Universal Sentence Encoder Visually Explained (https://amitness.com/2020/06/universal-sentence-encoder/)

7 minute read

With transformer models such as BERT and friends taking the NLP research community by storm, it might be tempting to just throw the latest and greatest model at a problem and declare it done. However, in industry, we have compute and memory limitations to consider and might not even have a dedicated GPU for inference.

Thus, it's useful to keep simple and efficient models in your NLP problem-solving toolbox. Cer et. al (https://arxiv.org/abs/1803.11175) proposed one such model called "Universal Sentence Encoder".

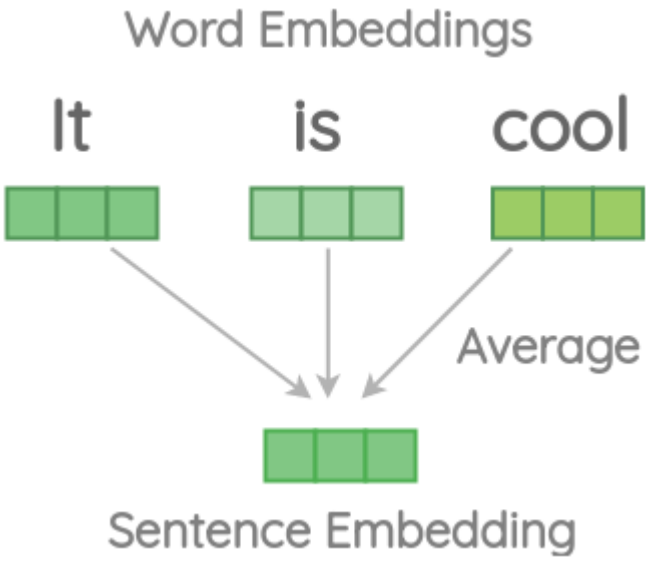
In this post, I will explain the core idea behind "Universal Sentence Encoder" and how it learns fixed-length sentence embeddings from a mixed corpus of supervised and unsupervised data.

Goal

We want to learn a model that can map a sentence to a fixed-length vector representation. This vector encodes the meaning of the sentence and thus can be used for downstream tasks such as searching for similar documents.

Why Learned Sentence Embeddings?

A naive technique to get sentence embedding is to average the embeddings of words in a sentence and use the average as the representation of the whole sentence. This approach has some challenges.



Let's understand these challenges with some code examples using the spacy library. We first install spacy and create an `nlp` object to load the medium version of their model.

```
# pip install spacy
# python -m spacy download en_core_web_md

import en_core_web_md
nlp = en_core_web_md.load()
```

- **Challenge 1: Loss of information**
  If we calculate the cosine similarity of documents given below using averaged word vectors, the similarity is pretty high even if the second sentence has a single word `It` and doesn't have the same meaning as the first sentence.

```
>>> nlp('It is cool').similarity(nlp('It'))
0.8963861908844291
```

- **Challenge 2: No Respect for Order**
  In this example, we swap the order of words in a sentence resulting in a sentence with a different meaning. Yet, the similarity obtained from averaged word vectors is 100%.

```
>>> nlp('this is cool').similarity(nlp('is this cool'))
1.0
```

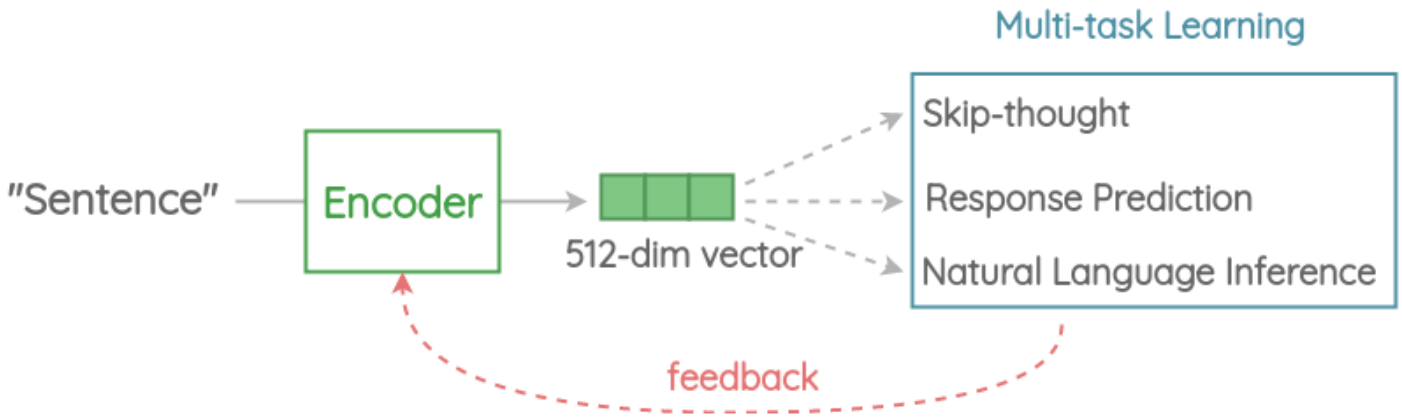
We could fix some of these challenges with hacky manual feature engineering like skipping st words, weighting the words by their TF-IDF scores, adding n-grams to respect order when averaging, concatenating embeddings, stacking max pooling and averaged embeddings and

A different line of thought is training an end-to-end model to get us sentence embeddings:

> *What if we could train a neural network to figure out how to best combine the word embeddings?*

Universal Sentence Encoder(USE)

On a high level, the idea is to design an encoder that summarizes any given sentence to a 512-dimensional sentence embedding. We use this same embedding to solve multiple tasks and based on the mistakes it makes on those, we update the sentence embedding. Since the same embedding has to work on multiple generic tasks, it will capture only the most informative features and discard noise. The intuition is that this will result in an generic embedding that transfers universally to wide variety of NLP tasks such as relatedness, clustering, paraphrase detection and text classification.



Let's now dig deeper into each component of Universal Sentence Encoder.

1. Tokenization

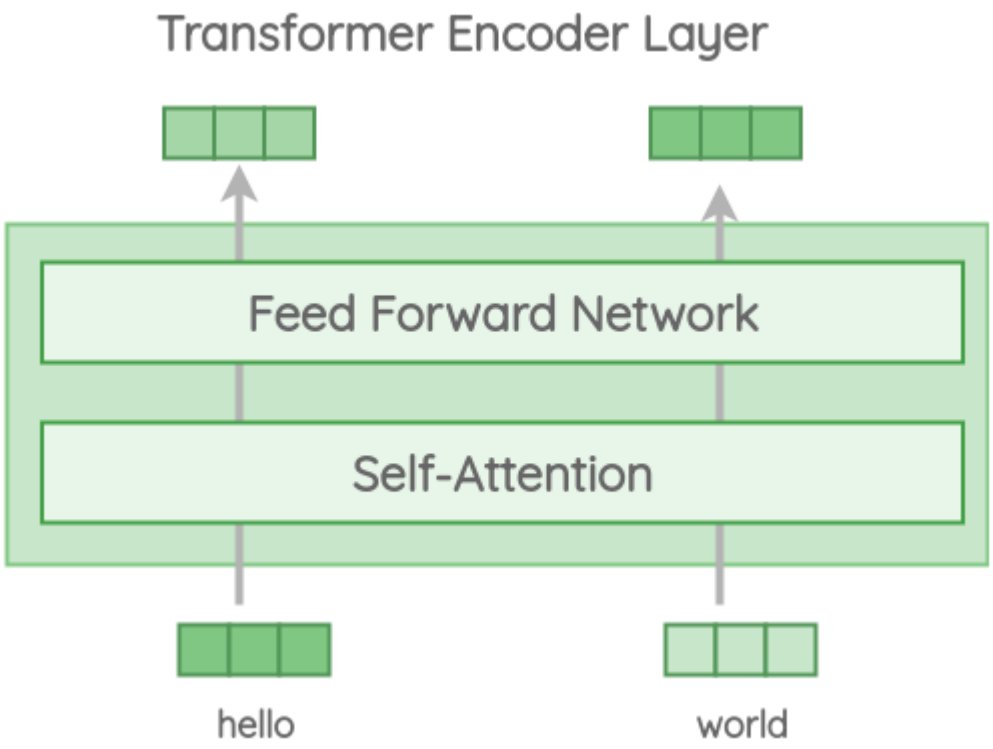
First, the sentences are converted to lowercase and tokenized into tokens using the Penn Treebank(PTB) tokenizer.

2. Encoder

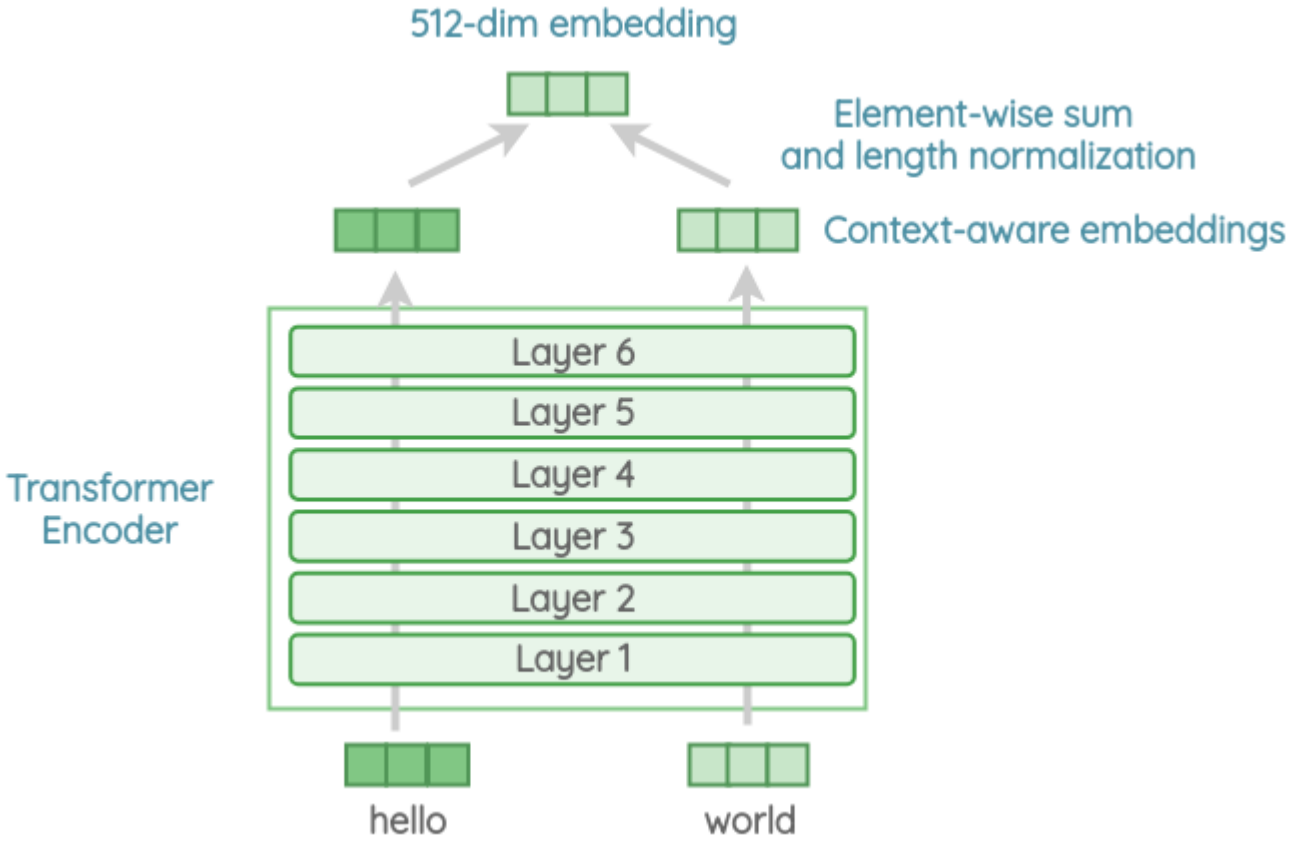
This is the component that encodes a sentence into fixed-length 512-dimension embedding. In the paper, there are two architectures proposed based on trade-offs in accuracy vs inference speed.

Variant 1: Transformer Encoder

In this variant, we use the encoder part of the original transformer architecture. The architecture consists of 6 stacked transformer layers. Each layer has a self-attention module followed by a feed-forward network.



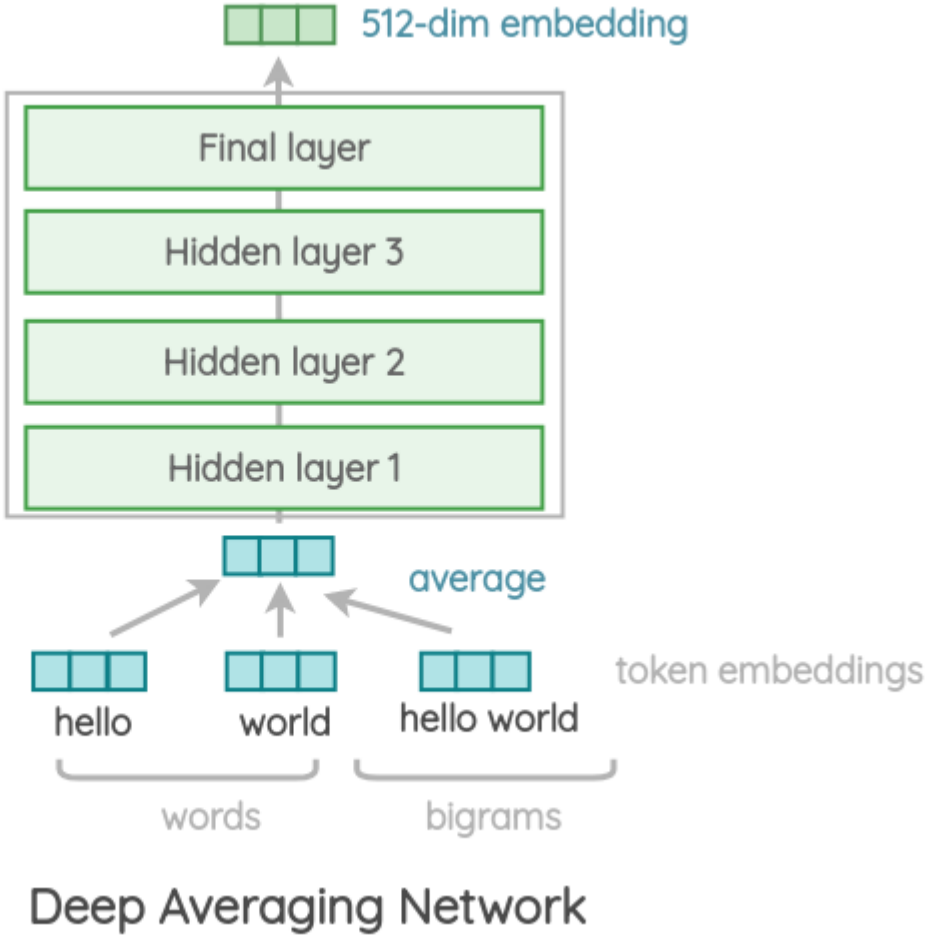
The self-attention process takes word order and surrounding context into account when generating each word representation. The output context-aware word embeddings are added element-wise and divided by the square root of the length of the sentence to account for the sentence-length difference. We get a 512-dimensional vector as output sentence embedding.



This encoder has better accuracy on downstream tasks but higher memory and compute resource usage due to complex architecture. Also, the compute time scales dramatically with the length of sentence as self-attention has $O(n^2)$ time complexity with the length of the sentence. But for short sentences, it is only moderately slower.

Variant 2: Deep Averaging Network(DAN)

In this simpler variant, the encoder is based on the architecture proposed by Iyyer et al. (https://people.cs.umass.edu/~miyyer/pubs/2015_acl_dan.pdf). First, the embeddings for word and bi-grams present in a sentence are averaged together. Then, they are passed through 4-layer feed-forward deep DNN to get 512-dimensional sentence embedding as output. The embeddings for word and bi-grams are learned during training.



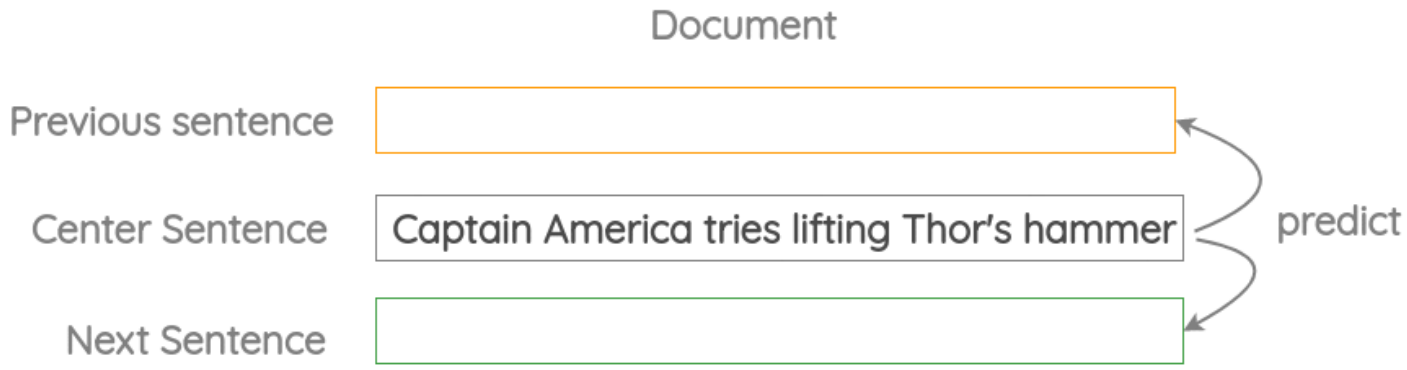
It has slightly reduced accuracy compared to the transformer variant, but the inference time is very efficient. Since we are only doing feedforward operations, the compute time is of linear complexity in terms of length of the input sequence.

3. Multi-task Learning

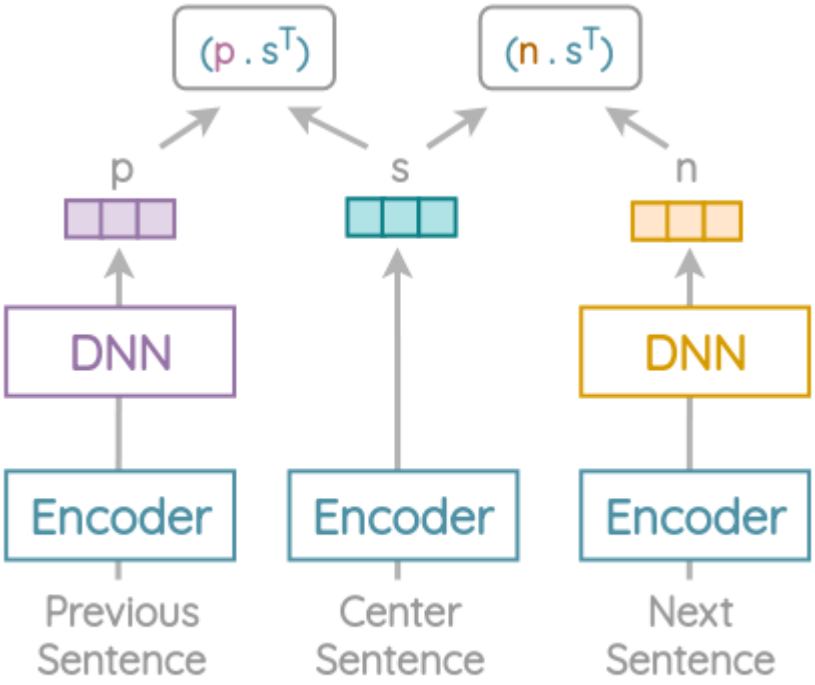
To learn the sentence embeddings, the encoder is shared and trained across a range of unsupervised tasks along with supervised training on the SNLI corpus. The tasks are as follows:

a. Modified Skip-thought

The idea with original skip-thought paper from Kiros et al. (https://arxiv.org/abs/1506.06726) was to use the current sentence to predict the previous and next sentence.



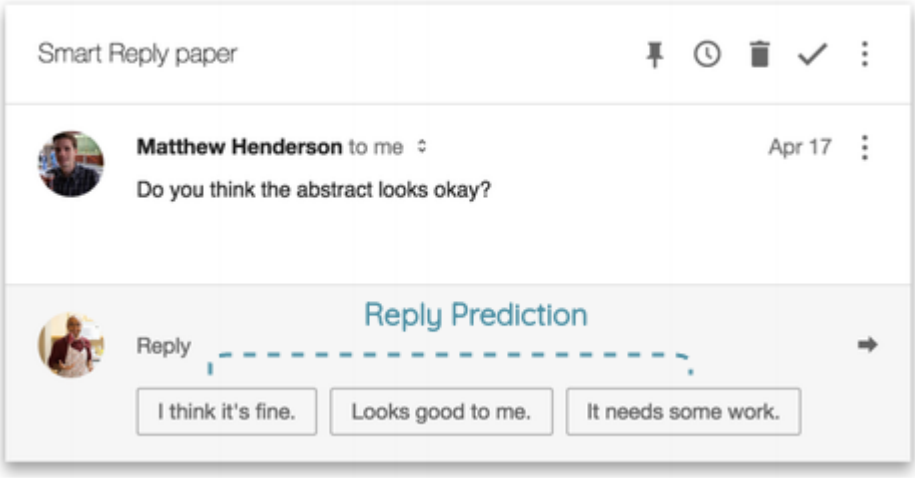
In USE, the same core idea is used but instead of LSTM encoder-decoder architecture, only an encoder based on transformer or DAN is used. USE was trained on this task using the Wikipedia and News corpus.



Skip-thought Task Structure

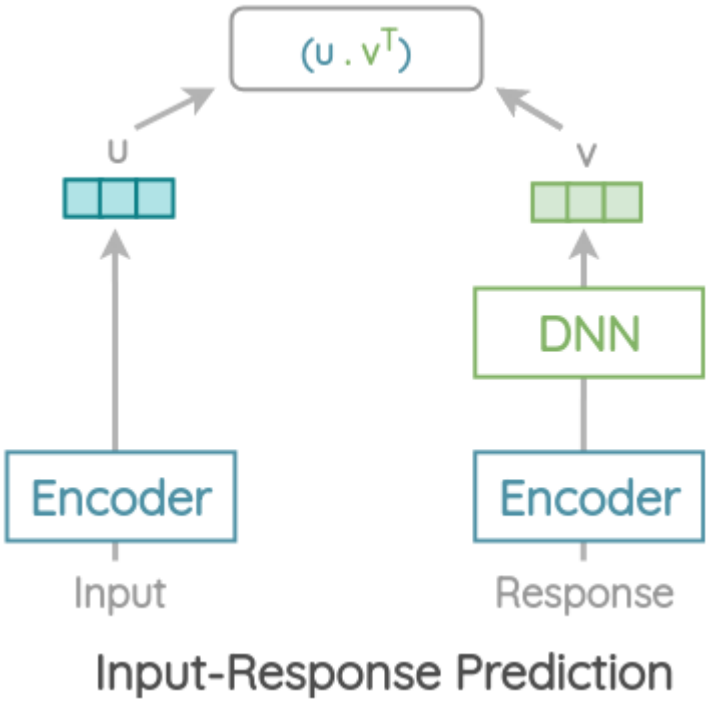
b. Conversational Input-Response Prediction

In this task, we need to predict the correct response for a given input among a list of correct responses and other randomly sampled responses. This task is inspired by Henderson et al. (https://arxiv.org/abs/1705.00652) who proposed a scalable email reply prediction architecture. This also powered the "Smart Reply" feature in "Inbox by Gmail".



Source: Henderson et al. (https://arxiv.org/abs/1705.00652)

The USE authors use a corpus scraped from web question-answering pages and discussion forums and formulate this task using a sentence encoder. The input sentence is encoded into a vector u. The response is also encoded by the same encoder and response embeddings are passed through a DNN to get vector v. This is done to model the difference in meaning of input and response. The dot product of this two vectors gives the relevance of an input to response.



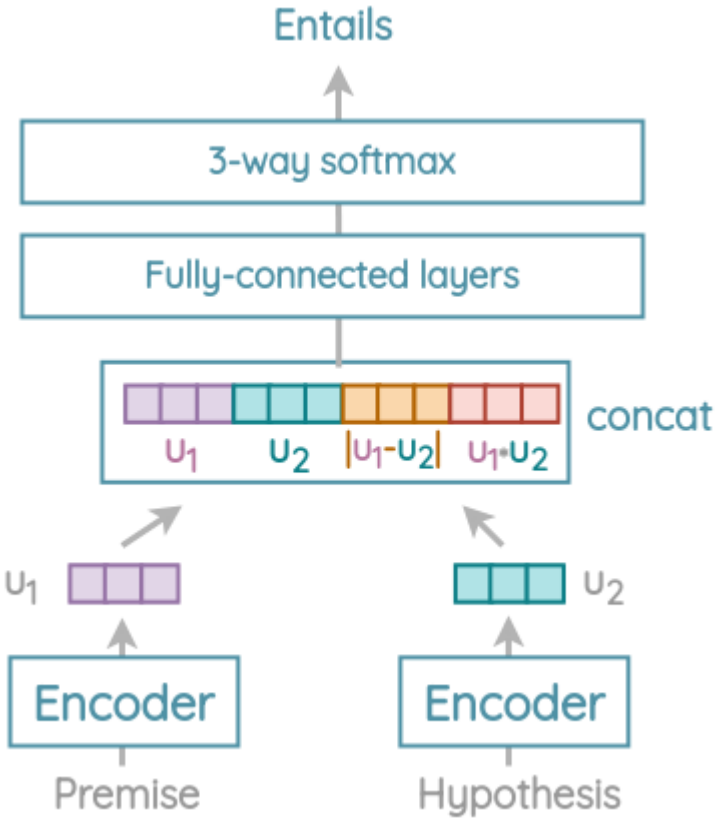
Training is done by taking a batch of K randomly shuffled input-response pairs. In each batch, for a input, its response pair is taken as the correct response and the remaining responses are treated as incorrect. Then, the dot product scores are calculated and converted to probabilities using a softmax function. Model is trained to maximize the log likelihood of the correct response for each inr

c. Natural Language Inference

In this task, we need to predict if a hypothesis entails, contradicts, or is neutral to a premise. The authors used the 570K sentence pairs from SNLI (https://nlp.stanford.edu/projects/snli/) corpus to train USE on this task.

Premise	Hypothesis	Judgement
A soccer game with multiple males playing	Some men are playing a sport	entailment
I love Marvel movies	I hate Marvel movies	contradiction
I love Marvel movies	A ship arrived	neutral

The sentence pairs are encoded using shared Transformer/DAN encoders and the output 512-dim embeddings u1 and u2 are obtained. Then, they are concatenated along with their L1 distance and their dot product(angle). This concatenated vector is passed through fully-connected layers and softmax is applied to get probability for entailment/contradiction/neutral classes.



The idea to learn sentence embedding based on SNLI seems to be inspired by the InferSent (https://arxiv.org/abs/1705.02364) paper though the authors don't cite it.

4. Inference

Once the model is trained using the above tasks, we can use it to map any sentence into fixed-length 512 dimension sentence embedding. This can be used for semantic search, paraphrase detection, clustering, smart-reply, text classification, and many other NLP tasks.

Results

One caveat with the USE paper was that it doesn't have a section on comparison with other competing sentence embedding methods over standard benchmarks. The paper seems to be written from an engineering perspective based on learnings from products such as Inbox by Gmail and Google Books.

Implementation

The pre-trained models for "Universal Sentence Encoder" are available via Tensorflow Hub. You can use it to get embeddings as well as use it as a pre-trained model in Keras. You can refer to my article on tutorial on Tensorflow Hub (https://amitness.com/2020/02/tensorflow-hub-for-transfer-learning/) to learn how to use it.

Conclusion

Thus, Universal Sentence Encoder is a strong baseline to try when comparing the accuracy gains of newer methods against the compute overhead. I have personally used it for semantic search, retrieval, and text clustering and it provides a decent balance of accuracy and inference speed.

References

- Daniel Cer et al., "Universal Sentence Encoder" (https://arxiv.org/abs/1803.11175)
- Iyyer et al., "Deep Unordered Composition Rivals Syntactic Methods for Text Classification" (https://people.cs.umass.edu/~miyyer/pubs/2015_acl_dan.pdf)
- Ryan Kiros et al., "Skip-Thought Vectors" (https://arxiv.org/abs/1506.06726)
- Matthew Henderson et al., "Efficient Natural Language Response Suggestion for Smart Reply" (https://arxiv.org/abs/1705.00652)
- Yinfei Yang et al., "Learning Semantic Textual Similarity from Conversations" (https://arxiv.org/abs/1804.07754)
- Alexis Conneau et al., "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data" (https://arxiv.org/abs/1705.02364)
- Google AI Blog, "Advances in Semantic Textual Similarity" (https://ai.googleblog.com/2018/05/advances-in-semantic-textual-similarity.html)

Categories: nlp

Updated: June 15, 2020

COMMENTS

Appreciate the post.
But I wondering it is possible to add my own text set and re-train with to universal sentence encoder to receive vectors with account the data corpus features.

amitness commented on Sep 2, 2020

@Petrolvaniuk Sadly it's not clear on their end. As far as I understand, you can't retrain it with your own vocab.

You can read more in this thread

👍 1

DarianHarrison commented on Nov 13, 2020

thank you Amit, really appreciate you writing this. It's very informative

bsathish11 commented on Apr 16, 2021

Hi Amit, for one of my project 'Document Search', i have used this multilingual USE. but model is over trained fo the most occurring word in the document. Suppose, if i ask question like "Why do we need ABC" ,it is throwing t response. but similarly, when i ask, "Need of ABC", I am expecting the same response. but it is not throwing the same response. i.e, it is throwing other responses of ABC with the highest cosine similarity score. Based on the semantic and context, i want it to be trained. Any help would be better..

PratyushGoel commented on Jul 13, 2022

Hi Amit, Thanks for such an informative article. In the DAN variant of universal sentence encoder you mentioned feeds the average of words and bigrams token embeddings. Can you shed some light on how these token embeddings are generated for each word or unigram? It would be helpful if you could point to that respective paper.

Write Preview

Sign in to comment

Styling with Markdown is supported

Sign in