Representation of Words in NLP: A Historical Perspective

From Symbols to Contextualized Embeddings

# Introduction

NLP stands for Natural Language Processing, and it is a subfield of Artificial Intelligence (AI) that focuses on the interaction between computers and human languages.

It is a cross-disciplinary field between computer science and linguistics, and it consists of mainly 5 subareas: Natural Language Understanding, Natural Language Generation, Speech or Voice recognition, Machine Translation, Spelling Correction and Grammar Checking.

Its historical evolution follows the typical path of most AI subfields: from symbolical models to statistical models to current neural networks, accordingly, changing how words were represented.

# 1950s – early 1960s

1949: Weaver’s Memorandum on Machine Translation (MT) à beginning of NLP research

1954: Georgetown-IBM experiment: English-Russian translation algorithm, 60 sentences

Basically dictionary-lookup algorithms

# Late 1960s-1970s-Early 1980s

**Late 60’s**

Following Chomsky’s introduction of generative grammar, the theoretical focus was on how to represent meaning and create computationally tractable theories of grammar. From the practical point of view, the main paradigm was the creation of explicit rule-based systems.

An example of these systems is ELIZA (1996): a rule-based simulation of a Rogerian psychotherapist.

**70’s**

The 70’s witnessed the introduction of conceptual ontologies: real-world information was structured into computer-understandable data, reflecting concepts and their relations. In these years the first chatterbots were created.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteAn example is PARRY (1972): a concept and beliefs-based simulation of a person with paranoid schizophrenia, which also included a conversational strategy.

Here we can see fragments of a conversation in 1972 between ELIZA and PARRY. We can see that the conversation has a lo of repetition and, in particular for ELIZA, it can be ungrammatical.

**Early 80’s**

Until the early 80’s symbolic approaches dominated the scene: rule-based systems, formal grammars, parsing algorithms, predicate logic.

## Word representation: symbols

Words up to this point are not represented through numbers but through strings of characters.

**Pros and cons**

+ These representations are highly interpretable and allow explicit knowledge representation through rules. They also allow rule manipulation and customization and can easily incorporate domain-specific knowledge.

- These representations are complex to create ad require high maintenance, becoming a time-consuming task and lacking scalability to real world applications. They also capture limited semantic information and cannot effectively resolve word or sentence ambiguities.

**When to use**

Even though rarely used, symbolic models can be applied in all tasks where explicit and interpretable rules are needed: parsing, basic machine translation, tasks involving grammar and syntax checking, specialized domains with well specified rules (medical or legal texts).

# Late 1980s – 1990s – 2000s

**Late 80s**

In the late 80’s, the dominance of Chomskyan theories decreased. At the same time, more computational power was at disposal. The statistical approach to language thus became more popular.

Instead of writing explicit rules, the focus was on extracting probability distributions from text data and create language models which would compute the probability of either a word or a sequence of words.

**90s**

The 90’s saw a rise in corpora linguistics and use of datasets. This decade saw the introduction of supervised machine learning techniques and the statistical approach was shown to be successful for machine translation, but the amount of data available was still limited.

**2000s**

By 2000, previous efforts resulted in the introduction of Penn Treebank and other annotated large corpora, allowing for a more widespread usage of statistical parsing techniques.

With the growth of the web, there was a great increase in raw (not annotated) data, shifting the research toward semi-supervised/unsupervised learning, which still is an open field.

## Word representations: Bag of Words (BoW)

In order to use statistical techniques, words need to be represented through vector encodings (informally, a sequence of numbers obtained by means of calculations).

The most intuitive approach is to treat a corpus as a “bag of words”: create a sorted vocabulary of all unique words present in the corpus and represent each word with a fixed length vector obtained based on counting frequency. There are different types of encoding in this approach.

**One-hot encoding**

Each word is represented by a vector of vocabulary length with all zeros, except for its position where it has a 1.

Example:  
Let’s take the document: [“today it rained. today I ate pizza”]. My vocabulary will be [ate, I, it, pizza, rained, today]. The encoding of the word “ate” is [1,0,0,0,0,0], the encoding of the word “pizza” is [0,0,0,1,0,0].

This type of encoding can be easily extended to sentences: sequences of 0 and 1, based on whether a word is present or not.

Example:  
“today I ate pizza” is encoded as [1,1,0,1,0,1].

One-hot encoding do not reflect the frequency of words in a document or throughout a corpus, just their presence.

**Count vector**

In order to incorporate frequency information, the corpus is represented as a matrix D\*V, where D is the number of Documents and V is the vocabulary length.

Example:  
Let’s take a corpus of three documents: D1 = [“today I ate pizza”], D2 = [“ I ate pizza and you ate pasta”], D3 = [“today it rained”]. The vocabulary will be [and, ate, I, it, pasta, pizza, rained, today, you], made of 9 words. The resulting count matrix will have 3 rows and 9 columns.

Immagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente

The entry (i,j) of the matrix represents the number of occurrences of the j-th word in the i-th document.

Example:  
The entry (2,1) is the number of times the word “and” is present in D2 = 1.  
The entry (2,2) is the number of times the word “ate” is present in D2 = 2.

Instead of a binary encoding, each word is then represented by the corresponding column in the matrix, resulting in a vector of length D where each entry is the count of that word in each document.

Example:  
The encoding of “ate” is [1,2,0].

This method allows also the encoding of single documents through the corresponding row in the matrix, resulting in a vector of length V, where each entry is the count of each word in that document.

Example:  
The encoding of D2, [“I ate pizza and you ate pasta”], is [1, 2, 1, 0, 1, 1, 0, 0, 1].

This method, despite considering frequency, treats all word as equivalent. However, we know that function words are less important to understand a document, despite being way more frequent. It can also happen that all documents in a corpus are about the same topic and words related to that topic become less relevant for discerning them.

**TF-IDF vectors**

TF-IDF algorithm produce representations which consider the importance of a word across the document and the corpus. Each entry of the corpus matrix considers two aspects: Term Frequency (the count we have seen before divided by the number of words in the document) and Inverse Document Frequency (the logarithm of the total number of documents divided by the number of documents which contain the word). These two numbers are then multiplied.

Example:  
Taking the same corpus as before, let’s compute the entry (1,2): TF-IDF for the word “ate” in D1. TF=1/4 (1 “and” over 4 words), IDF=log(3/2) (3 documents, 2 contain the word). Entry (2,1)= 1/4\*log(3/2) ≈ 0.044.

Words and documents are represented by columns/rows as before.

**Pros and cons**

+ BoW encodings are simple and easy to implement, they produce sparse vectors (mostly made of zeros) which are efficient to store in memory, they are scalable for many documents. No training is required: the encodings are just based on counting, making the representation more interpretable.

- BoW encodings produce high dimensional (very long) vectors, resulting in easy overfitting of machine learning methods starting from them (the models learn corpus-specific noise). They fail to capture order and grammatical structure and don’t yield any semantic information. It is difficult to introduce Out Of Vocabulary (OOV) words.

**When to use**

BoW embeddings can be used in all downstream tasks which do not require structure or sematic understanding, relying just on word frequency: classification tasks (e.g., spam filtering), information retrieval (keyword matching), simple sentiment analysis, document labelling, keyword extraction.

## Word representations: Co-occurrences

To encode semantic information, we can base the encodings on the idea underlying distributional semantics: similar meaning will lead to words being found in similar context.

We can thus create a co-occurrence matrix of dimension V\*V, where each entry corresponds to a couple of words and it is the number of times in which the two words occur together, within a fixed distance called context window.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteExample:  
Let’s take the same corpus of before and let’s fix the context window to 2 (count how many times the words are near each other.   
The co-occurrence of the couple (I,ate) is 2, the co-occurrence of the couple (end,ate) is 0. If we broaden the context window to 3, the co-occurrence of (end,ate) becomes 1.

Here is the full co-occurrence matrix for the example. We can see that most of the entries are zeros.

Sometimes, the matrix can be reduced to dimension V\*N, where N is a subset of words in the vocabulary obtained by removing function words.

The co-occurrence matrix is then usually decomposed through mathematical methods into smaller dimension matrixes, thus allowing to obtain lower dimensionality, dense vectors (opposite of sparse: almost no zero entries), which capture the same meaning (decomposition selects the components with higher variance) and are taken as the final encodings.

**Pros and cons**

+ Co-occurrence encodings require no training, are low dimensional and can be used as features for machine learning algorithms. They also partially capture semantic similarity and introduce context understanding into the encodings.

- Co-occurrence matrixes are large and require a lot of memory to be stored. Moreover, decomposition can be computationally expensive. The patterns represented can be influenced by the choice of context windows and word order is still not considered.

**When to use**

Co-occurrence encodings are useful in tasks where capturing semantic information and similarity between words is important: analogy, machine translation, recommendation systems. They can also be applied to clustering tasks (grouping similar documents together), word sense disambiguation, Named Entity Recognition.

# Excursus: vector spaces and cosine similarity

With the last statistical approach, we talk about dense vectors and word similarity. But how do we represent it?

Let’s visualize it in lower dimension: imagine our words are encoded only 2 numbers. We can represent each word with a point in cartesian plane, the vector corresponding to the word is an arrow from the origin of the axes to the point.

Immagine che contiene linea, diagramma, Diagramma, Parallelo

Descrizione generata automaticamenteIdeally, we will have words with similar meaning represented by points that are closer together. But how do we measure the similarity between two vectors? Through cosine similarity: a measure of the angle between them. The smaller the angle, the more similar the two vectors (and, theoretically, the meanings they represent).

Small example of vectors and angles

# 2010s

With the beginning of the 2010s, the field of Deep Learning began to emerge. Until 2012, attempts to scale up neural networks to more than few layers have been very unsuccessful. That year, however, Hinton and his team managed to win an important challenge in the field of Computer Vision (ImageNet) by using a deep-layered neural network, showing for the first time that this kind of approach to ML was not only feasible, but also amazingly successful.

This led to a revolution for NLP as well. Neural Networks started to be applied to tackle traditional problems in this field.

NNs started to be applied also to distributional semantics, as engines that could be trained to learn numerical representations of words. These are Word embedding models.

Generally, word embeddings models adopt different strategies, but they have some key points in common. They are trained on a corpus which contains the instances of words within a variety of contexts: the higher the vocabulary (the number of unique words in the corpus is) and the variety of instances for each word, the better. From this corpus, they learn to represent the meaning of each word as a numerical vector, using different strategies.

* Idea of neural embeddings: predict rather than count.

NOTE: fundamental difference compared to encoding methods presented before

**Tokenization**

Word tokenization is the process of splitting a sentence into individual words or tokens. This process is a fundamental step in natural language processing (NLP) and is used to prepare text data for further analysis. The history of word tokenization dates to the early days of NLP research, and there have been many approaches to this problem over the years.

One of the earliest methods for word tokenization was to simply split text on whitespace characters such as spaces and tabs. However, this approach has several limitations, such as not being able to handle punctuation marks correctly. Later, more sophisticated methods were developed that could handle punctuation marks and other special characters more effectively.

With the development of neural embeddings, more work has been done to improve the tokenization even further: algorithms started to split words more smartly, splitting words into subwords, by separating roots from their prefixes or suffixes using heuristic strategies.

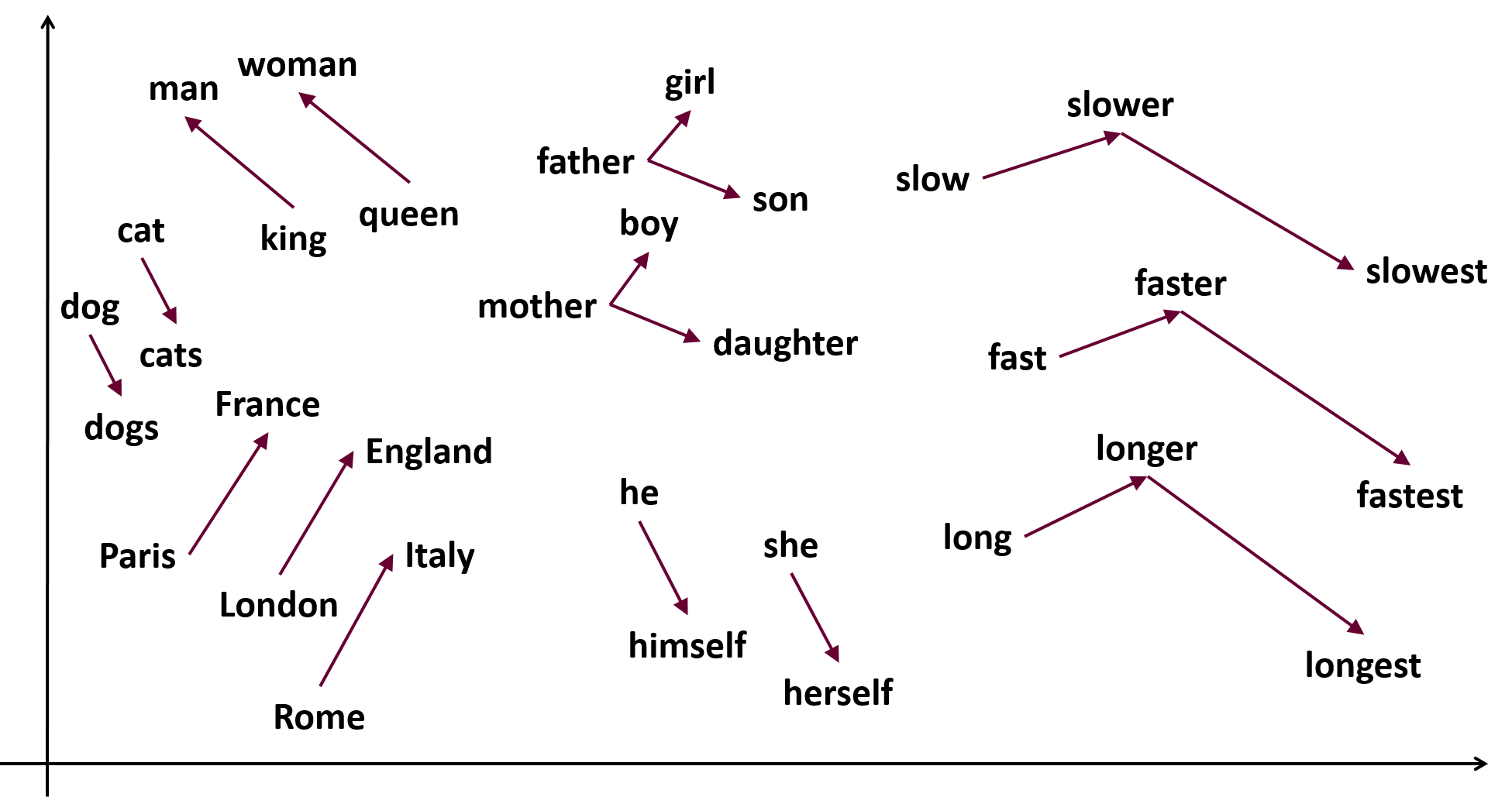
A perfect tokenizer yet doesn’t exist.

Let’s see some of the most important word embedding models:

**Word2Vec (2013)**

* One of the first word embedding models
* Idea:

1. Train a classifier (self-supervised) to predict current or context words.
2. Discard the classifier.
3. Use the learned weights as word embeddings.

* Two algorithms proposed:
  + **CBOW**: predict the current word given the context.
  + **SkipGram**: predict context words given the current word.
* 

**FastText (2017)**

* An extension of Word2Vec that generates also subword-embeddings.
* Idea:

1. Learn representations for character n-grams.
2. Represent words as the sum of the n-gram vector.

* Improvement: accounts for the internal structure of words (“dog”, “dogs”, “dogcatcher”, ..)
  + How?
    - Each word is represented as a bag of character n-grams.
    - Example: w=where and n=3 => {<wh, whe, her, ere, re>, <where>}
    - Note: “<” and “>” represent respectively starting word and ending word

Pros and cons

Pros: advantages of Word2Vec, but more precise: it also generates subword embeddings.

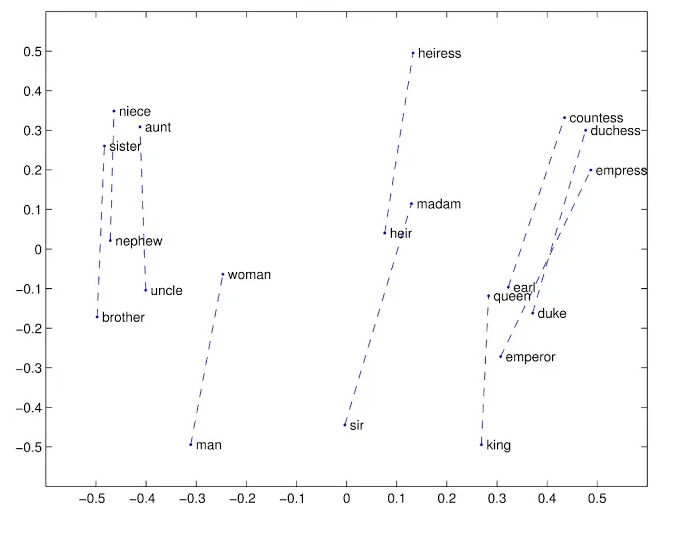
Cons: the same as word2vec, the problems have not been solved, but only slightly reduced.

**GloVe (2014)**

* Unsupervised algorithm
* Idea:

1. Generate word embeddings by aggregating word-word co-occurrence matrix from a corpus.
2. The resulting embeddings show interesting linear substructures of the word in vector space.

* An example of the vector space learned by GloVe.



* Pros and cons
  + Pros:
    - Highly scalable
    - Captures linear substructures in the vector space.
    - More context info compared to Word2Vec or FastText
  + Cons:
    - The context information contained in the embeddings are still not sufficiently good.

**The drawbacks of using static embeddings**

The development of these first neural embeddings allowed models to reach unprecedented levels of accuracy. However, they presented a fundamental problem. They were **context-independent** or **static**: the word embeddings were computed once and made available for download.

This means that the value associated to each word was always the same regardless of the context in which the word was present. If, for example, I had a dataset containing the word “bank”, and I transformed it by using one of these embeddings, every instance of the word “bank” in the dataset would be converted into the same vector everywhere. This is not desirable because it’s an ambiguous word, and we would like to take this into consideration when we build our embeddings.

**Contextualized embeddings**

* Embeddings generated by considering the context in which words are present.
* Two phases: RNN-based and Transformer-based embeddings.

**RNN-based contextualized embeddings**

**CoVe (Context Vectors) (2017)**

* Uses a bi-directional LSTM to generate 1 layer as a representation.
* Idea:

1. Start from GloVe representations.
2. Train a LSTM model for Machine Translation (MT-LSTM)

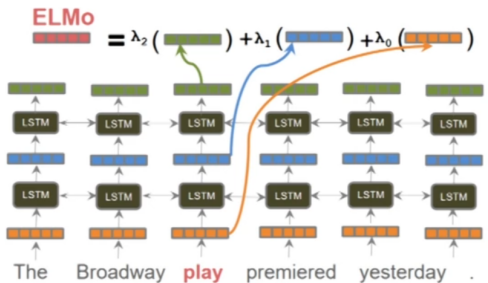
* Output: the encoder are CoVe vectors

**ELMo (2018)**

* Uses a deep bi-directional LSTM to generate multiple layers as a representation.
* Idea:

1. Start from raw word vectors computed via char-CNN.
2. Values go through 2-layer bidirectional LSTM.

* The final embedding is a task-specific weighting of all the layer outputs.



**Transformer-based contextualized embeddings**

The idea of computing contextualized embeddings is the most popular today.

However, instead of RNNs we use Transformer architectures.

In short, these architectures rely completely on the mechanism of Attention.

**The Attention Mechanism**

* Attention is the mechanism that governs Transformers and their learning: it allows these architectures to focus on specific parts of the input sequences and assign a weight to each word so that the words that are more important for the understanding of the sequence should have higher weights, while trivial ones (such as articles or conjunctions for example) should have a lower weight.

**The difference between Transformer-based embeddings and previous methods**

* Compared to the past, where we had models which were explicitly implemented for learning word embeddings, now learning embeddings is not necessarily a task that is performed by specific architectures but can be performed by the same network that learns the goal task without additional modules.
* In many cases, however, we don’t train a model from scratch, and we use as a base a large Transformer model which provide us with pre-trained embeddings. Starting from this base, we may fine-tune it to obtain representations that are more specific for our task.

**BERT** (2018)

* Transformer-based architecture
* It has been trained with MLM using jointly left and right contexts (bi-directional)
* It uses subword tokenization (wordpiece), words are split into tokens and subtokens according to a strategy that has not been disclosed by Google.

**GPT** (2018)

* Another Transformer-based architecture
* Trained with a generative pretraining objective, it learns what is the most probable word to output given the previous context. (uni-directional)
* It uses an implementation of the BPE tokenizer: it keeps the frequent words in the vocabulary, and split words into meaningful subwords.

# 2020s: Advancements in Word Embedding Techniques and Interpretability

By the turning of the decade, contextualized Transformer-based embeddings would replace completely older methods, and they continue to be the most relevant method today (2023).

However, new proposals about how to embed linguistic data continue to emerge, and it is not yet known what will replace them in the future.

Some of the current lines of research:

* **Neurosymbolic approaches**: combination of neural strategies with older symbolic. An expected advantage: higher interpretability and generalization capabilities. Experimental evidence seems to confirm this (Emb2Sym, but still too early to be certain).
* **Multimodal Embeddings**: combination of information from images and text together to enhance the contextualization of the embeddings.

**Focus on Multimodal Embeddings**

A very active line of research today attempts to use information from various modalities (images, videos, audio, neural data). The learning of better word embeddings of is one of them.

The modality that has been more frequently combined with text for this purpose is the visual one. Vision is one of the fundamental capabilities that a human has and plays a role in the way humans learn concepts.

The embeddings that are learned from combining visual and text data has been used both for empowering the capabilities of models for NLP and CV tasks, as well as combined tasks such as image retrieval.

Thanks to this, word embedding becomes even more contextualized: information from the images is encoded along with that from the surrounding words within sentences to ground a word into a scene.

However, these embeddings have a long way to go. They require more effort to be learned, as they require also visual data, and they still present clear limitations. They still struggle, for example, evidently for example to encode verb meanings.

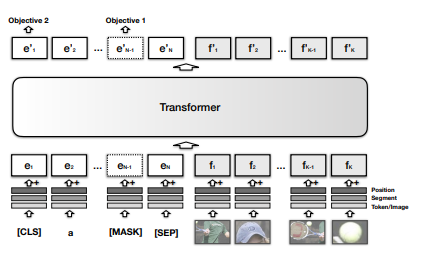
Let’s look at a couple of them.

**VisualBERT** (2020)

* Idea: combines BERT embeddings with visual information obtained by an object detector
* How:

1. Obtain the output from two models: BERT (language) and an object detector.
2. Combine them together.
3. Pass them to a Transformer that should learn useful alignments between them.

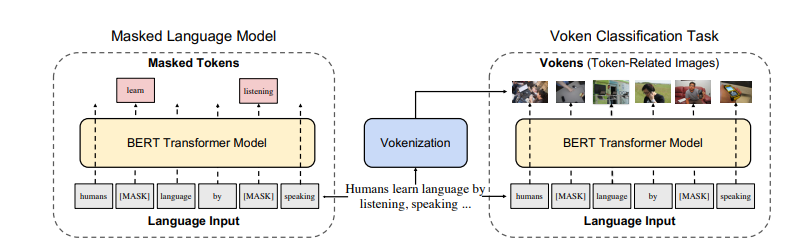
* The embedding will be the output of this Transformer.



**Vokenization** (2020)

* Idea: contextually map language tokens to their related images by taking language tokens as input and use token-related images as language supervision.
* How:

1. Train a BERT model with two objectives: Masked Language Modeling (predict the missing word in the middle of a sentence) and Voken Classification (assign to each input word the right Voken, which is defined by the authors as a visual image corresponding to a token)
2. Optimize the two objectives simultaneously and share the weights.

* The final embedding will be the output of the model.

# Recap

**Symbolic representations**

Represent words as strings of characters. Highly interpretable but time consuming and not scalable.

Even though rarely used, they can be applied in all tasks where explicit and interpretable rules are needed: parsing, basic machine translation, tasks involving grammar and syntax checking, specialized domains with well specified rules (medical or legal texts).

**Statistical representation: Bag of words**

First numerical representation: words as vectors, computed from the dataset. Treat documents as vocabulary, losing all order and semantic information. Efficient to store but too high dimensional.

All downstream tasks which do not require structure or sematic understanding, relying just on word frequency, are suitable for using these representations: classification tasks (e.g., spam filtering), information retrieval (keyword matching), simple sentiment analysis, document labelling, keyword extraction.

**Statistical representation: Co-occurrence**

Based on co-occurrence of words within a context window: capture local semantic information and structure. Still lacking order information, computationally expensive and highly influenced by length of window, but allow for dense representation in low dimensionality.

They are useful in tasks where capturing semantic information and similarity between words is important: analogy, machine translation, recommendation systems. They can also be applied to clustering tasks (grouping similar documents together), word sense disambiguation, Named Entity Recognition.

**Neural representation: Static**

The first generation of neural embeddings. They are generated by training models with large corpora, and they aim to extract embeddings for the words by leveraging the information contained within these corpora.

The embeddings learned are vectors within a space.

These embeddings present fundamental problems: they are built independently from the target task, and therefore they are always the same for every application. Moreover, they lack context: they struggle to represent the different meanings that a word might have.

However, they are useful when task-specific data or resources are limited and as initialization or baseline for downstream tasks (transfer learning). They can be used when the task does not involve rapid semantic change or ambiguity.

**Neural representation: Contextualized**

Given the problems of static embeddings, contextualized embeddings aim to capture the different meanings of words better by capturing the information in the context of each instance of a word. This results in considering words not just as vocabulary entries separated from the rest, but as a function of their context. Thanks to this, a word meaning is not global but becomes contextualized, and cannot be separated by its context.

The techniques for learning these embeddings have evolved during time, and continue to evolve today, (RNN-based, Transformer-based, multimodal), however the principles that led to their conception continue to be relevant.

They are employed in most NLP tasks, when a deep understanding of nuances and context-specific differences is essential.

**Neural representation: Multimodal**

Classical contextualized embeddings still present limitations in the context information that they encode. For this reason, enhancing these embeddings with information from other modalities and from images has gained popularity in the recent years.

The assumption is: humans ground language into what they see, hear, perceive. Models do not. We must allow models to ground language into reality like humans to improve their learning.

These embeddings have provided interesting results; however, they have not yet replaced those learned only on text. Training becomes more complicate, and there is a lack of communication between NLP and CV subfields of Deep Learning, although this has improved in recent years.

# REFERENCES

<https://www.icts.res.in/sites/default/files/media/media-library/NLPIntro.pdf> (historical overview)

<https://medium.com/@antoine.louis/a-brief-history-of-natural-language-processing-part-1-ffbcb937ebce> (historical overview)

<https://aclanthology.org/www.mt-archive.info/90/MTNI-1999-Hutchins.pdf> (Weaver’s memorandum)

<https://aclanthology.org/www.mt-archive.info/Garvin-1967.pdf> (georgetown experiment)

<https://dl.acm.org/doi/pdf/10.1145/365153.365168> (ELIZA)

[https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/abs/on-the-generality-of-parry-colbys-paranoia-model/CAEAAACBB8894DF9BB0ABC7CD7519D7C#](https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/abs/on-the-generality-of-parry-colbys-paranoia-model/CAEAAACBB8894DF9BB0ABC7CD7519D7C) (PARRY)

<https://www.rfc-editor.org/rfc/rfc439.html> (Eliza-Parry conversation)

<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/> (statistical word representation)

<https://arxiv.org/abs/1301.3781> (Word2Vec)

<https://www.samyzaf.com/ML/nlp/nlp.html>

<https://arxiv.org/pdf/1607.04606.pdf> (Fasttext)

<https://aclanthology.org/D14-1162/> (GloVe)

<https://medium.com/analytics-vidhya/word-vectorization-using-glove-76919685ee0b>

<http://ai.stanford.edu/blog/contextual/>

<https://arxiv.org/pdf/1708.00107.pdf> (CoVe)

<https://arxiv.org/pdf/1802.05365.pdf> (ELMo)

<https://arxiv.org/pdf/1810.04805.pdf> (BERT)

<https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf> (GPT)

[Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings (kr.org)](https://proceedings.kr.org/2022/44/kr2022-0044-aspis-et-al.pdf) (Emb2Sym)

[1908.03557.pdf (arxiv.org)](https://arxiv.org/pdf/1908.03557.pdf) (VisualBERT)

[arxiv.org/pdf/2010.06775.pdf](https://arxiv.org/pdf/2010.06775.pdf) (Vokenization)