Mining Unsupervised Data: Word Vectors

Professor: Salvador Medina Herrera
salvador.medina.herrera AT upc.edu

Outline

Question

What do you know about Word Vectors or Word Embeddings?

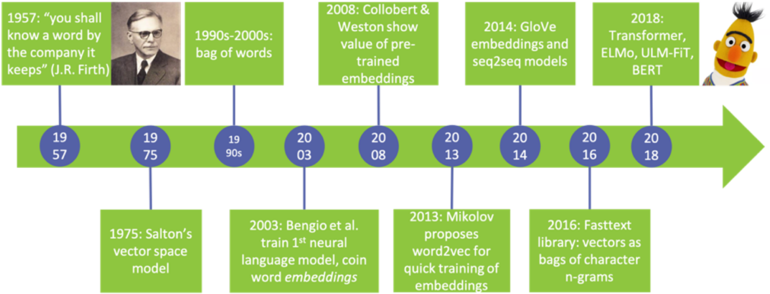
A Word embedding is a numerical representation of a word

- Word embeddings allow for arithmetic operations on a text:
  - Example: $time + flies$
  - Example (II): $king - man + woman \approx queen$
- Word embeddings have been referred to as:
  - Semantic Representation of Words
  - Word Vector Representations

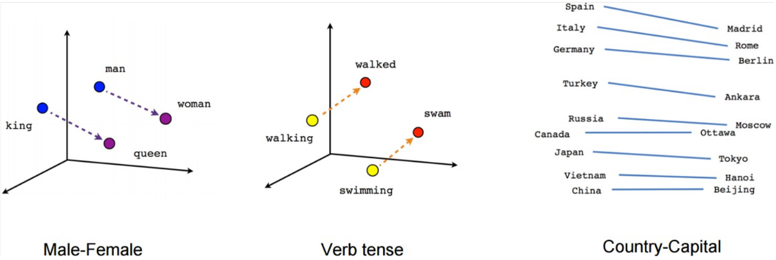
From Tokens to Meaning

- **Tokenization** gave us the building blocks (words, phrases).
- **PoS Tagging** helped us understand grammatical roles.
- **Lexical Semantics** (e.g., WordNet) provided structured meaning.
- **Now:** How can we represent words numerically to capture their meaning in context?
  - Word embeddings bridge the gap between discrete tokens and continuous vector spaces.
  - They generalize beyond fixed dictionaries (e.g., WordNet) by learning from data.

Timeline


1957: "you shall know a word by the company it keeps" (J.R. Firth)
1990s-2000s: bag of words
2008: Collobert & Weston show value of pre-trained embeddings
2014: GloVe embeddings and seq2seq models
2018: Transformer, ELMo, ULM-FiT, BERT
19 57
19 75
19 90s
20 03
20 08
20 13
20 14
20 16
20 18
1975: Salton's vector space model
2003: Bengio et al. train 1st neural language model, coin word *embeddings*
2013: Mikolov proposes word2vec for quick training of embeddings
2016: Fasttext library: vectors as bags of character n-grams

Word vectors



Distributional Hypothesis Contextuality

(Frege, 1884)

Never ask for the meaning of a word in isolation, but only in the context of a sentence

(Wittgenstein, 1953)

For a large class of cases... the meaning of a word is its use in the language

(Firth, 1957)

You shall know a word by the company it keeps

(Harris, 1954)

Words that occur in similar contexts tend to have similar meaning

Key Idea: Word embeddings capture meaning through context.

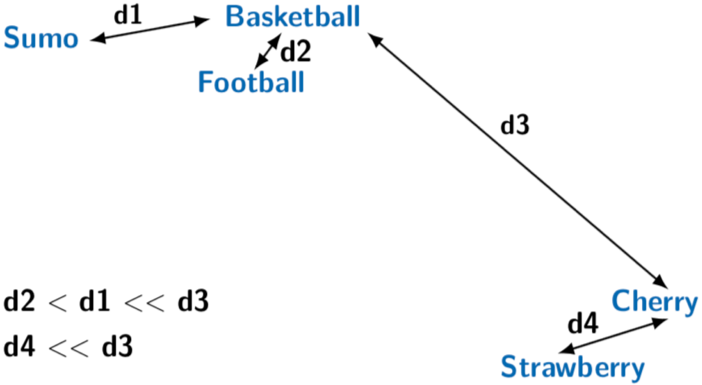
Words Embeddings allow to process sentences with Machine Learning

Sentences are sequences of symbols:
Word vectors (word embeddings) are vector representations of words, the "natural" unit for solving natural language processing tasks.

id	qid1	qid2	question1	question2	is_duplicate
447	895	896	What are natural numbers?	What is a least natural number?	0
1518	3037	3038	Which pizzas are the most popularly ordered pizzas on Domino's menu?	How many calories does a Dominos pizza have?	0
3272	6542	6543	How do you start a bakery?	How can one start a bakery business?	1
3362	6722	6723	Should I learn python or Java first?	If I had to choose between learning Java and Python, what should I choose to learn first?	1

Words Embeddings allow to process sentences with Machine Learning

Vector representations can help us finding **similar meanings**
...but we need to define a concept of **distance**.



$d2 < d1 << d3$

$d4 << d3$

Outline

How to represent a word: One-hot vectors

- **One-hot vector** (dim == vocabulary size)
  - Very large vector (millions of words in some applications)
  - Sparse, orthogonal representations
  - No information about how words are related
  - No useful vector distance
  - Huge use of memory (if sparse matrices are not used)
  - Usual coding of categorical variables for Linear models and SVMs with the standard kernels

$$
\begin{array}{lll}
\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & \dots\end{bmatrix} & \text{to} & (1) \\
\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & \dots\end{bmatrix} & \text{be} & (3) \\
\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & \dots\end{bmatrix} & \text{or} & (2) \\
\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1 & \dots\end{bmatrix} & \text{not} & (5) \\
\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & \dots\end{bmatrix} & \text{to} & (1) \\
\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & \dots\end{bmatrix} & \text{be} & (3)
\end{array}
$$

Outline

Vectors and Documents

- **Document-term matrix:** number of times a term (row) appears in a document (column)
- Originally defined as a means of finding similar documents for the task of document information retrieval
- We can use document vectors to find other similar documents

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Vectors and Documents (II)

- **Term-document matrix:** number of times a term (row) appears in a document (column)
- Similar words have similar vectors because they tend to occur in similar documents

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Problems:
  - Hard to get meaningful results for frequent words (the, it...)
  - 'good' appears frequently in different contexts
- Solution:
  - **tf-idf** (term frequency-inverse document frequency)

Outline

TF-IDF Vectors

- TF-IDF is a numerical representation of documents based on the importance of terms within them.
- Term Frequency (TF) measures the frequency of a term in a document.
- Inverse Document Frequency (IDF) measures the importance of a term in the entire corpus.
- The TF-IDF score combines both TF and IDF to determine the relevance of a term in a document.

TF-IDF Vectors (II)

$$
\begin{aligned}
\text{Term Frequency (TF)}: &\quad \mathsf{TF}_{ij} = \frac{n_{ij}}{n_{\text{total}}} \\
\text{Inverse Document Frequency (IDF)}: &\quad \mathsf{IDF}_i = \log\left(\frac{N}{n_i}\right) \\
\text{TF-IDF Score}: &\quad \mathsf{TF\text{-}IDF}_{ij} = \mathsf{TF}_{ij} \times \mathsf{IDF}_i
\end{aligned}
$$

where:

- n_{ij} is the frequency of term i in document j.
- n_{total} is the total number of terms in document j.
- N is the total number of documents in the corpus.
- n_i is the number of documents that contain term i.

TF-IDF Vectors (III)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Outline

Beyond one-hot: Type of word vectors

- Based on human knowledge
- Based on context words: "You shall know a word by the company it keeps" (J.R. Firth, 1957)
  - Example:
    - I will go to the **cinema** on Sunday.
    - Pop-up **cinema** to enjoy films about local cuisine.
    - Concerning eyesight, photography, **cinema**, television.
  - Types:
    - Count-based methods (co-occurrence counts)
    - Direct prediction / Deep learning methods
    - Hybrid (GloVe vectors)

Outline

Word vectors based on human knowledge

Based on human-created linguistic resources, e.g. Wordnet, a thesaurus containing lists of **synonym** sets and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Question

What problems can you imagine with this approach?

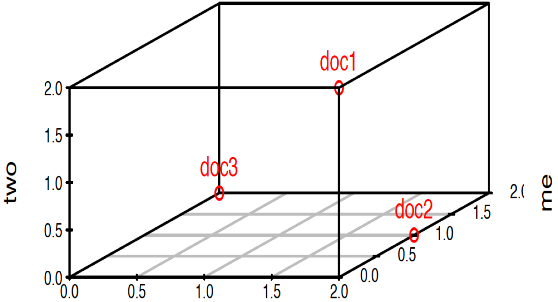
Word vectors based on human knowledge (continued)

- Problems:
  - No straightforward way to compute similarity between words.
  - Missing nuance: binary relationships (e.g., synonyms only in some contexts).
  - Limited number of words.
  - Impossible to keep up-to-date.
  - Subjective.
  - Costly human labor to create and adapt.
- **However**, knowledge-based approaches can still be effective:
  - For specific tasks, such as clustering or similarity in ancient languages.
  - When embeddings are not feasible (e.g., lack of data for training).
  - As a complement to other vector representations.

Outline

Based on context words: count-methods

- How do we do this? What we need is a collection of documents, and using these documents, we can use different methods...
- Starting by **term-frequency**... counting the number of words that appear in a document.



Based on context words: count-methods (II)

doc1	Two for tea and tea for two
doc2	Tea for me and tea for you
doc3	You for me and me for you

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

Based on context words

Count-based + SVD (reduced rank approx.)

- Count word co-occurrence counts:
  1. Window-based Word / Word co-occurrence matrix
  2. Pointwise Mutual Information

Word-Word Matrix

Context: ± 7 words

sugar, a sliced lemon, a tablespoonful of	**apricot**	preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

Resulting word-word matrix:

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Outline

Pointwise Mutual Information (PMI)

- PMI is a measure of the association between two words based on their co-occurrence in a corpus.
  - PMI captures the extent to which the observed co-occurrence of two words deviates from what would be expected if they were independent.
  - It provides a measure of the strength and directionality of the association between words.
  - Positive PMI values indicate a stronger association than expected, while negative PMI values indicate a weaker association than expected.

$$\text{PMI}(w_1, w_2) = \log\left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

- $P(w_1, w_2)$ is the joint probability of words w_1 and w_2 co-occurring together.
- $P(w_1)$ and $P(w_2)$ are the individual probabilities of words w_1 and w_2 occurring independently.

Positive Pointwise Mutual Information (PPMI)

- PPMI is a modified version of PMI that addresses some of its limitations, particularly the handling of low-frequency events and the problem of negative values.
  - PPMI only considers positive values and assigns higher weights to co-occurrences that are more significant.
  - PPMI measures the strength of association between two words based on their co-occurrence probabilities in a corpus.

$$\text{PPMI}(w_1, w_2) = \max(\log(\frac{\text{cooc}(w_1, w_2) \cdot N}{\text{freq}(w_1) \cdot \text{freq}(w_2)}), 0)$$

- $\text{cooc}(w_1, w_2)$ is the co-occurrence count of words w_1 and w_2 in a co-occurrence matrix.
- $\text{freq}(w_1)$ and $\text{freq}(w_2)$ are the frequencies of words w_1 and w_2 in the corpus.
- N is the total number of co-occurrences in the matrix.

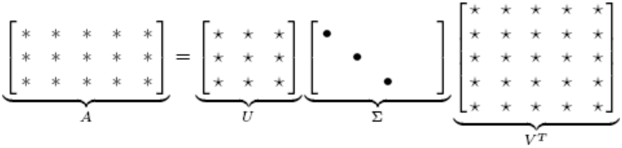
PPMI: Example

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Singular Value Decomposition

Count-based + SVD

- Count word co-occurrence counts: two options
  - Word / documents co-occurrence matrix
  - Window-based Word / Word co-occurrence matrix
- Singular Value Decomposition $X = USV^T$ to reduce the dimensionality (rank). The rows of U are the word embeddings.



Singular Value Decomposition (II)

Problems:

- Function words (the, you, is, ...) have a big impact.
- Solutions: modify raw counts (log tf-idf) or remove function words.
- High-dimensional matrix.
- Quadratic cost of SVD.
- Solutions: adaptive algorithms.

Based on context words: Direct prediction

- Continuous space representations or word embeddings.
- Small vector of real numbers (dimension 200–400).
- Linguistic or semantic similarity can be measured with the Euclidean distance or cosine similarity.
- Vector differences capture word relations.
- Standard choice for deep learning models.

```
(12424, 100)
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	-0.002272	0.015870	0.018349	0.022802	0.028364	-0.040064	-0.013263	0.136607	0.019667	0.033407	...	0.037663	-0.087140	0.073169	-0.028257
unto	0.034425	-0.102070	0.018051	0.017960	0.172954	-0.115672	-0.012632	0.096919	-0.049203	-0.040344	...	0.106373	-0.075703	0.013888	-0.134224
lord	0.051990	-0.113865	0.007226	0.031754	0.052963	-0.094523	-0.067664	0.001706	-0.112827	-0.078586	...	-0.041636	0.053685	0.041299	-0.026255
thou	-0.152183	-0.073681	-0.091472	0.022033	0.008415	-0.048438	-0.041181	0.082019	0.004648	0.044870	...	0.101531	-0.018404	-0.070462	-0.041363
thy	-0.257579	-0.023008	0.053303	0.013690	-0.083293	0.034279	0.078811	0.079851	-0.015215	-0.111211	...	-0.064527	0.112085	0.061625	0.026398

5 rows × 100 columns

Outline

Word2Vec: CBOW

- Direct prediction / Deep learning methods: Word2vec (Mikolov, Google 2013)
  - **Continuous bag-of-words (CBOW)**: prediction of a word using the context words (bag-of-words)

Word2Vec: CBOW (II)

FUN WITH FILL-INS

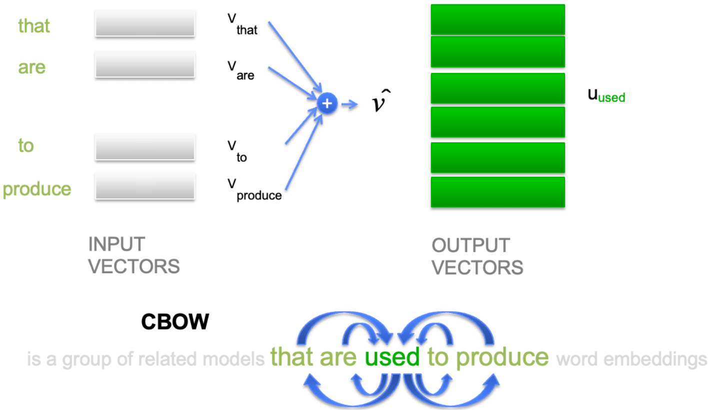
First Grade Sight Words

Choose the sight word from the Word List that will complete each sentence below.
Hint: Words can be used more than once.

Word List: are, good, now

1. Plums _________ in a tree.
2. Are the plums _________ now?
3. The plums are hard. They _________ not good.
4. Sun is good for plums. Rain is _________ for plums.
5. Are the plums good ___________?
6. The plums _________ soft.
7. _________ the plums are good!

Word2Vec: CBOW (III)



CBOW equations

- Continuous bag-of-words (CBOW)
- W is the word vocabulary
- Input vectors: v_w for each $w \in W$
- Output vectors: u_w for each $w \in W$

The 'predicted' output word vector is the sum over all context input vectors:

$$u_w = \sum_{\text{context words}} v_w$$

We use the dot product to compute the score vector (word similarity):

$$\text{score} = u_w \cdot v_c$$

And the softmax function to get probabilities:

$$p(w|c) = \frac{e^{\text{score}}}{\sum_{w' \in W} e^{\text{score}(w')}}$$

CBOW equations (II)

The standard choice for the loss function is the cross-entropy of the estimated probability $p(w)$ respect to the true probability $q(w)$:

$$\begin{aligned}\mathsf{CE}(q,p) &= E_q[-\log p(w)] \\ &= E_q[-\log p(w) + \log q(w) - \log q(w)] \\ &= E_q[\log p(w)] + E_q[-\log q(w)] \\ &= D_{KL}(q||p) + H(q)\end{aligned}$$

In our case, it is equivalent to the minimization of the negative log-likelihood of the target word vector given the context:

$$\text{minimize} \;\; -\log p(w_c|w_{\text{context}})$$

Outline

Word2Vec: Skip-gram

- Direct prediction / Deep learning methods: Word2vec (Mikolov, Google 2013)
  - **Continuous skip-gram architecture**: prediction of the context words using the current word

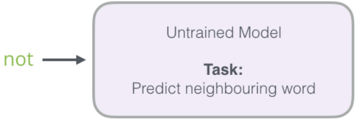
Step-by-step: skip-gram training with negative sampling

- Let's glance at how we use it to train a basic model that predicts if two words appear together in the same context.

Preliminary steps

- We start with the first sample in our dataset.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness



Note on efficiency of negative sampling

- We grab the feature and feed it to the untrained model asking it to predict if the words are in the same context or not (1 or 0).

Change Task from

To:

Negative Examples

- This can now be computed at blazing speed – processing millions of examples in minutes. But there's one loophole we need to close. If all of our examples are positive (target: 1), we open ourselves to the possibility of a smartass model that always returns 1 – achieving 100

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Negative Examples (II)

- For each sample in our dataset, we add negative examples. Those have the same input word and a 0 label.

input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

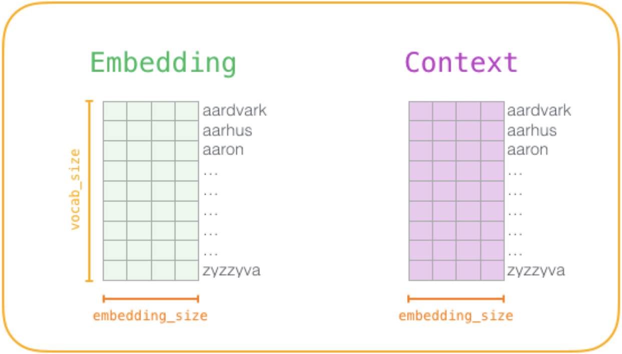
Negative examples

- We are contrasting the actual signal (positive examples of neighboring words) with noise (randomly selected words that are not neighbors). This leads to a great tradeoff of computational and statistical efficiency.

Training process

- Now that we've established the two central ideas of skipgram and negative sampling, we can proceed to look closer at the actual word2vec training process.
- Before the training process starts, we preprocess the text we're training the model against. In this step, we determine the size of our vocabulary (we'll call this vocab_size) and which words belong to it.
- At the start of the training phase, we create two matrices – an Embedding matrix and a Context matrix. These two matrices have an embedding for each word in our vocabulary (So vocab_size is one of their dimensions). The second dimension is how long we want each embedding to be (embedding_size – 300 is a common value).

Training process



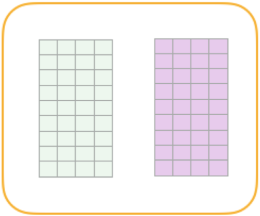
Training process: Step-by-step

1. At the start of the training process, we initialize these matrices with random values. Then we start the training process. In each training step, we take one positive example and its associated negative examples. Let's take our first group:

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

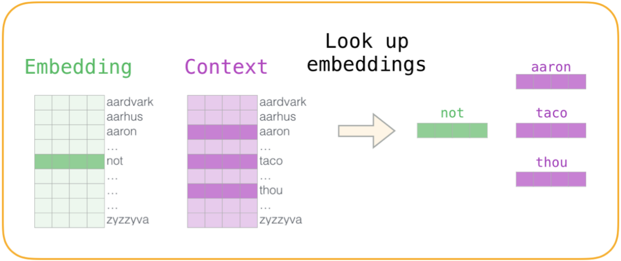
model



Training process: Step-by-step (II)

- Now we have four words: the input word "not" and the output/context words: "thou" (the actual neighbor), "aaron", and "taco" (the negative examples).

2. We proceed to look up their embeddings – for the input word, we look in the Embedding matrix. For the context words, we look in the Context matrix (even though both matrices have an embedding for every word in our vocabulary).



Training process: Step-by-step (III)

3. Then, we take the dot product of the input embedding with each of the context embeddings. In each case, that would result in a number that indicates the similarity of the input and context embeddings.
4. Now we need a way to turn these scores into something that looks like probabilities – we need them to all be positive and have values between zero and one. This is a great task for sigmoid, the logistic operation. And we can now treat the output of the sigmoid operations as the model's output for these examples.

input word	output word	target	input • output	sigmoid()
not	thou	1	0.2	0.55
not	aaron	0	-1.11	0.25
not	taco	0	0.74	0.68

Training process: Step-by-step (IV)

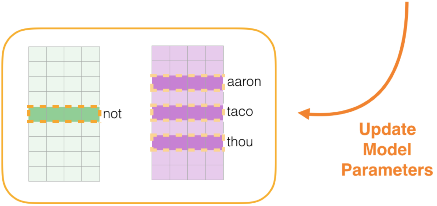
5. Now that the untrained model has made a prediction, and seeing as though we have an actual target label to compare against, let's calculate how much error is in the model's prediction. To do that, we just subtract the sigmoid scores from the target labels.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	−1.11	0.25	−0.25
not	taco	0	0.74	0.68	−0.68

Training process: Step-by-step (V)

6 Here comes the "learning" part of "machine learning". We can now use this error score to adjust the embeddings of "not", "thou", "aaron", and "taco" so that the next time we make this calculation, the result would be closer to the target scores.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	−1.11	0.25	−0.25
not	taco	0	0.74	0.68	−0.68



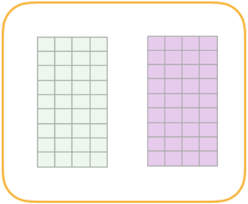
Training process: Step-by-step (VI)

7. This concludes the training step. We emerge from it with slightly better embeddings for the words involved in this step ("not", "thou", "aaron", and "taco"). We now proceed to our next step (the next positive sample and its associated negative samples) and do the same process again.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



Training process: Step-by-step (VII)

- The embeddings continue to be improved while we cycle through our entire dataset for a number of times. We can then stop the training process, discard the Context matrix, and use the Embeddings matrix as our pre-trained embeddings for the next task.

Outline

Other Language Units

- Phrase: Washington_Post is a newspaper.
  - Phrases can be automatically generated based on counts, e.g.:

$$\frac{\text{count}(w_i, w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}$$

- Character: W a s h i n g t o n _ P o s t _ i s _ a _ n e w s p a p e r
  - Create a word representation from its character
  - Fully character-level models
- Sub-word: Wash #ing #ton Post is a news #paper
  - N-grams, Byte Pair Encoding (BPE), Wordpiece, Sentencepiece

Sub-word Model: fastText

fastText (Facebook, 2016)

- Subword-based skip-gram architecture: the vector representation of a word is the sum of the embeddings of the character n-grams of the current word ($3 \leq n \leq 6$ by default).

Ex: the fastText representation of the word 'where' is the sum of 15 subwords (n-grams) embeddings:

- 3 grams: <wh, whe, her, ere, re>
- 4 grams: <whe, wher, here, ere>
- 5 grams: <wher, where, her>
- 6 grams: <where, where>
- $+$ the word itself: <where>

Improvements of fastText over word2vec

1. **Subword Modeling**: fastText uses a subword-based skip-gram architecture. The vector representation of a word is the sum of character n-gram embeddings. This allows fastText to capture morphological information and handle out-of-vocabulary (OOV) words effectively.
2. **Hashing Function**: fastText employs a hashing function to reduce memory usage. Instead of storing all possible n-grams explicitly, fastText applies a hashing trick to map n-grams into a fixed-size hash space. The hashing function is defined as follows:

$$\text{hash\_function}(\text{n-gram}) = \text{hash}(\text{n-gram}) \mod \text{bucket\_size}$$

3. **Flexible n-gram Selection**: fastText allows customizing the range of character n-grams considered during training to adjust it based on language or task characteristics.

Hybrid Model: GloVe

GloVe: Global Vectors for Word Representation

- Hybrid: co-occurrence counts + prediction
- Ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning.
- The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.
- The training objective is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence (ratio equals difference of logs).

Hybrid Model: GloVe (II)

- Co-occurrence Probability:

$$P_{ij} = \frac{X_{ij}}{X_i}$$

- Word Vector Dot Product:

$$\mathbf{v}_i \cdot \mathbf{v}_j = \log(P_{ij})$$

- Ratio of Co-occurrence Probabilities:

$$\frac{P_{ij}}{P_{ik}} = \frac{\exp(\mathbf{v}_i \cdot \mathbf{v}_j)}{\exp(\mathbf{v}_i \cdot \mathbf{v}_k)}$$

- Difference of Logs:

$$\mathbf{v}_i \cdot \mathbf{v}_j - \mathbf{v}_i \cdot \mathbf{v}_k = \log(P_{ij}) - \log(P_{ik})$$

GloVe: Training Algorithm

1. Initialize word vectors $\mathbf{v}_i$ and biases b_i.
2. Compute the ratio of co-occurrence probabilities for each word pair: $\frac{P_{ij}}{P_{ik}}$.
3. Define the loss function: $J = \sum_{i,j} f\left(P_{ij}\right)\left(\mathbf{v}_i \cdot \mathbf{v}_j - \log(P_{ij})\right)^2$.
4. Update word vectors and biases using gradient descent to minimize the loss function.
5. Repeat steps 2-4 until convergence.

Outline

Example

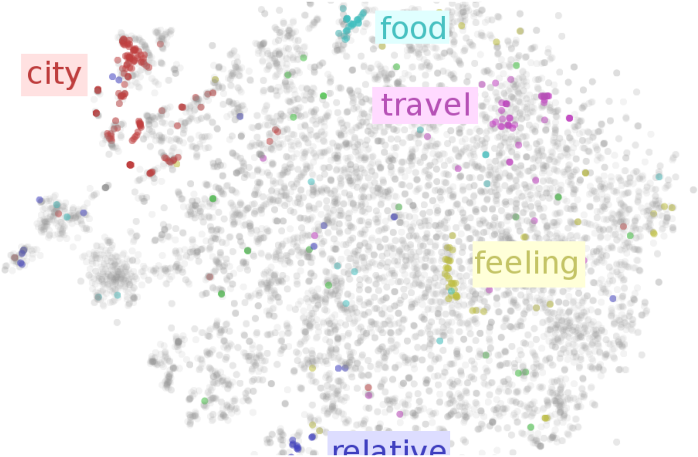
Closest words to the target word 'frog':

- frogs
- toad
- litoria
- leptodactylidae
- lizard
- eleutherodactylus

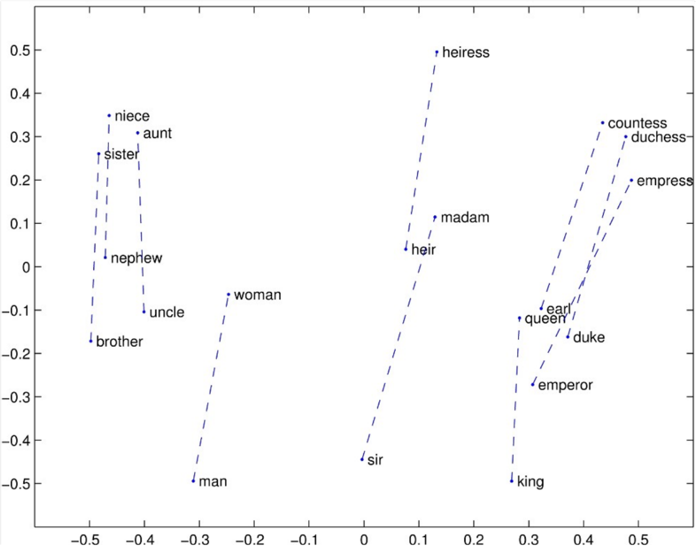
Translations of 'frog' (aligned word-embeddings):

- 'rana', 'granota'
- 'ranas', 'granotes'
- 'sapo', 'gripau'
- 'litoria', 'litòria'

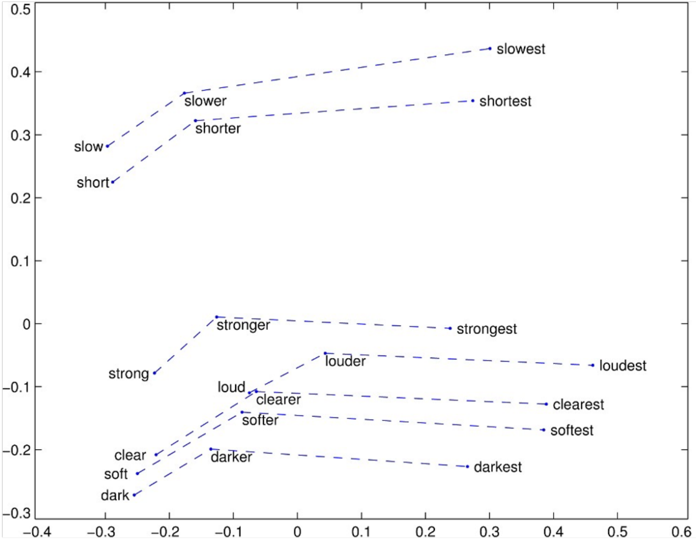
Visualizing Representations



Example: Linear structures man-woman



Example: Linear structures comparative - superlative



Example: Catalan word vectors (CBOW)

- 'dimecres' + ('dimarts' - 'dilluns') = 'dijous'
- 'tres' + ('dos' - 'un') = 'quatre'
- 'tres' + ('2' - 'dos') = '3'
- 'viu' + ('coneixia' - 'coneix') = 'vivia'
- 'la' + ('els' - 'el') = 'les'
- 'Polònia' + ('francès' - 'França') = 'polonès'

Question

How can we evaluate word vectors?

Evaluation

Intrinsic vs. Extrinsic evaluation:

- Intrinsic evaluation: evaluating word vectors based on their similarity, analogy, and distance.
- Extrinsic evaluation: evaluating word vectors in the context of downstream tasks such as translation and sentiment analysis.

Intrinsic evaluation methods include:

- Word similarity: finding the closest word to a target word.
- Word analogy: finding a word that completes an analogy (e.g., "a is to b as c is to...").
- Distance: measuring similarity using cosine similarity, Euclidean distance, or dot product.

Outline

Challenges of Word Vectors

The challenges of word vectors in neural networks for language include:

- Properly evaluating word vectors for similarity, analogy, and distance.
- Handling large datasets with millions or billions of words.
- Ensuring that mathematical operations encode meaning in word vectors.
- Capturing the meaning of a word based on its context and co-occurrence.

Summary

Meaning Word Embedding

Any technique mapping a word (or phrase) from its original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space - so one embeds the word in a different space.

Importance of Word Embedding

Word representations are a critical component of many natural language processing systems.

Summary: Take home message

- Similarity in meaning is reflected in similarity in vectors. Mathematics should be able to encode meaning.
- "You shall know a word by the company it keeps" - the environment of a word gives meaning to it.
- Use big datasets, especially neural models, require lots of data!

Motivation

Types of Word Vectors

Visualization and Evaluation

Summary

Annexes

Outline

WordNet vs. Embeddings in Clustering (Marcinczuk et al., 2021)

- The study compared WordNet-based similarity measures with TF-IDF, word2vec, and BERT embeddings for clustering Polish text documents.
- Results showed that WordNet-based measures (e.g., Wu-Palmer) can outperform or compete with modern embedding-based approaches.
- Key findings:
  - Wu-Palmer (WordNet) achieved the highest AMI score for the PPKZ dataset.
  - TF-IDF performed best for the Market and Higher Education datasets.
  - BERT underperformed due to limitations in handling long documents.

WordNet vs. Embeddings in Clustering (Marcinczuk et al., 2021) (II)

Table: Adjusted Mutual Information (AMI) Scores

Method	**ALL**	**PPKZ**	**Market**	**Higher Edu.**
Wu-Palmer (adj)	0.536	0.441	0.398	0.499
TF-IDF (adj)	0.529	0.289	0.460	0.507
doc2vec (allposes)	0.508	0.390	0.498	0.449
BERT	0.360	0.095	0.344	0.287