AYLIEN

RESEARCH

An overview of word embeddings and their connection to distributional semantic models

October 13, 2016 - Research

Facebook Twitter LinkedIn

Unsupervisedly learned word embeddings have seen tremendous success in numerous NLP tasks in recent years. So much so that in many NLP architectures, they are close to fully replacing more traditional distributional representations such as LSA features and Brown clusters.

You just have to look at last year's EMNLP and ACL conferences, both of which had a very strong focus on word embeddings, and a recent post in Communications of the ACM in which word embeddings is hailed as the catalyst for NLP's breakout. But are they worthy of the hype?

This post is a synopsis of two blogs written by AYLIEN Research Scientist, Sebastian Ruder. You can view Sebastian's original posts, and more, on Machine Learning, NLP and Deep Learning on his blog.

In this overview we aim to give an in-depth understanding of word embeddings and their effectiveness. We'll touch on where they originated, we'll compare popular word embedding models and the challenges associated with them and we'll and try to answer/debunk some common questions and misconceptions.

We will then look to disillusion word embeddings by relating them to literature in distributional semantics and highlighting the factors that actually account for the success of word embedding models.

A brief history of word embeddings

Vector space models have been used in distributional semantics since the 1990s. Since then, we have seen the development of a number models used for estimating continuous representations of words, Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA) being two such examples.

The term word embeddings was originally coined by Bengio et al. in 2003 who trained them in a neural language model together with the model's parameters. However, Collobert and Weston were arguably the first to demonstrate the power of pre-trained word embeddings in their 2008 paper *A unified architecture for natural language processing*, in which they establish word embeddings as a highly effective tool when used in downstream tasks, while also announcing a neural network architecture that many of today's approaches were built upon. It was Mikolov et al. (2013), however, who really brought word embedding to the fore through the creation of word2vec, a toolkit enabling the training and use of pre-trained embeddings. A year later, Pennington et al. introduced us to GloVe, a competitive set of pre-trained embeddings, suggesting that word embeddings was suddenly among the mainstream.

Word embeddings are considered to be among a small number of successful applications of unsupervised learning at present. The fact that they do not require pricey annotation is probably their main benefit. Rather, they can be derived from already available unannotated corpora.

Word embedding models

Naturally, every feed-forward neural network that takes words from a vocabulary as input and embeds them as vectors into a lower dimensional space, which it then fine-tunes through back-propagation, necessarily yields word embeddings as the weights of the first layer, which is usually referred to as *Embedding Layer*.

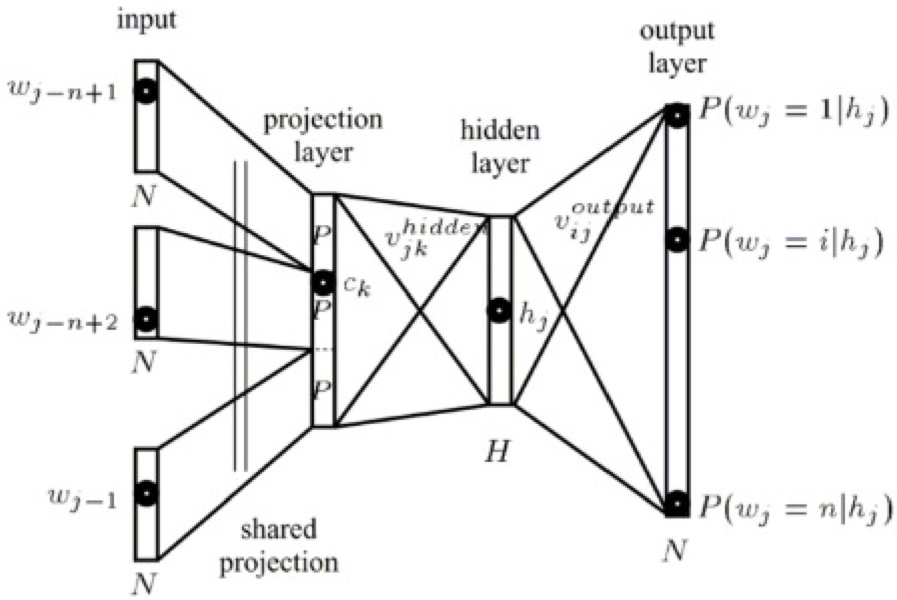


Figure 1: A neural language model (Bengio et al., 2006)

The key difference between a network like this and a method like word2vec is its computational complexity, which explains why it wasn't until 2013 that word embeddings became so prominent in the NLP space. The recent and rapid expansion and affordability in computational power has certainly aided its emergence.

The training objectives for GloVe and word2vec are another difference, with both geared towards producing word embeddings that encode general semantic relationships and can provide benefit in many downstream tasks. Regular neural networks, in comparison, generally produce task-specific embeddings with limitations in relation to their use elsewhere.

In comparing models, we will assume the following notational standards: We assume a training corpus containing a sequence of T training words $w_1, w_2, w_3, \cdots, w_T$ that belong to a vocabulary V whose size is $|V|$. Our models generally consider a context of n words. We associate every word with an input embedding v_w (the eponymous word embedding in the Embedding Layer) with d dimensions and an output embedding v'_w (another word representation whose role will soon become clearer). We finally optimize an objective function J_θ with regard to our model parameters θ and our model outputs some score $f_\theta(x)$ for every input x.

Classic neural language model

The classic neural language model proposed by Bengio et al. [1] in 2003 consists of a one-hidden layer feed-forward neural network that predicts the next word in a sequence as in Figure 2.

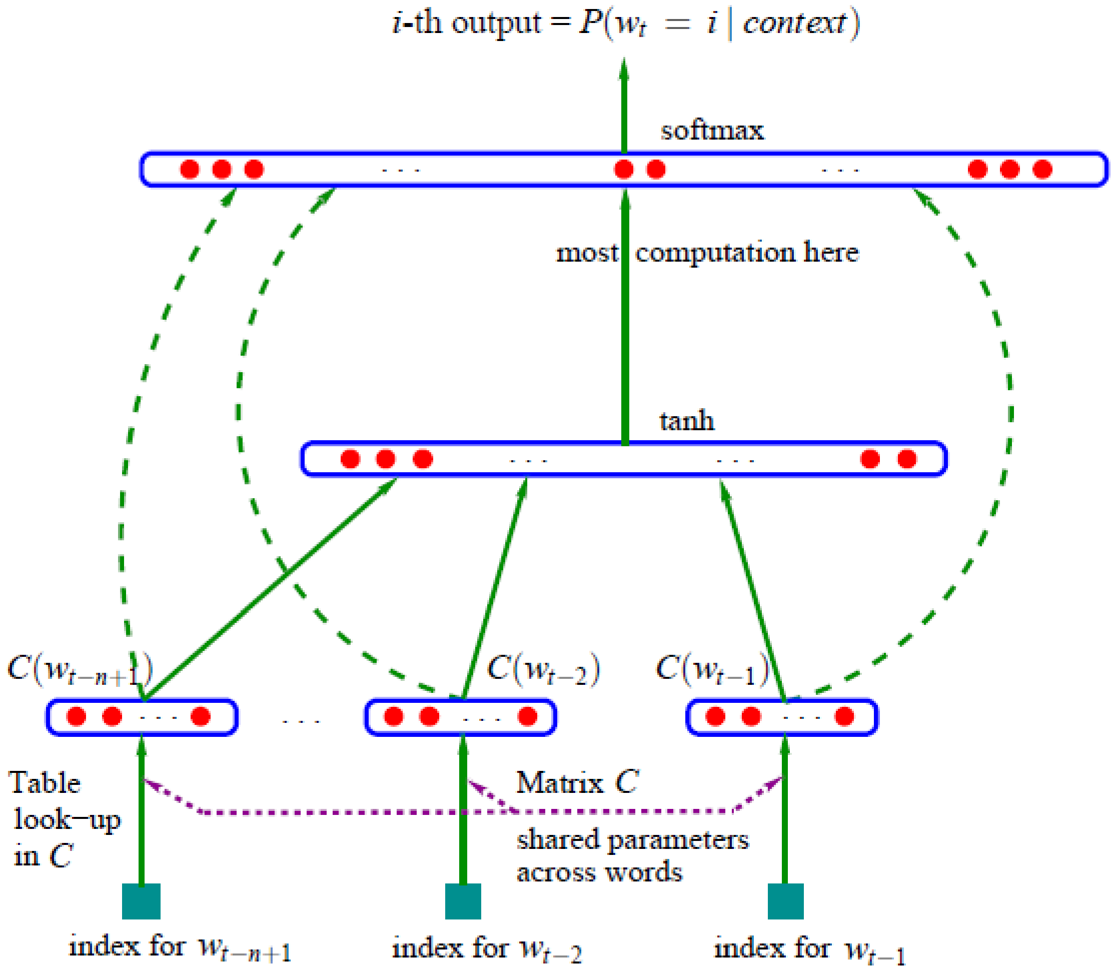


Figure 2: Classic neural language model (Bengio et al., 2003)

Their model maximizes what we've described above as the prototypical neural language model objective (For simplicity, the regularization term has been omitted):

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log f(w_t, w_{t-1}, \cdots, w_{t-n+1}).$$

$f(w_t, w_{t-1}, \cdots, w_{t-n+1})$ is the output of the model, i.e. the probability $p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})$ as computed by the softmax, where n is the number of previous words fed into the model.

Bengio et al. were among the first to introduce what has become to be known as a word embedding, a real-valued word feature vector in $\mathbb{R}$. The foundations of their model can still be found in today's neural language and word embedding models. They are:

1. Embedding Layer: This layer generates word embeddings by multiplying an index vector with a word embedding matrix;

2. Intermediate Layer(s): One or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of n previous words;

3. Softmax Layer: The final layer that produces a probability distribution over words in V.

Bengio et al. also determine two issues with current state-of-the-art-models:

– The first is that Layer **2.** is replaceable with an LSTM, which is used by state-of-the-art neural language models [6], [7].

– They also identify the final softmax layer (more precisely: the normalization term) as the network's main bottleneck, as the cost of computing the softmax is proportional to the number of words in V, which is typically on the order of hundreds of thousands or millions.

Discovering methods that alleviate the computational cost related to computing the softmax over a large vocabulary [9] is therefore one of the main challenges in both neural language and word embedding models.

C&W model

After Bengio et al.'s initial efforts in neural language models, research in word embeddings stalled as computational power and algorithms were not yet at a level that enabled the training of a large vocabulary.

In 2008, Collobert and Weston [4] (thus C&W) demonstrated that word embeddings trained on an adequately large dataset carry syntactic and semantic meaning and improve performance on downstream tasks. In their 2011 paper, they further expand on this [8].

In order to avoid computing the expensive softmax, their solution is to employ an alternative objective function: rather than the cross-entropy criterion of Bengio et al., which maximizes the probability of the next word given the previous words, Collobert and Weston train a network to output a higher score f_θ for a correct word sequence (a probable word sequence in Bengio's model) than for an incorrect one. For this purpose, they use a pairwise ranking criterion, which looks like this:

$$J_\theta \ = \sum\limits_{x \in X} \sum\limits_{w \in V} \text{max} \lbrace 0, 1 - f_\theta(x) + f_\theta(x^{(w)}) \rbrace .$$

They sample correct windows x containing n words from the set of all possible windows X in their corpus. For each window x, they then produce a corrupted, incorrect version $x^{(w)}$ by replacing x's centre word with another word w from V. Their objective now maximises the distance between the scores output by the model for the correct and the incorrect window with a margin of 1. Their model architecture, depicted in Figure 3 without the ranking objective, is analogous to Bengio et al.'s model.

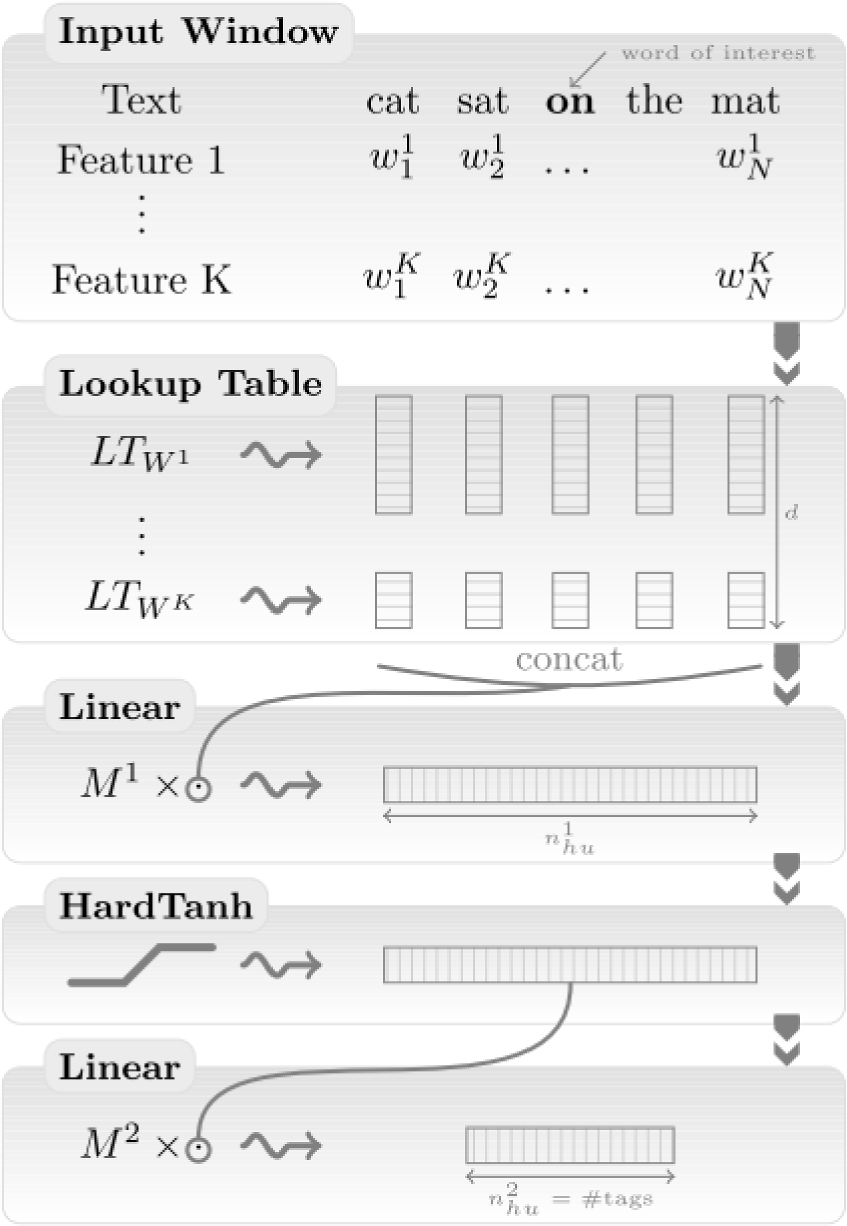


Figure 3: The C&W model without ranking objective (Collobert et al., 2011)

The resulting language model produces embeddings that already possess many of the relations word embeddings have become known for, e.g. countries are clustered close together and syntactically similar words occupy similar locations in the vector space. While their ranking objective eliminates the complexity of the softmax, they keep the intermediate fully-connected hidden layer (**2.**) of Bengio et al. around (the **HardTanh** layer in Figure 3), which constitutes another source of expensive computation. Partially due to this, their full model trains for seven weeks in total with $|V| = 130000$.

Word2Vec

Word2Vec is arguably the most popular of the word embedding models. Because word embeddings are a key element of deep learning models for NLP, it is generally assumed to belong to the same group. However, word2vec is not technically not be considered a component of deep learning, with the reasoning being that its architecture is neither deep nor uses non-linearities (in contrast to Bengio's model and the C&W model).

Mikolov et al. [2] recommend two architectures for learning word embeddings that, when compared with previous models, are computationally less expensive.

Here are two key benefits that these architectures have over Bengio's and the C&W model;

– They forgo the costly hidden layer.

– They allow the language model to take additional context into account.

The success their model can not only be attributed to these differences, it importantly also comes from specific training strategies, both of which we will now look at;

Continuous bag-of-words (CBOW)

Unlike a language model that can only base its predictions on past words, as it is assessed based on its ability to predict each next word in the corpus, a model that only aims to produce accurate word embeddings is not subject to such restriction. Mikolov et al. therefore use both the n words before and after the target word w_t to predict it as shown in Figure 4. This is known as a continuous bag of words (CBOW), owing to the fact that it uses continuous representations whose order is of no importance.

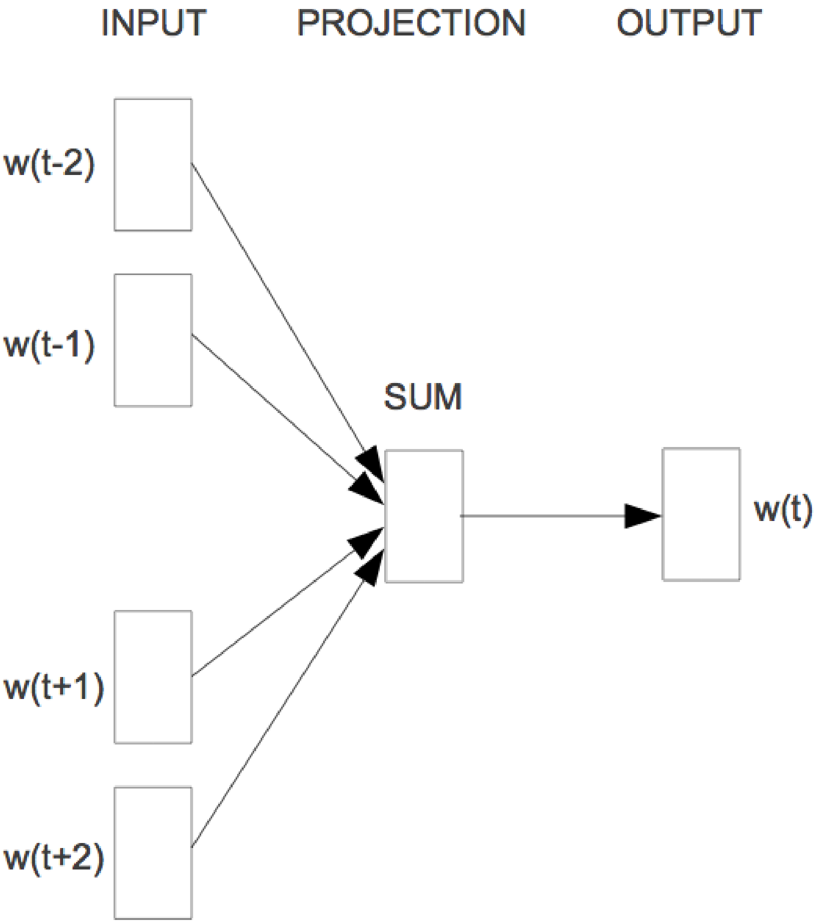


Figure 4: Continuous bag-of-words (Mikolov et al., 2013)

The purpose of CBOW is only marginally different than that of the the language model one:

$$J_\theta = \frac{1}{T}\sum\limits_{t=1}^T\ \text{log} \space p(w_t \: | \: w_{t-n} , \cdots , w_{t-1}, w_{t+1}, \cdots , w_{t+n}).$$

Rather than feeding n previous words into the model, the model receives a window of n words around the target word w_t at each time step t.

Skip-gram

While CBOW can be seen as a precognitive language model, skip-gram turns the language model objective on its head: rather than using the surrounding words to predict the centre word as with CBOW, skip-gram uses the centre word to predict the surrounding words as can be seen in Figure 5.

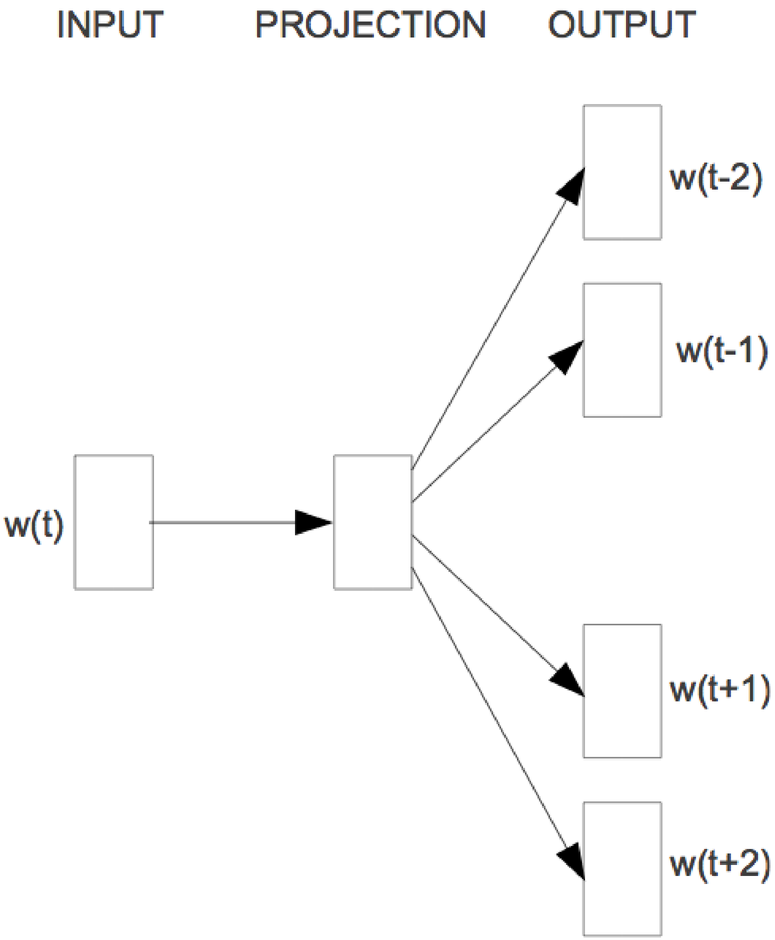


Figure 5: Skip-gram (Mikolov et al., 2013)

The skip-gram objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n,\neq 0}\log p(w_{t+j} \mid w_t).$$

GloVe

In contrast to word2vec, GloVe [5] seeks to make explicit what word2vec does implicitly: Encoding meaning as vector offsets in an embedding space — seemingly only a serendipitous by-product of word2vec — is the specified goal of GloVe.

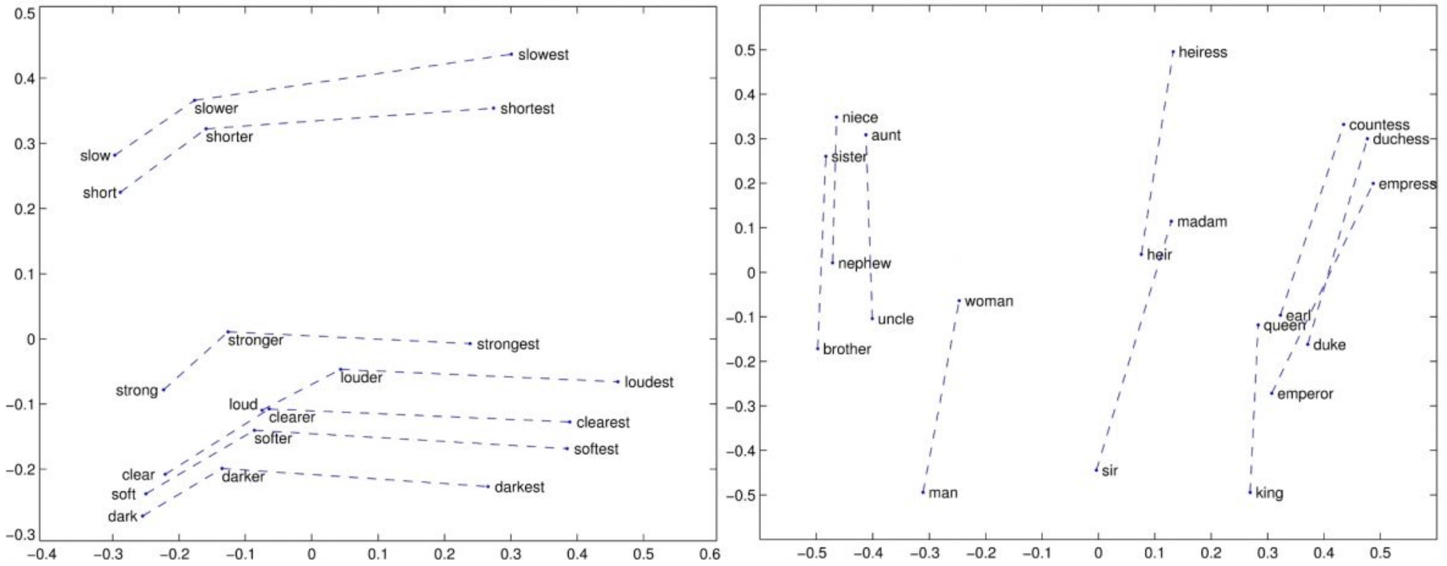


Figure 6: Vector relations captured by GloVe (Stanford)

To be specific, the creators of GloVe illustrate that the ratio of the co-occurrence probabilities of two words (rather than their co-occurrence probabilities themselves) is what contains information and so look to encode this information as vector differences.

For this to be accomplished, they propose a weighted least squares objective J that directly aims to reduce the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \, (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

where w_i and b_i are the word vector and bias respectively of word i, $\tilde{w}_j$ and b_j are the context word vector and bias respectively of word j, X_{ij} is the number of times word i occurs in the context of word j, and f is a weighting function that assigns relatively lower weight to rare and frequent co-occurrences.

As co-occurrence counts can be directly encoded in a word-context co-occurrence matrix, GloVe takes such a matrix rather than the entire corpus as input.

Word embeddings vs. distributional semantics models

Word embedding models such as word2vec and GloVe gained such popularity as they appeared to regularly and substantially outperform traditional Distributional Semantic Models (DSMs). Many attributed this to the neural architecture of word2vec, or the fact that it predicts words, which seemed to have a natural edge over solely relying on co-occurrence counts.

DSMs can be seen as *count* models as they “count” co-occurrences among words by operating on co-occurrence matrices. Neural word embedding models, in contrast, can be viewed as *predict* models, as they try to predict surrounding words.

In 2014, Baroni et al. [11] demonstrated that, in nearly all tasks, *predict* models consistently outperform *count* models, and therefore provided us with a comprehensive verification for the supposed superiority of word embedding models. Is this the end? No.

With GloVe, we have already seen that the differences are not as obvious: While GloVe is considered a predict model by Levy et al. (2015) [10], it is clearly factorizing a word-context co-occurrence matrix, which brings it close to traditional methods such as PCA and LSA. Even more, Levy et al. [12] demonstrate that word2vec implicitly factorizes a word-context PMI matrix.

While on the surface DSMs and word embedding models use varying algorithms to learn word representations – the former count, the latter predict – both types of model fundamentally act on the same underlying statistics of the data, i.e. the co-occurrence counts between words.

And so the question that we will focus on for the remainder of this post still remains:

Why do word embedding models still outperform DSM with very similar information?

Comparison models

To establish the elements that attribute to the success of neural word embedding models, and illustrate how they can be transferred to traditional processes, we will compare the following models;

Positive Pointwise Mutual Information (PPMI)

PMI is a typical measure for the strength of association between two words.It is defined as the log ratio between the joint probability of two words w and c and the product of their marginal probabilities: $PMI(w,c) = \log \frac{P(w,c)}{P(w)\ P(c)}$. As $PMI(w,c) = \log 0 = -\infty$ for pairs (w,c) that were never observed, PMI is in practice often replaced with *positive* PMI (PPMI), which replaces negative values with 0, yielding $PPMI(w,c) = \max(PMI(w,c),0)$.

Singular Value Decomposition (SVD)

SVD is among the more popular methods for dimensionality reduction and came about in NLP originally via latent semantic analysis (LSA). SVD factorizes the word-context co-occurrence matrix into the product of three matrices $U \cdot \Sigma \times V^T$ where U and V are orthonormal matrices (i.e. square matrices whose rows and columns are orthogonal unit vectors) and Σ is a diagonal matrix of eigenvalues in

decreasing order. In practice, SVD is often used to factorize the matrix produced by PPMI. Generally, only the top d elements of Σ are kept, yielding $W^{SVD} = U_d \cdot \Sigma_d$ and $C^{SVD} = V_d$, which are commonly used as the word and context representations respectively.

Skip-gram with Negative Sampling (SGNS)

Aka word2vec, as shown above.

Global Vectors (GloVe)

As shown earlier in this post.

Hyperparameters

We will focus on the following hyper-parameters:

Pre-processing

Word2vec suggests three methods of pre-processing a corpus, each of which can be applied to DSMs with ease.

Dynamic context window

Normally in DSMs, the context window is unweighted and of a unchanging size. Both SGNS and GloVe, however, use a scheme that assigns more weight to closer words, as closer words are generally considered to be more important to a word's meaning. Additionally, in SGNS, the window size is not fixed, but the actual window size is dynamic and sampled uniformly between 1 and the maximum window size during training.

Subsampling frequent words

SGNS dilutes very frequent words by randomly removing words whose frequency f is higher than some threshold t with a probability $p = 1 - \sqrt{\frac{t}{f}}$. As this subsampling is done *before* actually creating the windows, the context windows used by SGNS in practice are larger than indicated by the context window size.

Deleting rare words

During the pre-processing of SGNS, rare words are also deleted before creating the context windows, which increases the actual size of the context windows further. The actual performance impact of this is insignificant, however, according to Levy et al. (2015)

Association metric

In relation to measuring the association between two words, PMI is seen as useful metric. Since Levy and Goldberg (2014) have shown SGNS to implicitly factorize a PMI matrix, two variations stemming from this formulation can be introduced to regular PMI.

Shifted PMI

In SGNS, the greater the volume of negative samples k, the more data is being used and so the estimation of the parameters should therefore improve. k affects the shift of the PMI matrix that is implicitly factorized by word2vec, i.e. k k shifts the PMI values by log k.

If we transfer this to regular PMI, we obtain Shifted PPMI (SPPMI): $SPPMI(w,c) = \max(PMI(w,c) - \log k, 0)$.

Context distribution smoothing

In SGNS, the negative samples are sampled according to a _smoothed_ unigram distribution, i.e. an unigram distribution raised to the power of α, which is empirically set to $\frac{3}{4}$. This leads to frequent words being sampled relatively less often than their frequency would indicate.

We can transfer this to PMI by equally raising the frequency of the context words $f(c)$ to the power of α:

$PMI(w,c) = \log\frac{p(w,c)}{p(w)p_\alpha(c)}$ where $p_\alpha(c) = \frac{f(c)^\alpha}{\sum_c f(c)^\alpha}$ and $f(x)$ is the frequency of word x.

Post-processing

Just like in pre-processing, three methods can be used to modify the word vectors produced by an algorithm.

Adding context vectors

The authors of GloVe recommend the addition of word vectors and context vectors to create the final output vectors, e.g. $\vec{v}_{\text{cat}} = \vec{w}_{\text{cat}} + \vec{c}_{\text{cat}}$. This adds first-order similarity terms, i.e $w \cdot v$. This method, however, cannot be applied to PMI, as the vectors produced by PMI are too infrequent.

Eigenvalue weighting

SVD produces the following matrices: $W^{SVD} = U_d \cdot \Sigma_d$ and $C^{SVD} = V_d$. These matrices, however, have different properties: C^{SVD} is orthonormal, while W^{SVD} is not.

SGNS is more symmetric in contrast. We can thus weight the eigenvalue matrix Σ_d with an additional parameter p, which can be tuned, to yield the following:

$W^{SVD} = U_d \cdot \Sigma_d^p$.

Vector normalisation

Finally, we can also normalise all vectors to unit length.

Results

Levy et al. (2015) train all models on a dump of the English wikipedia and evaluate them on the commonly used word similarity and analogy datasets. You can read more about the experimental setup and training details in their paper. We summarise the most important results and takeaways below.

Takeaways

Levy et al. find that SVD — and not one of the word embedding algorithms — performs best on similarity tasks, while SGNS performs best on analogy datasets. They furthermore shed light on the importance of hyperparameters compared to other choices:

1. Hyperparameters vs. algorithms:
Hyperparameter settings are often more important than algorithm choice.
No single algorithm consistently outperforms the other methods.

2. Hyperparameters vs. more data:
Training on a larger corpus helps for some tasks.
In 3 out of 6 cases, tuning hyperparameters is more beneficial.

Debunking prior claims

Equipped with these insights, we can now debunk some generally held claims:

1. Are embeddings superior to distributional methods?
With the right hyperparameters, no approach has a consistent advantage over another.

2. Is GloVe superior to SGNS?
SNGS outperforms GloVe on all comparison tasks of Levy et al. This should necessarily be taken with a grain of salt as GloVe might perform better on other tasks.

3. Is CBOW a good word2vec configuration?
CBOW does not outperform SGNS on any task.

Recommendations

DON'T use shifted PPMI with SVD.

DON'T use SVD "correctly", i.e. without eigenvector weighting (performance drops 15 points compared to with eigenvalue weighting with $p = 0.5$).

DO use PPMI and SVD with short contexts (window size of 2).

DO use many negative samples with SGNS.

DO always use context distribution smoothing (raise unigram distribution to the power of $\alpha = 0.75$) for all methods.

DO use SGNS as a baseline (robust, fast and cheap to train).

DO try adding context vectors in SGNS and GloVe.

Conclusion

These results are in contrast to the general consensus that word embeddings are superior to traditional methods. Rather, they indicate that it typically makes no difference whatsoever whether word embeddings or distributional methods are used. What really matters is that your hyperparameters are tuned and that you utilize the appropriate pre-processing and post-processing steps.

Recent studies by Jurafsky's group [13], [14] reflect these findings and illustrate that SVD, rather than SGNS, is commonly the preferred choice accurate word representations is important.

We hope this overview of word embeddings has helped to highlight some fantastic research that sheds light on the relationship between traditional distributional semantic and in-vogue embedding models.

References

[1]: Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. The Journal of Machine Learning Research, 3, 1137–1155. http://doi.org/10.1162/153244303322533223

[2]: Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 1–12.

[3]: Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9.

[4]: Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. Proceedings of the 25th International Conference on Machine Learning – ICML '08, 20(1), 160–167. http://doi.org/10.1145/1390156.1390177

[5]: Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543. http://doi.org/10.3115/v1/D14-1162

[6]: Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-Aware Neural Language Models. AAAI. Retrieved from http://arxiv.org/abs/1508.06615

[7]: Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the Limits of Language Modeling. Retrieved from http://arxiv.org/abs/1602.02410

[8]: Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. Journal of Machine Learning Research, 12 (Aug), 2493–2537. Retrieved from http://arxiv.org/abs/1103.0398

[9]: Chen, W., Grangier, D., & Auli, M. (2015). Strategies for Training Large Vocabulary Neural Language Models, 12. Retrieved from http://arxiv.org/abs/1512.04906

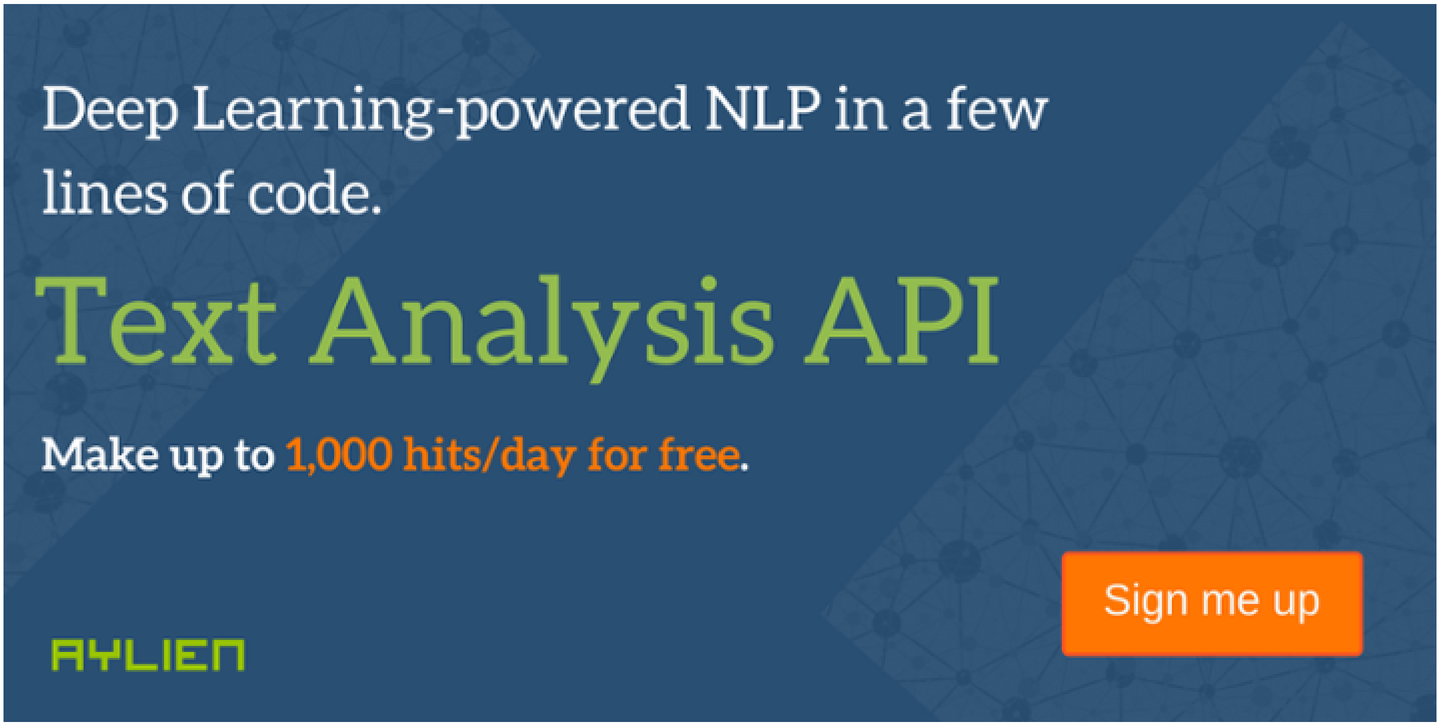
[10]: Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. Transactions of the Association for Computational Linguistics, 3, 211–225. Retrieved from https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/570

[11]: Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. ACL, 238–247. http://doi.org/10.3115/v1/P14-1023

[12]: Levy, O., & Goldberg, Y. (2014). Neural Word Embedding as Implicit Matrix Factorization. Advances in Neural Information Processing Systems (NIPS), 2177–2185. Retrieved from http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization

[13]: Hamilton, W. L., Clark, K., Leskovec, J., & Jurafsky, D. (2016). Inducing Domain-Specific Sentiment Lexicons from Unlabeled Corpora. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Retrieved from http://arxiv.org/abs/1606.02820

[14]: Hamilton, W. L., Leskovec, J., & Jurafsky, D. (2016). Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. arXiv Preprint arXiv:1605.09096.


Deep Learning-powered NLP in a few lines of code.
Text Analysis API
Make up to 1,000 hits/day for free.
Sign me up
AYLIEN

Tags: CBOW, GloVe, word embeddings, word2vec

Author

Sebastian Ruder

Research Scientist @ AYLIEN

Sebastian is a PhD student in Natural Language Processing at the Insight Research Centre for Data Analytics and a research scientist at AYLIEN. Previously, he worked with Microsoft, IBM Extreme Blue, and Google Summer of Code. His main research interest lies in using Deep Learning for domain adaptation in NLP. In his spare time, he loves to travel, read, blog, and learn new languages.

Twitter: @seb_ruder

YOU MIGHT ALSO LIKE THESE

Source Code Classification Using Deep Learnin...

Source Code Classification Using Deep Learnin...

Data Science, Research

Word Embeddings and Their Challenges

Word Embeddings and Their Challenges

Research

Modeling documents with Generative Adversaria...

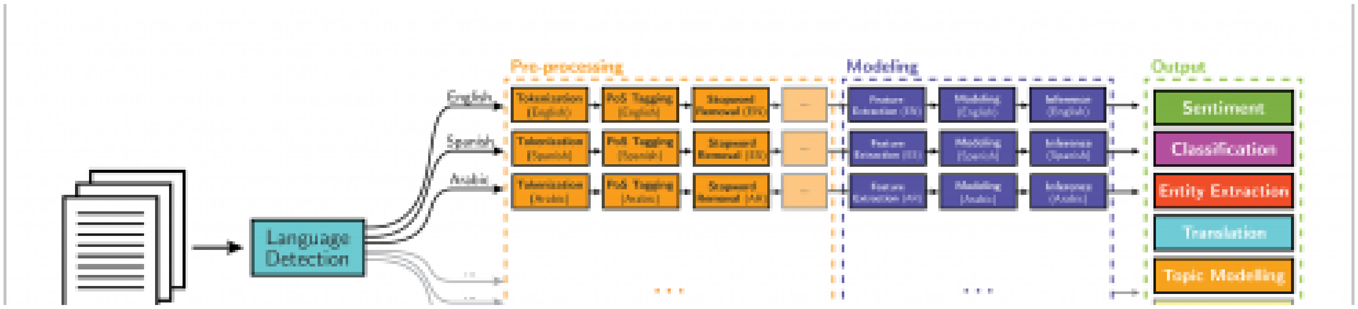
Modeling documents with Generative Adversaria...

Research

A Call for Research Collaboration at AYLIEN 2...

A Call for Research Collaboration at AYLIEN 2...

Research



Leveraging Deep Learning for Multilingual Sen...

Product, Research

A TensorFlow implementation of "A neural auto...

A TensorFlow implementation of "A neural auto...

Research

An introduction to Generative Adversarial Net...

An introduction to Generative Adversarial Net...

Research

Flappy Bird and Evolution Strategies: An Expe...

Flappy Bird and Evolution Strategies: An Expe...

Research

0 Comments **AYLIEN** 1 **Login**

♡ **Recommend** **Share** **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON **AYLIEN**

Text Analytics meets 2014 World Cup tweets – Part 1

1 comment • a year ago•

Nicolas Terral — Hello Mike,Can you explain me why you only have the tweets id in your tweets.csv ?

Building a Twitter Sentiment Analysis Process in RapidMiner

2 comments • a year ago•

Giulia Bosch Boschesi — Hi, before "Visualizing the results" the process make me a bug: "Duplicate attribute: Text".Can

Support Vector Machines for dummies; A Simple Explanation

1 comment • a year ago•

Matthew Simpson — The best description

Naive Bayes for Dummies; A Simple Explanation

3 comments • a year ago•

Jim F — This explanation makes no

About AYLIEN

AYLIEN is an AI, NLP & Machine Learning startup based in Dublin, Ireland. We provide Text Analysis & News APIs that allow users to make sense of human-generated content at scale. We provide a range of content analysis solutions to developers, data scientists, marketers and academics. Our core offerings include packages of Information Retrieval, Machine Learning, Natural Language Processing and Image Recognition APIs.

Recent Posts

Video Content in Online News – July Media Roundup
August 4, 2017

Your First Text Mining Project with Python in 3 steps
July 28, 2017

An Interactive History of Chatbots
July 21, 2017