CS-410: Text Information Systems

Technology Review

Sentence Embeddings

Discussion of some popular techniques

November 10th, 2020

Prepared By: **Pushpit Saxena (netid: pushpit2)**

Introduction

In the current era of blazing fast internet along with ease of acquiring a computing device (esp. mobile phones), humans are creating data at unprecedented volumes. One of the fastest growing datasets is unstructured textual data. A lot of daily tasks for humans involve interacting with this unstructured textual data (e.g. via web search). As the volume of unstructured dataset grows so do the need to automate the analysis, summarization and extraction of relevant information, in order to improve the efficiency and experience of interacting with the unstructured textual data, for the people.

Text classification, clustering, semantic similarity and relevance play an important role in many applications, e.g. document retrieval, web search, document recommendation, spam filtering etc. At the center of these applications are different machine learning and natural language processing (NLP) algorithms. Most of these algorithms require text to be represented as a fixed-length feature vector. One of the most common approaches to generate such vector representations for text is bag-of-words(BOW) due to its simplicity, efficiency and often surprising accuracy[1].

So, as the research progress in this area, people started using Word embeddings (e.g. GloVe[3], word2vec[4]), to generate representation of the sentences (one approach is to represent a sentence as the centroid of all its word vectors). This approach to represent a sentence based on the vectors of the words it is composed of is again simple, efficient and easy to implement (provides an easier way to transfer prior learned knowledge from word2vec model). But both bag-of-words and to some extent sentence vector representation based on static word embedding lacks at efficiently representing different syntactic structures within the sentence as well as it performs poorly in preserving the semantic information about the sentence. Hence, the research moved into the area of exploring techniques to enrich the vector representation of the sentence with syntactic as well as semantic information. This area is still a hotbed for new research but there are already some state-of-the-art techniques that are proven to work and we will review some of those techniques as part of this paper:

1. Doc2Vec[1]
2. SBert - Sentence BERT [5, 6]

Sentence Embeddings: Doc2Vec

Doc2Vec [1] can be considered as an extension of word2vec[4]. Doc2Vec introduces *Paragraph Vector*, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. The algorithm represents each document by a dense vector which is trained to predict words in the document. Its construction gives the algorithm the potential to overcome the weaknesses of bag-of-words models[1].

The basic idea behind Doc2Vec is to augment word2Vec with paragraph vectors. The author[1] described two approaches to learn the paragraph vectors:

➔ **PV-DM (Paragraph Vector: Distributed Memory Model)**

This idea of learning paragraph vectors is based on the methods for learning the word vectors. As in word2vec, the learned word vectors are used in prediction tasks about the next word in the sentence, similarly here the paragraph vectors are also used to contribute to the prediction task of the next word given many contexts sampled from the paragraph[1].

Specifically in this framework (see Fig. 1) , each paragraph is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W. The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context.

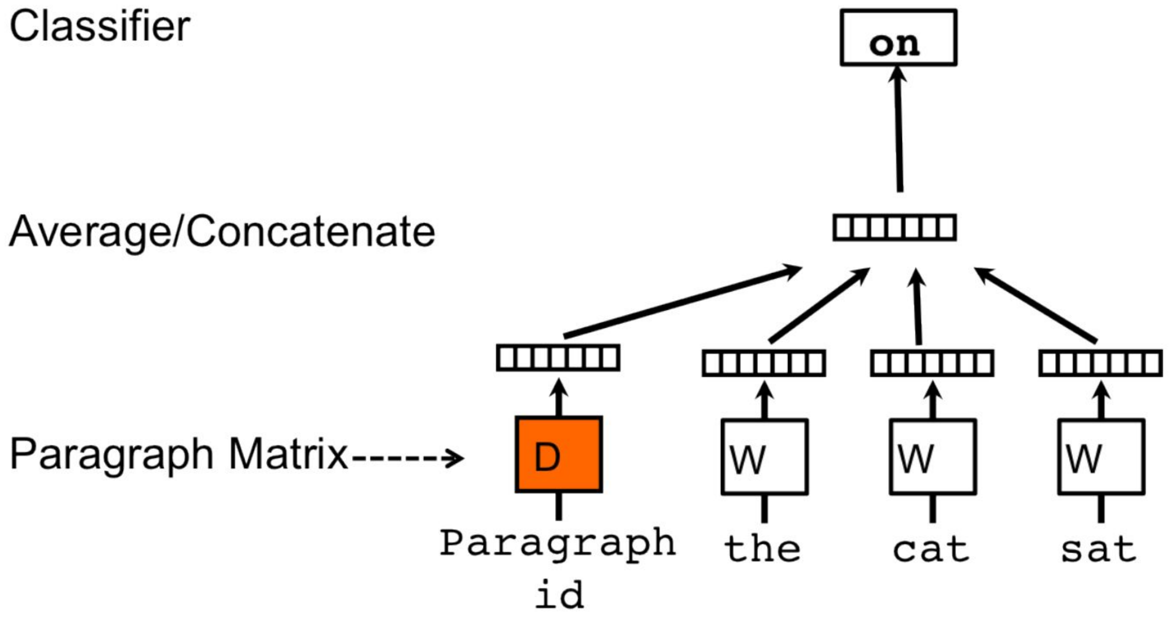


Fig.1: A framework for learning paragraph vector[1].

For generating sentence embeddings, a paragraph vector is assigned to a sentence while word vectors are shared among all sentences. Then the word vector & paragraph vector are either averaged or concatenated to get the final vector representation for the sentence. As one can notice, this approach is similar to CBOW (Continuous Bag-of-Word) word2vec where the next word is predicted based on the given set of words. In PV-DM[1], the next sentence is predicted given a set of sentences.

➔ **PV-DBOW (Paragraph Vector: Distributed bag of words)**

This is the second approach described by the author[1]. Similar to PV-DM, PV-DBOW is another extension over the word2vec basically SG (Skip-gram) word2vec[4]. In this approach, at each iteration the words of the sentence are randomly sampled and then the model is trained to predict that which sentence the words came from, a classification task (Fig. 2).

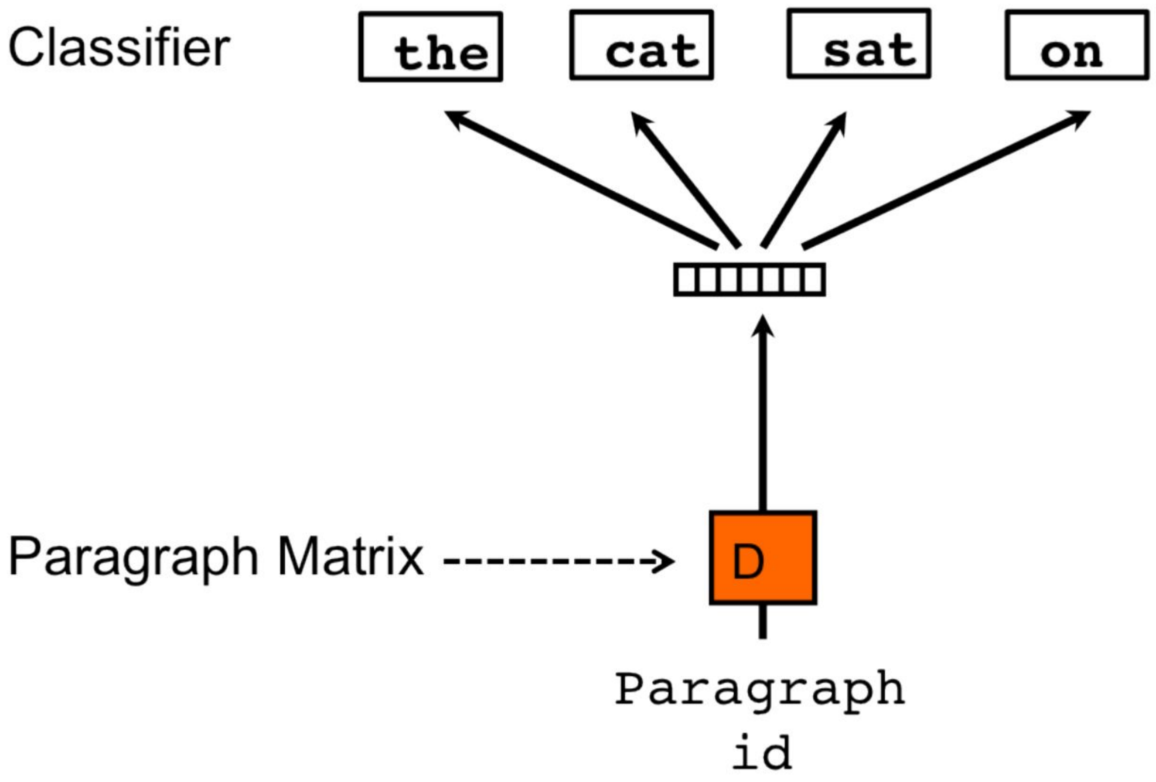


Fig.2: Distributed Bag of Words version of paragraph vectors. In this version, the paragraph vectors is trained to predict the words in a sentence[1]

As per the author[1], PV-DM alone usually works well for most tasks (with state-of-art performances), but its combination with PV-DBOW is usually more consistent across many tasks.

Doc2Vec (simple hands-on exercise in python):

We can use Gensim[7] to show an example of how Doc2Vec can be used.

Install gensim

```
!pip install gensim
```

Sample sentence collection

```
sentences = ['I played basketball',
             'All three of us like to play field sport',
             'All my friends like to play football',
             'I like to play soccer',
             'In India cricket is the most popular sport',
             'Soccer is a sport where each team has 11 players']
```

Tokenize sentences

```
from nltk.tokenize import word_tokenize

# Tokenization of each document
tokenized_sentences = []
for s in sentences:
    tokenized_sentences.append(word_tokenize(s.lower()))
tokenized_sentences
```

```
[['i', 'played', 'basketball'],
 ['all', 'three', 'of', 'us', 'like', 'to', 'play', 'field', 'sport'],
 ['all', 'my', 'friends', 'like', 'to', 'play', 'football'],
 ['i', 'like', 'to', 'play', 'soccer'],
 ['in', 'india', 'cricket', 'is', 'the', 'most', 'popular', 'sport'],
 ['soccer',
  'is',
  'a',
  'sport',
  'where',
  'each',
  'team',
  'has',
  '11',
  'players']]
```

Represent each sentence as a TaggedDocument (containing a list of the words in it and an associated tag)

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

```
[TaggedDocument(words=['i', 'played', 'basketball'], tags=[0]),
 TaggedDocument(words=['all', 'three', 'of', 'us', 'like', 'to', 'play', 'field', 'sport'],
tags=[1]),
 TaggedDocument(words=['all', 'my', 'friends', 'like', 'to', 'play', 'football'], tags=[2]),
 TaggedDocument(words=['i', 'like', 'to', 'play', 'soccer'], tags=[3]),
 TaggedDocument(words=['in', 'india', 'cricket', 'is', 'the', 'most', 'popular', 'sport'],
tags=[4]),
 TaggedDocument(words=['soccer', 'is', 'a', 'sport', 'where', 'each', 'team', 'has', '11',
'players'], tags=[5])]
```

Train Doc2Vec model

```
● doc2vecModel = Doc2Vec(tagged_data, vector_size=20, window = 2, min_count =1 ,
  epochs = 200)
● doc2vecModel.wv.vocab
```

```
{'i': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a730>,
 'played': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a790>,
 'basketball': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a7f0>,
 'all': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a850>,
 'three': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a8b0>,
 'of': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a910>,
 'us': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a640>,
 'like': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a9a0>,
 'to': <gensim.models.keyedvectors.Vocab at 0x7faef1b5aa00>,
 'play': <gensim.models.keyedvectors.Vocab at 0x7faef1b5aa60>,
 'field': <gensim.models.keyedvectors.Vocab at 0x7faef1b5aac0>,
 'sport': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ab20>,
 'my': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ab80>,
 'friends': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ac10>,
 'football': <gensim.models.keyedvectors.Vocab at 0x7faef1b5aca0>,
 'soccer': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ad00>,
 'in': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ad90>,
 'india': <gensim.models.keyedvectors.Vocab at 0x7faef1b5adf0>,
 'cricket': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ae50>,
 'is': <gensim.models.keyedvectors.Vocab at 0x7faef1b5aeb0>,
 'the': <gensim.models.keyedvectors.Vocab at 0x7faef1b5af10>,
 'most': <gensim.models.keyedvectors.Vocab at 0x7faef1b5af40>,
 'popular': <gensim.models.keyedvectors.Vocab at 0x7faef1b5afa0>,
 'a': <gensim.models.keyedvectors.Vocab at 0x7faef1b5ad60>,
 'where': <gensim.models.keyedvectors.Vocab at 0x7faef1b5a580>,
 'each': <gensim.models.keyedvectors.Vocab at 0x7faef1b67070>,
 'team': <gensim.models.keyedvectors.Vocab at 0x7faef1b670d0>,
 'has': <gensim.models.keyedvectors.Vocab at 0x7faef1b67130>,
 '11': <gensim.models.keyedvectors.Vocab at 0x7faef1b67190>,
 'players': <gensim.models.keyedvectors.Vocab at 0x7faef1b671f0>}
```

Testing a new sentence(query) and finding most similar sentences (from collection)

```
test_sen_1 = word_tokenize("I like swimming".lower())
test_sen_1_embedding = doc2vecModel.infer_vector(test_sen_1)

preds = doc2vecModel.docvecs.most_similar(positive=[test_sen_1_embedding])

for i, j in preds:
    print(f"Tag:{i}, Similarity: {j}, Sentence: {sentences[i]}")
```

```
Tag:1, Similarity: 0.6197482943534851, Sentence: All three of us like to play field sport
Tag:0, Similarity: 0.5684641003608704, Sentence: I played basketball
Tag:5, Similarity: 0.49407148361206055, Sentence: Soccer is a sport where each team has 11
players
Tag:3, Similarity: 0.48013174533843994, Sentence: I like to play soccer
Tag:2, Similarity: 0.4671724736690521, Sentence: All my friends like to play football
Tag:4, Similarity: 0.4393582344055176, Sentence: In India cricket is the most popular sport
```

**Sentences from the collection are displayed in decreasing order of similarity with the test sentence (query)*

Sentence Embeddings: SentenceBERT

SentenceBert(SBERT) [5] is the one of the most sophisticated techniques and currently a leader in the field of learning sentence embeddings. SBERT is a modification of the BERT[8] network using siamese and triplet networks that is able to derive semantically meaningful sentence embeddings[2] . This enables BERT to be used for certain new tasks, which were not applicable for BERT earlier. These tasks include large-scale semantic-similarity comparison, clustering and information retrieval via semantic search[5].

At the center of the SBERT is the BERT based model with following key concepts:

- Attention
- Transformers
- BERT
- Siamese Network

SBERT is a *twin* network which allows it to process two sentences simultaneously and in the same way. These two twin networks are identical down to every parameter (i.e. their weights are tied). Basically, SBERT uses a Siamese network like architecture (**Fig. 3 & Fig. 4**) to provide two sentences as the input. These two sentences are then passed to BERT models and a pooling layer to generate their embeddings. Then we can use the embeddings for the pair of sentences as inputs to calculate the cosine similarity or use the embeddings for classification tasks.

[2] Semantically meaningful means that semantically similar sentences are close in vector space.

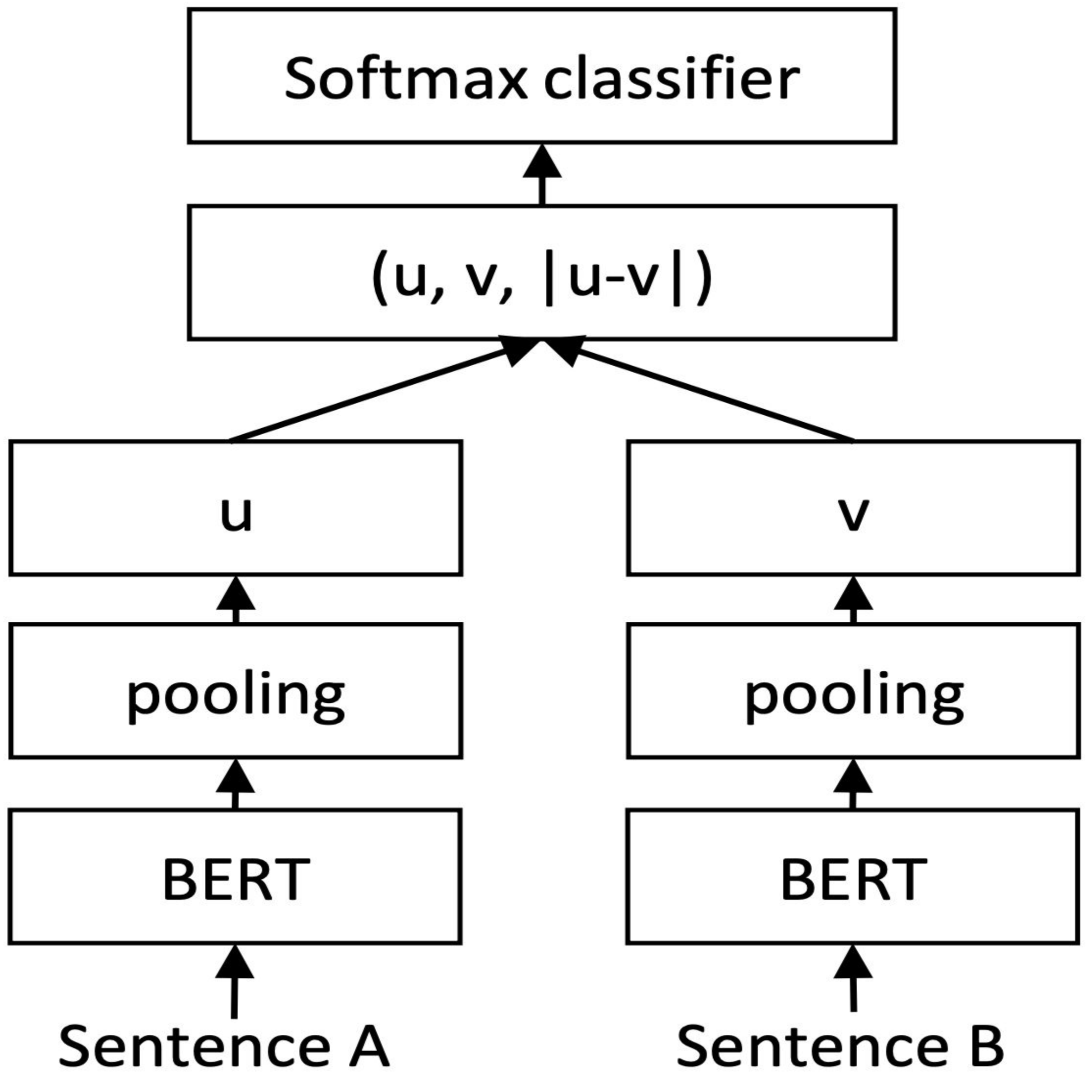


Fig.3: SBERT architecture with classification objective function. The two BERT networks have tied weights (siamese network structure)[5]

Classification Objective Function[5]:

$$o = \mathrm{softmax}(W_t(u, v, |u - v|))$$

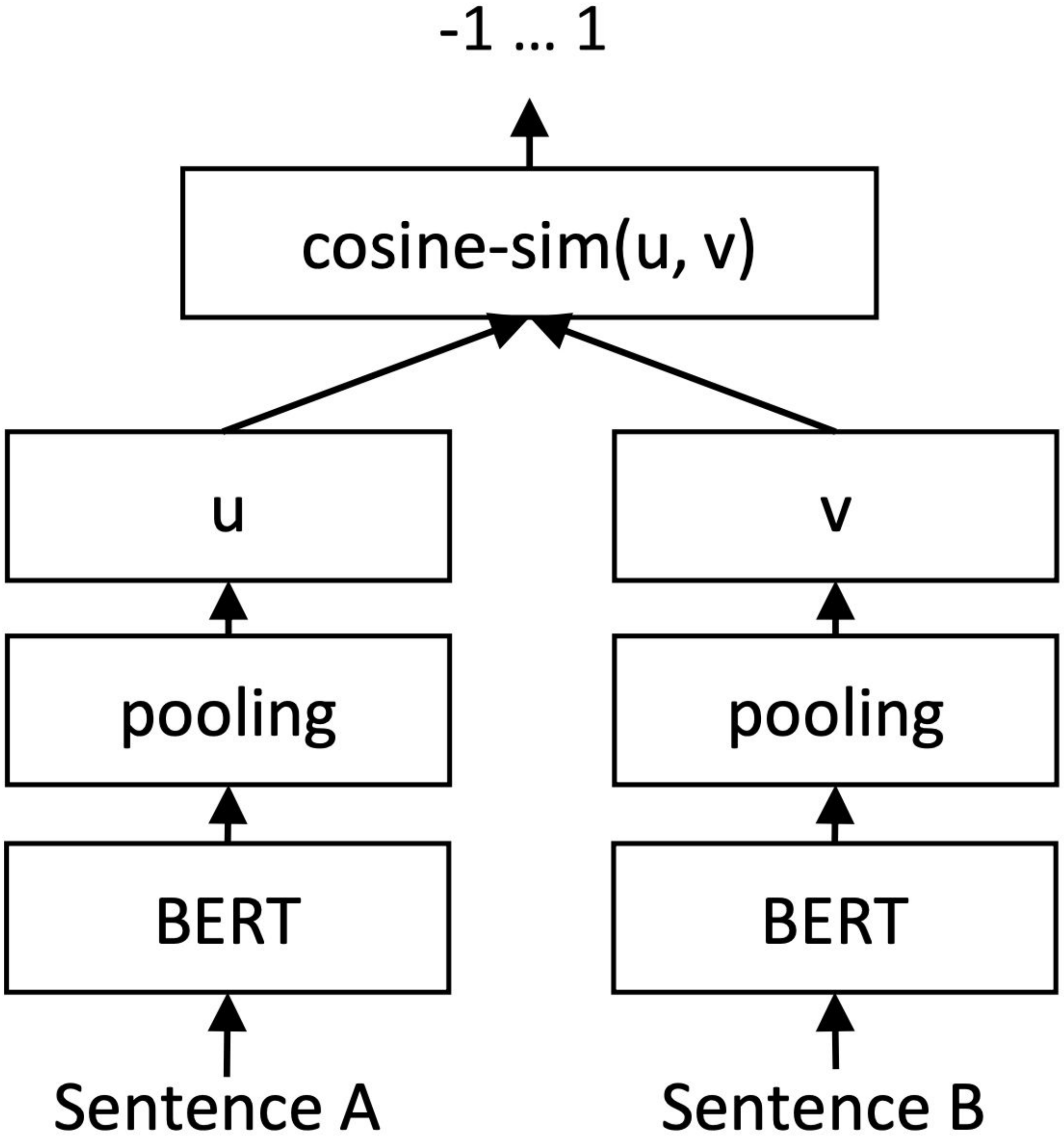


Fig.4: SBERT architecture at inference, for e.g. to compute similarity scores. This architecture is also used with the regression objective function.[5]

SBERT (simple hands-on exercise in python):

We can experiment & use SBERT using sentence-transformers[6, 9]:

Installing library

```
!pip install sentence-transformers
```

Load a pre-trained BERT model. There are many pre-trained models available, the full list can be seen here.

```
from sentence_transformers import SentenceTransformer
sbertModel = SentenceTransformer('distilbert-base-nli-stsb-mean-tokens')
```

Encode sentence collection to generate embeddings. *Sentence vectors can be seen by uncommenting the code below. Also we have used the same sentence collection that we have used in Doc2Vec.*

```
sentence_embeddings = sbertModel.encode(sentences)
#print(f'BERT embedding vector(length):{len(sentence_embeddings[0])}')
#print(f'BERT embedding vector: {sentence_embeddings[0]}')
```

Testing with a query (encode the query using the same model).

```
query = "I like swimming"
query_vec = sbertModel.encode([query])[0]
```

Computing the similarity of the test query with sentences in the collection.

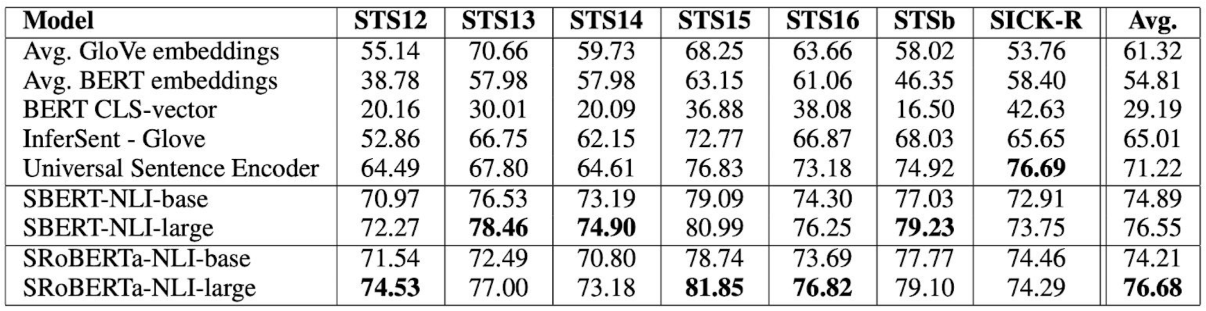
```
from scipy.spatial.distance import cosine
sim = [(sentences[i], 1 - cosine(query_vec, embeddings)) for i, embeddings in
enumerate(sentence_embeddings)]
for sen, sim in sorted(sim, key=lambda x: -x[1]):
        print(f"Sentence: {sen}, Similarity: {sim}")
```

```
Sentence: I like to play soccer, Similarity: 0.5140874981880188
Sentence: In India cricket is the most popular sport, Similarity: 0.32369759678840637
Sentence: All my friends like to play football, Similarity: 0.20060335099697113
Sentence: All three of us like to play field sport, Similarity: 0.1608773171901703
Sentence: I played basketball, Similarity: 0.06079213693737984
Sentence: Soccer is a sport where each team has 11 players, Similarity: 0.012044277042150497
```

Above, we have obtained similarity between sentences in the collection and the test query.

SBERT[5] is currently the leader of the pack when it comes to sentence embedding which can be seen from the evaluation results[5] below:

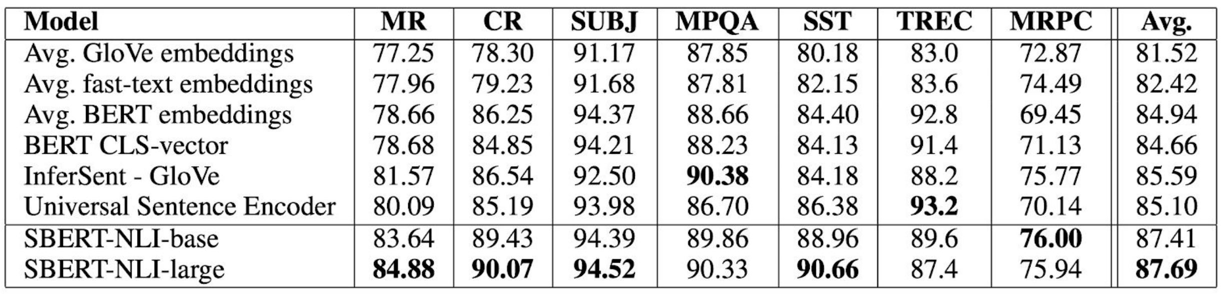
Evaluation - STS (Semantic Textual Similarity)



Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as ρ × 100. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.[5]

Evaluation - SentEval



Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	Avg.
Avg. GloVe embeddings	77.25	78.30	91.17	87.85	80.18	83.0	72.87	81.52
Avg. fast-text embeddings	77.96	79.23	91.68	87.81	82.15	83.6	74.49	82.42
Avg. BERT embeddings	78.66	86.25	94.37	88.66	84.40	92.8	69.45	84.94
BERT CLS-vector	78.68	84.85	94.21	88.23	84.13	91.4	71.13	84.66
InferSent - GloVe	81.57	86.54	92.50	**90.38**	84.18	88.2	75.77	85.59
Universal Sentence Encoder	80.09	85.19	93.98	86.70	86.38	**93.2**	70.14	85.10
SBERT-NLI-base	83.64	89.43	94.39	89.86	88.96	89.6	**76.00**	87.41
SBERT-NLI-large	**84.88**	**90.07**	**94.52**	90.33	**90.66**	87.4	75.94	**87.69**

Table 2: Evaluation of SBERT sentence embeddings using the SentEval toolkit. SentEval evaluates sentence embeddings on different sentence classification tasks by training a logistic regression classifier using the sentence embeddings as features. Scores are based on a 10-fold cross-validation.

Conclusion

We can conclude that learning sentence embeddings (Doc2Vec[1], SBERT[5]), provides substantial advantages over the traditional bag-of-words approaches. The good performance of models (Doc2Vec[1], SBERT[5]) that we have discussed also demonstrates their merits in capturing the semantics of the sentences.

As part of this review, we also experimented with some basic code for two of the models. Please note that the code above is just for the demonstration purpose, in practice we need to pre-process (stemming/stop word removal etc.) the sentences first and then transform them into embeddings.

We have explored some other models also, and wish to include them in any future tech-review we will do for the field of sentence embeddings. Some of the interesting ones ares:

- InferSent [10], [11]
- Universal Sentence Encoder[12]
- Discourse Informed Sen2Vec [2]

To summarize, we have given an overview of the architecture of Doc2Vec[1] and SBERT[5] and we are very excited to explore more on how these sentence embedding techniques will help us in enhancing the models to understand natural language better.

References:

[1] Distributed Representations of Sentences and Documents. Quoc Le, Tomas Mikolov (https://cs.stanford.edu/~quocle/paragraph_vector.pdf)

[2] Dis-S2V: Discourse Informed Sen2Vec. Tanay Kumar Saha, Shafiq Joty, Naeemul Hassan and Mohammad Al Hasan (https://arxiv.org/pdf/1610.08078.pdf)

[3] GloVe: Global Vectors for Word Representation. Jeffrey Pennington, Richard Socher, Christopher D. Manning (https://nlp.stanford.edu/pubs/glove.pdf)

[4] Distributed Representations of Words and Phrases and their Compositionality, (https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf)

[5] Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks Nils Reimers and Iryna Gurevych (https://arxiv.org/pdf/1908.10084.pdf)

[6] Sentence-transformers (Python toolkit for SBERT, https://pypi.org/project/sentence-transformers/)

[7] Gensim (https://pypi.org/project/gensim/)

[8] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova.
(https://arxiv.org/abs/1810.04805)

[9] SBERT (sentence-transformers) https://github.com/UKPLab/sentence-transformers

[10] Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. (https://arxiv.org/pdf/1705.02364.pdf)

[11] InferSent (https://github.com/facebookresearch/InferSent)

[12] Universal Sentence Encoder. Daniel Cera , Yinfei Yanga , Sheng-yi Konga , Nan Huaa , Nicole Limtiacob , Rhomni St. Johna , Noah Constanta , Mario Guajardo-Cespedes ´ a , Steve Yuanc , Chris Tara , Yun-Hsuan Sunga , Brian Stropea , Ray Kurzweila
(https://arxiv.org/pdf/1803.11175.pdf)