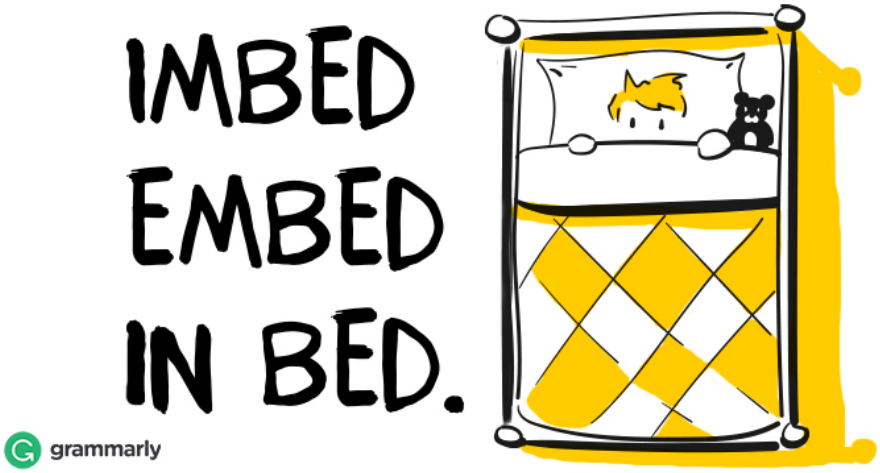
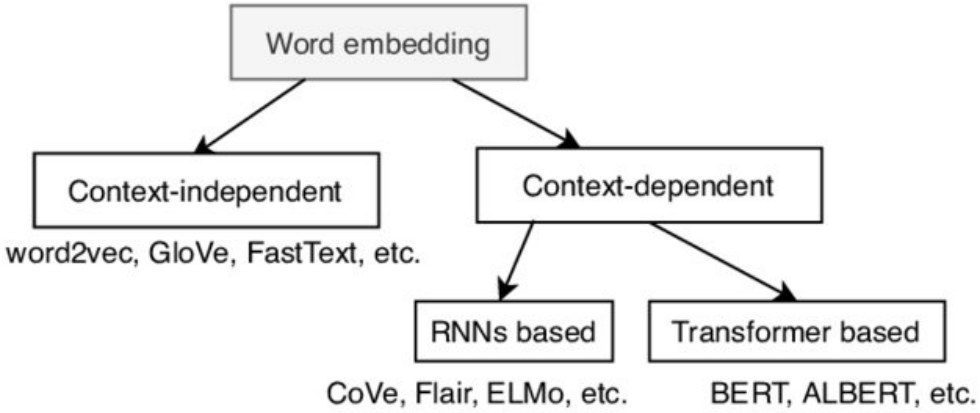


Word Embeddings

An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models.



Context-independent

Context-independent methods are known as “classic” word embeddings, which learn representations through language model (LM) based shallow neural networks or co-occurrence matrix factorisation [34]. The learned representations are characterised by being unique and distinct for each word without considering the word’s context. Hence, these are usually pre-trained on general text corpora and are distributed in the form of downloadable files, which may be directly applied to initialise the embedding weights for downstream language tasks.

Context-dependent

In contrast with context-independent word embeddings, context dependent methods learn different embeddings for the same word that is dependent on the context in which it is used. For example, the polysemy “bank”, will have multiple embeddings depending on whether it is used in a river-related context or finance-related one. This feature has brought context-dependent embeddings into the mainstream in recent times. As research has progressed, the learning methods for context-dependent word embedding have mainly evolved in two categories. One category consists of those approaches based on LM-based Recurrent Neural Networks

(RNNs), such as CoVe [20], Flair [2] and ELMo [24]. Alternatively, the recently developed Transformer-based models [28], such as BERT [7] and ALBERT [17], have been demonstrated to learn efficient contextualised word representations.

Examples (start with something simple)

OneHotEncoder

In [1]:

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
_ = enc.fit(X)

enc.categories_
```

Out[1]:

```
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
```

In [2]:

```
enc.transform([['Female', 1], ['Male', 4]]).toarray()
```

Out[2]:

```
array([[1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

Term Frequency inverse Document Frequency

In [3]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

dataset = [
    "I enjoy reading about Machine Learning and Machine Learning is my PhD subject",
    "I would enjoy a walk in the park",
    "I was reading in the library"
]
```

In [4]:

```
tfIdfVectorizer = TfidfVectorizer(use_idf=True)
tfIdf = tfIdfVectorizer.fit_transform(dataset)

tfIdf[0].T.todense()
```

Out[4]:

```
matrix([[0.25685987],
        [0.25685987],
        [0.19534855],
        [0.        ],
        [0.25685987],
        [0.51371974],
        [0.        ],
        [0.51371974],
        [0.25685987],
        [0.        ],
        [0.25685987],
        [0.19534855],
        [0.25685987],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ]])
```

In [5]:

```
df = pd.DataFrame(tfIdf[0].T.todense(), index=tfIdfVectorizer.get_feature_names(), colu
df = df.sort_values('TF-IDF', ascending=False)

print (df.head(25))
```

```
            TF-IDF
machine   0.513720
learning  0.513720
about     0.256860
subject   0.256860
phd       0.256860
and       0.256860
my        0.256860
is        0.256860
reading   0.195349
enjoy     0.195349
library   0.000000
park      0.000000
in        0.000000
the       0.000000
walk      0.000000
was       0.000000
would     0.000000
```

Examples

Word2Vec

In [6]:

```
from gensim.test.utils import common_texts
from gensim.models import Word2Vec

model = Word2Vec(sentences=common_texts, vector_size=20, window=5, min_count=1, workers
```

In [7]:

```
common_texts
```

Out[7]:

```
[['human', 'interface', 'computer'],
 ['survey', 'user', 'computer', 'system', 'response', 'time'],
 ['eps', 'user', 'interface', 'system'],
 ['system', 'human', 'system', 'eps'],
 ['user', 'response', 'time'],
 ['trees'],
 ['graph', 'trees'],
 ['graph', 'minors', 'trees'],
 ['graph', 'minors', 'survey']]
```

In [8]:

```
model.wv['human']
```

Out[8]:

```
array([-0.03532419, -0.02431326, -0.01889282, -0.042681  ,  0.03977803,
       -0.02421969,  0.04211806,  0.02631285, -0.03275013,  0.01978935,
        0.02735075, -0.03713268, -0.0370286 , -0.01237615, -0.04312863,
       -0.00790787, -0.00201716,  0.01649842,  0.0072094 , -0.00440711],
      dtype=float32)
```

In [9]:

```
model.wv.most_similar('computer', topn=5)
```

Out[9]:

```
[('eps', 0.12416858971118927),
 ('user', 0.08889889717102051),
 ('graph', 0.07645358890295029),
 ('human', 0.04767070710659027),
 ('trees', 0.029164940118789673)]
```

In [10]:

```
model.wv.similarity('human', 'computer')
```

Out[10]:

```
0.047670692
```

Sentence Embeddings

Sentence embedding techniques represent entire sentences and their semantic information as vectors. This helps the machine in understanding the context, intention, and other nuances in the entire text.

Doc2Vec

In [11]:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
import numpy as np
```

```
[nltk_data] Downloading package punkt to C:\Users\Lucas Zanco
[nltk_data]     Ladeira\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [12]:

```
sentences = ["I ate dinner.",
       "We had a three-course meal.",
       "Brad came to dinner with us.",
       "He loves fish tacos.",
       "In the end, we all felt like we ate too much.",
       "We all agreed; it was a magnificent evening."]
```

In [13]:

```
tokenized_sent = []
for s in sentences:
    tokenized_sent.append(word_tokenize(s.lower()))

tokenized_sent
```

Out[13]:

```
[['i', 'ate', 'dinner', '.'],
 ['we', 'had', 'a', 'three-course', 'meal', '.'],
 ['brad', 'came', 'to', 'dinner', 'with', 'us', '.'],
 ['he', 'loves', 'fish', 'tacos', '.'],
 ['in',
  'the',
  'end',
  ',',
  'we',
  'all',
  'felt',
  'like',
  'we',
  'ate',
  'too',
  'much',
  '.'],
 ['we', 'all', 'agreed', ';', 'it', 'was', 'a', 'magnificent', 'evening',
'.']]
```

In [14]:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

tagged_data = [TaggedDocument(d, [i]) for i, d in enumerate(tokenized_sent)]
tagged_data
```

Out[14]:

```
[TaggedDocument(words=['i', 'ate', 'dinner', '.'], tags=[0]),
 TaggedDocument(words=['we', 'had', 'a', 'three-course', 'meal', '.'], tags=
[1]),
 TaggedDocument(words=['brad', 'came', 'to', 'dinner', 'with', 'us', '.'], t
ags=[2]),
 TaggedDocument(words=['he', 'loves', 'fish', 'tacos', '.'], tags=[3]),
 TaggedDocument(words=['in', 'the', 'end', ',', 'we', 'all', 'felt', 'like',
'we', 'ate', 'too', 'much', '.'], tags=[4]),
 TaggedDocument(words=['we', 'all', 'agreed', ';', 'it', 'was', 'a', 'magnif
icent', 'evening', '.'], tags=[5])]
```

In [15]:

```
model = Doc2Vec(tagged_data, vector_size=20, window=2, min_count=1, epochs=100)
```

In [16]:

```
test_doc = word_tokenize("I had pizza and pasta".lower())
test_doc_vector = model.infer_vector(test_doc)

test_doc_vector
```

Out[16]:

```
array([-0.00832267, -0.00017435, -0.02142651, -0.00225561,  0.00838164,
       -0.01318515,  0.01797567,  0.02194282, -0.00714004, -0.01010755,
        0.01935871, -0.00529   , -0.04461143, -0.02153678, -0.01111172,
       -0.00474056, -0.01153947, -0.01720854, -0.03053446, -0.00500126],
      dtype=float32)
```

In [17]:

```
ms = model.docvecs.most_similar(positive=[test_doc_vector])
ms
```

```
<ipython-input-17-67cb82d55d88>:1: DeprecationWarning: Call to deprecated `d
ocvecs` (The `docvecs` property has been renamed `dv`.).
  ms = model.docvecs.most_similar(positive=[test_doc_vector])
```

Out[17]:

```
[(3, 0.5333281755447388),
 (5, 0.5016788840293884),
 (1, 0.47435587644577026),
 (2, 0.45826831459999084),
 (4, 0.3940267264842987),
 (0, 0.3520171046257019)]
```

In [18]:

```
sentences[ms[0][0]]
```

Out[18]:

```
'He loves fish tacos.'
```

SentenceBERT

In [19]:

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('bert-base-nli-mean-tokens')
```

In [20]:

```
sentence_embeddings = model.encode(sentences)

query = "I had pizza and pasta"
query_vec = model.encode([query])[0]
```

In [22]:

```
import numpy as np

def cosine(u, v):
    return np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))
```

In [23]:

```
for sent in sentences:
    sim = cosine(query_vec, model.encode([sent])[0])
    print("Sentence = ", sent, "; similarity = ", sim)
```

```
Sentence =  I ate dinner. ; similarity =  0.7173461
Sentence =  We had a three-course meal. ; similarity =  0.6371337
Sentence =  Brad came to dinner with us. ; similarity =  0.5897907
Sentence =  He loves fish tacos. ; similarity =  0.62239325
Sentence =  In the end, we all felt like we ate too much. ; similarity =  0.
41980493
Sentence =  We all agreed; it was a magnificent evening. ; similarity =  0.1
8081592
```

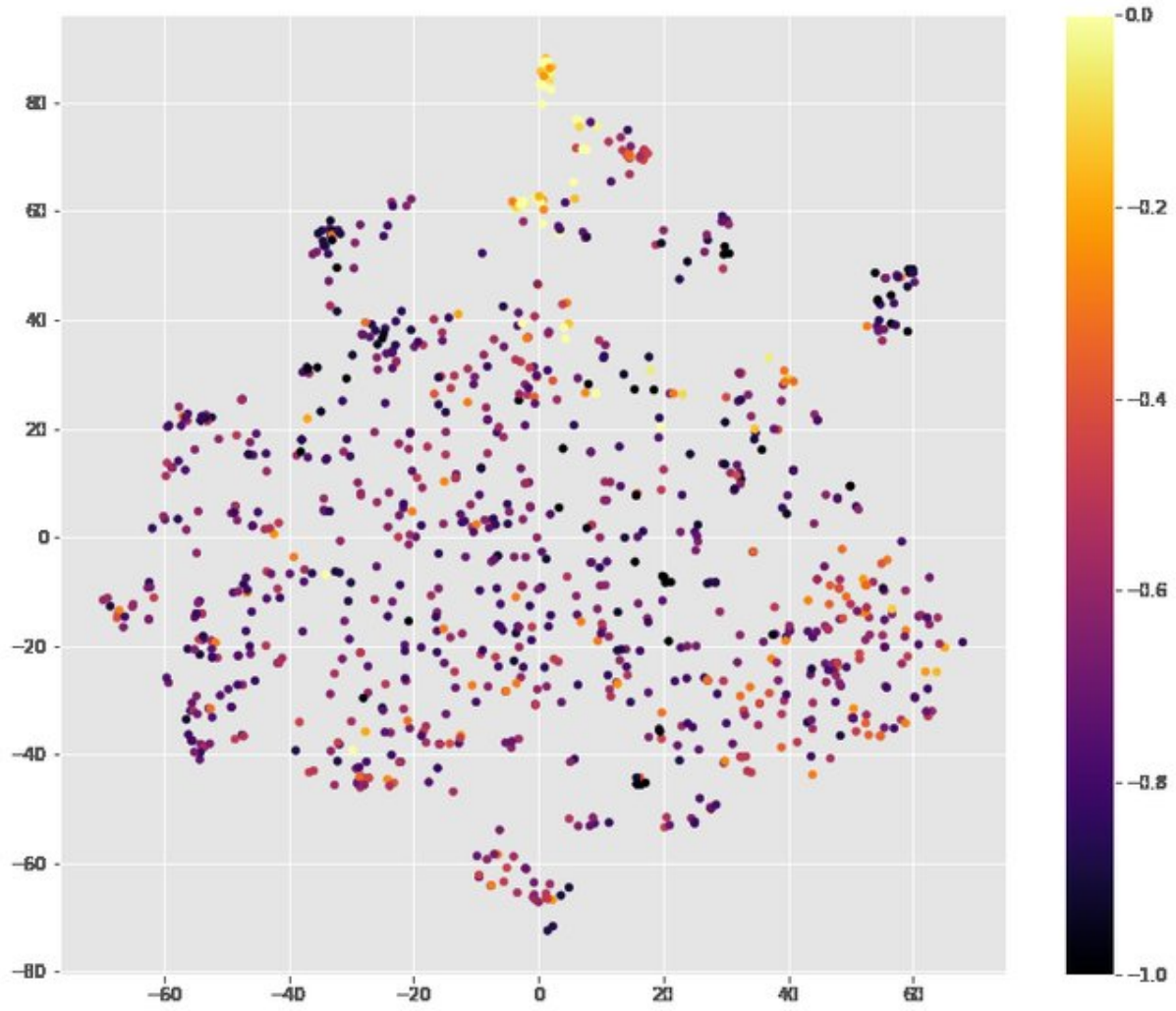
PhD

Data augmentation make small modifications to texts to generate new data. These modifications may comprehend to add words, switch words, translate, and more. Some models may be used to find synonyms and close words to switch them. However, literature techniques use only the textual representation of sentences (despite translation) and not the vector representation.

Embedding algorithms are evolving fast, which already enable multi-language and language agnostic vectorization. So, how do we apply transformations to sentence embeddings?

Hypothesis: With a big enough embedding, we may apply transformations within vector space.

That is one of the things I want to explore in my PhD. How it really works and propose transformations in vector space.



References

- https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture (https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture)
- https://www.researchgate.net/publication/348946675_A_Comparative_Study_on_Word_Embeddings_in_Dee (https://www.researchgate.net/publication/348946675_A_Comparative_Study_on_Word_Embeddings_in_De
- https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275 (https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275)
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)
- https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa)
- https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92 (https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92)
- https://radimrehurek.com/gensim/models/word2vec.html (https://radimrehurek.com/gensim/models/word2vec.html)
- https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/ (https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/)
- https://www.researchgate.net/publication/335701251_Analysing_Representations_of_Memory_Impairment_i (https://www.researchgate.net/publication/335701251_Analysing_Representations_of_Memory_Impairment_


QUESTION
DOES ANYONE HAVE QUESTIONS?