Beyond Word Embedding: Key Ideas in Document Embedding

<= Previous post

Next post =>

Tags: LDA, NLP, Topic Modeling, Trends, Word Embeddings

This literature review on document embedding techniques thoroughly covers the many ways practitioners develop rich vector representations of text -- from single sentences to entire books.

Latest News

- Data Extraction Software Octoparse 8 vs Octoparse 7: Wh...
- Model Evaluation Metrics in Machine Learning
- Taming Complexity in MLOps
- 5 Machine Learning Papers on Face Recognition
- Top tweets, May 20-26: The Best NLP with Deep Learnin...
- Are Tera Operations Per Second (TOPS) Just hype? Or Dar...

Top Stories Last Week

Most Popular

1. The Best NLP with Deep Learning Course is Free

comments

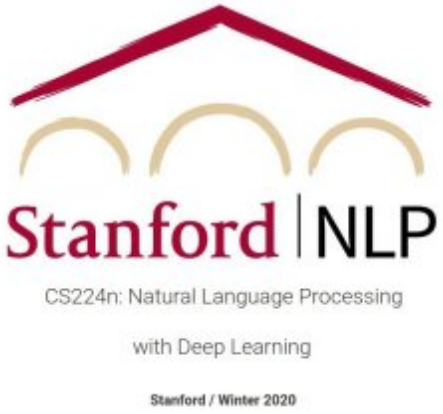
By Shay Palachy, Data Science Consultant.

Word embedding — the mapping of words into numerical vector spaces — has proved to be an incredibly important method for natural language processing (NLP) tasks in recent years, enabling various machine learning models that rely on vector representation as input to enjoy richer representations of text input. These representations preserve more semantic and syntactic information on words, leading to improved performance in almost every imaginable NLP task.

Both the novel idea itself and its tremendous impact have led researchers to consider the problem of how to provide this boon of richer vector representations to larger units of texts — from sentences to books. This effort has resulted in a slew of new methods to produce these mappings, with various innovative solutions to the problem and some notable breakthroughs.

This post is meant to present the different ways practitioners have come up with to produce document embeddings.

> ***Note:*** *I use the word* **document** *here to refer to* **any sequence of words**, *ranging from sentences and paragraphs through social media posts all way up to articles, books and more complexly structured text documents (e.g. forms).*



2. Build and deploy your first machine learning web app
3. Automated Machine Learning: The Free eBook
4. Complex logic at breakneck speed: Try Julia for data science
5. Easy Text-to-Speech with Python
6. An easy guide to choose the right Machine Learning algorithm
7. What they do not tell you about machine learning

Most Shared

1. The Best NLP with Deep Learning Course is

Figure 1: A common example of embedding documents into a wall.

In this post, I will touch upon not only approaches that are direct extensions of word embedding techniques (e.g., in the way *doc2vec* extends *word2vec*), but also other notable techniques that produce — sometimes among other

2. Automated Machine Learning: The Free eBook
3. Sparse Matrix Representation in Python
4. Build and deploy your first machine learning web app
5. 13 must-read papers from AI experts
6. An easy guide to choose the right Machine Learning algorithm
7. Appropriately Handling Missing Values for Statistical Modelling and Prediction

More Recent Stories

Are Tera Operations Per Second (TOPS) Just hype? Or Dark AI Si...

Deepmind's Gaming Streak: The Rise of AI Dominance

Best GIS Courses in 2020

Faster machine learning on larger graphs with NumPy and Pandas

KDnuggets 20:n21, May 27: The Best NLP with Deep Learning C...

How to Rock a Virtual Data Interview

Dataset Splitting Best Practices in Python

I will also try to provide links and references to both the original papers and code implementations of the reviewed methods whenever possible.

> ***Note:*** *This topic is somewhat related, but not equivalent, to the problem of learning structured text representations (e.g., Liu & Lapata, 2018).*

Applications of document embeddings

The ability to map documents to informative vector representations has a wide range of applications. What follows is only a partial list.

[Le & Mikolov, 2014] demonstrated the capabilities of their *paragraph vectors* method on several text classification and sentiment analysis tasks, while [Dai et al, 2015] examined it in the context of document similarity tasks and [Lau & Baldwin, 2016] benchmarked it against a forum question duplication task and the Semantic Textual Similarity (STS) SemEval shared task.

[Kiros et al., 2015] has demonstrated the use of their *skip-thought* vectors for semantic relatedness, paraphrase detection, image-sentence ranking, question-type classification, and four sentiment and subjectivity datasets. [Broere, 2017] used them to predict POS tags and dependency relations.

[Chen et al., 2018] showed *BioSentVec*, their set of sentence embeddings trained on biomedical texts, to perform well on sentence pair similarity tasks (official Python implementation).

Finally, the Deep Semantic Similarity Model was used by various authors for information retrieval and web search ranking, ad selection/relevance, contextual entity search and interestingness tasks, question answering, knowledge inference, image captioning, and machine translation tasks.

Prominent approaches and trends

Machine Fairness: How to assess AI system's fairness and mit...

LinkedIn Open Sources a Small Component to Simplify the Tensor...

10 Useful Machine Learning Practices For Python Developers

Python For Everybody: The Free eBook

Top Stories, May 18-24: The Best NLP with Deep Learning Course...

The Best NLP with Deep Learning Course is Free

Appropriately Handling Missing Values for Statistical Modellin...

A Holistic Framework for Managing Data Analytics Projects

Build and deploy your first machine learning web app

Top tweets, May 13-19: Linear algebra and optimization and ...

Caserta Announces Pro-Bono Data and

I'm writing this part last, having dedicated a lot of time to think on how to structure this article, how the various techniques covered in the following sections can be grouped into prominent approaches and what trends emerge when examining how the different works in the field relate to each other and the way the followed one another.

Do note, however, that while the problem of document embedding is old, many of the currently influential solutions are young, and this field has seen a resurgence very recently (around 2014), directly following the success of contemporary encoder-decoder-based word embedding techniques, so this is very much still early days. Having said that, I hope this part can put the following sections into a wider context and frame them in a meaningful manner.

Document embedding approaches

A possible way to map the field is into the following four prominent approaches:

1. **Summarizing word vectors**
   This is *the*classic approach. *Bag-of-words* does exactly this for one-hot word vectors, and the various weighing schemes you can apply to it are variations on this way to summarizing word vectors. However, this approach is also valid when used with the most state-of-the-art word representations (usually by averaging instead of summing), especially when word embeddings are optimized with this use in mind, and can stand its ground against any of the sexier methods covered here.
2. **Topic modelling**
   While this is not usually the main application for topic modeling techniques like LDA and PLSI, they i*nherently generate a document*

Component Analysis (PCA)

hidden in the data, and are thus useful in our context. I don't really cover this approach in this post (except a brief intro to LDA) because I think that it is both well represented by LDA and well known generally.

3. **Encoder-decoder models**
   This is the newest unsupervised addition to the scene, featuring the likes of *doc2vec*and *skip-thought.* While this approach has been around since the early 2000's — under the name of *neural probabilistic language models* — it has gained new life recently with its successful application to word embedding generation, with current research focusing on how to extend its use to document embedding. This approach gains more than others from the increasing availability of large unlabeled corpora.

4. **Supervised representation learning**
   This approach owes its life to the great rise (or resurgence) of neural network models, and their ability to learn rich representations of input data using various non-linear multi-layer operators, which can approximate a wide range of mappings. By simply inputting ye olde bag-of-words into a neural network learning to solve some supervised text-related problem, you get a model where hidden layers hold rich representations of the input text, which is exactly what we're after.

There are a couple of unsupervised approaches that don't fit any of the above groups (specifically, *quick-thought* and *Word Mover's Distance* come to mind), but I think most techniques do fall into one of these four broad categories.

> ***Note:*** *While it is tempting to point out the classic bag-of-words technique as suffering from a unique absence of order information, this is actually the rule rather than the exception. The main information gained by most of the newer methods reviewed here is extending the distributional*

Trends and challenges

There are several broad trends arising when examining both research and application of document embedding techniques as a whole, as well as several challenges one might identify.

1. **Encoder-Decoder Optimization:** A notable part of the research focuses on optimizing both the exact architecture (e.g., NN/CNN/RNN) and some components/hyper-parameters (e.g., n-grams, projection functions, weighing, etc.) of the unsupervised encoder-decoder approach to learning document embeddings. Although part of the goal of this fine-tuning is to improve success metrics on various tasks, the ability to train models over larger corpora or in shorter time is also a target.
2. **Learning objective design:** The crux of unsupervised (or self-supervised) representation learning is in designing a learning objective that exploits labels that are freely available within the data in a way that generates representations that prove to be useful to downstream tasks. This is, to me, the most exciting trend, and I think the one with the most potential for impact on NLP tasks that might equate the one word embedding techniques had. At the moment, I count only *quick-thought* and *Word Mover's Distance* as offering an alternative to the encoder-decoder approach. Another appealing aspect of this trend is that innovations here might be applicable to the problem of word embeddings as well.
3. **Benchmarking:** Part of a field-wide trend in machine learning research generally, document embedding, perhaps due to it being a young sub-field, demonstrates well the increased focus of research on benchmarking of techniques on a wide range and a large number of tasks (see the GLUE leaderboard). However, with almost every paper on the topic declaring comparable or superior results to current SOTA

techniques, this has yet to result in a clear leader emerging ahead of the pack.

4. **Open-sourcing:** Again, part of a wider trend, the fervent releasing of easily usable code implementing techniques — and often also experiments — enables reproducibility and drives both engagement with the wider data science community outside of academia and use on real-world problems.
5. **Cross-task applicability:** This is perhaps more the case with supervised embedding learning although not all unsupervised techniques were benchmarked with the same level of comprehensiveness. In any case, the wide range of extremely varied NLP tasks, relying on different types of information in text data, makes this issue a prominent one. Joint learning of embeddings from several tasks is one interesting way in which supervised approaches might tackle this challenge.
6. **Labeled corpora:** The limited availability of very large labeled corpora is also an issue for supervised approaches going forward. This might represent the true edge unsupervised approaches will have on supervised representation learning in coming years.

> ***Note:*** *If you found this part a bit out of context, I suggest you revisit it after going through a good portion of the techniques covered in this post.*

Classic techniques

This section briefly covers two established techniques for document embedding: *bag-of-words* and *latent Dirichlet allocation*. Feel free to skip it.

Bag-of-words

Presented in [Harris, 1954], this method represents text as the bag (multiset) of its words (losing grammar and ordering information). This is done by deciding on a set of *n* words that will form the vocabulary supported by the mapping, and assigning each word in the vocabulary a unique index. Then, each document is represented by a vector of length *n*, in which the *i*-th entry contains the number of occurrences of the word *i* in the document.

Figure 2: A bag-of-words representation of an example sentence.

For example, the sentence “dog eat dog world, baby!” (after cleaning punctuation) might be represented by a 550-length vector *v* (assuming a vocabulary of 550 words was chosen), which is zero everywhere except the following entries:

- V_{76}=1, as the 76th word of the vocabulary is *world*.
- V_{200}=2, as the 200th word of the vocabulary is *dog*.
- V_{322}=1, as the 332nd word of the vocabulary is eat.
- The word *baby* was not selected for inclusion in the vocabulary, so it induces a value of 1 on no entry of the vector.

grow quickly to support rich vocabularies, this technique was used almost exclusively and with great success on a huge range of NLP tasks for decades. Even with the significant progress in vector representation for text in recent years, common slight variations of this method — covered below — are still used today, and not always as only the first baseline to be quickly surpassed.

Bag-of-n-grams

To gain back some of the word order information lost by the bag-of-words approach, the frequency of short word sequences (of length two, three, etc.) can be used (additionally or instead) to construct word vectors. Naturally, bag-of-words is a private case of this method, for *n=1*.

For the sentence of “dog eat dog world, baby!”, the word pairs are “dog eat”, “eat dog”, “dog world” and “world baby” (and sometimes also “<START> dog” and “baby <END>”), and the vocabulary is made of (or enhanced with) all successive word pairs in the input corpus.

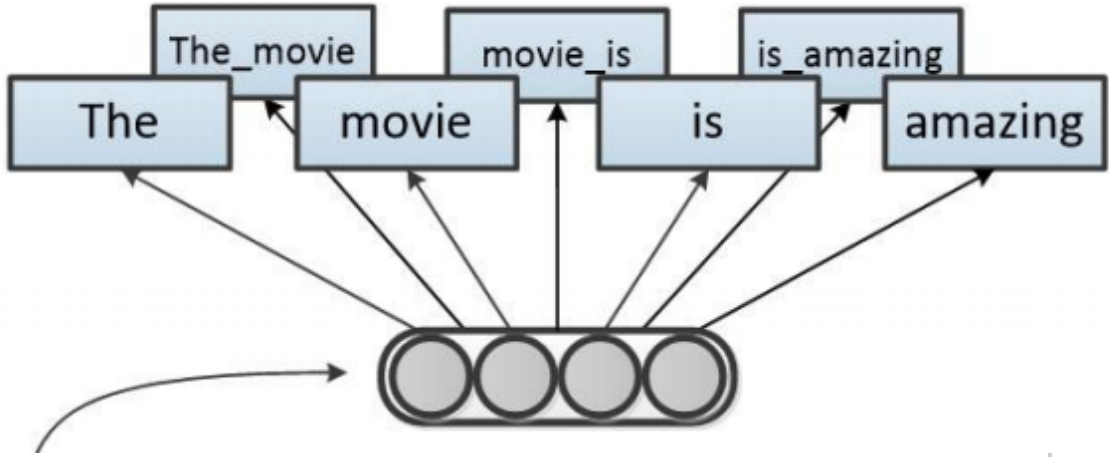


Figure 3: 2-grams representation of the sentence “The movie is amazing.”

One major downside of this approach is the non-linear dependence of the vocabulary size on the number of unique words, which can be very large for large corpora. Filtering techniques are commonly used to reduce the vocabulary size.

tf-idf weighting

A final related technique worth mentioning in the context of bag-of-words is *term frequency–inverse document frequency*, commonly denoted as *tf-idf*. This method re-weights the above word (or n-gram) frequency vectors with the *inverse document frequency* (IDF) of each word. The IDF of a word is simply the logarithm of the number of documents in the corpus divided by the number of documents in which that word appears in.

$$IDF_i = log\left(\frac{\#\text{ of documents in corpus}}{\#\text{ of documents in which word i appears in}}\right)$$

In short, the TF term grows as the word appears more often, while the IDF term increases with the word's rarity. This is meant to adjust the frequency scores for the fact that some words appear more (or less) frequently in general. See [Salton & Buckley, 1988] for a thorough overview of term-weighting approaches.

Latent Dirichlet allocation (LDA)

LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.

To connect this back to bag of words, the former approach can be thought of

The bag-of-words vector then represents the best approximation we have for the unnormalized distribution-of-words in each document; but document here is the basic probabilistic unit, each a single sample of its unique distribution.

The crux of the matter, then, is to move from this simple probabilistic model of documents as distributions over words to a more complex one by adding a latent (hidden) intermediate layer of *K* topics.

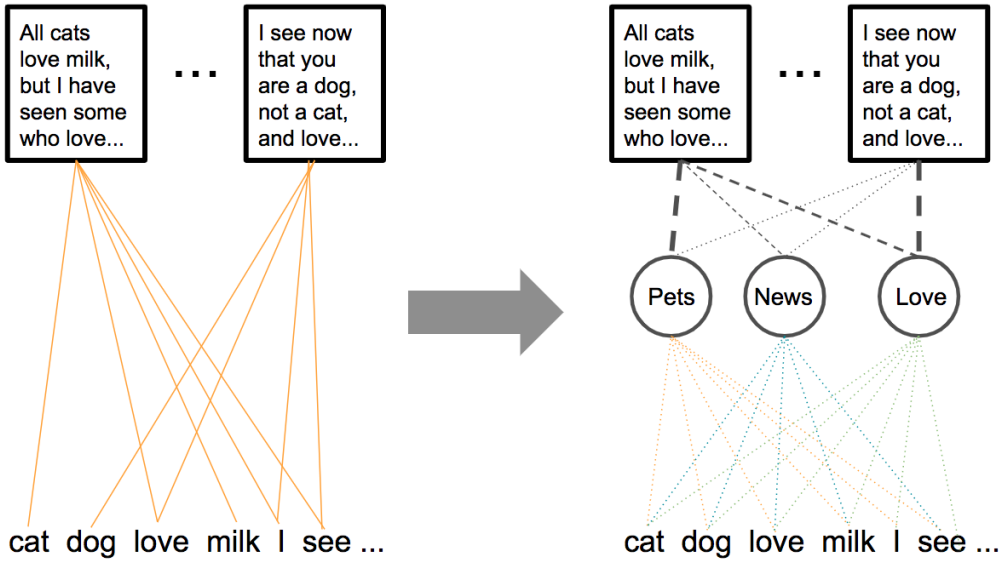


Figure 4: The probabilistic model shift from bag-of-words to LDA.

Topics are now characterized by distributions over words, while documents are distributions over topics. This probabilistic model of a document corresponds to a generative model of documents. To generate a set of *M* documents of lengths $\{N_i\}$, assuming a predetermined number of *K* topics, where *Dir()* denotes a Dirichlet distribution:

2. For each document *i*, sample a topic distribution (or mixture) $\theta_i \sim Dir(\alpha)$.

3. To generate document *i* of length N_i, for each word *j*:
   3.1. Sample a topic $z_{ij} \sim Multinomial(\theta_i)$ for word *j*.
   3.2. Sample word $j \sim Multinomial(z_{ij})$.

Given this model and a corpus of documents, the problem becomes one of inference, and approximations of the various distributions mentioned above are found in the inference process. Among these are θ_i, the topic distribution for each document *i*, vectors of dimension *K*.

So in the process of inferring the model, a vector space of dimension *K* was inferred, one which captures in some way the topics or themes in our corpus and the way they are shared between documents in it. This, of course, can be taken as an embedding space for these documents, and — depending on the choice of *K* — it can be of a significantly smaller dimension than vocabulary-based ones.

Indeed, while the main use case for LDA is unsupervised topic/community discovery, other cases include the use of the resulting latent topic space as an embedding space for the document corpus. Also, note that other topic modeling techniques — such as non-negative matrix factorization (NMF) and probabilistic latent semantic indexing (PLSI) — can be used in a similar manner to learn document embedding spaces.

> ***Note:*** *The main issue practitioners have with probabilistic topic models is their stability. Since training a topic model requires sampling of probability distributions, models of the same corpus can be expected to differ as seeds of the random number generator vary. This issue is compounded by the sensitivity of topic models to relatively small corpus changes.*

Many of the methods presented in this section are inspired by prominent word embedding techniques, chief among them *word2vec*, and they are sometimes even direct generalizations of these methods. These word embedding techniques are sometimes also called *Neural Probabilistic Language Models*; these are not identical terms, as a probabilistic language model is *a probability distribution over word sequences*, but as this approach was introduced as a way to learn language models in [Bengio, 2003], they are closely associated.

As such, a basic understanding of word embedding techniques is essential for understanding this section. If you are not familiar with the topic, the well-written two-part tutorial of *word2vec* by Chris McCormick is an excellent starting point (part 2), as is the Scholarpedia article on neural net language models by Prof. Joshua Bengio (also see Hunter Heidenreich's post for a more general and concise overview of word embeddings in general, and Alex Minnar's two-part post for a more in-depth mathematical deep dive). However, for a profound grasp of the details I urge you to read the seminal papers by [Bengio, 2003], [Mikolov et al., 2013a] and [Pennington et al, 2014] on the topic, which in many ways shaped this sub-field.

Even assuming your familiarity with *word2vec*, I still wish to note an important assumption this model makes, and which is carried forward by perhaps each and every one of the models reviewed here: *The Distributional Hypothesis*. Here is a brief description from Wikipedia:

> *The **distributional hypothesis** in linguistics is derived from the semantic theory of language usage, i.e., words that are used and occur in the same contexts tend to purport similar meanings. The underlying idea that "a word is characterized by the company it keeps" was popularized by Firth. The distributional hypothesis is the basis for statistical semantics.*

Indeed, it is easy to see that *word2vec*, and other self-supervised methods for learning word representations, rely heavily on this hypothesis; the crux of the model, after all, is that word representations learned while learning to predict the context of a word from the word itself (or vice versa) represent a vector space capturing deep semantic and syntactic concepts and phenomena. Meaning, learning from the context of a word can teach us about both its meaning and its syntactic role.

In this section, covering self-supervised document representation learning, you will see that all such methods both maintain this assumption for words and extend it in some way to larger units of texts.

n-gram embeddings

[Mikolov et al., 2013b] extended *word2vec*'s skip-gram model to handle short phrases by identifying a large number of short phrases — the authors focus on two- and three-word phrases — using a data-driven approach, and then treating the phrases as individual tokens during the training of the *word2vec* model. Naturally, this is less suitable for learning longer phrases — as the size of the vocabulary explodes when increasing phrase length — and *is bound to not generalize to unseen phrases* as well as the methods that follow it.

Moshe Hazoom wrote a wonderful practical review of this approach, used by his employer for a search engine focused on the finance domain.

Averaging word embeddings

There is a very intuitive way to construct document embeddings from meaningful word embeddings: Given a document, perform some vector arithmetics on all the vectors corresponding to the words of the document to summarize them into a single vector in the same embedding space; two such

Building upon this, you can perhaps already imagine that extending the encoder-decoder architecture of *word2vec* and its relatives to learn *how* to combine word vectors into document embeddings can be interesting; the methods that follow this one fall into this category.

A second possibility is to use a fixed (unlearnable) operator for vector summarization — e.g., averaging — and learn word embeddings in a preceding layer, using a learning target that is aimed at producing rich document embeddings; a common example is using a sentence to predict context sentences. Thus the main advantage here is that word embeddings are optimized for averaging into document representations.

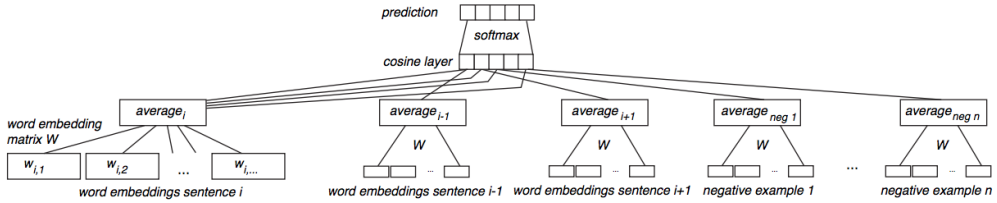


Figure 5: Siamese CBOW network architecture from [Kenter et al, 2016].

[Kenter et al., 2016] did exactly that, using a simple neural network over an averaging of word vectors, learning the word embeddings by predicting, given a sentence representation, its surrounding sentences. They compare the results to both averaged *word2vec* vectors and to *skip-thoughts* vectors (see the appropriate sub-section below). [Hill et al., 2016] compare a plethora of methods, including training CBOW and skip-gram word embeddings, while optimizing for sentence representation (here using element-wise addition of word vectors). [Sinoara et al., 2019] also propose a straightforward composition of word embedding vectors and other knowledge sources (like word-sense vectors) into their centroid to represent documents.

using a smooth inverse frequency weighting scheme, and (2) removing the common discourse component from word vectors; this component is found using PCA, and it is used as a correction term for the most frequent discourse, presumably related to syntax. The authors provide a Python implementation.

> ***Note:*** *Another demonstration of the power of correctly-averaged word "embeddings" can perhaps be found when looking at attention-based machine translation models. The one-directional decoder RNN gets the previous translated word as input, plus not just the "embedding" (i.e., the bi-directional activations from the encoder RNN) of the current word to translate, but also those of words around it; these are averaged in a weighted manner into a context vector. It is teaching that this weighted averaging is able to maintain the complex compositional and order-dependent information from the encoder network's activations (recall, these are not isolated embeddings like in our case; each is infused with the context of previous/following words).*

Sent2Vec

Presented in [Pagliardini et al., 2017] and [Gupta et al., 2019] (including an official C++-based Python implementation), this technique is very much a combination of the two above approaches: The classic CBOW model of *word2vec* is both extended to include word n-grams *and* adapted to optimize the word (and n-grams) embeddings for the purpose of averaging them to yield document vectors.

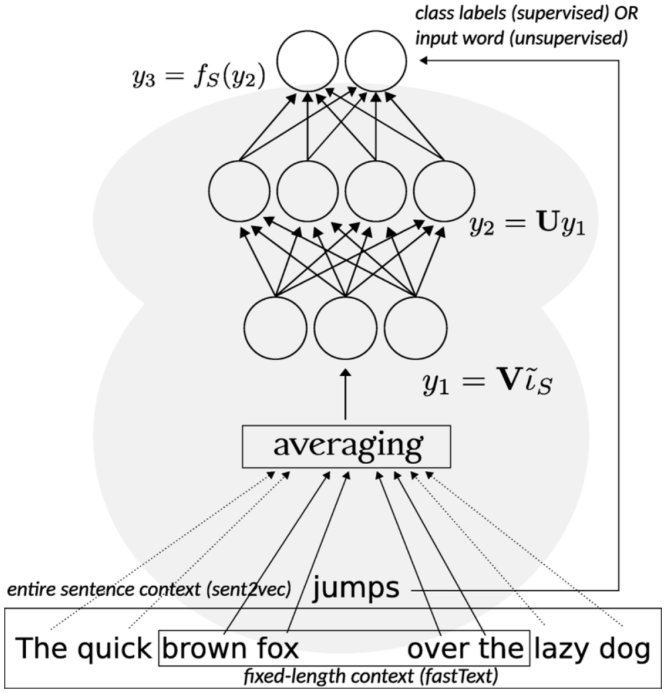


Figure 6: sent2vec can be thought of as an unsupervised version of fastText.

In addition, the process of input subsampling is removed, considering the entire sentence as context instead. This means both that **(a)** the use of frequent word subsampling is discarded — so as not to prevent the generation of n-grams features — and **(b)** the dynamic context windows used by *word2vec* are made away with: the entire sentence is considered as the context window, instead of sampling the context window size for each subsampled word uniformly between 1 and the length of the current sentence.

Another way to think of *sent2vec* is as an unsupervised version of *fastText* (see Figure 6), where the entire sentence is the context, and

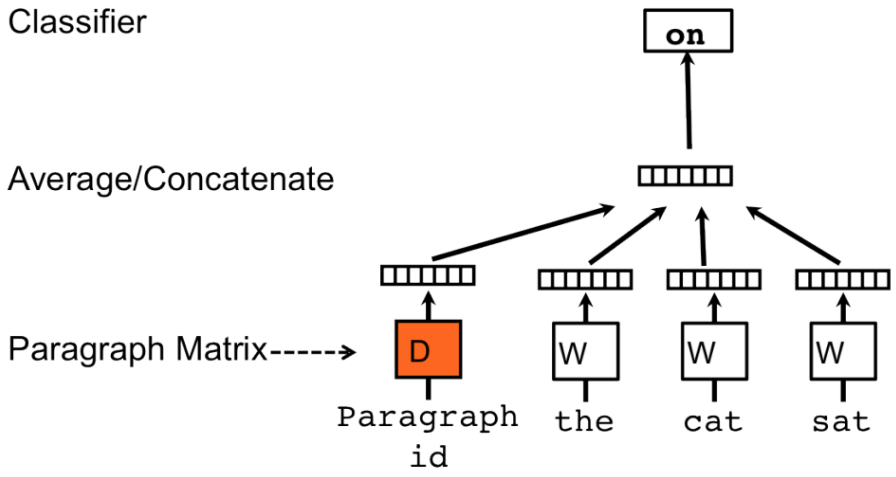
using *sent2vec* vectors as features to that of *fastText*, against the task of biomedical sentence classification.

Paragraph vectors (doc2vec)

Sometimes referred to as *doc2vec*, this method, presented in [Le & Mikolov, 2014] is perhaps the first attempt to generalize *word2vec* to work with word sequences. The authors introduce two variants of the p*aragraph vectors* model: *Distributed Memory* and *Distributed Bag-of-Words.*

Paragraph Vectors: Distributed Memory (PV-DM)

The PV-DM model augments the standard encoder-decoder model by adding a memory vector, aimed at capturing the topic of the paragraph, or context from the input. The training task here is quite similar to that of a *continuous bag of words*; a single word is to be predicted from its context. In this case, the context words are the preceding words, not the surrounding words, as is the paragraph.



To achieve this, every paragraph is mapped to a unique vector, represented by a column in a matrix (denoted by *D*), as is each word in the vocabulary. The contexts are fixed-length and sampled from a sliding window over the paragraph. The paragraph vector is shared across all contexts generated from the same paragraph but not across paragraphs. Naturally, word embeddings are global, and pre-trained word embeddings can be used (see *implementations and enhancements* below).

As in *word2vec*, vectors must be summarized in some way into a single vector; but unlike *word2vec*, the authors use concatenation in their experiments. Notice that this preserves order information. Similar to *word2vec*, a simple softmax classifier (in this case, actually hierarchical softmax) is used over this summarized vector representation to predict the task output. Training is done the standard way, using stochastic gradient descent and obtaining the gradient via backpropagation.

Notice that only the paragraphs in the training corpus have a column vector from *D* associated with them. At prediction time, one needs to perform an inference step to compute the paragraph vector for a new paragraph: The document vector is initialized randomly. Then, repeatedly, a random word is selected from the new document, and gradient descent is used to adjust input-to-hidden-layer weights such that softmax probability is maximized for the selected word, while hidden-to-softmax-output weights are fixed. This results in a representation of the new document as a mixture of training corpus document vectors (i.e., columns of *D*), naturally residing in the document embedding space.

Paragraph Vectors: Distributed Bag of Words (PV-DBOW)
The second variant of *paragraph vectors*, despite its name, is perhaps the parallel of *word2vec*'s *skip-gram* architecture; the classification task is to

single random word is sampled from that window, forming the below classification task.

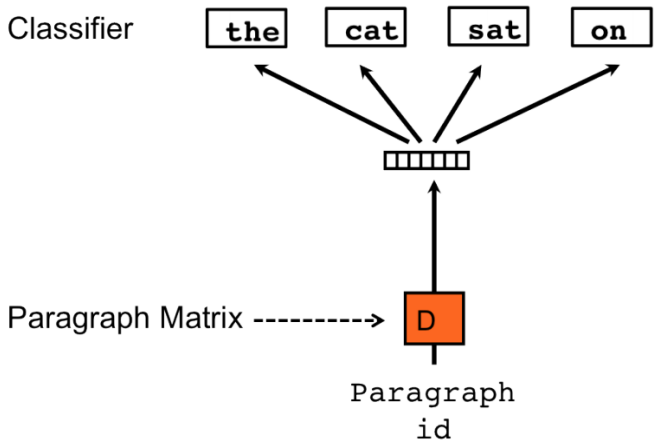


Figure 8: The Distributed Bag of Words model of Paragraph Vectors (PV-DBOW).

Training is otherwise similar, except for the fact that word vectors are not jointly learned along with paragraph vectors. This makes both memory and runtime performance of the PV-DBOW variant much better.

> ***Note:*** *In its Gensim implementation, PV-DBOW uses randomly initialized word embeddings by default; if dbow_words is set to 1, a single step of skip-gram is run to update word embeddings before running dbow. [Lau & Baldwin, 2016] argue that even though dbow can, in theory, work with randomized word embeddings, this degrades performance severely in the tasks they have examined.*
>
> *An intuitive explanation can be traced back to the model's objective function, which is to maximize the dot product between the document*

> *are randomly distributed, it becomes more difficult to optimize the document embedding to be close to its more critical content words.*

Applications, implementations and enhancements

[Le & Mikolov, 2014] demonstrated the use of *paragraph vectors* on several text classification and sentiment analysis tasks, while [Dai et al., 2015] examined it in the context of document similarity tasks and [Lau & Baldwin, 2016] benchmarked it against a forum question duplication task and the *Semantic Textual Similarity (STS) SemEval* shared task. Both later papers present an extended evaluation of the method (the former focusing on the PV-DBOW variant), comparing it to several other methods, and also giving practical advice (the later including code).

The method has a Python implementation, as part of the gensim package, and a PyTorch implementation. Again, [Lau & Baldwin, 2016] also supplied the code used for their examination.

Finally, various enhancements to the method have been proposed. For example, [Li et al., 2016] extend the method to also incorporate n-gram features, while [Thongtan & Phienthrakul, 2019] suggest using cosine similarity instead of dot product when computing the embedding projection (also providing a Java implementation).

Doc2VecC

[Chen, 2017] presented an interesting approach inspired by both the distributed memory model of the paragraph vectors approach (PV-DM) and approaches that average word embeddings to represent documents.

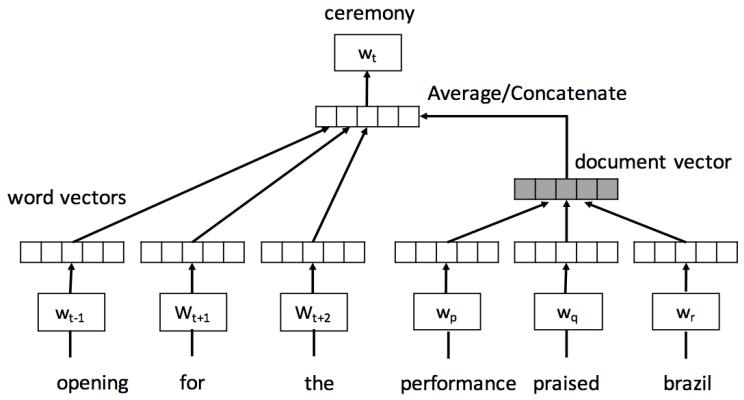


Figure 9: The architecture of the Doc2VecC model.

Similar to *paragraph vectors*, *Doc2VecC* (an acronym of *document vector through corruption*) consists of an input layer, a projection layer and an output layer to predict the target word ("ceremony" in the above example). The embeddings of neighboring words (e.g., "opening," "for," "the") provide local context while the vector representation of the entire document (shown in grey) serves as the global context. In contrast to *paragraph vectors*, which directly learns a unique vector for each document, *Doc2VecC* represents each document as an average of the embeddings of words randomly sampled from the document (e.g. "performance" at position *p*, "praised" at position *q*, and "brazil" at position *r*).

Additionally, the authors choose to corrupt the original document by randomly removing a significant portion of words, representing the document by averaging only the embeddings of the remaining words. This corruption mechanism allows a speedup during training as it significantly reduces the number of parameters to update in backpropagation. The authors also show how it introduces a special form of regularization, which they believe results in the observed performance improvement.

and a semantic relatedness task versus a plethora of state-of-the-art document embedding techniques.

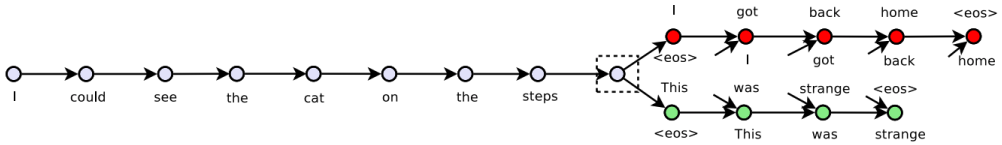
An open-source C-based implementation of the method and code to reproduce the experiments in the paper can be found in a public Github repository.

The general idea of corrupting, or adding noise, to the document embedding learning process to produce a more robust embedding space has also been applied by [Hill et al., 2016] to the *skip-thought* model (see the following sub-section) to create their sequential denoising autoencoder (SDAE) model.

Skip-thought vectors

Presented in [Kiros et al., 2015], this is another early attempt to generalize *word2vec*, and was published with an official pure Python implementation (and recently also boasting implementations for PyTorch and TensorFlow).

This, however, extends *word2vec* — specifically the *skip-gram* architecture — in another intuitive way: the base unit is now sentences, and an encoded sentence is used to predict the sentences around it. The vector representations are learned using an encoder-decoder model trained on the above task; the authors use an RNN encoder with GRU activations and RNN decoders with a conditional GRU. Two different decoders are trained for the previous and next sentences.



and the next sentence s_{i+1}.

Vocabulary expansion in skip-thought

The *skip-thought* encoder uses a word embedding layer that converts each word in the input sentence to its corresponding word embedding, effectively converting the input sentence into a sequence of word embeddings. This embedding layer is also shared with both of the decoders.

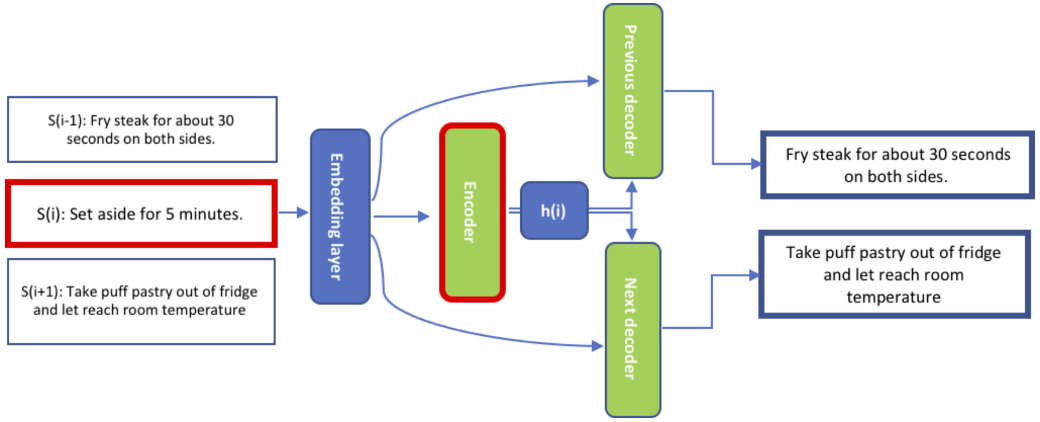


Figure 11: In the skip-thoughts model, sentence s_i is encoded by the encoder; the two decoders condition on the hidden representation of the encoder's output h_i to predict s_{i-1} and s_{i+1} [from Ammar Zaher's post].

However, the authors only use a small vocabulary of 20,000 words, and as a result, many unseen words might be encountered during use in various tasks. To overcome this, a mapping is learned from a word embedding space trained on a much larger vocabulary (e.g., *word2vec*) to the word embedding space of the *skip-thoughts* model, by solving an un-regularized *L2* linear regression loss for the matrix *W* parameterizing this mapping.

Applications, enhancements and further reading

classification, and four sentiment and subjectivity datasets. [Broere, 2017] further investigates the syntactic properties of *skip-thought* sentence representations by training logistic regression on them to predict POS tags and dependency relations.

[Tang et al., 2017a] propose a neighborhood approach to *skip-thought*, dropping ordering information and predicting both the previous and next sentence using a single decoder. [Tang et al., 2017b] expand this examination to propose three enhancements to the model they claim to provide comparable performance using a faster and lighter-weight model: **(1)** only learning to decode the next sentence, **(2)** adding an *avg+max* connection layer between encoder and decoder (as a way to allow non-linear non-parametric feature engineering), and **(3)** performing good word embedding initialization. Finally, [Gan et al., 2016] apply the same approach using a hierarchical CNN-LSTM-based encoder instead of an RNN-only one, across a broad range of applications.

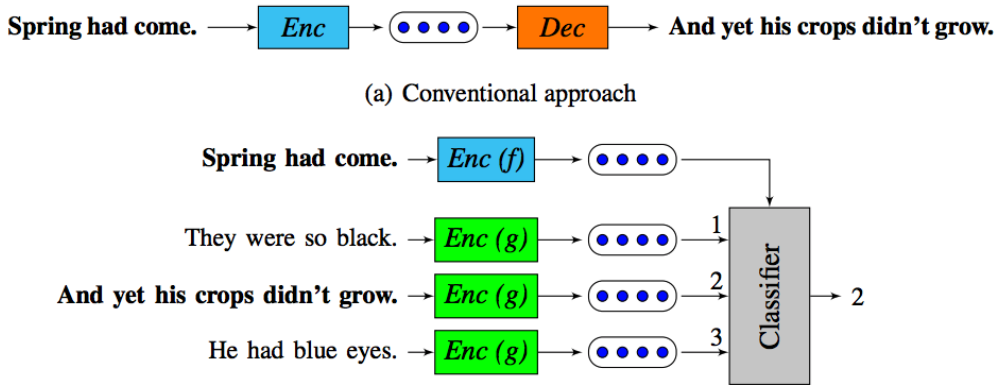
Another variation, presented in [Lee & Park, 2018], learns sentence embeddings by choosing, for each target sentence, influential sentences in the entire document based on document structure, thus identifying dependency structures of sentences using metadata or text styles. Additionally, [Hill et al., 2016] suggest the *sequential denoising autoencoder (SDAE)* model, a variant of *skip-thought* where input data is corrupted according to some noise function, and the model is trained to recover the original data from the corrupted data.

For further non-academic reading on the *skip-thought* model, Sanyam Agarwa gives a great detailed overview of the method on his blog, and Ammar Zaher demonstrates its use to construct an embedding space for cooking recipes.

[Hill et al., 2016] propose a significantly simpler variation on the *skip-thoughts* model; *FastSent* is a simple additive (log-bilinear) sentence model designed to exploit the same signal but at a much lower computational expense. Given a BOW representation of some context sentence, the model simply predicts adjacent sentences (also represented as BOW). More formally, *FastSent* learns a source u^w and target v^w embedding for each word *w* in the model vocabulary. For training example S_{i-1},S_i,S_{i+1} of consecutive sentences, S_i is represented as the sum of its source embeddings $\boldsymbol{s_i}=\sum u^w$ over $w \in S_i$. The cost of the example is then simply $\sum \phi(\boldsymbol{s_i},v^w)$ over $w \in S_{i-1} \cup S_{i+1}$, where ϕ is the softmax function. The paper is accompanied by an official Python implementation.

Quick-thought vectors

[Logeswaran & Lee, 2018] reformulate the document embedding task — the problem of predicting the context in which a sentence appears — as a supervised classification problem (see Figure 12b) rather than the prediction task of previous approaches (see Figure 12a).



(a) Conventional approach

(b) Proposed approach

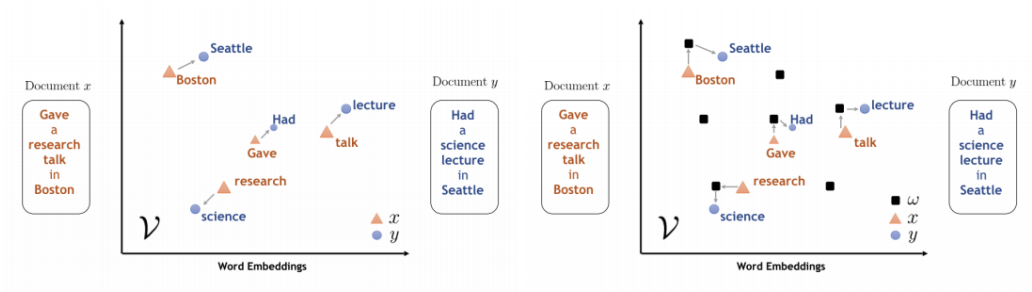
The gist is to use the meaning of the current sentence to predict the meanings of adjacent sentences, where meaning is represented by an embedding of the sentence computed from an encoding function; notice two encoders are learned here: *f* for the input sentence and *g* for candidates. Given an input sentence, it is encoded by an encoder (RNNs, in this case), but instead of generating the target sentence, the model chooses the correct target sentence from a set of candidate sentences; the candidate set is built from both valid context sentences (ground truth) and many other non-context sentences. Finally, the constructed training objective maximizes the probability of identifying the correct context sentences for each sentence in the training data. Viewing the former sentence prediction formulation as choosing a sentence from all possible sentences, this new approach can be seen as a discriminative approximation to the prediction problem.

The authors evaluate their approach on various text classification, paraphrase identification and semantic relatedness tasks, and also provide an official Python implementation.

Word Mover's Embedding (WME)

A very recent method, coming out of IBM research, is *Word Mover's Embedding* (WME), presented in [Wu et al., 2018b]. An official C-based, Python-wrapped implementation is provided.

[Kushner et al, 2015] presented W*ord Mover's Distance* (WMD); this measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to "travel" **in the embedding space** to reach the embedded words of another document (see Figure 13a). Additionally, [Wu et al., 2018a] proposed D2KE (distances to kernels and embeddings), a general methodology for the derivation of a positive-definite kernel from a given distance function.



(a) WMD (b) WME

Figure 13: Contrasting WMD with WME. (a) WMD measures the distance between two documents *x* and y, while (b) WME approximates a kernel derived from WMD with a set of random documents ω.

WME builds on three components to learn continuous vector representations for texts of varying lengths:

1. The ability to learn high-quality word embedding in an unsupervised manner (e.g., using *word2vec*).
2. The ability to construct a distance measure for documents based on said embeddings using W*ord Mover's Distance*(WMD).
3. The ability to derive positive-definite kernel from a given distance function using D2KE.

Using these three components, the following approach is applied:

1. Construct a positive-definite *Word Mover's Kernel*(WMK) via an infinite-dimensional feature map given by the *Word Mover's distance* (WMD) to random documents ω from a given distribution, using D2KE. Due to its use of the WMD, the feature map takes into account alignments of individual words between the documents in the semantic space given by

2. Based on this kernel, derive a document embedding via a random features approximation of the kernel, whose inner products approximate exact kernel computations.

This framework is extensible, since its two building blocks, *word2vec* and WMD, can be replaced by other techniques such as *GloVe* (for word embeddings) or S-WMD (for translation of the word embedding space into a document distance metric).

The authors evaluate WME on 9 real-world text classification tasks and 22 textual similarity tasks, and demonstrate that it consistently matches, and sometimes even outperforms, other state-of-the art techniques.

Sentence-BERT (SBERT)

2018 in NLP was marked by the rise of the transformers (see Figure 14), state-of-the-art neural language models inspired by the transformer model presented in [Vaswani et al. 2017] — a sequence model that dispenses of both convolutions and recurrence and uses attention instead to incorporate sequential information into sequence representation. This booming family includes BERT (and its extensions), GPT (1 and 2), and the XL-flavored transformers.

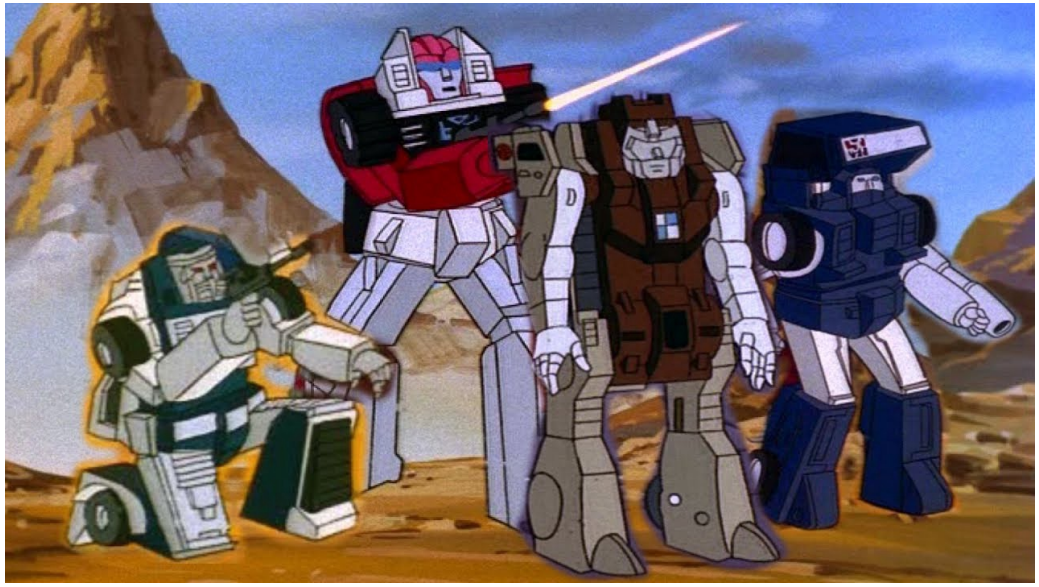


Figure 14: Rise of the transformers.

These models generate contextual embeddings of input tokens (commonly sub-word units), each infused with information of its neighborhood, but are not aimed at generating a rich embedding space for input sequences. BERT even has a special [CLS] token whose output embedding is used for classification tasks, but still turns out to be a poor embedding of the input sequence for other tasks. [Reimers & Gurevych, 2019]

Sentence-BERT, presented in [Reimers & Gurevych, 2019] and accompanied by a Python implementation, aims to adapt the BERT architecture by using use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity (see Figure 15).

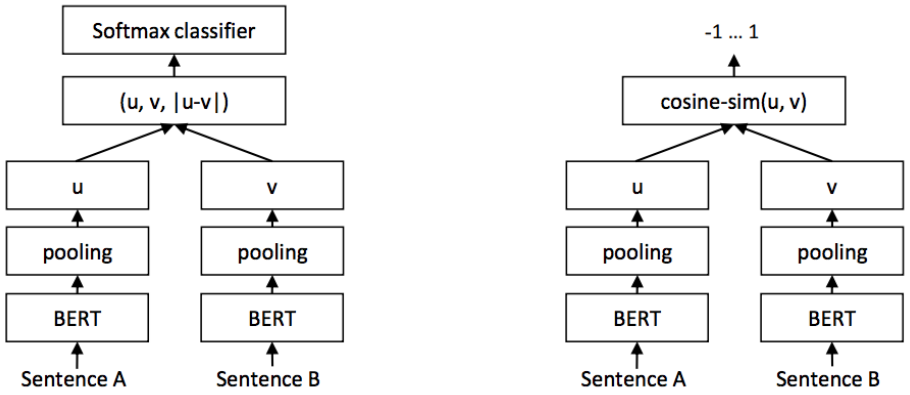


Figure 15: The SBERT architecture in training on a classification objective (left) and inference (right).

Supervised document embedding techniques

The unsupervised methods covered in the previous section allowed us to learn useful representations from large unlabelled corpora. Not unique to natural language processing, this approach focuses on learning representations by designing learning objectives that exploit labels that are freely available within the data. The strength and robustness of these methods thus depend heavily not only on the learning framework but also on how well the artificially designed learning objective requires or brings about the learning of meaningful features or knowledge that would prove useful in various downstream tasks. For example, we expect both semantic and syntactic information to be captured well by word and document embedding spaces.

The contrasting approach to learning meaningful representations of data — in our case word sequences — is to utilize explicit labels (almost always generated by human annotators in some way). Here, the relevancy to various

application, and again, how well does this task bring about the learning of generalizable features and knowledge.

We'll see that supervised approaches range from those that directly utilize a specific labeled task to learn representations, to those that restructure tasks or extract new labeled tasks from them to elicit better representations.

Learning document embeddings from labeled data

There have been various attempts to use labeled or structured data to learn sentence representations. Specifically, [Cho et al., 2014a] and [Sutskever et al., 2014] are perhaps the first attempts to apply the encoder-decoder approach to explicitly learn sentence/phrase embeddings with labelled data; the first using using *Europarl*, a parallel corpus of phrases for statistical machine translation, the second using the English to French translation task from the WMT-14 dataset. Another such notable attempt is presented in [Wieting et al., 2015] and [Wieting & Gimpel, 2017], where both word embeddings and their mapping into document embeddings are jointly learned to minimize cosine similarity between pairs of paraphrases (from the PPDB dataset). [Hill et al., 2015] trained neural language models to map dictionary definitions to pre-trained word embeddings of the words defined by those definitions. Finally, [Conneau et al., 2017] trained NN encoders of various architectures on the Stanford Natural Language Inference task (see Figure 16).

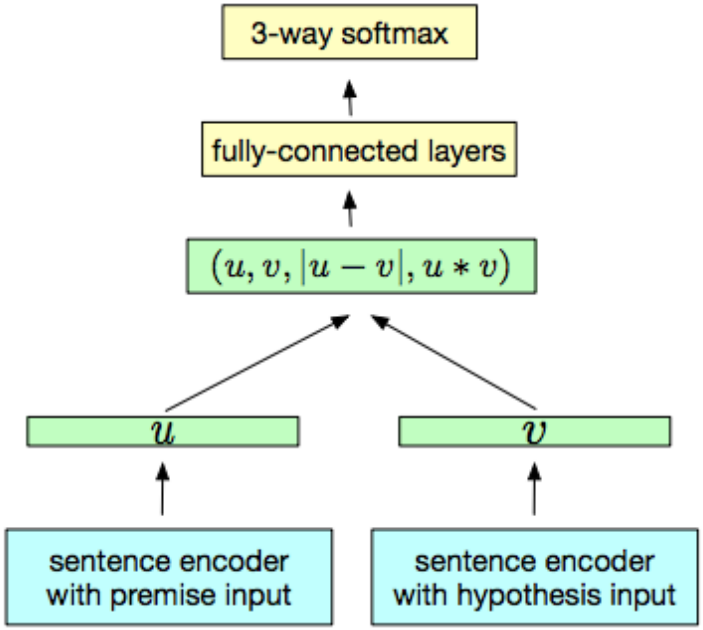


Figure 16: Generic NLI training scheme.

Contextual embeddings for document similarity

A specific case of the above approach is one driven by document similarity. [Das et al., 2016] showcase document embeddings learned to maximize the similarity between two documents via a siamese network for community Q/A. (see Figure 17)

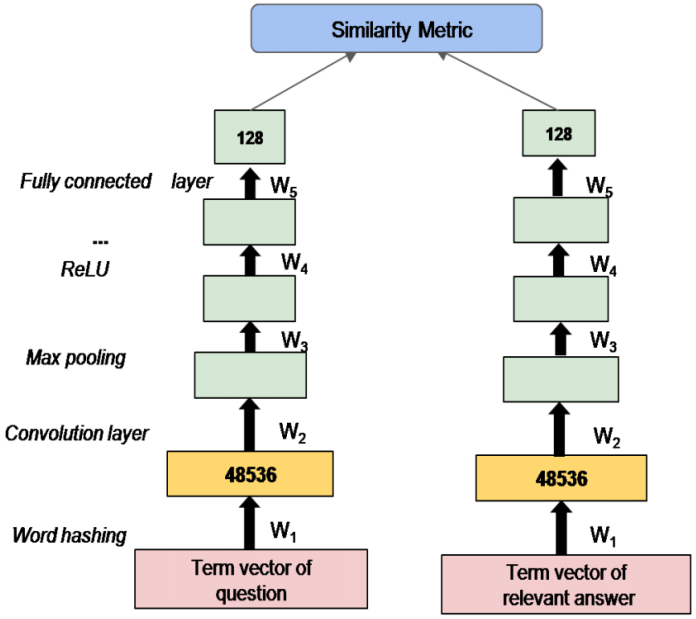


Figure 17: The SCQA network consists of repeating convolution, max pooling and ReLU layers, and a fully connected layer. Weights W1 to W5 are shared between the sub-networks.

Similarly, [Nicosia & Moschitti, 2017] use siamese networks to produce word representations while learning binary text similarity, considering examples in the same category as similar. (see Figure 18)

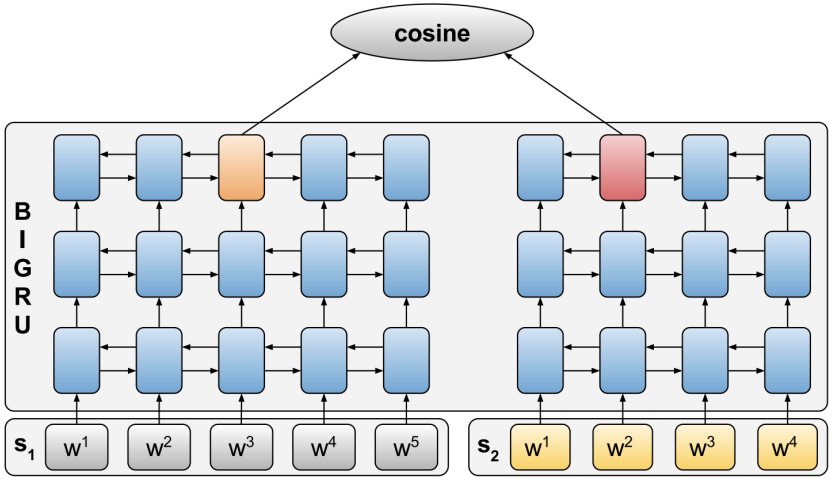


Figure 18: The architecture of the siamese network from [Nicosia & Moschitti, 2017]. Word embeddings of each sentence are consumed by a stack of 3 Bidirectional GRUs. Both network branches share parameter weights.

Crosslingual reduced-rank ridge regression (Cr5)

[Josifoski et al., 2019] introduce a method for embedding documents written in any language into a single, language-independent vector space. This is done by training a ridge-regression-based classifier that uses language-specific bag-of-words features in order to predict the concept that a given document is about. When constraining the learned weight matrix to be of low rank, the authors show it can be factored to obtain the desired mappings from language-specific bags-of-words to language-independent embeddings. An official Python implementation is provided.

Task-specific supervised document embeddings

A common supervised method to produce document embeddings uses various neural network architectures, learning composition operators that map word vectors to document vectors; these are passed to a supervised

task and depend on a class label in order to back-propagate through the composition weights (see Figure 19).

Therefore, almost all hidden layers of the network can be considered to produce a vector embedding of an input document, with the prefix of the network up to that layer being the learned mapping from word vectors to the embedding space. A rigorous examination of the different ways to learn sentence vectors based on word vectors and a supervised learning task can be found in [Wieting et al, 2015].

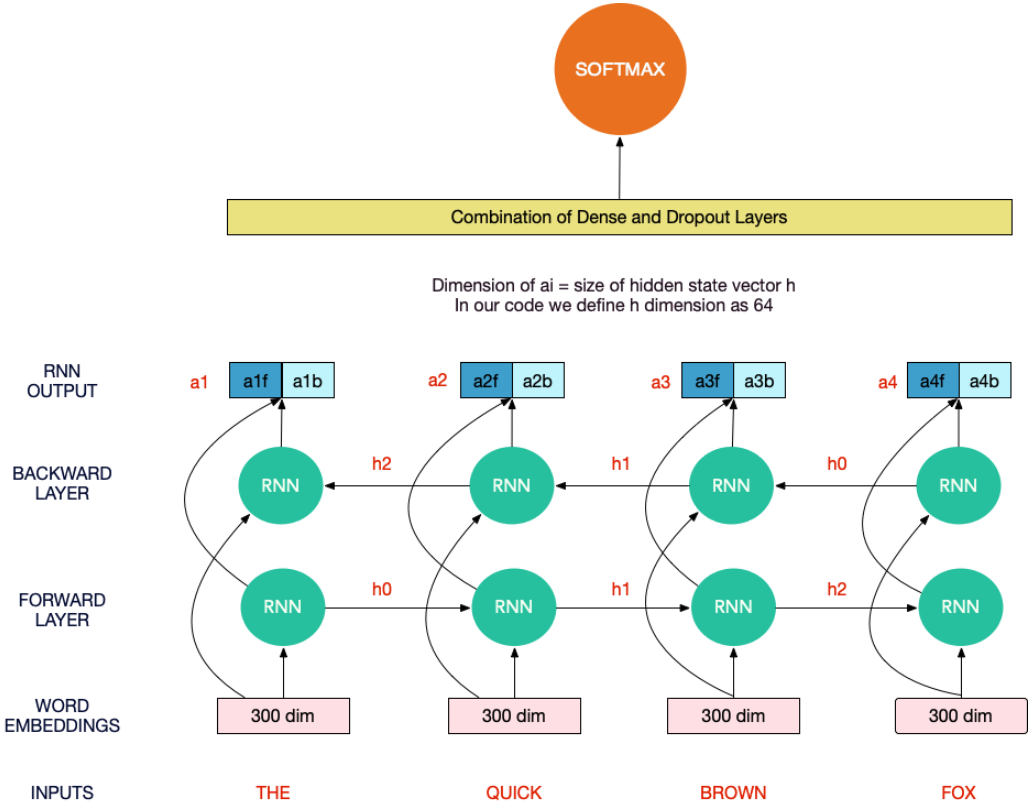


Figure 19: Neural networks implicitly learn to map word embedding

Note that while the word embeddings used can be pre-generated and task-agnostic (to a degree, at least), the mapping learned from them to document embedding is task-specific. While these can be useful for related tasks, this method is bound to be less robust and generalized than unsupervised ones, at least in theory. [Kiros et al, 2015]

Notable uses include sentiment classification using RNNs [Socher et al, 2013], various text classification tasks using CNNs [Kalchbrenner et al, 2014] [Kim, 2014] and both machine translation and text classification tasks using recursive-convolutional neural networks [Cho et al., 2014a, 2014b] [Zhao et al, 2015].

GPT

[Radford et al., 2018] presented the *generative pre-training* (GPT) approach (accompanied by a Python implementation), combining unsupervised and supervised representation learning, using the transformer model presented in [Vaswani et al., 2017] to learn an unsupervised language model on unlabeled corpora, and then fine-tuning its use for each task separately using supervised data. They later presented GPT-2 in [Radford et al., 2019], focusing on bolstering the unsupervised learning portion of their work, an again releasing an official Python implementation.

Deep Semantic Similarity Model (DSSM)

A Microsoft Research project, DSSM is a deep neural network modeling technique for representing text strings in a continuous semantic space and modeling semantic similarity between two text strings (see Figure 20).

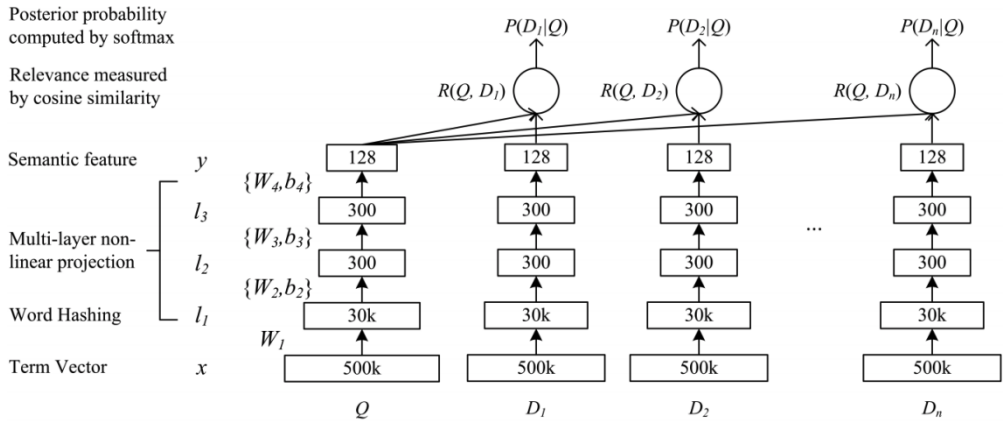


Figure 20: The architecture of a DSSM neural network.

DSSM was used, among other applications, to develop latent semantic models that project entities of different types (e.g., queries and documents) into a common low-dimensional semantic space for a variety of machine learning tasks such as ranking and classification. For example, [Huang et al., 2013] use it project queries and documents into a common low-dimensional space where the relevance of a document given query is computed as the distance between them.

Implementations include TensorFlow, Keras, and two PyTorch variations.

Jointly learning sentence representations

[Ahmad et al., 2018] suggest that jointly learning sentence representations from multiple text classification tasks and combining them with pre-trained word-level and sentence-level encoders result in robust sentence representations that are useful for transfer learning

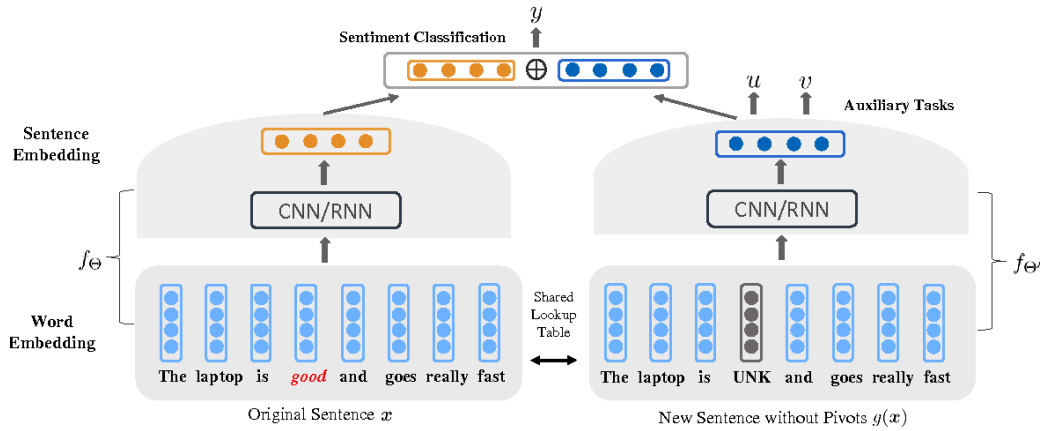


Figure 21: Jointly learning sentence embeddings using auxiliary tasks.

[Yu & Jiang, 2016] similarly show that using two auxiliary tasks to help induce a sentence embedding supposedly works well across domains for sentiment classification, jointly learning this sentence embedding together with the sentiment classifier itself (Figure 21).

Universal Sentence Encoder

Presented in [Cer et al., 2018a] and [Cer et al., 2018b], and accompanied by a TensorFlow implementation, this method actually includes two possible models for sentence representation learning: The *Transformer* model and the *Deep Averaging Network (DAN)* model. Both are designed to allow multi-task learning, with supported tasks including (1) a s*kip-thought* like task for unsupervised learning; (2) a conversational input-response task for the inclusion of parsed conversational data; and (3) classification tasks for training on supervised data (see the following section for a review of this approach). The authors focus on experiments with transfer learning tasks and benchmark their models versus simple CNN and DAN baselines. The method was later extended to address multilingual settings.

The *transformer* model is directly based on the transformer model presented in [Vaswani et al 2017], the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention (see Figure 22).

The model constructs sentence embeddings using the encoding sub-graph of the transformer architecture. The encoder uses attention to compute context-aware representations of words in a sentence that take into account both the ordering and identity of other words. The context-aware word representations are averaged together to obtain a sentence-level embedding.

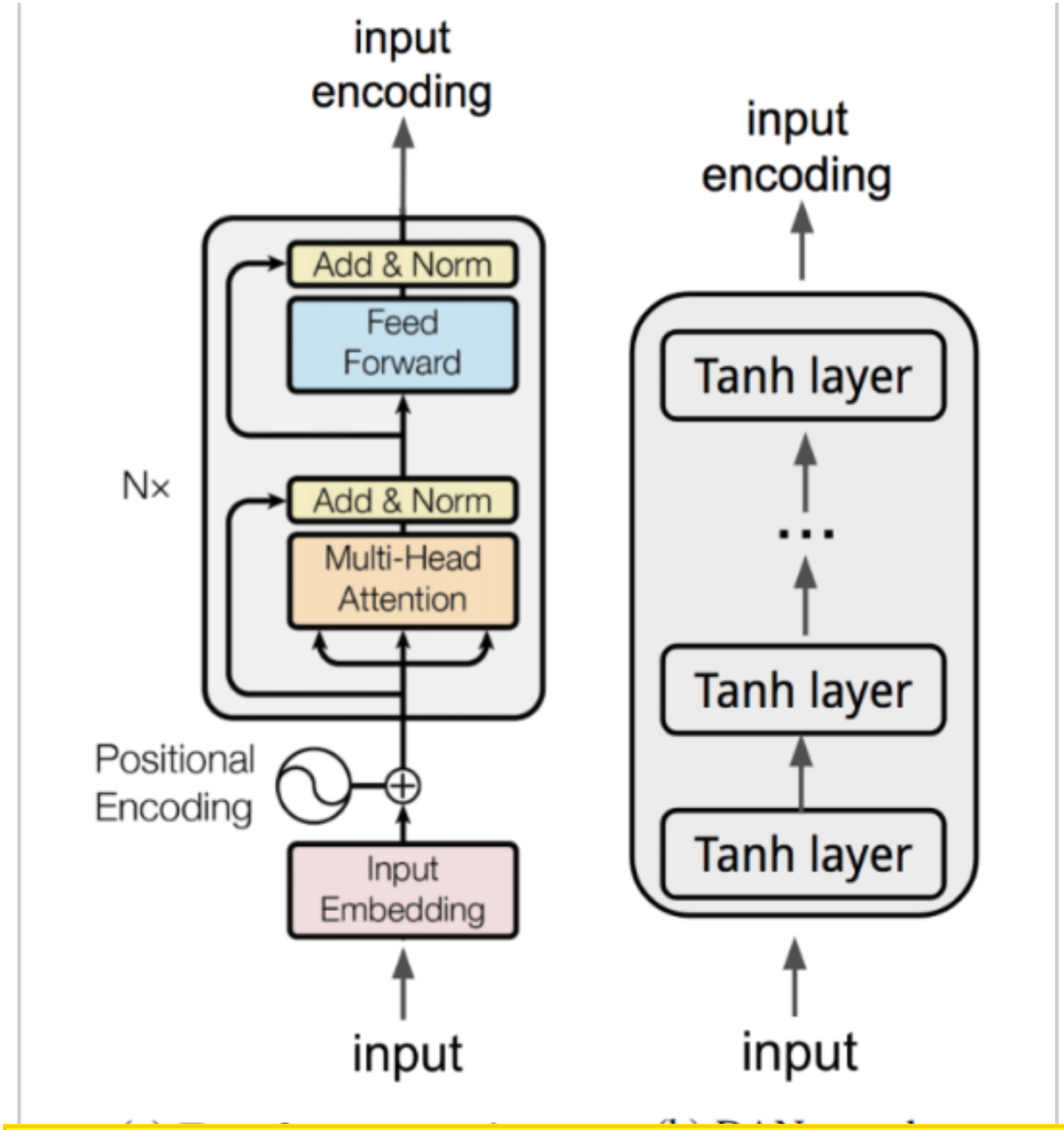


Figure 22: The two models of the Universal Sentence Encoder: (a) Transformer and (b) DAN.

Conversely, in the DAN model, presented in [Iyyer et al., 2015], input embeddings for words and bi-grams are first averaged together and then passed through a feedforward deep neural network (DNN) to produce sentence embeddings (see Figure 22).

GenSen

Much like the Universal Sentence Encoder, the GenSen approach, presented in [Subramanian et al., 2018] together with an official Python implementation, combines multiple supervised and unsupervised learning tasks to train an RNN w/ GRU based encoder-decoder model from which the embedding is extracted. The four supported tasks are: (1) *Skip-thought* vectors, (2) neural machine translation, (3) constituency parsing, and (4) natural language inference (a 3-way classification problem; given a premise and a hypothesis sentence, the objective is to classify their relationship as either entailment, contradiction, or neutral). An official Python implementation was published.

How to choose which technique to use

I have no easy answers here, but here are a few possible takeaways:

1. **Averaging word vectors is a strong baseline**, so a good idea is to start your quest for good document embeddings by focusing on generating very good word vectors, and simply averaging them at first. Undoubtedly, much of the power of document embeddings come from the word vectors upon which they are built, and I think it is safe to say there is a significant delta of information to optimize in that layer before moving forwards. You can try different pre-trained word embeddings, exploring

Then, perhaps extending this slightly by trying different summarization operators or other tricks (like those in [Arora et al., 2016]) can prove to be enough.

2. **Performance can be a key consideration**, especially without a clear leader among the methods. In that case, both averaging word vectors, and some lean methods like *sent2vec*and *FastSent*, are good candidates. In contrast, the real-time vector representation inference required for each sentence when using *doc2vec* might prove costly given application constraints. SentEval, an evaluation toolkit for sentence representations presented in [Conneau & Kiela, 2018] is a tool worth mentioning in this context.
3. **Consider the validity of the learning objective for your task**. The different self-supervised techniques covered above extended *the distributional hypothesis*in different ways, with *skip-thought* and *quick-thought* modeling a strong relation between sentences/paragraphs based on their distance in a document. This perhaps applies trivially for books, articles, and social media posts, but might not apply as strongly to other sequences of texts, especially structured ones, and might thus project your documents into an embedding space which does not apply to them. Similarly, the word-alignment approach, which WME relies on might not apply to every scenario.
4. **Open-source implementations are abundant**, so benchmarking different approaches against your task might be feasible.
5. **There are no clear task-specific leaders.**Papers often benchmark different methods against classification, paraphrasing, and semantic relatedness tasks. Nevertheless, the above conclusion arises both when considering the entirety of the literature on the topic, and specifically when

thought method, and the second by [Wu et al., 2018b] as part of their paper on *Word Mover's Embedding*.

Final words

That's it! As always, I'm sure that the posts I write are not complete, so feel free to suggest corrections and additions to the above overview, either by commenting here or contacting me directly. :)

I also want to thank both Adam Bali and Ori Cohen, who have provided very valuable feedback. Go read their posts!

Finally, I found it worthwhile mentioning that *Papers With Code* has a task dedicated to document embedding and that Facebook Research has open-sourced SentEval, an evaluation toolkit for sentence representations presented in [Conneau & Kiela, 2018].

Now sit back, and let the references overwhelm you.

References

Agibetov, A., Blagec, K., Xu, H., & Samwald, M. (2018). Fast and scalable neural embedding models for biomedical sentence classification. *BMC bioinformatics*, *19*(1), 541.

Ahmad, W. U., Bai, X., Peng, N., & Chang, K. W. (2018). Learning Robust, Transferable Sentence Representations for Text Classification. *arXiv preprint arXiv:1810.00681*.

Arora, S., Liang, Y., & Ma, T. (2016). A simple but tough-to-beat baseline for sentence embeddings. [unofficial implementation]

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic

Broere, (2017). Syntactic properties of skip-thought vectors. *Master's thesis, Tilburg University*.

Cer, D., Yang, Y., Kong, S. Y., Hua, N., Limtiaco, N., John, R. S., ... & Sung, Y. H. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Cer, D., Yang, Y., Kong, S. Y., Hua, N., Limtiaco, N., John, R. S., ... & Strope, B. (2018, November). Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 169–174).

Chen, M. (2017). Efficient vector representation for documents through corruption. *arXiv preprint arXiv:1707.02377*.

Chen, Q., Peng, Y., & Lu, Z. (2018). BioSentVec: creating sentence embeddings for biomedical texts. arXiv preprint arXiv:1810.09302.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Conneau, A., & Kiela, D. (2018). Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*.

Das, A., Yenala, H., Chinnakotla, M., & Shrivastava, M. (2016, August). Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 378–387).

Gan, Z., Pu, Y., Henao, R., Li, C., He, X., & Carin, L. (2016). Unsupervised learning of sentence representations using convolutional neural networks. *arXiv preprint arXiv:1611.07897*.

Gan, Z., Pu, Y., Henao, R., Li, C., He, X., & Carin, L. (2016). Learning generic sentence representations using convolutional neural networks. *arXiv preprint arXiv:1611.07897*.

Gupta, P., Pagliardini, M., & Jaggi, M. (2019). Better Word Embeddings by Disentangling Contextual n-Gram Information. *arXiv preprint arXiv:1904.05033*.

Harris, Z. S. (1954). Distributional structure. Word, 10(2–3), 146–162.

Hill, F., Cho, K., Korhonen, A., & Bengio, Y. (2015). Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, *4*, 17–30.

Hill, F., Cho, K., & Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. *arXiv preprint arXiv:1602.03483*.

Huang, P. S., He, X., Gao, J., Deng, L., Acero, A., & Heck, L. (2013, October). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (pp. 2333–2338). ACM.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., & Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification

Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers) (Vol. 1, pp. 1681–1691).

Josifoski, M., Paskov, I. S., Paskov, H. S., Jaggi, M., & West, R. (2019, January). Crosslingual Document Embedding as Reduced-Rank Ridge Regression. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (pp. 744–752). ACM.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kenter, T., Borisov, A., & De Rijke, M. (2016). Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640*.

Kim, Yoon. "Convolutional neural networks for sentence classification." *arXiv preprint arXiv:1408.5882* (2014).

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems* (pp. 3294–3302).

Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. (2015, June). From word embeddings to document distances. In *International conference on machine learning* (pp. 957–966).

Lau, J. H., & Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*. [code]

Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196).

Lee, T., & Park, Y. (2018). UNSUPERVISED SENTENCE EMBEDDING USING DOCUMENT STRUCTURE-BASED CONTEXT.

Logeswaran, L., & Lee, H. (2018). An efficient framework for learning sentence representations. arXiv preprint arXiv:1803.02893.

Li, B., Liu, T., Du, X., Zhang, D., & Zhao, Z. (2015). Learning document embeddings by predicting n-grams for sentiment classification of long movie reviews. *arXiv preprint arXiv:1512.08183*.

Liu, Y., & Lapata, M. (2018). Learning structured text representations. Transactions of the Association for Computational Linguistics, 6, 63–75.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

Nicosia, M., & Moschitti, A. (2017, August). Learning contextual embeddings for structural semantic similarity using categorical information. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*(pp. 260–270).

Pagliardini, M., Gupta, P., & Jaggi, M. (2017). Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*.

Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*(pp. 1532–1543).

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding with unsupervised learning*. Technical report, OpenAI.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8).

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084*.

Rudolph, M., Ruiz, F., Athey, S., & Blei, D. (2017). Structured embedding models for grouped data. In *Advances in neural information processing systems* (pp. 251–261).

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, *24*(5), 513–523.

Sinoara, R. A., Camacho-Collados, J., Rossi, R. G., Navigli, R., & Rezende, S. O. (2019). Knowledge-enhanced document embeddings for text classification. *Knowledge-Based Systems*, *163*, 955–971.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631–1642).

Subramanian, S., Trischler, A., Bengio, Y., & Pal, C. J. (2018). Learning general purpose distributed sentence representations via large scale multi-task learning. *arXiv preprint arXiv:1804.00079*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing*

Tang, S., Jin, H., Fang, C., Wang, Z., & de Sa, V. R. (2017). Rethinking skip-thought: A neighborhood based approach. *arXiv preprint arXiv:1706.03146*.

Tang, S., Jin, H., Fang, C., Wang, Z., & de Sa, V. R. (2017). Trimming and improving skip-thought vectors. *arXiv preprint arXiv:1706.03148*.

Thongtan, T., & Phienthrakul, T. (2019, July). Sentiment Classification using Document Embeddings trained with Cosine Similarity. In *Proceedings of the 57th Conference of the Association for Computational Linguistics: Student Research Workshop* (pp. 407–414).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *In Advances in neural information processing systems* (pp. 5998–6008).

Wieting, J., Bansal, M., Gimpel, K., & Livescu, K. (2015). Towards universal paraphrastic sentence embeddings. arXiv preprint arXiv:1511.08198.

Wieting, J., & Gimpel, K. (2017). Revisiting recurrent networks for paraphrastic sentence embeddings. *arXiv preprint arXiv:1705.00364*.

Wu, L., Yen, I. E. H., Xu, F., Ravikumar, P., & Witbrock, M. (2018). D2ke: From distance to kernel and embedding. *arXiv preprint arXiv:1802.04956*.

Wu, L., Yen, I. E., Xu, K., Xu, F., Balakrishnan, A., Chen, P. Y., ... & Witbrock, M. J. (2018). Word Mover's Embedding: From Word2Vec to Document Embedding. *arXiv preprint arXiv:1811.01713*.

Yu, J., & Jiang, J. (2016, November). Learning sentence embeddings with auxiliary tasks for cross-domain sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 236–246).

data, *6*(1), 52.

Zhao, H., Lu, Z., & Poupart, P. (2015, June). Self-adaptive hierarchical sentence model. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Original. Reposted with permission.

Bio: Shay Palachy works as a machine learning and data science consultant after being the Lead Data Scientist of ZenCity, a data scientist at Neura, and a developer at Intel and Ravello (now Oracle Cloud). Shay also founded and manages NLPH, an initiative meant to encourage open NLP tools for Hebrew.

Related:

- Word Embeddings in NLP and its Applications
- Word Embeddings & Self-Supervised Learning, Explained
- How to solve 90% of NLP problems: a step-by-step guide

What do you think?

3 Responses

Upvote | Funny | Love | Surprised | Angry

Sad

1 Comment | KDnuggets | Disqus' Privacy Policy | Login

Recommend 1 | Tweet | Share | Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

Thomas • 8 months ago

On the topic of document embedding: Document Embedding with BERT

Reply • Share ›

Subscribe | Add Disqus to your site | Do Not Sell My Data | DISQUS

Top Stories Past 30 Days

Most Popular

1. Five Cool Python Libraries for Data Science
2. The Super Duper NLP Repo: 100 Ready-to-Run Colab Notebooks
3. Natural Language Processing Recipes: Best Practices and Examples
4. The Best NLP with Deep Learning Course is Free
5. 24 Best (and Free) Books To Understand Machine Learning
6. Free High-Quality Machine Learning & Data Science Books & Courses: Quarantine Edition
7. Deep Learning: The Free eBook

Most Shared

1. The Best NLP with Deep Learning Course is Free
2. Automated Machine Learning: The Free eBook
3. Beginners Learning Path for Machine Learning
4. AI and Machine Learning for Healthcare
5. Natural Language Processing Recipes: Best Practices and Examples
6. Deep Learning: The Free eBook
7. Start Your Machine Learning Career in Quarantine