Natural language processing course

Neural networks in NLP. Words embeddings. Representing Words Meaning in Natural Language Processing

Yaroslav Shalivskyy

AI-2

2020

Introduction

Natural language processing tasks have changed dramatically over the past few years, largely thanks to machine learning. As linguistics researchers in the ’80s moved away from transformational grammar — a theoretical approach that argues certain hard rules are responsible for generating grammatically-correct sentences — the floor opened for statistical approaches.

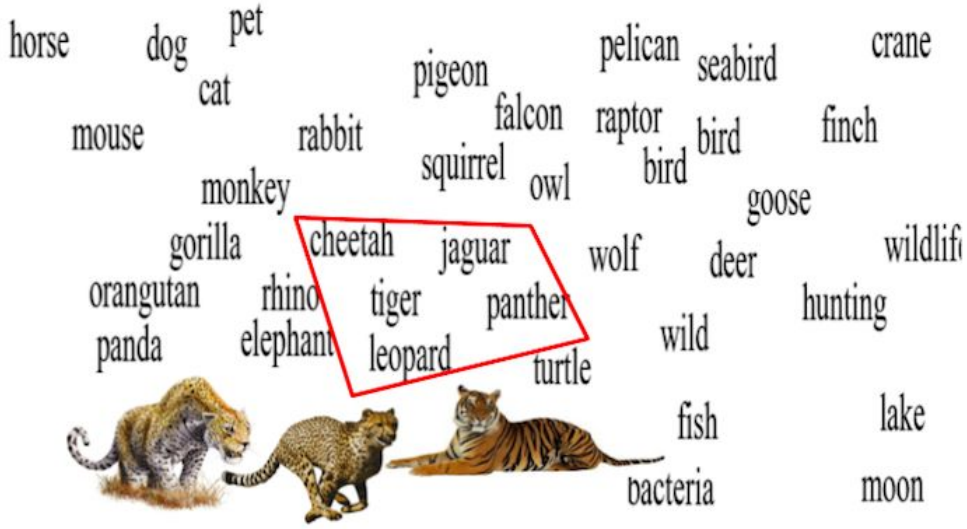
Many of these methods had early successes but still embraced a rules-based approach at some level. Decision trees, for example, essentially just discover different rules for stringing words together. Markov chains — another early and popular variant for text generation — rely on a flat probability of observing the next word in a sequence given all prior words. In other words, they create a rule for what word should come after a given sequence.

Naturally, this approach reached its limit. The probabilities it uncovers tell us nothing about what a word actually means. Generally speaking, statistical NLP hit a wall.

Circa 2010, neural networks blew open natural language processing with state-of-the-art performance on text generation and analysis. Neural nets can discover non-linear and non-rules-based solutions, so they were able to capture the ephemeral qualities of human language by representing context. That is precisely what word embeddings technics (e.g. Word2Vec [1], GloVe [2] or FastText [3]) gives us.

Word embeddings and neural networks

First, what are the word embeddings? Word embeddings are (roughly) dense vector representations of wordforms in which similar words are expected to be close in the vector space. For example, in the figure below, all the big cats (i.e. cheetah, jaguar, panther, tiger, and leopard) are really close in the vector space. Word embeddings represent one of the most successful applications of unsupervised learning, mainly due to their generalization power. The construction of these word embeddings varies, but in general, a neural language model is trained on a large corpus and the output of the network is used to learn word vectors (e.g. Word2Vec [4]).



How can we apply neural networks and word embeddings in text data? This depends on the task, although the general encoding of a text is fairly similar regardless. Given a text, each vector corresponding to each word is passed directly to what we call an embedding layer. Then, there may be zero, one or more hidden layers and an output layer at the end representing the output of the system in a given task. Depending on the specific model, the learning process is carried out in one way or another, but what needs to be learned are the weights representing the connections (word embeddings are also generally updated

throughout the learning process), which is generally achieved through backpropagation. This is a very simplified overview of how neural networks can be applied to text data.

Neural network architectures are useful in Natural Language Processing to solve some various problems comparing to traditional approaches:

- preprocessing techniques like lemmatization, while useful for linear models [8,9], are not really necessary in neural architectures [10]
- eliminates the problem of **absent lemmas in training corpus** (or occurs only a few times). Here is where word embeddings come into play. If you look again at the image from the beginning of the post, you will see that the "leopard" is very close in the vector space to words like "tiger" or "panther". This means that **properties across these similar words are shared**, and therefore we can infer decisions when "leopard" occurs, even if this word has not been explicitly seen during training.
- Finally, relying on **simple word-based features may work fine for detecting the topic of a text, but may fail in other complex tasks.** For example, understanding the syntactic structure of a sentence is generally essential to succeed in a task like sentiment analysis.

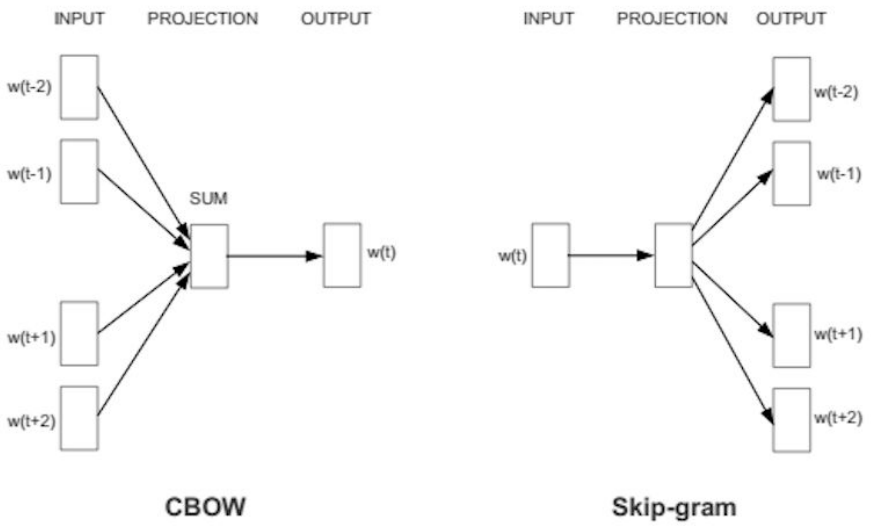
Instead of adding complex features for dealing with all cases (practically impossible), neural architectures take the whole sorted sequence into account, and not each word in isolation. With the appropriate amount of training data (not always trivial in many tasks and domains though), they are expected to learn these nuances.

Word2Vec

Word2Vec is a shallow, two-layer neural network which is trained to reconstruct linguistic contexts of words. It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector

space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2Vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text. It comes in two flavors, the Continuous Bag-of-Words (CBOW) model and the Skip-Gram model. Algorithmically, these models are quite similar.



Continuous Bag-of-Words (CBOW)

CBOW predicts target words (e.g. ‘mat’) from the surrounding context words (‘the cat sits on the’). Statistically, it has the effect that CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation). For the most part, this turns out to be a useful thing for smaller datasets.

Skip-Gram

Skip-gram predicts surrounding context words from the target words (inverse of CBOW). Statistically, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

Word2Vec uses a trick you may have seen elsewhere in machine learning. Word2Vec is a simple neural network with a single hidden layer, and like all neural networks, it has weights, and during training, its goal is to adjust those weights to reduce a loss function. However, Word2Vec is not going to be used for the task it was trained on, instead, we will just take its hidden weights, use them as our word embeddings, and toss the rest of the model.

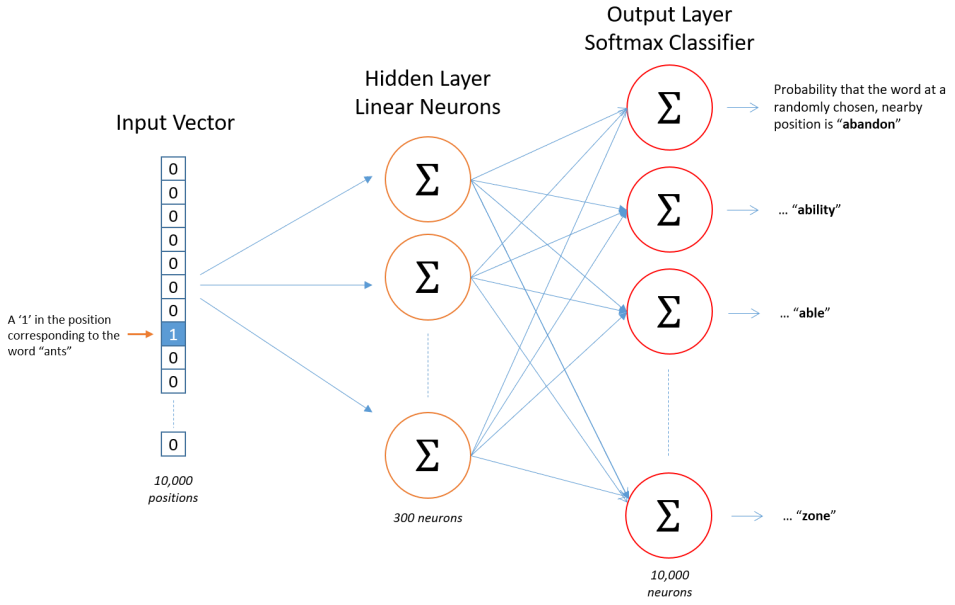
Architecture

The architecture is similar to an autoencoder's one, you take a large input vector, compress it down to a smaller dense vector and then instead of decompressing it back to the original input vector as you do with autoencoders, you output probabilities of target words.

First of all, we cannot feed a word as a string into a neural network. Instead, we feed words as one-hot vectors, which is basically a vector of the same length as the vocabulary, filled with zeros except at the index that represents the word we want to represent, which is assigned "1".

The hidden layer is a standard fully-connected (Dense) layer whose weights are the word embeddings.

The output layer outputs probabilities for the target words from the vocabulary.



Practical methodology

The use of different model parameters and different corpus sizes can greatly affect the quality of a word2vec model.

Accuracy can be improved in a number of ways, including the choice of model architecture (CBOW or Skip-Gram), increasing the training data set, increasing the number of vector dimensions, and increasing the window size of words considered by the algorithm. Each of these improvements comes with the cost of increased computational complexity and therefore increased model generation time.

In models using large corpora and a high number of dimensions, the skip-gram model yields the highest overall accuracy, and consistently produces the highest accuracy on semantic relationships, as well as yielding the highest syntactic accuracy in most cases. However, the CBOW is less computationally expensive and yields similar accuracy results.

Accuracy increases overall as the number of words used increases, and as the number of dimensions increases. Doubling the amount of training data results in an equivalent increase in computational complexity as doubling the number of vector dimensions.

Sub-sampling

Some frequent words often provide little information. Words with a frequency above a certain threshold (e.g ‘a’, ‘an’ and ‘that’) may be subsampled to increase training speed and performance. Also, common word pairs or phrases may be treated as single “words” to increase training speed.

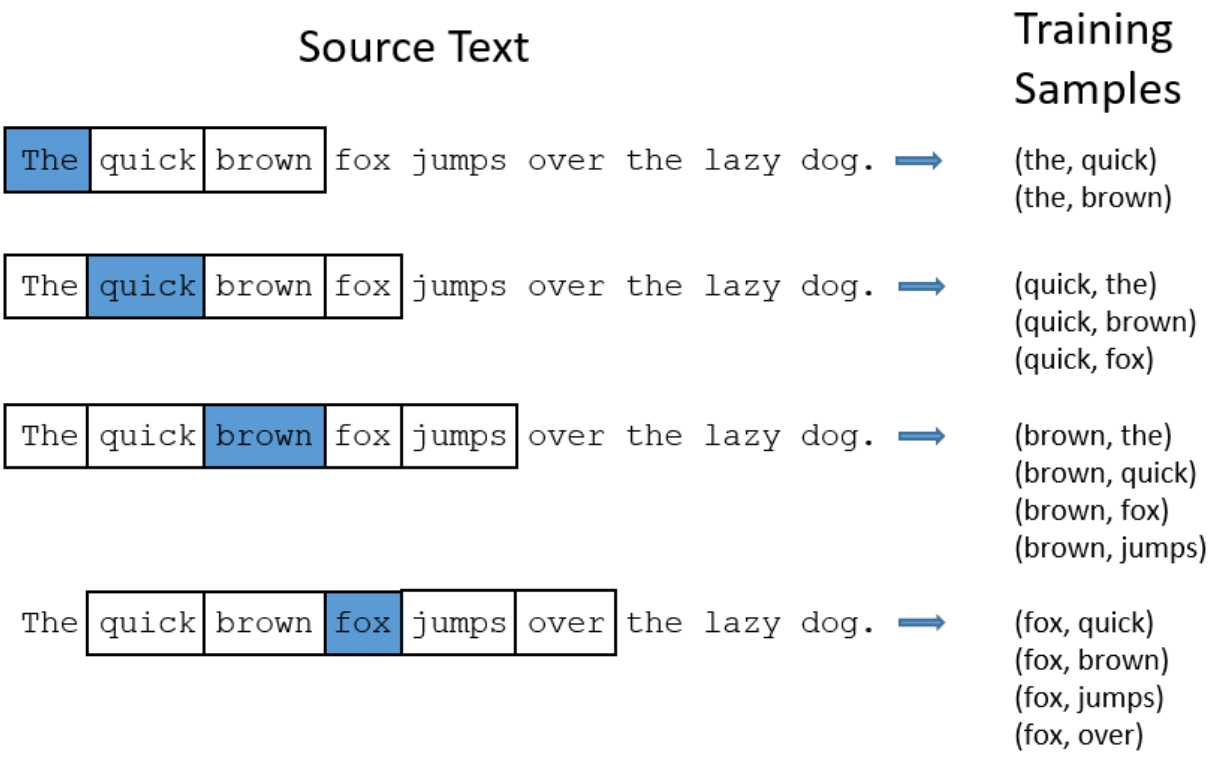
Dimensionality

Quality of word embedding increases with higher dimensionality. However, after reaching some threshold, the marginal gain will diminish. Typically, the dimensionality of the vectors is set to be between 100 and 1,000.

Context window

The size of the context window determines how many words before and after a given word would be included as context words of the given word. According to the authors' note, the recommended value is 10 for skip-gram and 5 for CBOW.

Here is an example of Skip-Gram with context window of size 2:



GloVe

GlobalVectors (GloVe) is a word vector technique that rode the wave of word vectors after a brief silence. Just to refresh, word vectors put words to a nice vector space, where similar words cluster together and different words repel. The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.

The similarity metrics used for nearest neighbor evaluations produce a single scalar that quantifies the relatedness of two words. This simplicity can be problematic since two given words almost always exhibit more intricate relationships than can be captured by a single number. For example, a man may be regarded as similar to a woman in that both words describe human beings; on the other hand, the two words are often considered opposites since they highlight a primary axis along which humans differ from one another.

In order to capture in a quantitative way the nuance necessary to distinguish man from woman, it is necessary for a model to associate more than a single number to the word pair. A natural and simple candidate for an enlarged set of discriminative numbers is the vector difference between the two-word vectors. GloVe is designed in order that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words.

The underlying concept that distinguishes man from woman, i.e. sex or gender may be equivalently specified by various other word pairs, such as king and queen or brother and sister. To state this observation mathematically, we might expect that the vector differences man - woman, king - queen, and brother - sister might all be roughly equal. This property and other interesting patterns can be observed in the above set of visualizations.

GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. For example, consider the co-occurrence probabilities for target words ice and steam with

various probe words from the vocabulary. Here are some actual probabilities from a 6 billion word corpus:

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

As one might expect, ice co-occurs more frequently with solid than it does with gas, whereas steam co-occurs more frequently with gas than it does with solid. Both words co-occur with their shared property water frequently, and both co-occur with the unrelated word fashion infrequently. Only in the ratio of probabilities does noise from non-discriminative words like water and fashion cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam. In this way, the ratio of probabilities encodes some crude form of meaning associated with the abstract concept of the thermodynamic phase.

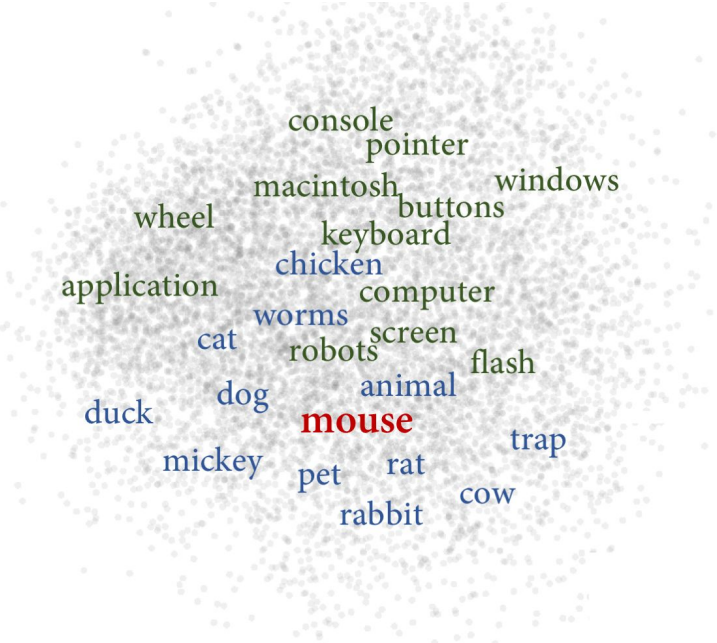
The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associate (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well.

Meaning conflation deficiency

Despite their flexibility and success in capturing semantic properties of words, the effectiveness of word embeddings is generally hampered by an important limitation, known as the meaning conflation deficiency: the inability to discriminate among different meanings of a word.

A word can have one meaning (monosemous) or multiple meanings (ambiguous). For instance, the noun mouse can refer to two different meanings depending on the context: an animal or a computer device. Hence, a mouse is said to be ambiguous. According to the Principle of Economical Versatility of Words [4], frequent words tend to have more senses, which can cause practical problems in downstream tasks. Moreover, this meaning conflation has additional negative impacts on accurate semantic modeling, e.g., semantically unrelated words that are similar to different senses of a word are pulled towards each other in the semantic space [5,6].

In our previous example, the two semantically-unrelated words rat and screen are pulled towards each other in the semantic space for their similarities to two different senses of the mouse. This, in turn, contributes to the violation of the triangle inequality in euclidean spaces [5,7]. Therefore, accurately capturing the semantics of ambiguous words plays a crucial role in the language understanding of NLP systems.



In order to deal with the meaning conflation deficiency, a number of approaches have attempted to model individual word *senses*. The main distinction of these approaches is in how they model meaning and where they obtain it from:

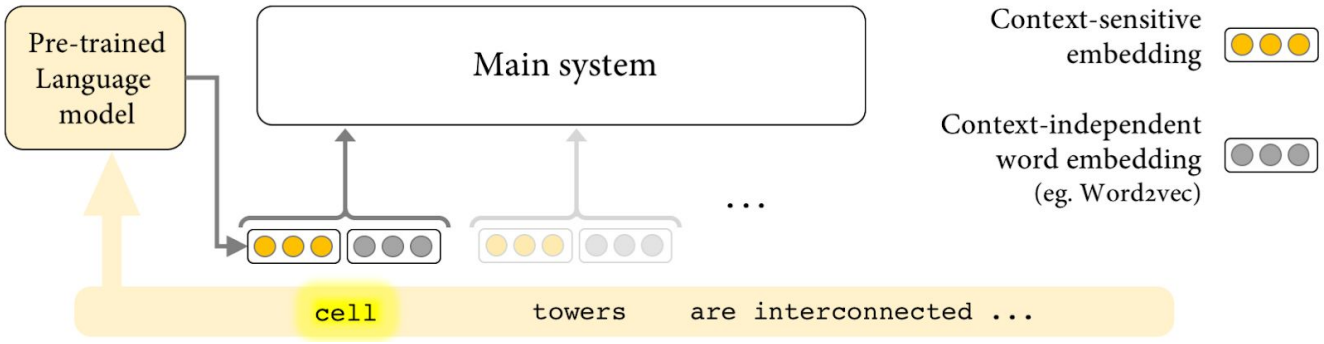
- **Unsupervised** models [5,8,9] directly learn word senses from text corpora.
- **Knowledge-based** techniques [6,10,11] exploit the sense inventories of knowledge resources (e.g. WordNet or BabelNet) as their main source for representing meanings.

Each approach has its own advantages and disadvantages. For instance, knowledge-based models have the advantage of better interpretability and the fact that they can benefit from knowledge encoded in lexical resources (e.g., definitions, properties or images). However, all senses have to be contained in a knowledge base, which has to be constantly updated. On the other hand, unsupervised models are easily adaptable to different domains, although this makes them highly biased to the underlying training corpus, which can potentially result in missing specialized senses.

Moreover, both kinds of approaches suffer from the difficulty of linking learned representations to text representations, as this step requires accurate disambiguation models. In the case of knowledge-based systems, current disambiguation systems are far from perfect [12], as state-of-the-art supervised systems need vast amounts of training data. This data is highly expensive to obtain in practice, which causes the so-called **knowledge-acquisition bottleneck** [13].

As a practical way to deal with this issue, an emerging branch of research has focused on directly integrating unsupervised representations (learned by leveraging language models) into downstream applications. Instead of learning a fixed number of senses per word, **contextualized word embeddings** learn “senses” dynamically, i.e., their representations dynamically change depending on the context in which a word appears. Context2vec [14] is

one of the pioneers for this type of representation. The model represents the context of a target word by extracting the output embedding of a multi-layer perceptron built on top of a bi-directional LSTM language model. More recently, this branch has been popularized by ELMo [15], where seamless integration of these representations into neural NLP systems was proposed, and more recently BERT [16]. At test time, a word's contextualized embedding is usually concatenated with its static embedding and fed to the main model, as shown in the following figure.



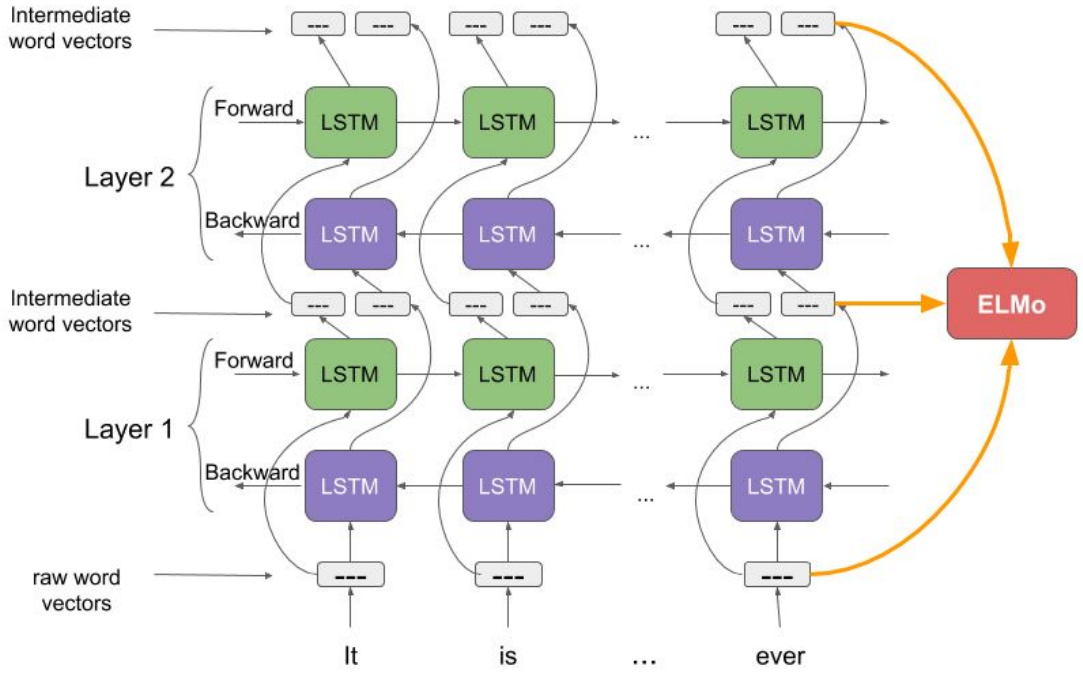
This way the main system benefits from static and dynamic word representations at the same time, and without the need for disambiguation. In fact, the integration of contextual word embeddings into neural architectures has led to consistent improvements over important NLP tasks such as sentiment analysis, question answering, reading comprehension, textual entailment, semantic role labeling, coreference resolution or dependency parsing.

ELMO

ELMo is a novel way to represent words in vectors or embeddings. These word embeddings are helpful in achieving state-of-the-art (SOTA) results in several NLP tasks:

Task	Previous SOTA		ELMo + Baseline
SQuAD	SAN	84.4	85.8
SNLI	Chen et al (2017)	88.6	88.7 +/- 0.17
SRL	He et al (2017)	81.7	84.6
Coref	Lee et al (2017)	67.2	70.4
NER	Peters et al (2017)	91.93 +/- 0.19	92.22 +/- 0.10
Sentiment (5-class)	McCann et al (2017)	53.7	54.7 +/- 0.5

ELMo word vectors are computed on top of a two-layer bidirectional language model (biLM). This biLM model has two layers stacked together. Each layer has 2 passes — forward pass and backward pass.



- The architecture above uses a character-level convolutional neural network (CNN) to represent the words of a text string into raw word vectors.
- These raw word vectors act as inputs to the first layer of biLM.
- The forward pass contains information about a certain word and the context (other words) before that word.
- The backward pass contains information about the word and the context after it.
- This pair of information, from the forward and backward pass, forms the intermediate word vectors.
- These intermediate word vectors are fed into the next layer of biLM.
- The final representation (ELMo) is the weighted sum of the raw word vectors and the 2 intermediate word vectors.

As the input to the biLM is computed from characters rather than words, it captures the inner structure of the word. For example, the biLM will be able to figure out that terms like “beauty” and “beautiful” are related at some level without even looking at the context they often appear in.

ELMo difference from other words embeddings

Unlike traditional word embeddings such as word2vec and GLoVe, the ELMo vector assigned to a token or word is actually a function of the entire sentence containing that word. Therefore, the same word can have different word vectors in different contexts.

Suppose we have a couple of sentences:

1. I read the book yesterday.
2. Can you read the letter now?

The verb “read” in the first sentence is in the past tense. And the same verb transforms into the present tense in the second sentence. This is a case of Polysemy wherein a word could have multiple meanings or senses.

Traditional word embeddings come up with the same vector for the word “read” in both the sentences. Hence, the system would fail to distinguish between the **polysemous** words. These word embeddings just cannot grasp the context in which the word was used.

ELMo word vectors successfully address this issue. ELMo word representations take the entire input sentence into the equation for calculating the word embeddings. Hence, the term “read” would have different ELMo vectors under different contexts.

Conclusion

Even with these groundbreaking results, there is definitely room for improvement. In a recent study was analyzed how well sense and contextualized representations capture word meaning in context (WiC). The results show how state-of-the-art sense and contextualized representation techniques fail at accurately distinguishing meanings in context, performing only slightly better than a simple baseline, while significantly lagging behind the human inter-rater agreement of the dataset (best model at around 65% accuracy with respect to the human performance over 80%). This suggests that the improvements of contextualized embeddings may not be due to a perfectly accurate capturing of meaning, but rather the first step towards this direction. In fact, static (context-independent) word embeddings cannot be completely replaced by contextualized embeddings, as they still play an important role in these systems.

Links

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. CoRR abs/1301.3781. https://code.google.com/archive/p/word2vec/

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In Proceedings of EMNLP, pages 1532–1543. https://nlp.stanford.edu/projects/glove/

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. Transactions of the Association of Computational Linguistics, 5(1):135–146. https://github.com/facebookresearch/fastText

[4] George Kingsley Zipf. 1949. Human behavior and the principle of least effort: An introduction to human ecology. Addison-Wesley, Cambridge, MA.

[5] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. 2014. Efficient non-parametric estimation of multiple embeddings per word in vector space. In Proceedings of EMNLP. Doha, Qatar, pages 1059–1069.

[6] Mohammad Taher Pilehvar and Nigel Collier. 2016. De-conflated semantic representations. In Proceedings of EMNLP, Austin, TX, pages 1680–1690.

[7] Amos Tversky and Itamar Gati. 1982. Similarity, separability, and the triangle inequality. Psychological review 89.2: 123.

[8] Joseph Reisinger and Raymond J. Mooney. 2010. Multi-prototype vector-space models of word meaning. In Proceedings of ACL, pages 109–117.

[9] Jiwei Li and Dan Jurafsky. 2015. Do multi-sense embeddings improve natural language understanding? In Proceedings of EMNLP. Lisbon, Portugal, pages 683–693.

[10] Massimiliano Mancini, Jose Camacho-Collados, Ignacio Iacobacci, and Roberto Navigli. 2017. Embedding words and senses together via joint knowledge-enhanced training. In Proceedings of CoNLL, Vancouver, Canada, pages 100–111.

[11] Sascha Rothe and Hinrich Schütze. 2015. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In Proceedings of ACL. Beijing, China, pages 1793–1803.

[12] Alessandro Raganato et al. 2017. Word sense disambiguation: A unified evaluation framework and empirical comparison. In Proceedings of EACL. Valencia, Spain, pages 99–110. http://lcl.uniroma1.it/wsdeval/

[13] William A. Gale, Kenneth Church, and David Yarowsky. 1992. A method for disambiguating word senses in a corpus. Computers and the Humanities 26:415–439.

[14] Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. Context2vec: Learning generic context embedding with bidirectional LSTM. In Proceedings of CoNLL, Berlin, Germany, pages 51–61.

[15] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In Proceedings of NAACL, New Orleans, LA, USA, pages 2227–2237. https://allennlp.org/elmo

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. https://arxiv.org/abs/1810.04805