THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Evaluation on Sentence Embeddings by Semantic Similarity

Kahang Ngau
Qingyuan Xie

Word Embedding vs Sentence Embedding

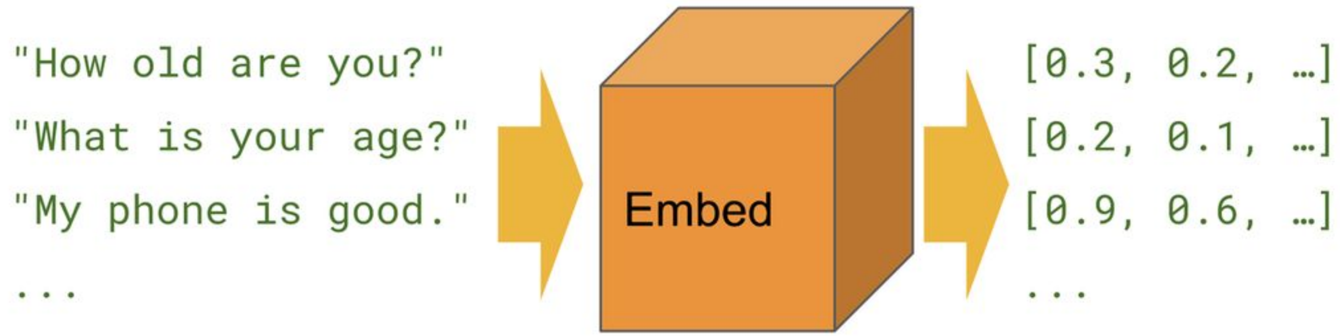
What is Word Embedding?

- Built vector representation to identify the semantic and syntaxes of the word, why?
- Calculate the distance to figure out if two data points are similar to each other or not
- Popular techniques are Word2Vec, GloVe, and etc.

Sentence Embedding

- Extension of Word Embedding
- Represents the entire sentences semantic information as vectors
- Allows the machine to understand the context, intension, and other nuances from the sentence

Sentence Embedding

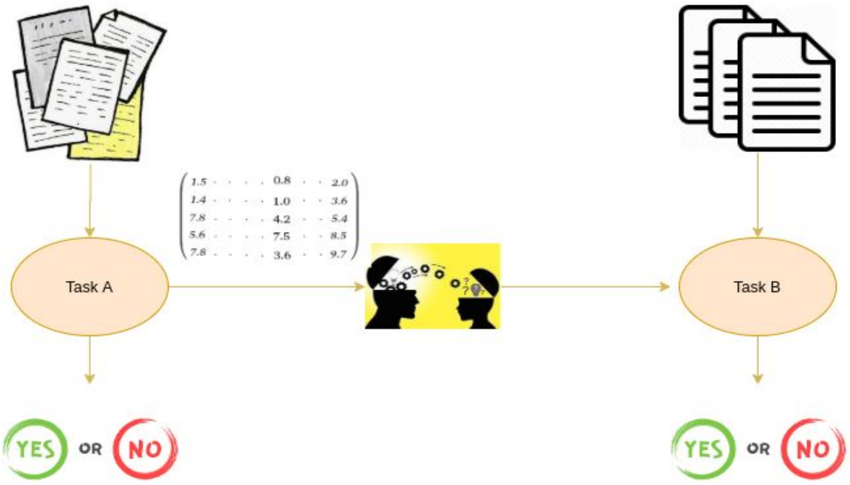


Objective

To compare the performance of different pre-trained models on calculating the semantic similarity score with the same input sentences

Transfer Learning and Pre-Trained Word Embeddings

- Pre-trained embedding is a form of transfer learning.
- It captures both the semantic and syntactic meaning when it is trained on large datasets.
- Time saving and accessible



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

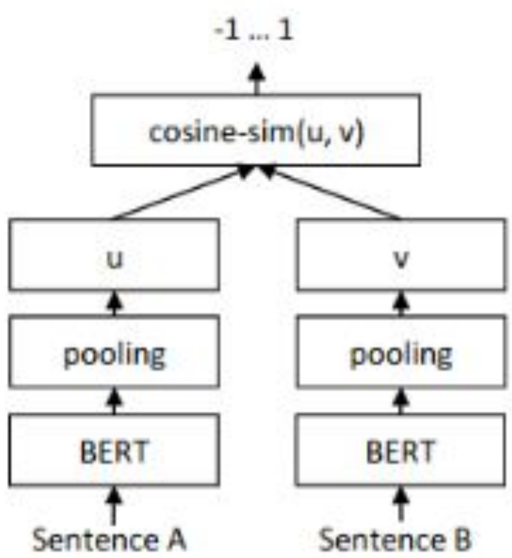
Pre-Trained Models

1. Sentence-BERT
2. Doc2Vec
3. Universal Sentence Encoder
    a. Transformer
    b. Deep Averaging Network (DAN)

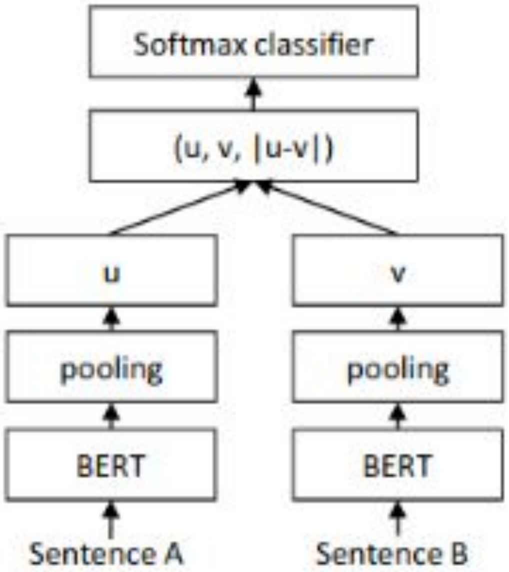
Sentence BERT

SentenceBERT

- Siamese network like architecture(build on minimal training data) with 2 sentence as an input
- regression or classification models



Sentence-BERT for sentence similarity(Regression Task)

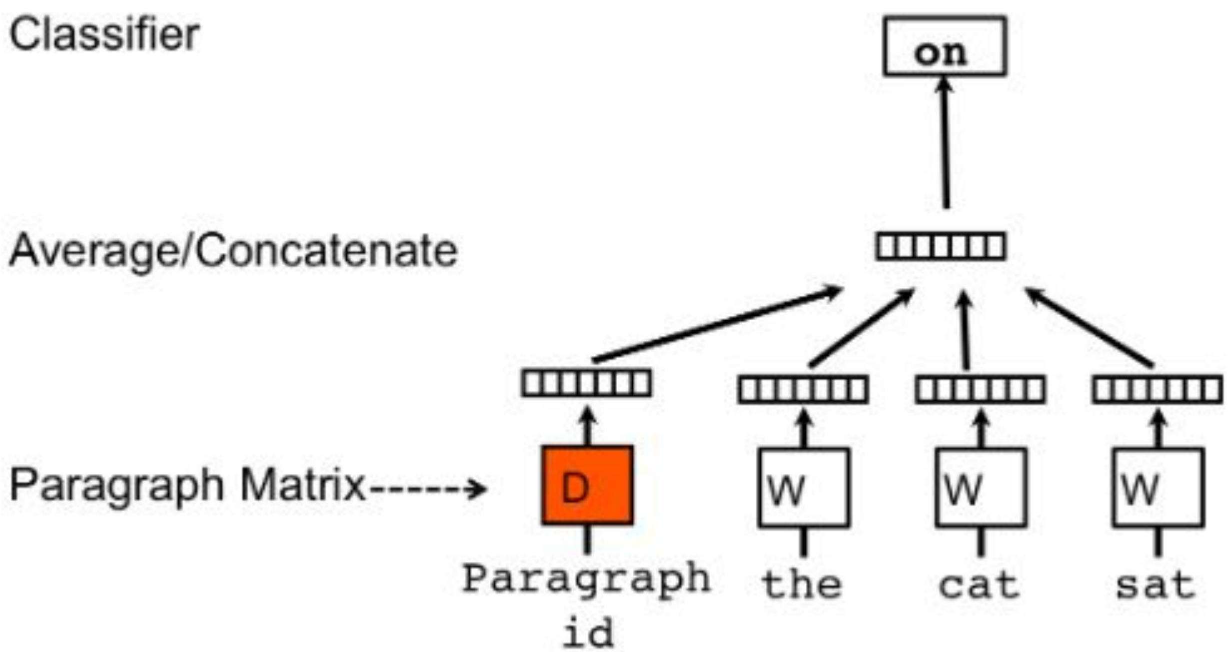


Sentence-BERT for Classification task

Doc2Vec

- Unsupervised learning approach to represent paragraph or sentence in vectors based off Word2Doc

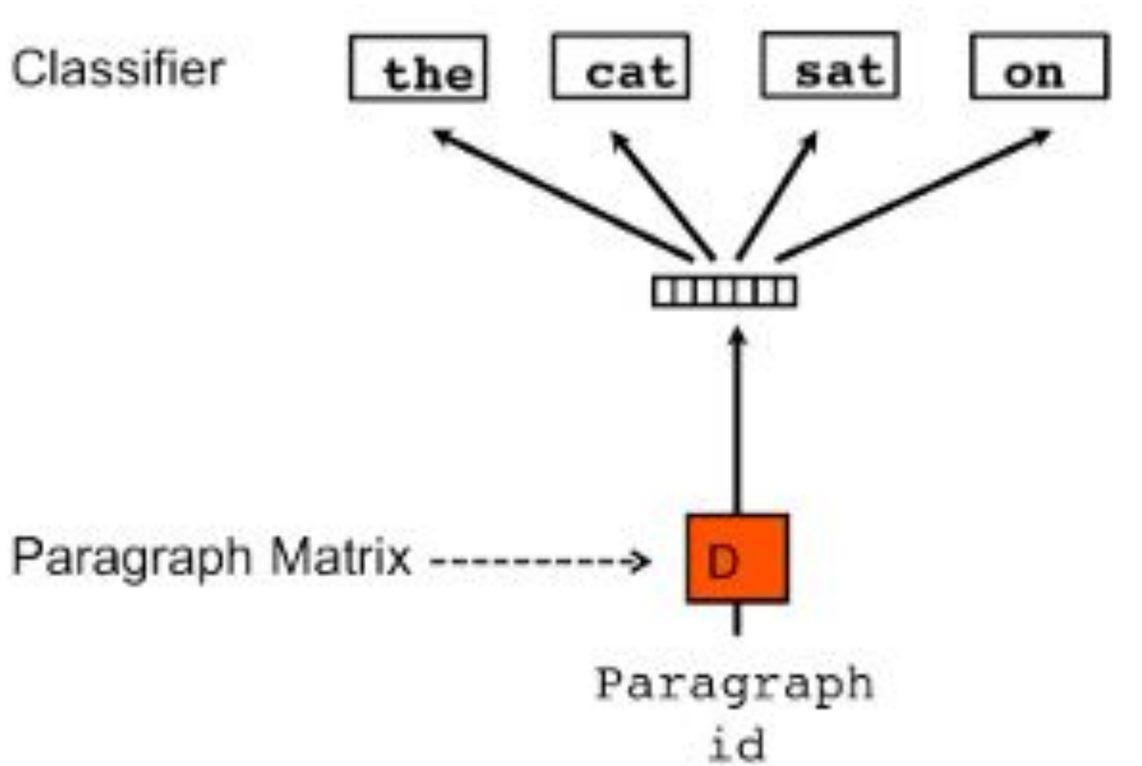
Distributed memory model(PVDM)



Doc2Vec

Distributed Bag of Words

- **extract features of a paragraph through optimization with sampled words**



Universal Sentence Encoder

- One of the most well-performing sentence embedding techniques
- Sentence gets converted to a 512-dimensional vector
- Feasible for multi-tasking
- Two types of model:
  - Transformer
  - Deep Averaging Network (DAN)

	Transformer model	Deep Averaging Network (DAN) model
Vector Length	512	512
Encoding time with sentence length	Non-Linear	Linear
Memory usage	High	Medium
Accuracy	Very High	High

Implementation Doc2Vec

```
# Install TF-Hub&libraries
!pip3 install --upgrade tensorflow-gpu
!pip3 install tensorflow-hub

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

#input
sentences = ["Did you watch the movie Glass?",
             "Be careful, I just broke a glass cup",
             "This window is made of glass.",
             "He doesn't usually wear glasses."]
```

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
tagged_data = [TaggedDocument(d, [i]) for i, d in enumerate(tokenized_sent)]
tagged_data
```

```
[TaggedDocument(words=['did', 'you', 'watch', 'the', 'movie', 'glass', '?'], tags=[0]),
 TaggedDocument(words=['be', 'careful', ',', 'i', 'just', 'broke', 'a', 'glass', 'cup'], tags=[1]),
 TaggedDocument(words=['this', 'window', 'is', 'made', 'of', 'glass', '.'], tags=[2]),
 TaggedDocument(words=['he', 'does', "n't", 'usually', 'wear', 'glasses', '.'], tags=[3])]
```

```
test_doc = word_tokenize("This glass cup is fragile".lower())
test_doc_vector = model.infer_vector(test_doc)
model.docvecs.most_similar(positive = [test_doc_vector])
```

```
[(0, -0.05732543021440506),
 (2, -0.07792545855045319),
 (1, -0.10838853567838669),
 (3, -0.13605502247810364)]
```

Implementation Sentence-BERT

```
#load model
from sentence_transformers import SentenceTransformer
sbert_model = SentenceTransformer('bert-base-nli-mean-tokens')
```

```
#function that returns cosine similarity between the two vectors
def cosine(u, v):
    return np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))
```

```
query = "This glass cup is fragile"
query_vec = sbert_model.encode([query])[0]
```

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

```
Sentence =  Did you watch the movie Glass? ; similarity =  0.2366257
Sentence =  Be careful, I just broke a glass cup ; similarity =  0.46764034
Sentence =  This window is made of glass. ; similarity =  0.6351793
Sentence =  He doesn't usually wear glasses. ; similarity =  0.6418279
```

Implementation USE

```
module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
model = hub.load(module_url)

# For the first message
sentence_embeddings = model(messages)
query = "The bright spot in the sky at night is a star"
query_vec = model([query])[0]

random.seed(10)
for sent in messages:
  sim = cosine(query_vec, model([sent])[0])
  print("Sentence = ", sent, "; similarity = ", sim)
```

```
Sentence =  That person is a real rock star ; similarity =  0.18940803
Sentence =  I did not know the sun was a star ; similarity =  0.5535026
Sentence =  twinkle twinkle little star ; similarity =  0.4428834
Sentence =  please have this star shaped cake ; similarity =  0.2860058
```

```
# For the Second message
sentence_embeddings = model(messages1)
query = "This glass cup is fragile"
query_vec = model([query])[0]

random.seed(10)
for sent in messages1:
  sim = cosine(query_vec, model([sent])[0])
  print("Sentence = ", sent, "; similarity = ", sim)
```

```
Sentence =  Did you watch the movie Glass ; similarity =  0.3120871
Sentence =  Be careful, I just broke a glass cup ; similarity =  0.6632582
Sentence =  This window is made of glass ; similarity =  0.6765582
Sentence =  He does not usually wear glasses ; similarity =  0.1990158
```

```
USE_large = hub.load("https://tfhub.dev/google/universal-sentence-encoder-large/5")

# For the first message
random.seed(10)
query = "The bright spot in the sky at night is a star"
query_vec = USE_large([query])[0]

for sent in messages:
  sim = cosine(query_vec, USE_large([sent])[0])
  print("Sentence = ", sent, "; similarity = ", sim)
```

```
Sentence =  That person is a real rock star ; similarity =  0.2584718
Sentence =  I did not know the sun was a star ; similarity =  0.57124144
Sentence =  twinkle twinkle little star ; similarity =  0.37080938
Sentence =  please have this star shaped cake ; similarity =  0.22781506
```

```
# For the second message
random.seed(10)
query = "This glass cup is fragile"
query_vec = USE_large([query])[0]

for sent in messages1:
  sim = cosine(query_vec, USE_large([sent])[0])
  print("Sentence = ", sent, "; similarity = ", sim)
```

```
Sentence =  Did you watch the movie Glass ; similarity =  0.29692373
Sentence =  Be careful, I just broke a glass cup ; similarity =  0.5774079
Sentence =  This window is made of glass ; similarity =  0.6119121
Sentence =  He does not usually wear glasses ; similarity =  0.17812516
```

Results

"The bright spot in the sky at night is a star"

```
messages = [
    "That person is a real rock star",
    "I did not know the sun was a star",
    "twinkle twinkle little star",
    "please have this star shaped cake"
]
```

	Sentence 1	Sentence 2	Sentence 3	Sentence 4
Sentence-BERT	0.442	0.587	0.568	0.455
Doc2Vec	0.289	0.249	0.505	0.125
USE Transformer	0.189	0.554	0.443	0.286
USE DAN	0.258	0.571	0.371	0.228

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Results

"This glass cup is fragile"

```
messages1 = [
    "Did you watch the movie Glass",
    "Be careful, I just broke a glass cup",
    "This window is made of glass",
    "He does not usually wear glasses"
]
```

	Sentence 1	Sentence 2	Sentence 3	Sentence 4
Sentence-BERT	0.251	0.468	0.617	0.645
Doc2Vec	0.282	0.112	0.218	0.162
USE Transformer	0.312	0.663	0.677	0.199
USE DAN	0.297	0.577	0.612	0.178

Conclusion & Future Work

- Each model has its best performing text tasks, so better to choose model based on the task(e.g. semantic relatedness or sentiment analysis)
- There is a consensus between the two sub-models understand USE
- Doc2Vec model seems to have a different similarity scores comparing to other models
- USE seems to be most reliable models (among the three models) in presenting semantic similarity scores
- Discover more pre-trained models, like stsb-mpnet-base-v2 and stsb-roberta-base-v2 to compare the results

References

1. TensorFlow Hub
2. Guide To Universal Sentence Encoder With TensorFlow
3. Pretrained Word Embeddings | Word Embedding NLP
4. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
5. 4 Sentence Embedding Techniques One Should Know| With Python Codes
6. https://arxiv.org/pdf/2004.09813.pdf
7. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data
8. facebookresearch/SentEval: A python tool for evaluating the quality of sentence embeddings.
9. Sentence level embeddings from BERT | DAIR.AI
10. Paper Summary: Evaluation of sentence embeddings in downstream and linguistic probing tasks