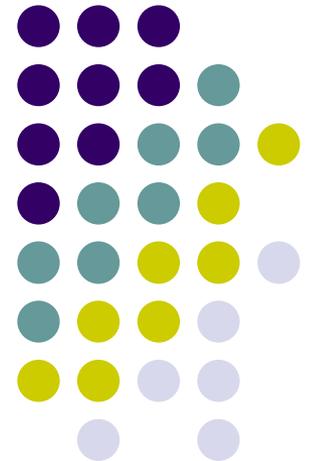


---

A practical introduction to  
word embedding techniques -  
and why social scientists  
should be interested in them





# References

- ✓ Rodriguez Pedro L. and Spirling Arthur (2022). Word Embeddings: What works, what doesn't, and how to tell the difference for applied research, *Journal of Politics*, 84(1), 101-115

The slides, script, packages to install and the dataset we will use this afternoon, are available here:

<http://www.luigicurini.com/applied-scaling--classification-techniques-in-political-science1.html>

# Beyond bag-of-words

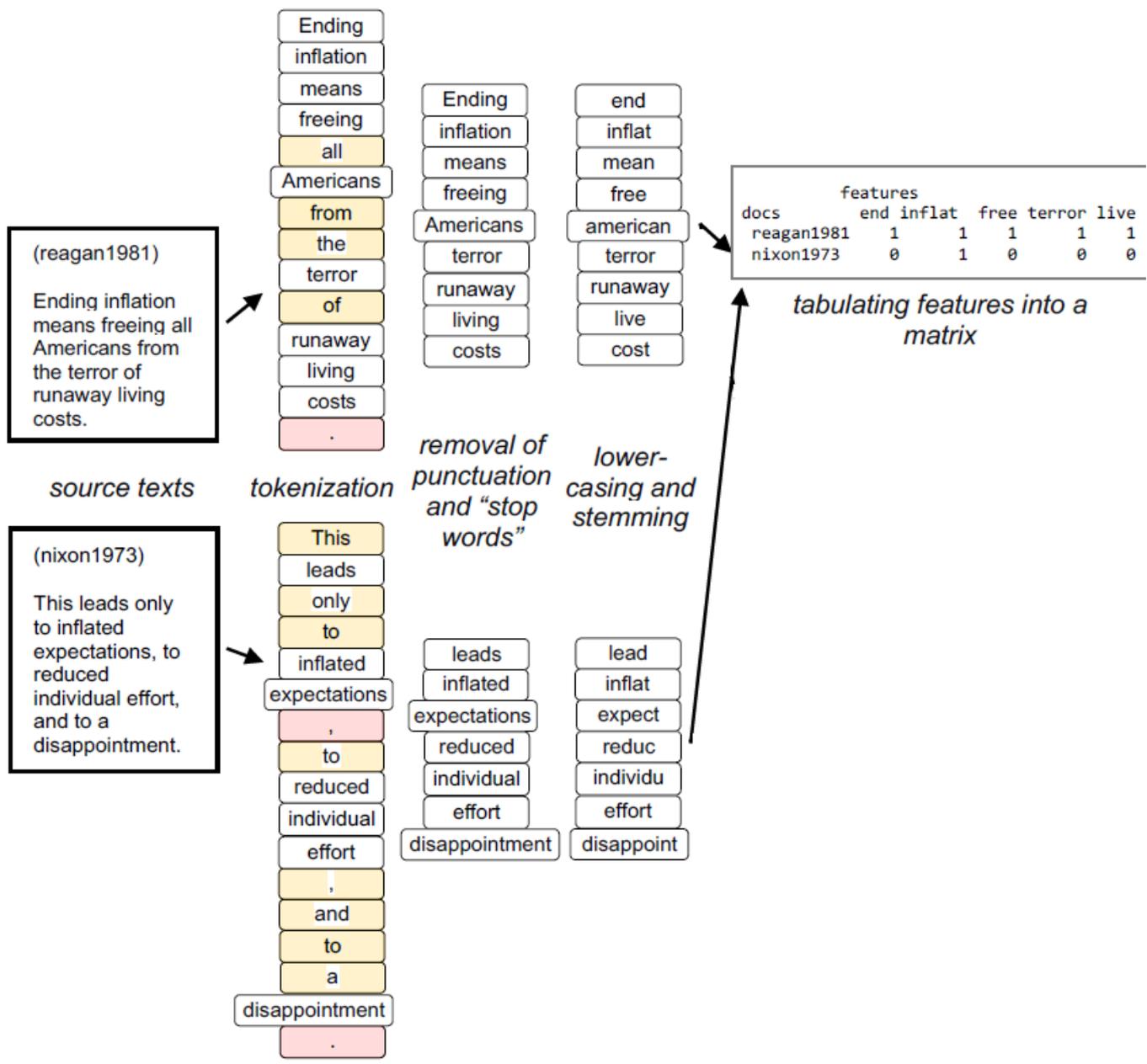


The **Bag-of-words (BoW)** approach has been for long time the “mainstream” one when dealing with text-analytics

The representation of texts according to BoW

- ✓ Focusing on frequency of features
- ✓ Ignoring context, grammar, word order

Multiple document vectors (one vector for each text) are then put together in a **document-feature matrix**, where each **row** represents a document and each **column** represents a unique word, or term



# Beyond bag-of-words



BoW approach discards therefore much linguistic information regarding the **surrounding syntactic and semantic context** of a given word in a sentence

This is wrong, but often useful!

Still, how bringing back the context in which a word appears? And does it matter?

One alternative: **word embeddings (WE)**

**WE approach** allows us in particular to move from non-positional to positional text-analysis



# What is word-embedding?

It's all began with...: “*You shall know a word by the company it keeps*” (Firth, 1957)

More concretely, a word's meaning can be garnered from its *contextual information*: literally, the other words that appear near it in text

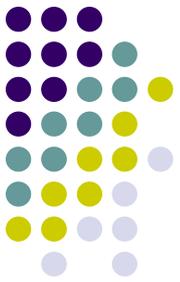
Here, a “context” means a *symmetric window* of terms around the word of interest

# What is word-embedding?

Let's consider the following three sentences:

1. *Kitty likes milk and huggings*
2. *Cat likes milk and fish*
3. *Lion eats meat not fish*

Let's create a **co-occurrence matrix (COM)**



# What is word-embedding?



If there are  $k$  features, a **co-occurrence matrix** is a  $k \times k$  matrix, where each cell tells us how frequently  $\text{word}_i$  occurs with  $\text{word}_j$  in some given window

In such instance the *window-size* determines the number of words, on either side of the focus word, to be included in its context

Let's consider a window size of 2

In Quanteda the command for doing it is `fcm`

# What is word-embedding?



|          | kitty | likes | milk | and | huggings | cat | fish | lion | eats | meat | not |
|----------|-------|-------|------|-----|----------|-----|------|------|------|------|-----|
| kitty    | 0     | 1     | 1    | 0   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| likes    | 1     | 0     | 2    | 2   | 0        | 1   | 0    | 0    | 0    | 0    | 0   |
| milk     | 1     | 2     | 0    | 2   | 1        | 1   | 1    | 0    | 0    | 0    | 0   |
| and      | 0     | 2     | 2    | 0   | 1        | 0   | 1    | 0    | 0    | 0    | 0   |
| huggings | 0     | 0     | 1    | 1   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| cat      | 0     | 1     | 1    | 0   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| fish     | 0     | 1     | 0    | 1   | 0        | 0   | 0    | 0    | 0    | 1    | 1   |
| lion     | 0     | 0     | 0    | 0   | 0        | 0   | 0    | 0    | 1    | 1    | 0   |
| eats     | 0     | 0     | 0    | 0   | 0        | 0   | 0    | 1    | 0    | 1    | 1   |
| meat     | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 1    | 1    | 0    | 1   |
| not      | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 0    | 1    | 1    | 0   |

The semantically similar words will have similar looking vectors (i.e., similar looking rows) to each other (at least more similar looking than the semantically dissimilar words) – compare for example *cat*, *kitty* and *lion* here. This usually becomes more evident as we increase our corpus size

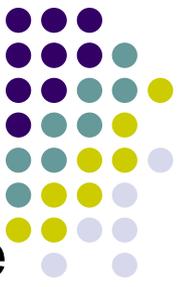
# What is word-embedding?



|          | kitty | likes | milk | and | huggings | cat | fish | lion | eats | meat | not |
|----------|-------|-------|------|-----|----------|-----|------|------|------|------|-----|
| kitty    | 0     | 1     | 1    | 0   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| likes    | 1     | 0     | 2    | 2   | 0        | 1   | 0    | 0    | 0    | 0    | 0   |
| milk     | 1     | 2     | 0    | 2   | 1        | 1   | 1    | 0    | 0    | 0    | 0   |
| and      | 0     | 2     | 2    | 0   | 1        | 0   | 1    | 0    | 0    | 0    | 0   |
| huggings | 0     | 0     | 1    | 1   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| cat      | 0     | 1     | 1    | 0   | 0        | 0   | 0    | 0    | 0    | 0    | 0   |
| fish     | 0     | 1     | 0    | 1   | 0        | 0   | 0    | 0    | 0    | 1    | 1   |
| lion     | 0     | 0     | 0    | 0   | 0        | 0   | 0    | 0    | 1    | 1    | 0   |
| eats     | 0     | 0     | 0    | 0   | 0        | 0   | 0    | 1    | 0    | 1    | 1   |
| meat     | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 1    | 1    | 0    | 1   |
| not      | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 0    | 1    | 1    | 0   |

- ✓ Formalizing this idea, the “*distributional hypothesis*” suggests that words which appear in similar contexts are likely to share similar meanings (Harris, 1970)

# What is word-embedding?



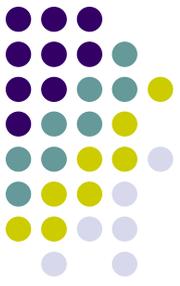
Compare it with a bag-of-words approach. In our case we would have generated the following document feature matrix (DfM) via the command `d_fm` in Quanteda

|        | kitty | likes | milk | and | huggings | cat | fish | lion | eats | meat | not |
|--------|-------|-------|------|-----|----------|-----|------|------|------|------|-----|
| text 1 | 1     | 1     | 1    | 1   | 1        | 0   | 0    | 0    | 0    | 0    | 0   |
| text 2 | 0     | 1     | 1    | 1   | 0        | 1   | 1    | 0    | 0    | 0    | 0   |
| text 3 | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 1    | 1    | 1    | 1   |

The main difference between a DfM and a COM is the **type of contextual information** they use

- A DfM use *documents as contexts*
- A COM instead use *words as contexts*, which can be considered as more natural from a linguistic perspective

# What is word-embedding?



But what should we do with a COM?

We could for example perform some matrix factorization technique (a Factor Analysis or a Principal Component Analysis for example) on it to reduce its *dimensionality*

The outcome would be that each word will now have a *vector* of numeric values to represent it

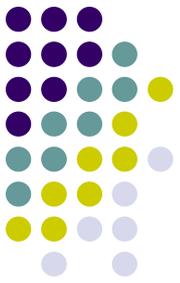
# What is word-embedding?



For example, if we extract 3-dimensions out of the previous COM via a PCA, the word “like” now will be represented as a 3-dimensional vector which may look like (0.3, 0.4, -0.2). And this is true for each other word included in our corpus

The length of this vector corresponds to the nature and complexity of the multidimensional space in which we are seeking to **embed** the word (more on this later on)

# What is word-embedding?

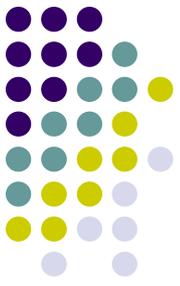


Instead of using vectors of word counts (i.e., our traditional bag-of-words approach), words are therefore now represented as **positions on a latent multidimensional vector space**

In such space, words that are mapped closer to each other have similar meaning (remember the *Distributional hypothesis*: words that appear in the same context – i.e., sharing a similar vector in the COM - share semantic meaning!)

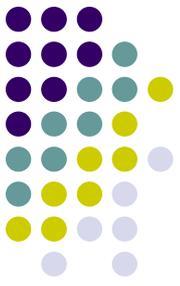


# What is word-embedding?



Through that, we can give a more meaningful numeric representation of a text that contains both information regarding the *word itself* as well as information regarding the *linguistic context* of that word

# What is word-embedding?



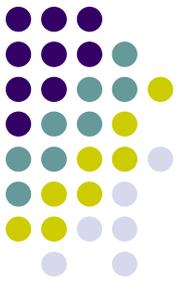
**Word embeddings** give us therefore a way to use an efficient, dense representation in which similar words have a similar encoding

GloVe (“Global Vectors of words”) uses a more sophisticated approach to construct word vectors compared to PCA or FA (Pennington, Socher, and Manning, 2014)

GloVe is a count-based model

Count-based models learn vectors by applying, once again, a dimensionality reduction on a COM

# What is word-embedding?



This is achieved by minimizing a “**reconstruction loss**” which looks for lower-dimensional representations that can explain most of the variance in the high-dimensional data

More in details, GloVe’s aim is to **minimize the difference** between the *dot* product of a given word pair’s embeddings and the log of their global co-occurrence count

The *dot* product, also called scalar product, is a measure of how closely two vectors align, in terms of the directions they point. As the alignment is higher (i.e., when the angle between the two vectors is lower), the dot product increases, and viceversa

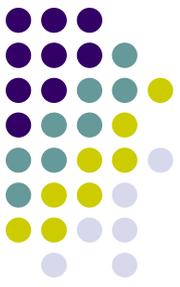
# What is word-embedding?



The GloVe's aim therefore makes sense!

If in fact meaning is connected to co-occurrence, **optimizing the correspondence** between the embedding vectors (that are supposed to capture semantic meaning after all) and the global co-occurrence statistics appears to be a very reasonable thing!

That is, when two words tend to co-occur often within a given window, they should also have a larger dot product in their respective embedding vector



# What is word-embedding?

GloVe aims to minimize the following loss function (with some simplification):

$$\sum_{i \in V} \sum_{j \in V} f(x_{ij}) (\mathbf{u}_j \cdot \mathbf{v}_i - \log(x_{ij}))^2$$

where:  $i$  is the center word;  $j$  is the context word (within a given window)

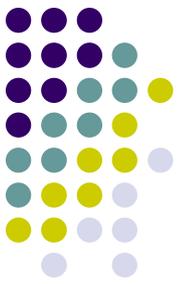
$V$  is the size of the vocabulary within COM

$\mathbf{u}_j$  and  $\mathbf{v}_i$  are the embedding vectors for the two words

$x_{ij}$  is the global co-occurrence count of word  $j$  and word  $i$  in the same context window in the entire corpus

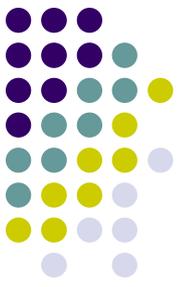
# What is word-embedding?

$$\sum_{i \in V} \sum_{j \in V} f(x_{ij}) (\mathbf{u}_j \cdot \mathbf{v}_i - \log(x_{ij}))^2$$



Finally  $f(x_{ij})$  is a weight function that satisfies three conditions:

1. it becomes 0 if  $x_{ij}$  is 0 (a far from negligible possibility in a COM – a condition therefore very well needed, given that you cannot compute  $\log(x_{ij})$  when  $x_{ij}=0!$ )
2. rare co-occurrences are not overweighted, given that co-occurrences that happen rarely or never are typically noisy and carry less information than the more frequent ones
3. frequent co-occurrences are not overweighted



# What is word-embedding?

The typical class of weighting functions can be parameterized as:

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

where:  $x_{max}$  (i.e., maximum number of co-occurrences) is a parameter that can be chosen by the researcher;  $\alpha$  is .75 by default (but this is a tuning parameter)

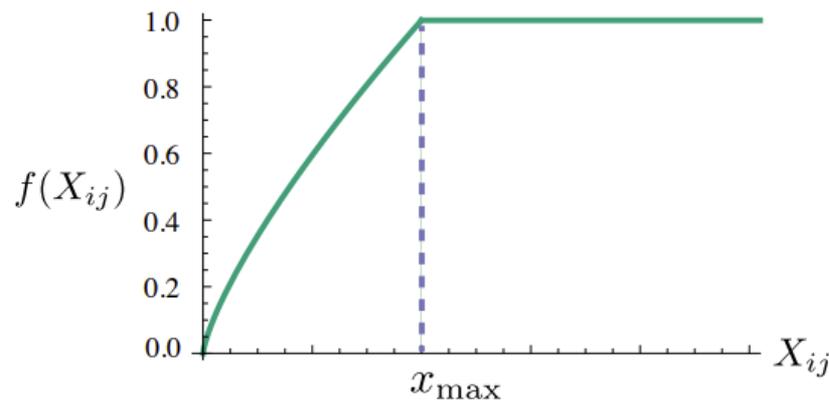


Figure 1: Weighting function  $f$  with  $\alpha = 3/4$ . and  $x_{max}=8$

# What is word-embedding?

$$\sum_{i \in V} \sum_{j \in V} h(x_{ij}) (\mathbf{u}_j \cdot \mathbf{v}_i - \log(x_{ij}))^2$$



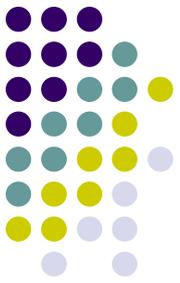
Note moreover the following: if word  $i$  appears in the context window of word  $j$ , then *vice versa*. Therefore,  $x_{ij} = x_{ji}$

However in practice, owing to different initialization values, the same word may still get (usually slightly) different values in these two vectors after training

As a result of that, GloVe learns two sets of word vectors

While both of word-vectors matrices can be used, it could also be a good idea to average or take a sum of main and context vector to reduce overfitting and noise and generally improve results

# What is word-embedding?

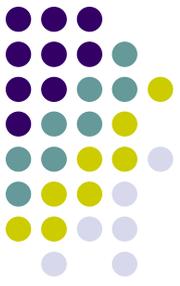


The technical details related to *GloVe* are not however (too much) important for applied users to benefit from them

The aim remains the already discussed one:

- ✓ building a **low-dimensional vector representation** out of a COM, which preserves the *contextual similarity* of words, such that the **semantically similar** words are mapped close to each other

# What is word-embedding?

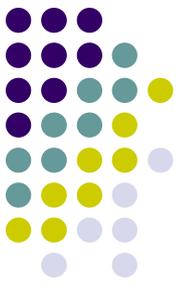


`word2vec` is the main alternative to GloVe among the *context independent algorithms* (more on this later...)

`word2vec` is however a *predictive model*: it uses a neural net model (not going to discuss about it, unless you really want it...) to estimate the probability that a word is “close” to another given word in a given window

In predictive models, the *word vectors* are learnt by trying to improve on the predictive ability, i.e., minimizing the loss between the target word and the context word

For example, in our COM above initially *kitty* and *cat* could have been randomly assigned as too distant. However in order to minimize its loss with the context words (“*like*” and “*milk*”), both words will have to be close to each other in the vector space



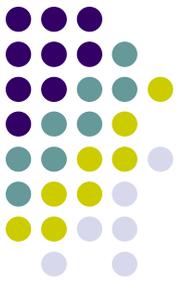
# What is word-embedding?

Two options: the CBOW (continuous bag-of-words) or the continuous skip-gram model

*CBOW model:* it predicts the middle/focus word based on surrounding context words (included in our window). The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words

In our example above, given the input (“kitty”, “milk”, “cat”) we want to maximize the probability of getting “likes” as the output

# What is word-embedding?

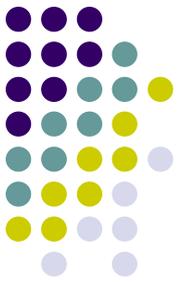


Two options: the CBOW (continuous bag-of-words) or the continuous skip-gram model

*Continuous skip-gram model*: predicts words within a certain range before and after the current word in the same sentence

That is, while the CBOW model predicts a word given the neighboring context, a skip-gram model predicts the context (or neighbors) of a word, given the word itself

# What is word-embedding?



Contrary to GloVe, that works directly on the entire co-occurrence matrix, `word2vec` follows therefore an *online learning approach*, that is, the model is trained as the local window is moved from word to word along the corpus

As a result, GloVe captures the counts of overall statistics how often it appears, while `word2vec` captures co-occurrence one window at a time

Despite this difference, the two approaches are not mathematically very different

This should not be all that surprising since the number of times `word2vec` will observe a given target-context pair once the moving window has covered the entire corpus is exactly equal to the total co-occurrence count used in GloVe

# What is word-embedding?



Any so which one to choose?

According to Rodriguez and Spirling (2022), Word2Vec is likely to be less “robust,” i.e., embeddings will tend to be more corpus specific, than GloVe

Moreover, GloVe appears to fare better than Word2Vec in several different tasks, such as detecting word-analogy (word-what? Give me a minute...)

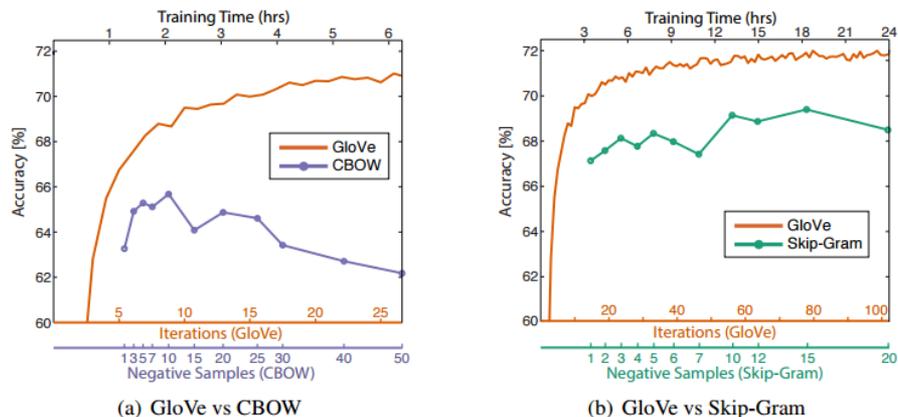
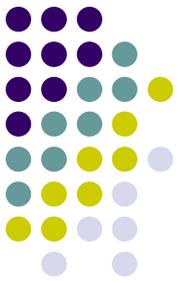


Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.

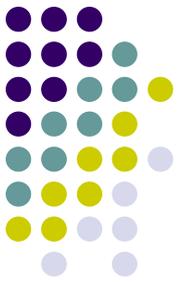
# What is word-embedding?



Finally, to properly understand the logic behind `Word2Vec` we should go in greater details into the architecture of a neural networks

So we will focus on `GloVe` in the lab (plus: it is more easy to implement within `Quanteda...`)

# Parameter Choices



You should be called, at the very least, to decide about 3 parameters in any embedding model:

1. how large a **window size** you want the model to use
2. how **large an embedding** you wish to use to represent your words (i.e., the number of extracted dimensions)
3. whether to fit the embedding models **locally**, or to apply **pre-trained embeddings** to your corpus

# Window-size



**Window-size** determines the number of words, on either side of the focus word to be included in its context

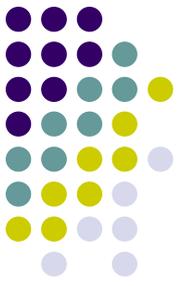
Intuitively, larger windows provide more information to discriminate between different words

Take, for example, the following two sentences: “*cows eat grass*” and “*lions eat meat*”

A window-size of 1 does not provide enough information to distinguish between cows and lions (we know they both eat, but we don't know what) whereas a window size of 2 does

Larger windows (usually 5 or above) tend therefore to produce better quality embeddings although with decreasing returns

# Embedding dimensions



This parameter determines the **dimensions** of the embedding vectors which usually range between 50-500

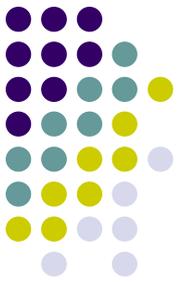
We can think of these dimensions as capturing different aspects of “meaning” or semantics that can be used to organize words

Though none of the dimensions of the estimated vector are named, the “loading” of each term on the dimensions indeed often captures substantively important relationships

# Embedding dimensions

For instance, the words “king” and “queen” might have a similar mapping on dimension 1 (a dimension related to the *concept of royalty?*) but deviate on dimension 2 (a dimension related to *gender?*)

The coordinates of the dimensional-vector represent in other words **distinct topics**



# Embedding dimensions



*Too few* dimensions - imagine the extreme of 1 - and there can be no meaningful separability of words; *too many*, and some dimensions are likely to be redundant (go unused)

Factors such as vocabulary size and topical specificity of the corpus are likely to play a role, although theoretical work in this area remains scant

# Advice to Practitioners



- ✓ To summarize: with the possible exception of small corpora, one should avoid using very few dimensions (below 100) and small window-sizes ( $< 5$ )
- ✓ While performance improves with larger window-sizes and more dimensions, both exhibit decreasing returns: i.e., improvements are marginal beyond 300 dimensions and window-size of 6
- ✓ Given the trade-off between more dimension/larger window-size and computation time, the choice of 5 or 6 (window-size) and between 100 and 300 (dimensions) seems therefore reasonable

# Pre-Trained Versus Going Local



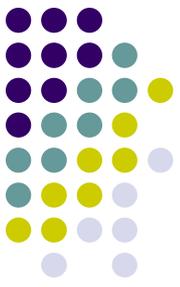
Embedding models can be data hungry, meaning they need a lot of data to produce ‘useful’ results

Consequently, researchers with small corpora rather than building word-embeddings **locally** out of their corpus, can decide to use generic (**pre-trained**) **embeddings** trained on much larger corpora

These off-the-shelf embedding corpora can be used like dictionaries, but instead of translations or meanings, they return vector embeddings for the requested words

Therefore, the usage of pre-trained embeddings does not require any further computing time

# Pre-Trained Versus Going Local



For example, pre-trained word vectors are available that have been estimated from large corpora, such as that trained on six billion tokens from Wikipedia and the Gigaword corpus (Pennington, Socher and Manning, 2014)

This allows a researcher to represent his or her texts not just from the corpus at hand, but also augmented with quantitative measures of the words' semantic representations fitted from other contexts

# Pre-Trained Versus Going Local

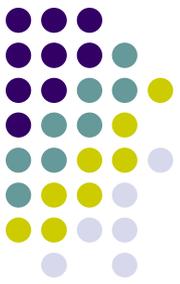


Employing pre-trained word vectors is an example of what is called in the NLP literature as a “*transfer learning approach*”

Transfer learning refers to a set of learning procedures in which knowledge, that has been obtained by training on a *source task* in a *source domain*, is transferred to the learning process of the *target task* in a *target domain*, where either the target task is not the same task as the source task or the target domain is not the same as the source domain

There are however some *trade-offs* to keep in mind

# Pre-Trained Versus Going Local



Intuitively, we would want to use pre-trained embeddings trained on a corpus generated by a *similar “language model”* to that which generated our corpus of interest

The more similar the two language models, the more similar the underlying semantics. For a highly specific local corpus - a corpus in Old English for example - generic pre-trained embeddings may not be all that useful!

This becomes a problem because large pretraining tends to be very expensive

As a result, it is much more convenient and cost-effective for applied researchers to make use of an already pretrained model (even when the “distance” between source and task is pretty large...)

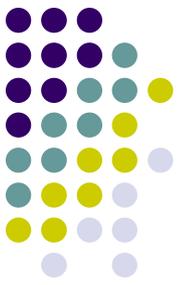
# Pre-Trained Versus Going Local



Another problematic aspect is that any “*transfer learning approach*” reflects the representational biases (including stereotypes related to chauvinist or racial issues, underrepresentations) encoded in the data they have been pretrained on (Bommasani et al., 2021)

As soon as a model is adapted to some target task, these biases can materialize with serious negative consequences

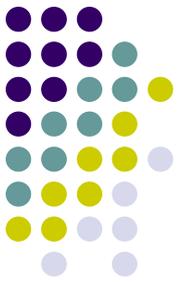
# *What for* Word embedding models?



Operations with word-vectors are meaningful per-se, i.e., embeddings can be a **direct object of interest** for studying word usage and meaning

For example, **similarities** between word embeddings may be substantively informative: if the distance between “immigrants” and “hardworking” is smaller for liberals than for conservatives, we learn something about their relative worldviews!

# *What for* Word embedding models?



This also allows us to explore the possible existence of societal *stereotypes* encoded in the language

To take an example, if you analyze the Italian legislative speeches since 1948 to nowadays, the most similar word to “woman” in the ‘50s was “mother” and “wife”; then over the years it changed to “worker” and “person”

# What for Word embedding models?

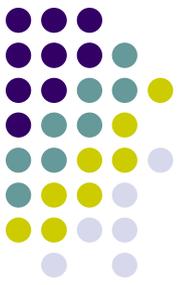
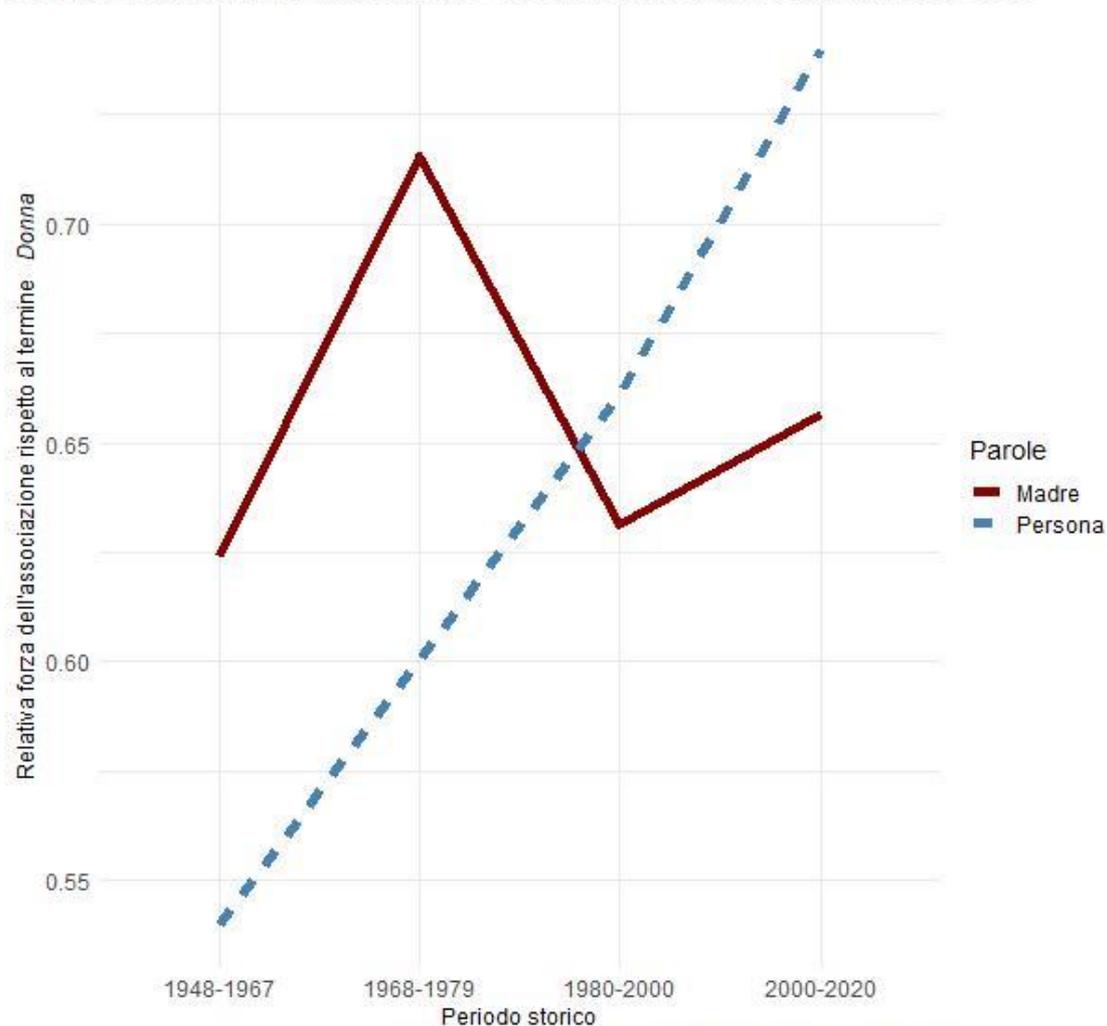


Fig.3: Associazione tra Donna vs. Madre e Persona nel Parlamento italiano 1948-2020



Fonte: Analisi sul database del Portale Storico della Camera dei deputati

# *What for Word embedding models?*



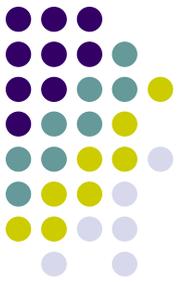
You can also do addition in a word embeddings framework to explore *analogies*

Analogies are **equations** of the form *A is to B as C is to D*

Given the terms A, B, C, the model must return the word that correctly stands for D in the given analogy

A classic example is the following one: «King - male + female ~ queen». That is, “man is to king as woman is to queen”. Why?

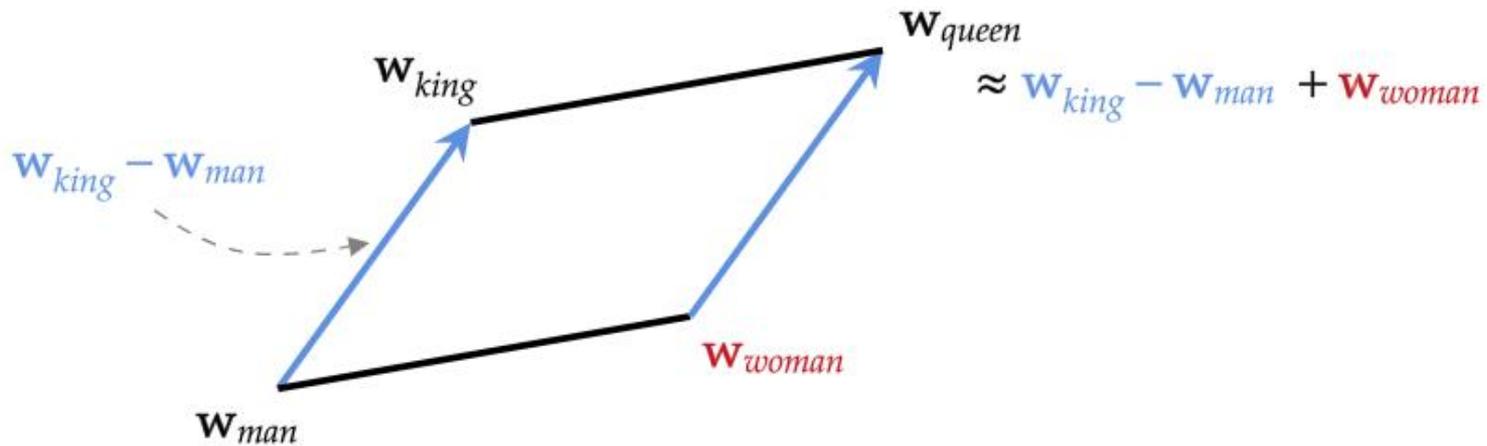
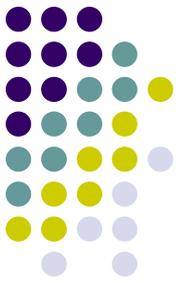
# What for Word embedding models?



As the difference between vector *king* and vector *queen* is similar to the difference between vector *male* and vector *female*, then "King - queen ~ male - female", therefore, "King - male + female ~ queen"

Subtracting the **man** vector from the **king** vector and adding **female**, the most similar word to this would be **queen**

# What for Word embedding models?



| word  | $D_1$ | $D_2$ | $D_3$ | ... | $D_N$ |
|-------|-------|-------|-------|-----|-------|
| man   | 0.46  | 0.67  | 0.05  | ... | ...   |
| woman | 0.46  | -0.89 | -0.08 | ... | ...   |
| king  | 0.79  | 0.96  | 0.02  | ... | ...   |
| queen | 0.80  | -0.58 | -0.14 | ... | ...   |

# *What for* Word embedding models?

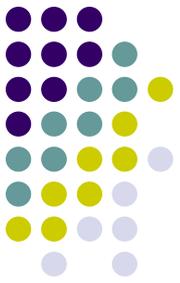


It may be surprising that you are able to get these results, without any label on the meaning of words

"*Co-occurrence*" is however the key here! It is in fact the only information that you are using

By building a sense of one *word's proximity* to other similar words, which do not necessarily contain the same letters, we can move beyond hard tokens to a smoother and more *general sense of meaning*

# *What for* Word embedding models?



Word embedding models can moreover be thought not simply as a **method on their own** for analyzing text as data, but also as extremely **useful complements to representations of text as data based on word counts**

How?

# The advantages of Embeddings

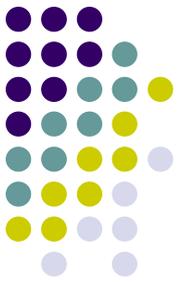


In a bag-of-words approach (i.e., in a context-blind approach) features are completely independent from one another (remember?). The feature “kitty” is as dissimilar to “cat” than it is to “lion”

On the contrary, using a word embeddings approach, each feature is embedded into a  $d$  dimensional space, and represented as a vector in that space

And as we have already discussed, similar features will have similar vectors – i.e., information is shared between similar features (“kitty” will be reasonably spatially closer to “cat” than to “lion”)

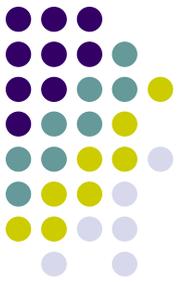
# The advantages of Embeddings



What's the implication of that?

Let's suppose that we summarize the position of each text in our corpus by calculating its *mean vector* over each of the dimensions in which its words are embedded

# The advantages of Embeddings



Let's go back to our previous example where we extracted 3 dimensions via a PCA (not via the much more efficient *GloVe* – but this is just an illustration!)

The table shows for each feature its vector of length 3

|          | 1st D | 2nd D | 3rd T |
|----------|-------|-------|-------|
| kitty    | 0.38  | -0.14 | 0.41  |
| likes    | 0.30  | 0.40  | -0.18 |
| milk     | 0.28  | -0.16 | -0.40 |
| and      | 0.35  | -0.26 | 0.36  |
| huggings | 0.32  | 0.49  | 0.01  |
| cat      | 0.38  | -0.14 | 0.41  |
| fish     | 0.08  | 0.63  | 0.20  |
| lion     | -0.26 | 0.12  | 0.35  |
| eats     | -0.27 | 0.22  | 0.19  |
| meat     | -0.31 | -0.06 | 0.17  |
| not      | -0.28 | 0.04  | 0.35  |

# The advantages of Embeddings



Using the previous table (and knowing the DfM) we can for each text taking the average of the vectors of each words included in it along the 3 dimensions to represent the text in the same embedding world than the words

|          | 1st D | 2nd D | 3rd T |
|----------|-------|-------|-------|
| kitty    | 0.38  | -0.14 | 0.41  |
| likes    | 0.30  | 0.40  | -0.18 |
| milk     | 0.28  | -0.16 | -0.40 |
| and      | 0.35  | -0.26 | 0.36  |
| huggings | 0.32  | 0.49  | 0.01  |
| cat      | 0.38  | -0.14 | 0.41  |
| fish     | 0.08  | 0.63  | 0.20  |
| lion     | -0.26 | 0.12  | 0.35  |
| eats     | -0.27 | 0.22  | 0.19  |
| meat     | -0.31 | -0.06 | 0.17  |
| not      | -0.28 | 0.04  | 0.35  |



|        | kitty | likes | milk | and | huggings | cat | fish | lion | eats | meat | not |
|--------|-------|-------|------|-----|----------|-----|------|------|------|------|-----|
| text 1 | 1     | 1     | 1    | 1   | 1        | 0   | 0    | 0    | 0    | 0    | 0   |
| text 2 | 0     | 1     | 1    | 1   | 0        | 1   | 1    | 0    | 0    | 0    | 0   |
| text 3 | 0     | 0     | 0    | 0   | 0        | 0   | 1    | 1    | 1    | 1    | 1   |



|        | 1st D | 2nd D | 3rd T |
|--------|-------|-------|-------|
| text 1 | 0.33  | 0.07  | 0.04  |
| text 2 | 0.28  | 0.09  | 0.08  |
| text 3 | -0.21 | 0.19  | 0.25  |

# The advantages of Embeddings



As a result, a text (like text 1) that includes that word “kitty”, ceteris paribus, will report a mean vector in the embedded world more similar (i.e., spatially closer) to a text including the word “cat” (like text 2) than it is to a text including the word “lion” (like text 3)

This is exactly what it happens in the previous example

Let’s further suppose that in our *training-set* we have text 1 and text 3, but not text 2, i.e., we have a text including the word “kitty” but not the word “cat”

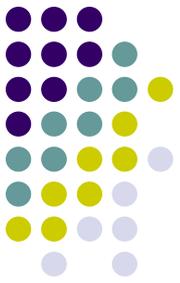
# The advantages of Embeddings



Now suppose that we train a Random Forest (RF) algorithm to predict some class-label of the texts (say: 1=if a text discuss about a pet; 0=otherwise), and let's use as our IVs the position of the texts within the training-set along the 3-dimensions of our embedding world

Finally, let's also suppose that the mean vector of the text including the word "kitty" is important in affecting the prediction of our RF algorithm for the class "pet" for example

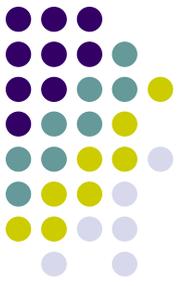
# The advantages of Embeddings



As a result of that, our RF algorithm can take advantage of the word “cat” during the prediction stage to classify a text as “pet”, although **the word “cat” wasn’t included in any text within the training set!**

This is because “cat” is spatially very close to “kitty”, and as a result of that, the mean vector of a text including the word “cat” (ceteris paribus) is going also to be spatially closer, ceteris paribus, to the mean vector of a text including the word “kitty”

# The advantages of Embeddings



The main benefit of a *dense real-valued vector of number representation* (such as the one we obtain via WE), rather than a *sparse matrix* (such as a DfM or also a COM) lies therefore in its **generalization power**

- ✓ if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities

# The advantages of Embeddings



Through that, WE may increase the accuracy of classification models as they also provide information (i.e., vector representations) on words **that are not represented** in the training data based on their similarity with other words

This is of course impossible in a bag-of-words approach in which new words encountered in the texts of the test-set (i.e., “cat”) are a *nuisance*

# What for Word embedding models?



WE can also be very useful to expand *existing dictionaries*

Imagine you want to expand a dictionary of *uncivil words*

By looking for other words with semantic similarity to each of the terms included in the original dictionary, we can identify words that we may not have thought of in the first place, either because they are slang, new words or just misspellings of existing words

# What for Word embedding models?



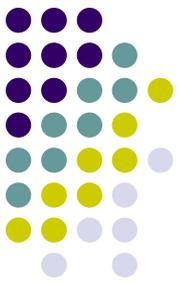
An example:

Douglas R. Rice and Christopher Zorn. [Corpus-based dictionaries for sentiment analysis of specialized vocabularies](#), *Political Science Research and Methods* (2021), 9, 20–35

Replication data:

<https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/4EKHFM>

# What for Word embedding models?



Similarly, we can use WE to estimate a semi-supervised scaling algorithm

For example Watanabe (2021) propose a Latent semantic scaling based on WE

- ✓ first you have to identify a list of seed-words related to the two polarity of a unidimensional scale you are interested about (say positive/negative)
- ✓ then you assign a polarity score to each word included in a given text based on the semantic proximity between words in a corpus and the seed-words employing precisely word-embedding techniques
- ✓ Once you have a score for each word, you can locate documents along the previous unidimensional scale

# What for Word embedding models?

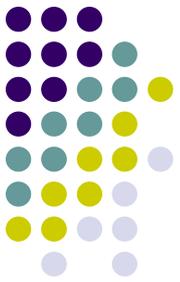


Watanabe, K. (2021). Latent semantic scaling: A semisupervised text analysis technique for new domains and languages. *Communication Methods and Measures*, 15(2), 81-102.

The nice thing is that there is a package that you can run within Quanteda that implements such method:

```
library(LSX)
```

# Evaluation



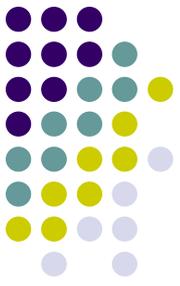
Bu how to evaluate embedding models?

Two possible evaluation tasks: *intrinsic* and *extrinsic*

These correspond to the two main use cases of embeddings: as models of semantics and as feature inputs

Such evaluation tasks allow you not only to evaluate one single WE, but also to choose between different WE settings (i.e., with different window-size or different dimensions), similarly to what you do via a grid-search

# Evaluation



*Intrinsic* tasks evaluate embeddings as models of semantics

These include:

- ✓ checking if the WE estimated retrieve some very well known *word analogy* (such as the king-queen discussed above...)
- ✓ checking if they provide meaningful *word similarities* (including comparing similarity ratings computed using WE vs. similarity ratings computed by humans)

These tasks require human generated data

# Evaluation



*Extrinsic* tasks evaluate embeddings generally via supervised approaches

For example, if you want to use WE as an input for a ML classification, you can compare different WE (for example, with a different number of dimensions / window-size) via cross-validation on the training-set!

See Rodriguez and Spirling (2021) for some further alternatives in this respect

Of course, note two things:

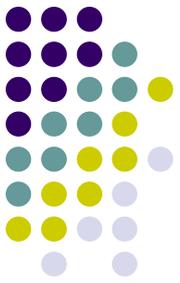
- a) evidence of good performance need not generalize (it is corpus specific!)
- b) performance in extrinsic tasks need not translate into good performance in intrinsic tasks (and vice-versa)

# Evaluation

Another possible metric to compare different models is focusing on the **reconstruction loss** at the point of convergence and selecting that model (i.e., combination of # dimensions and window size) that minimizes it



# Recent advancements



GloVe (as well as `word2vec`) embeddings are *context independent*

That is, these models output just one embedding for each word, combining all the different senses of the word into one single vector *regardless* of where the words occurs in a sentence and *regardless* of the different meanings they may have

# Recent advancements



For instance, after we train GloVe on a corpus we get as output one vector representation for, say the word “cell”

So even if we had a sentence like “*He went to the prison cell with his cell phone to extract blood cell samples from inmates*”, where the word cell has different meanings based on the sentence context, these models just collapse them all into one vector for “cell” in their output

This single vector is likely to blend these two meanings (having the effect that the vector will be located somewhere between the two different meanings in space)

# Recent advancements



In recent years, this issue has been addressed in NLP by learning *contextualized representations*

*Contextualized representations* account for the observation that the (exact) meaning of a word arises from its context

More in details, a contextualized representation is a representation of a word and is a function of the words that precede and/or proceed word at

Hence, two identical words that occur in different contexts will have a different representation

Through that, we are able to capture information from surrounding words

# Recent advancements

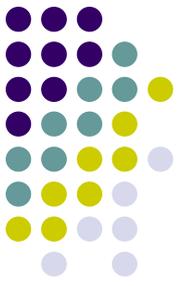


In other words, algorithms that implements contextualized representations can generate different word embeddings for the same word to capture the *context of a word* - that is its **position** in a sentence

The most popular class of algorithms implementing contextualized representations is called...



# Recent advancements



Within this class of Transformer-based models, we have algorithms such as BERT or RoBERTa (having fun names is a must nowadays for very complex algorithms...perhaps to sound less threatening?)

Their aim is to acquire a highly general, close to universal language representation model

The pretrained general-purpose representation model then can be used as an input to a target task of interest

# Recent advancements

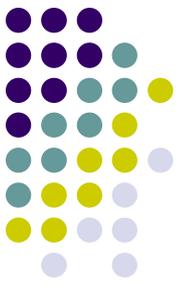


Going back to previous example, these algorithms would generate different vectors for the word “cell”

The first cell (*prison cell case*), for instance would be plausibly closer to words like incarceration, crime etc. whereas the second “cell” (*phone case*) would be plausibly closer to words like iPhone, Android, etc.

To repeat: this difference arises cause `GloVe` and `word2vec` do not take into account word order in their training, while `BERT` or `RoBERTa` does

# Recent advancements



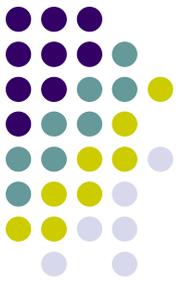
An implication of this difference is that we can use GloVe vectors trained on a corpus directly for some tasks. All we need is the vectors for the words. There is no need for the model itself that was used to train these vectors

Moreover, as discussed, we can also train a GloVe model locally

In the case of BERT or RoBERTa, on the contrary, since they are context dependent, we need the model that was used to train the vectors even after training, since the models generate the vectors for a word based on context

Moreover, to fully leverage the power of such model, researchers require access always to already pretrained models that they can fine-tune on their specific tasks

# Recent advancements



A wonderful article discussing all those issues:

Wankmüller, S. (2022). Introduction to Neural Transfer Learning With Transformers for Social Science Text Analysis. *Sociological Methods & Research*

<https://doi.org/10.1177/00491241221134527>