corpus of text

Word2Vec

- Proposed in **Efficient Estimation of Word Representations in Vector Space**, 2013 by a research group of Google
- **Goal of the paper**: to introduce techniques that can be used for learning high-quality word vectors from huge data sets with **billions** of words
- **General idea**: based on a word occurrences in the text, it is possible to estimate its meaning

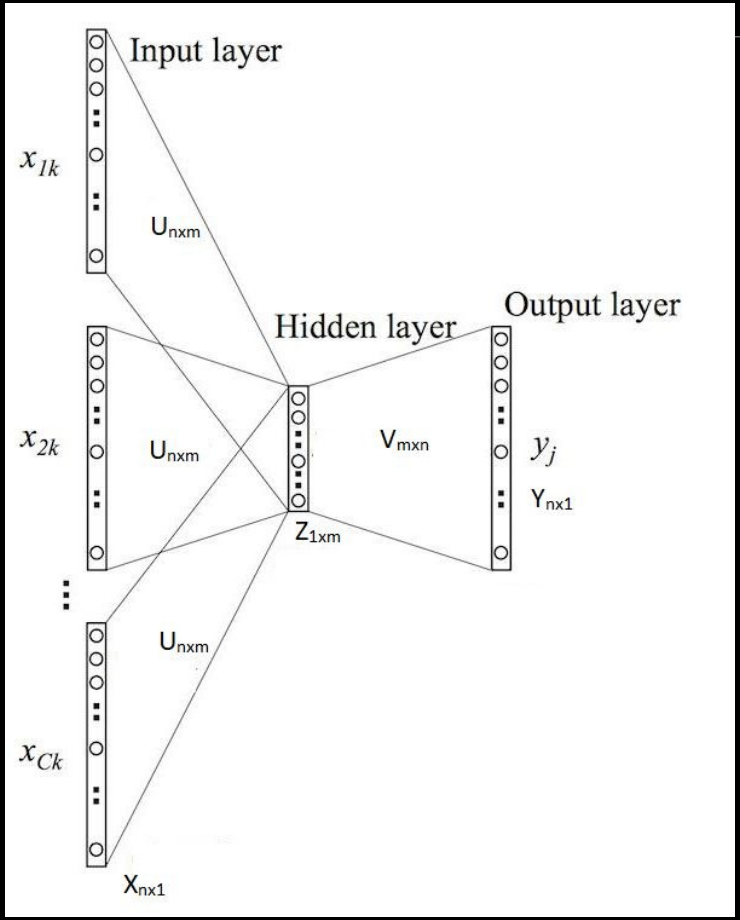
Word2Vec: visual representation



Word2Vec: architecture

- Word2Vec has two possible implementations (both are neural networks):
  - **Continuous Bag of Words (CBOW)**: the objective of this model is to predict the probability of a word appearing given the context
  - **Continuous Skip-Gram Model:** the objective of this model is to predict a context given the position of a word in a sentence

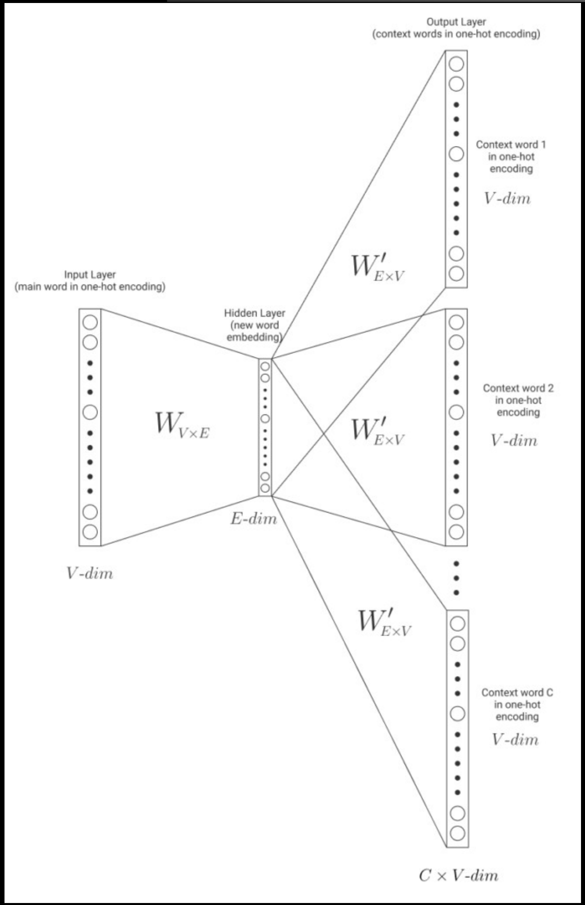
Word2Vec: CBoW



Word2Vec: CBoW

- Input layer: multiple 1-hot vectors where the 1 corresponds to the word in our text corpus that occupies the position corresponding to the vector
- Output layer: 1-hot vector where the 1 corresponds to the word in our text corpus for which we are learning the word embeddings
- Hidden layer: vector of weights from the context. It is what we are trying to learn and what will be actually stored as a vector

Word2Vec: Continuous Skip-Gram



Word2Vec: Continuous Skip-Gram

- Input layer: 1-hot vector where the 1 corresponds to the word in our text corpus for which we are learning the word embeddings
- Output layer: multiple 1-hot vectors where the 1 corresponds to the word in our text corpus that occupies the position corresponding to the vector
- Hidden layer: vector of weights from the context. It is what we are trying to learn and what will be actually stored as a vector

Word2Vec: the right terminology

- These kind of neural networks is usally referred as **Autoencoders**
- Their objective is to try to learn a compressed representation of an input
- Usually the hidden layer of a neural networks have a larger dimension compared to the input layer
- As seen for the two previous architectures, this is not the case for autoencoders

Word2Vec: advantages

- Both previously mentioned problem have been solved:
  - It is not necessary to store the entire text corpus and obtain only 1-hot vectors → only the hidden layers are now necessary
  - The vectors are able to capture part of the semantics of a term
  - Once a term has been encoded through word embeddings, it is possible to do any operation on it (e.g., feed it into classifiers, check synonyms, check for equality...)

Word2Vec: which is better

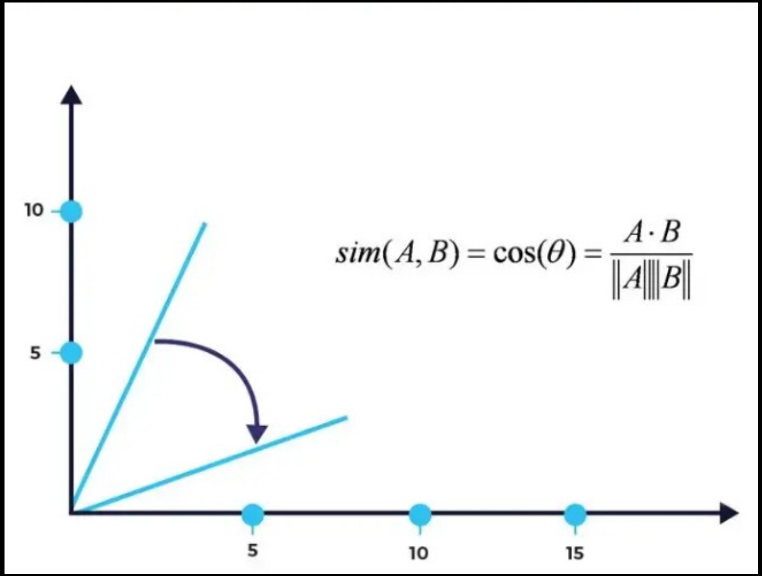
- The original paper reports the following:
  - Skip-Gram works well with small datasets, and can better represent less frequent words
  - CBOW is found to train faster than Skip-Gram, and can better represent more frequent words

Word2Vec: which is better

- The original paper reports the following:
  - Skip-Gram works well with small datasets, and can better represent less frequent words
  - CBOW is found to train faster than Skip-Gram, and can better represent more frequent words

Word2Vec: evaluate similarity

- The most simple method to evaluate the similarity between word embeddings is **Cosine similarity**



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Word2Vec: limitations

- A word to be evaluated must be in the training dataset → rare words may be unknown to the model
- Polysemy: a word may have multiple meanings based on the context (should correspond to multiple vectors at the same time)

Word2Vec: solutions

- fastText: is able to handle out of vocabulary terms, since it is based on n-grams instead of full words
- ELMo: takes into account the whole sentence to produce the embedding at runtime → but this IS NOT WORD EMBEDDING

Introducing Sentence Embeddings

- Usually, the meaning of an entire sentence can not be expressed through a single word
- A naive way to obtain sentence embeddings from word embeddings would be to sum or average each of the components in the phrase
- This, in the space of the word embeddings, simply summarize the entire sentence as a single word

Existing models for Sentence Embeddings

- **ELMo**: learns word embeddings on the fly and based on them generates sentence embeddings
- **InferSent**: word embeddings are not used, what is produced in directly the sentence embeddings
- **Sentence-BERT**: state of the art model, but very big and very slow (65 hours for 10,000 sentences)

The final push: language barriers

- Usually, both word embeddings and sentence embeddings are trained on a text corpus of a single language
- Using embeddings of a different language would be of no use (same word but context never seen)

The final push: possible solutions

- Naive: train only one language, translate all the languages into the central one and use its embeddings
- Align different spaces: some terms are general and easy to translate (animale → animal, linguaggio → language). Learning how to align these couples of vectors should leadt to the alignment of the entire space of the embeddings
- Both of these methos are not able to handle expression that are meaningful only for a specific language and lead to bad translations: **Once in a blue moon** → Una volta in una luna blu

The final push: state-of-the-art models

- These two methods are able to learn language agnostic embeddings through parallel sentences provided in the training set:
  - LASER: Language-Agnostic Sentence Representations
  - m-USE: Multilingual Universal Sentence Encoder