Natural Language Processing

Lecture 8
Static neural word embeddings

2021

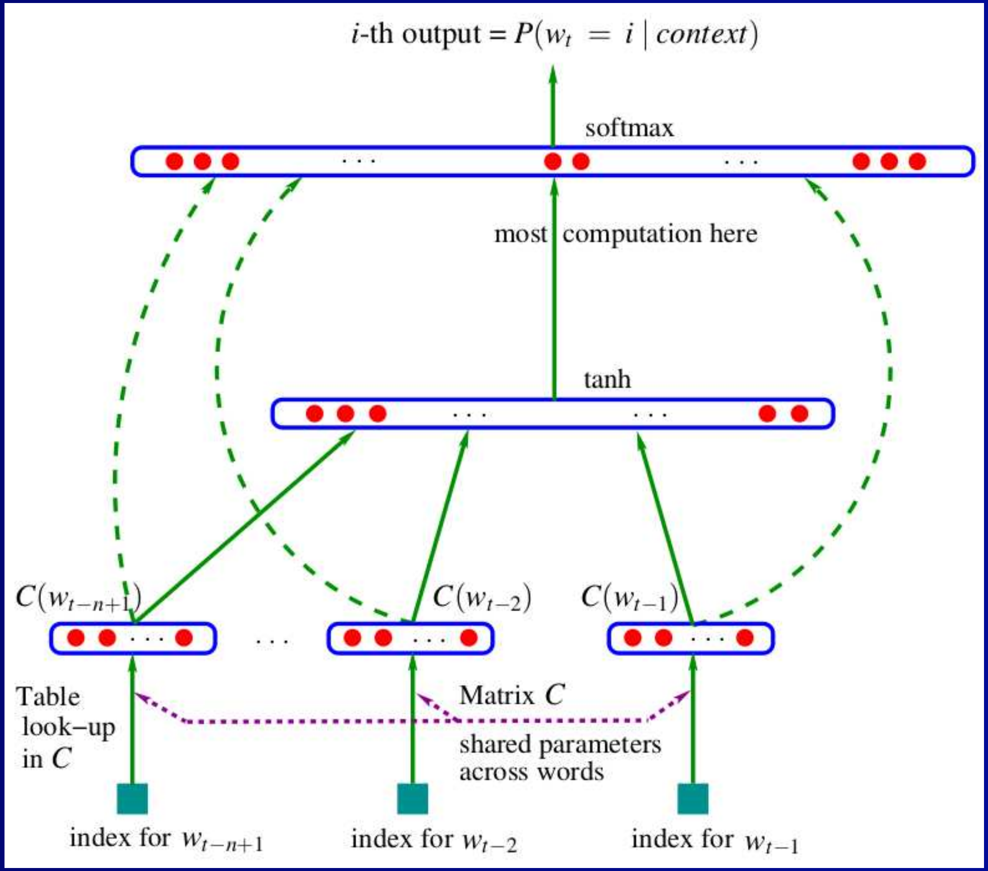
Word vectors and neural networks

The success of LSI and LSA showed that distribution based vector representations of words can be very useful for NLP tasks in general. In the case of neural network NLP models, continuous, dense word representations have been especially important, because they

- can be used as informative and economic representations instead of simply one-hot encoding words;
- can help decreasing the number of model parameters;
- can be learned by neural networks from text corpora in a self-supervised manner.

Word vectors and neural networks cont.

One of the first instances of word vectors learnt by a neural network can be found in the neural language model of Bengio et al. [2003]:



Word vectors and neural networks cont.

C is an embedding layer mapping word vocabulary indexes to vectors of real numbers:

$$C : [0, |V|) \rightarrow \mathbb{R}^d.$$

The d dimensionality of (static) word embeddings is typically within the 50–600 range.
Technically, embedding layers can be implemented in several ways, e.g., as a dense layer with one-hot encoded word indexes as input (in this case the word vectors are represented as the weight matrix of the layer), or as a look-up table represented as an array etc.

Word vectors and neural networks cont.

Important lessons regarding the embedding layer in Bengio et al. [2003]:

- Embeddings are *static*: tokens of the same type have identical embeddings regardless of their context.
- The model using the end-to-end trained word embedding layer performed better than traditional *n*-gram based language models.
- Using the first principal components of the word co-occurrence frequency matrix as word feature vectors instead of the trained embeddings did not have the same performance benefits.
- Learning word embeddings with neural nets is a viable way of scaling up the training corpus.

Word2vec

Distinguishing features

Word2vec, introduced by Mikolov et al. [2013a], is also a neural network family learning useful distributed representations of words from a corpus, but with several novel features:

- It is a dedicated architecture: *representation learning* is its sole goal.
- It is based on a new type of corpus-based, self-supervised predictive task.
- The architecture is kept intentionally very simple to make it possible to train on huge corpora with large vocabularies.

Skipgrams

Word2vec is based on *skipgrams*, which are a generalization of n-grams: while an n-gram is a *continuous*, n-long subsequence of a text, skipgrams can contain a certain amount of “jumps”: If the base sequence is $\langle w_1, \ldots, w_N \rangle$ then the set of n-long skipgrams with at most k total distance is

$$\{\langle w_{i_1}, \ldots, w_{i_n} \rangle \mid i_1, \ldots, i_n \in [1, N], \sum_{j=2}^{n} i_j - i_{j-1} \le k\}.$$

There can be additional restrictions, e.g. on the number and length of individual skips.

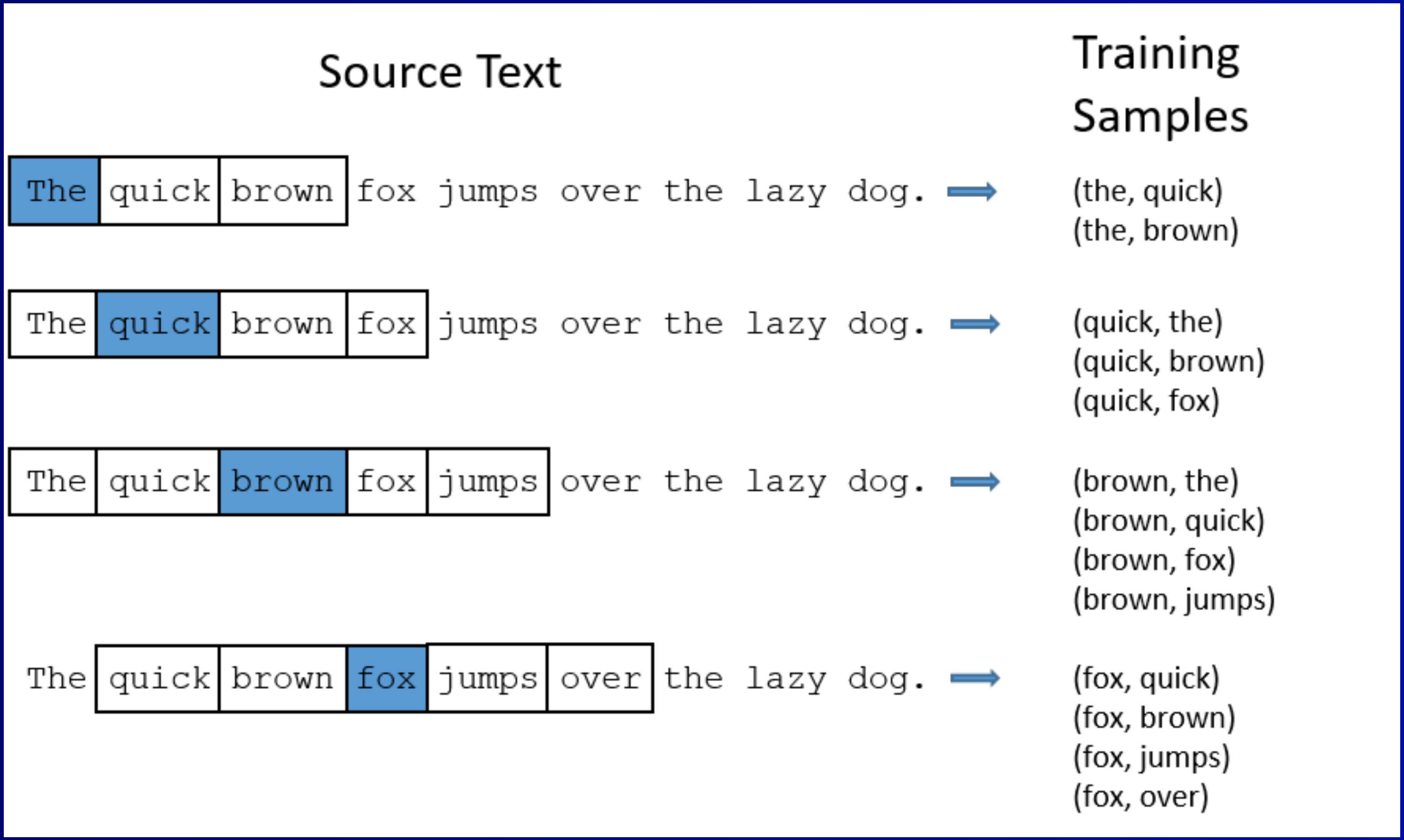
Word2vec tasks

The Word2vec task is specifically based on skipgrams of length $2c$ with a single one-word skip at the center. There are two task-variants with associated model architectures:

- *CBOW* (Continuous Bag of Words): predict the missing word at the center of a skipgram.
- *SkipGram*: predict the elements of the skipgram given the missing/skipped word. In contrast to the CBOW task, where each skipgram corresponds to exactly one classification example, the SkipGram task produces several $\langle$word-in-the-center word-from-the-skipgram$\rangle$ examples for each skipgram.

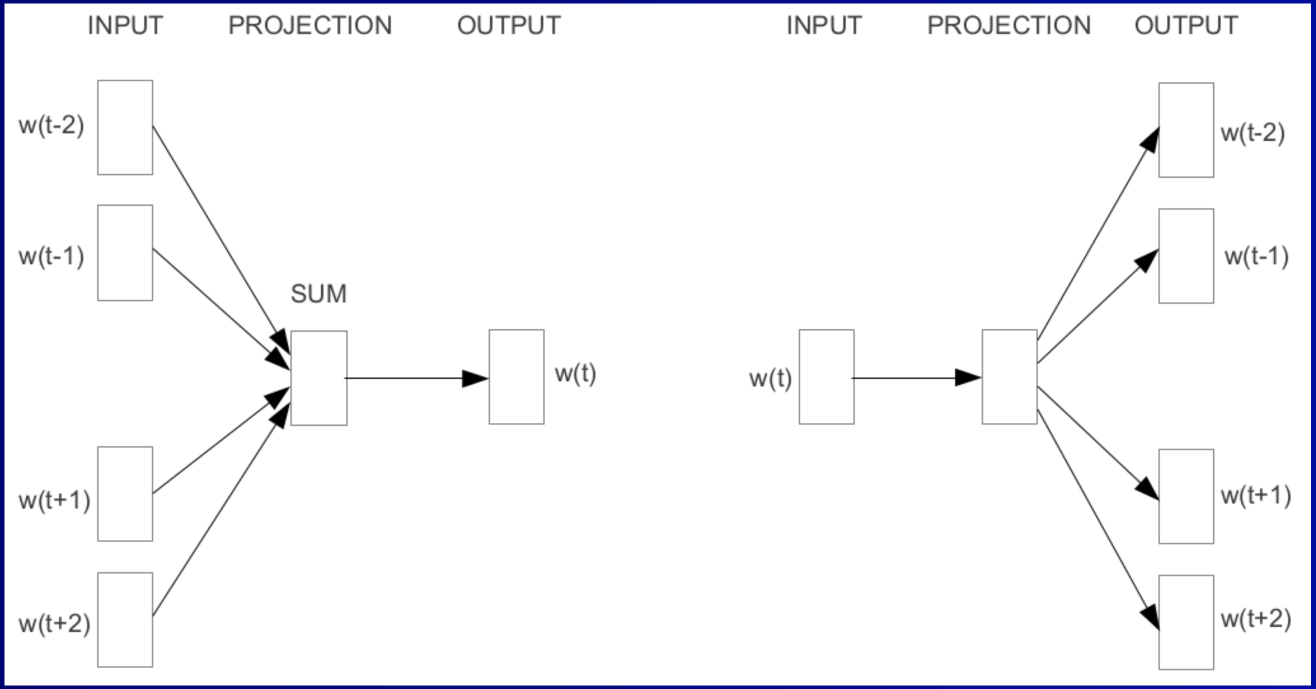
Word2vec tasks cont.

A simple illustration of the skipgram task:



(Figure from McCornick: Word2vec tutorial)

Architecture



While SkipGram simply applies the $E(\cdot)$ embedding mapping to its one-word input, CBOW embeds all words in the input skipgram and computes their sum.

Architecture cont.

After projecting the input into the word embedding space, both architectures use simply a linear projection with weight matrix $W \in \mathbb{R}^{|V|\times d}$ and a final softmax layer to produce prediction probabilities for all words in the vocabulary:

$$CBOW(\langle w_{t-c}, ..., w_{t+c}\rangle) = \mathrm{softmax}(W\sum_i E(w_i)),$$

$$SkipGram(w_t) = \mathrm{softmax}(WE(w_t)).$$

Both models are trained with standard negative log likelihood loss and SGD, but there are interesting differences regarding the sampling of examples.

Architecture cont.

It is worth noting that the $W \in \mathbb{R}^{|V|\times d}$ projection matrix can also be seen as an $E'(\cdot)$ embedding of the vocabulary into the same R^d space. With this notation, the logits (linear outputs) of the two models for a particular w_j word can be written simply as

$$CBOW_{linear}(\langle w_{t-c}, ..., w_{t+c}\rangle)[w_j] = \sum_i E(w_i) \cdot E'(w_j),$$

$$SkipGram_{linear}(w_t)[w_j] = E(w_t) \cdot E'(w_j)$$

As this formulation shows, minimizing the negative log likelihood training objective is a way of increasing the dot product of the embeddings of the input and that of the correct prediction.

Architecture cont.

Because of the symmetry shown by this formulation, it is an option to *tie the weights* of the two layers making $E(w) = E'(w)$ for all $w \in V$.
Although this is frequently done, it's also customary to keep them different and only use the vectors of the input embedding $E(\cdot)$ as the final result, or combine them, e.g., by taking their average.

Data point generation and sampling

For the CBOW variant, we simply slide the c-radius context window through the corpus and generate a

$$\langle\langle w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}\rangle, w_t\rangle$$

$\langle$input, correct output$\rangle$ data point at each step.

The process is more complicated for SkipGram, because at each step the actually used context window radius r is randomly chosen from the $[1, c]$ interval, and a $\langle w_t, w_i\rangle$ data point is generated for each $w_i \in \langle w_{t-r}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+r}\rangle$ word. The effect is that words closer to the target are sampled with a higher probability.

Avoiding full softmax

As computing full softmax for a $|V|$-long vector is expensive for large Vs, Word2vec implementations typically use cheaper output layer alternatives.

One solution is *hierarchical softmax*, which is based on a binary tree whose leaves are the vocabulary words. The linear outputs of the network correspond to internal nodes, and the probability assigned to a w word can be calculated by computing values of $\sigma(o)$ only for the o outputs that lie on the path to w. Using a balanced tree this trick reduces the softmax computation complexity during training from $\mathcal{O}(|V|)$ to $\mathcal{O}(\log|V|)$, and with clever tree structure this can be further decreased.

Hierarchical softmax

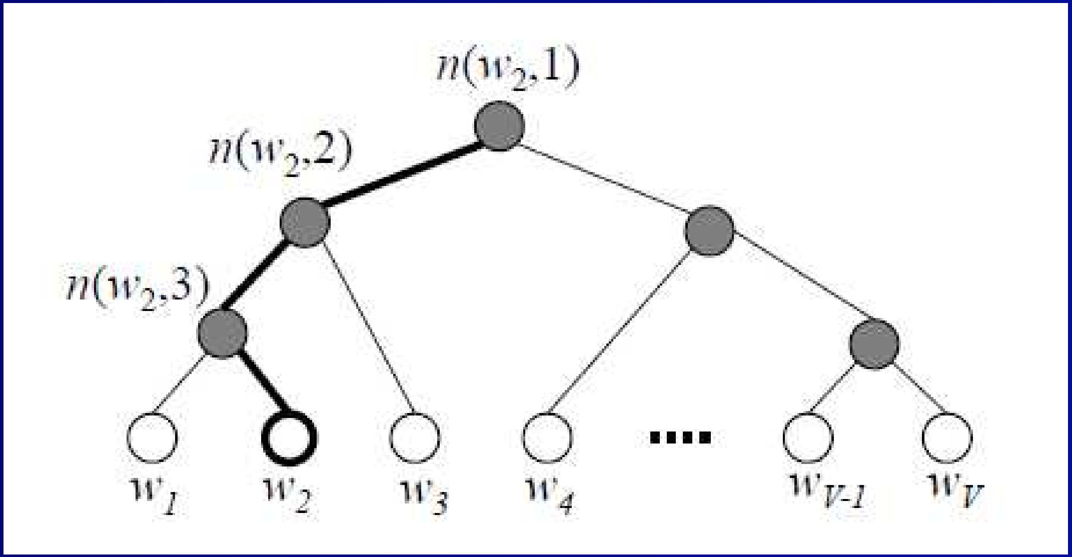


Illustration: if the linear outputs on the path are $o(w_2, 1), o(w_2, 2), o(w_2, 3)$ then the probability assigned to w_2 can be calculated as $(1 - \sigma(o(w_2, 1)))(1 - \sigma(o(w_2, 2)))\sigma(o(w_2, 3)) = \sigma(-o(w_2, 1))\sigma(-o(w_2, 2))\sigma(o(w_2, 3))$.

Negative sampling

The other alternative is *negative sampling*[1] This involves reformulating the SkipGram task as a binary classification problem.

- We consider earlier SkipGram $\langle$center word, context word$\rangle$ data points from the corpus as positive examples,
- and also generate a certain number of negative examples by sampling, for each center word, “fake context words” from a noise distribution representing the entire corpus.

[1] Mikolov et al. [2013b].

Negative sampling

The negative sampling trick makes it possible to simplify the network architecture to the point of

$$SGNS(w_t, w_c) = \sigma(E_t(w_t) \cdot E_c(w_c))$$

where $E_t(\cdot)$ is the target (center) word embedding while $E_c(\cdot)$ is the context word embedding. With k negative samples from a P_n noise distribution, the negative log likelihood loss for each real $\langle w_t, w_c \rangle$ data point will be

$$-[\log SGNS(w_t, w_c) + \sum_{\substack{i=1 \\ w_i \sim P_n}}^{k} \log(1 - SGNS(w_t, w_i))].$$

Word2vec as matrix factorization

After the success of Word2Vec, many studies investigated its relationship to counting based matrix factorization methods, and they turned out to be closely related: Levy and Goldberg [2014] showed that the SGNS objective is equivalent to factorizing the word co-occurrence based M matrix whose elements are

$$m_{ij} = \max(0, PMI(w_i, w_j) - \log k),$$

where $PMI(w_i, w_j)$ is the $\frac{P(w_i,w_j)}{P(w_i)P(w_j)} \approx \frac{C(w_i,w_j)}{C(w_i)C(w_j)}$ *pointwise mutual information* of w_i and w_j, and k is the number of negative samples.

GloVe

The GloVe algorithm

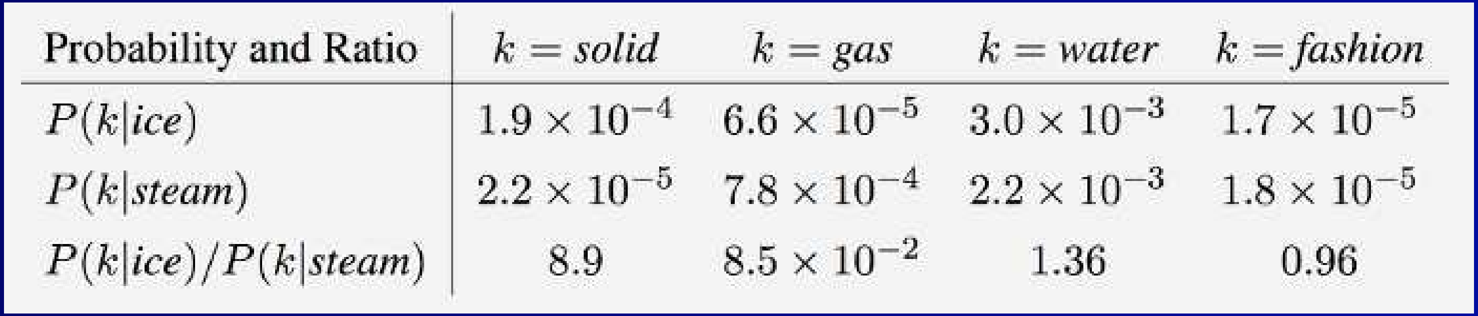
GloVe (Global Vectors)[2] is another algorithm for learning static word embeddings from a very large corpus. It is *not* a neural method, but discussed here because it was published (one year) after Word2vec as a reaction to it, and is one of its most important alternatives.

Similarly to LSA methods, GloVe is explicitly based on a low-rank factorization of a matrix based on word-occurrences in fixed-size context windows, but the matrix elements are actually the *logarithms* of word co-occurrences.

[2] Pennington et al. [2014].

The GloVe algorithm cont.

Focusing on the logarithm of co-occurrences is motivated by the observation that *ratios* of co-occurrence probabilities are very informative semantically:



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

The table shows that the ratios do a good job in distinguishing words that are relevant to the word pair *ice*, *steam*, i.e., *solid* and *gas* from noise words.

The GloVe algorithm cont.

Factorizing directly the co-occurrence log probability matrix would require for any w_i, w_j word pair

$$E_w(w_i) \cdot E_c(w_j) \approx \log(P(w_j \mid w_i)) \approx \log(C(w_i, w_j)) - \log(C(w_i)).$$

With $E_w(\cdot)$ word and $E_c(\cdot)$ context embeddings satisfying this requirement the $\log(P(w_k \mid w_i)/P(w_k \mid w_j))$ log probability ratio can be expressed simply as $(E_w(w_i) - E_w(w_j)) \cdot E_c(w_k)$, i.e., the semantically informative co-occurrence relationships correspond to simple geometric ones between the embeddings.

The GloVe algorithm cont.

Instead of trying to minimize the differences $E_w(w_i) \cdot E_c(w_j) + \log(C(w_i)) - \log(C(w_i, w_j))$ for $w_1, w_2 \in V$, GloVe minimizes the closely related

$$\sum_{i,j=1}^{|V|} f(C(w_i, w_j))(E_w(w_i) \cdot E_c(w_j) + b_w(w_i) + b_c(w_j) - \text{log}C(w_i, w_j))^2$$

objective. The differences are

- the $f(\cdot)$ weighting function downweighting rare co-occurrences,
- the learned b_w and b_c biases for each word, providing a symmetric replacement for $\log(C(w_i))$.

GloVe Training

Unlike Word2vec, which is trained with sliding context windows through the corpus, GloVe is trained in two steps:

1. assembling the global co-occurrence counts matrix;
2. optimizing the E_w, E_c, b_w, b_c parameters in the above objective by SGD, randomly sampling the elements of the co-occurrence matrix.

Compared to Word2vec, GloVe can have larger memory requirements because of working with the co-occurrence matrix (although it is sparse), but this can be compensated by the fact that optimization happens on the non-zero matrix elements afterwards, whose number is typically sublinear in corpus length.

Evaluation

Types of evaluation

How can the quality of word embeddings be measured? As learned representations, word vectors can be evaluated

- *intrinsically*, according to how well they correspond to human judgments about the semantic and morphological features of words, and
- *extrinsically*, according to how useful they are as components in solutions of downstream NLP tasks.

As for the intrinsic point of view, Word2vec with appropriately fine-tuned parameters and on a large, good quality corpus can produce embeddings whose geometrical characteristics are surprisingly close to human similarity judgments.

Intrinsic evaluation

The two most used intrinsic metrics are

- how well the *cosine similarity* of two representations, $\frac{E(w_1) \cdot E(w_2)}{\|E(w_1)\| \times \|E(w_2)\|}$, correlates with human-assigned similarity scores;
- how well similar *relationships* between words, e.g,. $king : queen$ and $man : woman$ correspond to similar geometric relationships between the vectors. It is typical to look at the *difference* between the vectors, i.e., whether $E(king) - E(queen) \approx E(man) - E(woman)$, or, in a slightly different formulation, whether the word whose embedding is the closest to $E(king) - E(queen) + E(woman)$ is man.

Intrinsic evaluation cont.

These intrinsic metrics can be nicely visualized using dimension reduction techniques such as t-SNE or UMAP, e.g.,

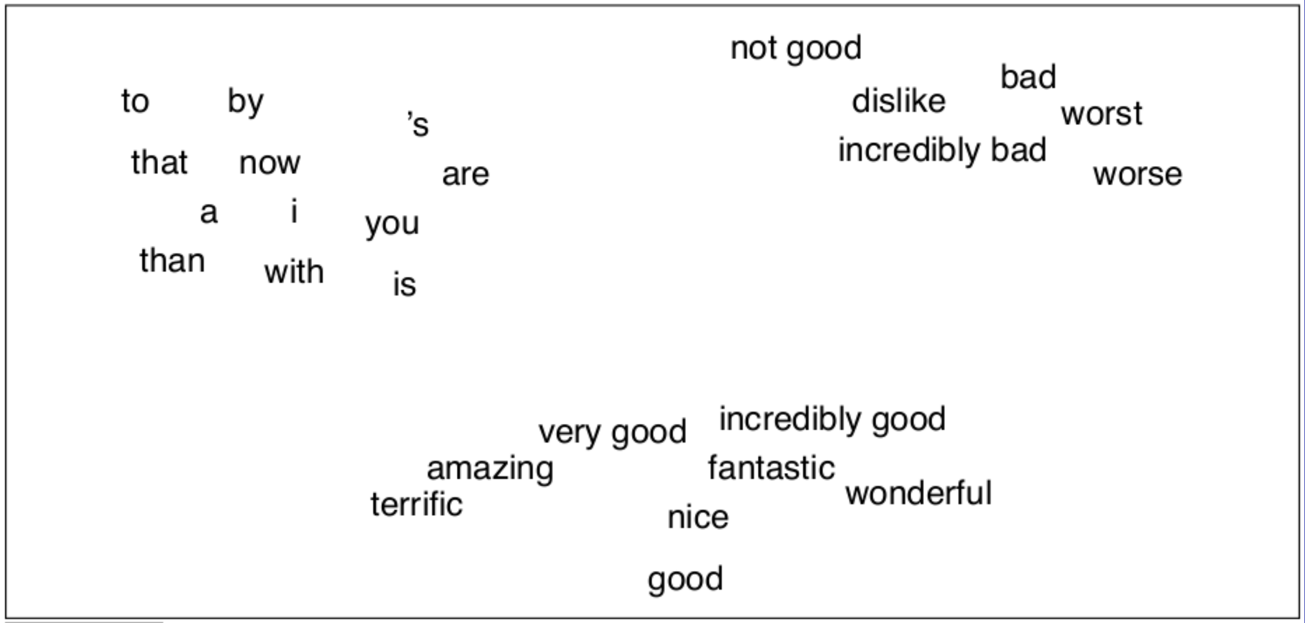


Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015).

(Figure from Jurafsky and Martin [2019, ch. 6].)

Intrinsic evaluation cont.

There are also purpose built *datasets* for intrinsic evaluation, e.g.,

- the WorsSim-353 dataset containing 353 English word pairs with semantic similarity scores between 1.0 and 10.0;
- BATS (The Bigger Analogy Test Set) containing 98000 analogy questions for testing the correspondence between word analogies and vector offsets.

Extrinsic evaluation

Extrinsic evaluation can be performed using any NLP task, but it is typical to use standard sequence labeling tasks such as named entity recognition.
Embeddings can be evaluated by switching between them in an embedding-based architecture while keeping everything else fixed.

There is an important difference between using the embeddings “as is”/“frozen” without changing them, or *fine tuning* them, i.e., training them on the task dataset together with the other parameters.

Evaluation results

According to the intrinsic evaluation results of Levy et al. [2015], there is no large difference in performance between Word2vec variants, GloVe and traditional SVD on certain co-occurrence matrixes. Most importantly, they found that

- hyperparameter tuning influences performance more than the choice of algorithm,
- SGNS proved to be a very strong baseline which did not “significantly underperform in any scenario”,
- the *sum* of the two learned embeddings (target and context) often performs significantly better than only one of them.

Utilizing internal word structure

The blackbox problem

The word embeddings we have discussed so far are based *exclusively* on distribution, the *internal structure* of words do not play any role. Consequently,

- out of vocabulary words, and
- words that are rare in the training corpus

do not have adequate embeddings, even if their internal morphological/character structure could provide ample information about their semantic and syntactic properties.

Apart from using *morpheme* embeddings, which requires a morphological analyser, a few self-supervised solutions also emerged.

fastText

The fastText algoritm [3] is based on SGNS, but adds n-grams $(3 \leq n \leq 6)$ to the vocabulary and models target words as the sum of the embeddings of all of its components. E.g., for the word *where* and $n = 3$ the components are `<wh`, `whe`, `her`, `ere`, `re>`, plus the whole word `<where>`. The SGNS architecture is modified to

$$\sigma\left(\sum_{w \in G(w_t)} E_t(w) \cdot E_c(w_c) \right)$$

where $G(w_t)$ is the set of all components of w_t.

[3] Bojanowski et al. [2017].

fastText cont.

On similarity tasks fastText vectors typically perform better than vanilla Word2vec, especially in case of morphologically rich languages.

An additional, important advantage is that with fastText it is possible to generate informative embeddings for unseen words by summing up the embeddings of their component n-grams.

Subword embeddings

A more radical solution to the blackbox problem is to switch to subword tokenization (e.g., using BPE) and generate embeddings only for the subwords in the vocabulary with one of the established algorithms.

PBEmb, for instance, uses GloVe to generate embeddings for the subwords resulting from BPE-based tokenization. Similarly to fastText, embeddings for OOV words can produced by combining the embeddings of component subwords (e.g., averaging them). Heinzerling and Strube [2017] found that while performing similarly, this type of solution requires a markedly smaller vocabulary than fastText.

References

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3: 1137–1155, 2003. URL `https://bit.ly/3rkKa4M`

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017

Benjamin Heinzerling and Michael Strube. Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages. *arXiv preprint arXiv:1710.02187*, 2017. URL `http://www.lrec-conf.org/proceedings/lrec2018/pdf/1049.pdf`

Daniel Jurafsky and James H Martin. Speech and language processing (3rd ed. draft). 2019. URL `https://web.stanford.edu/~jurafsky/slp3/`

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27:2177–2185, 2014. URL `https://www.aclweb.org/anthology/Q15-1016.pdf`

References cont.

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. URL `https://www.aclweb.org/anthology/Q15-1016.pdf`

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a. URL `https://arxiv.org/pdf/1301.3781.pdf`

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26:3111–3119, 2013b. URL `https://bit.ly/2WxL8MT`

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. URL `https://nlp.stanford.edu/pubs/glove.pdf`