Word Embeddings: A Survey

Felipe Almeida **Geraldo Xexéo***
Computer and Systems Engineering Program (PESC-COPPE)
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
{falmeida,xexeo}@cos.ufrj.br

Abstract

This work lists and describes the main recent strategies for building fixed-length, dense and distributed representations for words, based on the distributional hypothesis. These representations are now commonly called *word embeddings* and, in addition to encoding surprisingly good syntactic and semantic information, have been proven useful as extra features in many downstream NLP tasks.

1 Introduction

The task of representing words and documents is part and parcel of most, if not all, Natural Language Processing (NLP) tasks. In general, it has been found to be useful to represent them as vectors, which have an appealing, intuitive interpretation, can be the subject of useful operations (e.g. addition, subtraction, distance measures, etc) and lend themselves well to be used in many Machine Learning (ML) algorithms and strategies.

The Vector Space Model (VSM), generally attributed to Salton (1975) and stemming from the Information Retrieval (IR) community, is arguably the most successful and influential model to encode words and documents as vectors.

Another very important part of natural language-based solutions is, of course, the study of language models. A language model is a statistical model of language usage. It focuses mainly on predicting the next word given a number of previous words. This is very useful, for instance, in speech recognition software, where one needs to correctly decide what is the word said by the speaker, even when signal quality is poor or there is a lot of background noise.

These two seemingly independent fields have arguably been brought together by recent research on Neural Network Language Models (NNLMs), with Bengio et al. (2003)) having developed the first[1] large-scale language models based on neural nets.

Their idea was to reframe the problem as an unsupervised learning problem. A key feature of this solution is the way raw words vectors are first projected onto a so-called *embedding layer* before being fed into other layers of the network. Among other reasons, this was imagined to help ease the effect of the curse of dimensionality on language models, and help generalization (Bengio et al. (2003)).

With time, such **word embeddings** have emerged as a topic of research in and of themselves, with the realization that they can be used as standalone features in many NLP tasks (Turian et al. (2010)) and the fact that they encode surprisingly accurate syntactic and semantic word relationships (Mikolov et al. (2013a)).

More recently[2], other ways of creating embeddings have surfaced, which rely not on neural networks and embedding layers but on leveraging word-context matrices to arrive at vector representations for words. Among the most influential models we can cite the GloVe model (Pennington et al. (2014)).

These two types of model have something in common, namely their reliance on the assumption that words with similar contexts (other words) have the same meaning. This has been called the distributional hypothesis, and has been suggested some time ago by Harris (1954), among others.

This brings us to the definition of *word embeddings* we will use in this article, as suggested by the literature (for instance, Turian et al. (2010); Blacoe and Lapata (2012); Schnabel et al. (2015)),

* Geraldo Xexéo is also with the Mathematics Institute (IM-UFRJ), Federal University of Rio de Janeiro

[1] They claim this idea has been put forward before (Miikkulainen and Dyer (1991)), but not used at scale.

[2] Their roots, however, date back at least two decades, with the work of Deerwester et al. (1990).

according to which word embeddings are **dense, distributed, fixed-length word vectors, built using word co-occurrence statistics as per the distributional hypothesis**.

Embedding models derived from neural network language models have (Baroni et al. (2014)) been called *prediction-based* models, since they usually leverage language models, which predict the next word given its context. Other matrix-based models have been called *count-based* models, due to their taking into account global word-context co-occurrence counts to derive word embeddings. [3] These are described next.

This survey is structured as follows: in section 2 we describe the origins of statistical language modelling. In section 3 we give an overview of word embeddings, generated both by so-called prediction-based models and by count-based methods. In Section 4 we conclude and in Section 5 we provide some pointers to promising further research topics.

1.1 Motivation

To our knowledge, there is no comprehensive survey on word embeddings [4], let alone one that includes modern developments in this area. Furthermore, we think such a work is useful in the light of the usefulness of word embeddings in a variety of downstream NLP tasks (Turian et al. (2010)) and strikingly accurate semantic information encoded in such vectors (Mikolov et al. (2013a)).

1.2 Scope

We chose to include articles/strategies based on a mixture of citation count and reported impact on newer models.

2 Background: The Vector Space Model and Statistical Language Modelling

In order to understand the reasons behind the emergence and development of word embeddings, we think two topics are of utmost importance, namely the vector space model and statistical language modelling.

The vector space model is important inasmuch as it underpins a large part of work on NLP; it allows for the use of mature mathematical theory (such as linear algebra and statistics) to support our work. Additionally, vector representations are required for a wide range of machine learning algorithms and methods which are used to help address NLP tasks.

Modern research on word embeddings (particularly prediction-based models) has been, to some extent, borne out of attempts to make language modelling more efficient and more accurate. In fact, word embeddings (Bengio et al. (2003); Bengio and Senécal (2003); Mnih and Hinton (2007), to cite a few) have been treated as by-products of language models, and only after some time (arguably after Collobert and Weston (2008)) has the building of word embeddings been decoupled from the task of language models.

We give brief introductions to these two topics next.

2.1 The Vector Space Model

The first problem one encounters when trying to apply analytical methods to text data is probably that of how to represent it in a way that is amenable to operations such as similarity, composition, etc.

One of the earliest approaches to that end was suggested in the field of Information Retrieval (IR), with the work of Salton et al. (1975). They suggest an encoding procedure whereby each document in a collection is represented by a *t-dimensional* vector, each element representing a distinct term contained in that document. These elements may be binary or real numbers, optionally normalized using a weighting scheme such as TF-IDF, to account for the difference in information provided by each term.

With such a *vector space* in place, one can then proceed onto doing useful work on these vectors, such as calculating the similarity between document vectors (using even simple operations such as the inner-product between them), scoring search results (viewing the search terms as a pseudo document), etc.

Turney and Pantel (2010) provide a very thorough survey of different ways to leverage the VSM, while explaining the particular applications most suitable for them.

2.2 Statistical Language Modelling

Statistical language models are probabilistic models of the distribution of words in a language. For example, they can be used to calculate the likelihood of the next word given the words immedi-

[3] Note that a link between both types of models has been suggested by Levy and Goldberg (2014).

[4] There are, however, systematic studies on the performance of different weighting strategies and distance measures on word-context matrices, authored by Bullinaria and Levy (2007; 2012).

ately preceding it (its *context*). One of their earliest uses has been in the field of speech recognition (Bahl et al. (1983)), to aid in correctly recognizing words and phrases in sound signals that have been subjected to noise and/or faulty channels.

In the realm of textual data, such models are useful in a wide range of NLP tasks, as well as other related tasks, such as information retrieval.

While a full probabilistic model containing the likelihood of every word given all possible word contexts that may arise in a language is clearly intractable, it has been empirically observed that satisfactory results are obtained using a context size as small as 3 words (Goodman (2001)). A simple mathematical formulation of such an *n-gram model* with window size equal to T follows:

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t|w_1^{t-1}),$$

where w_t is the t-th word and w_i^T refers to the sequence of words from w_i to w_T, i.e. $(w_i, w_{i+1}, w_{i+2}...w_T)$. $P(w_t|w_1^{t-1})$ refers to the fraction of times w_t appears after the sequence w_1^{t-1}. Actual prediction of the next word given a context is done via maximum likelihood estimation (MLE), over all words in the vocabulary.

Some problems reported with these models have been (Bengio et al. (2003)) the high dimensionality involved in calculating discrete joint distributions of words with vocabulary sizes in the order of 100,000 words and difficulties related to generalizing the model to word sequences not present in the training set.

Early attempts of mitigating these effects, particularly those related to generalization to unseen phrases, include the use of smoothing, e.g. pretending every new sequence has count one, rather than zero in the training set (this is referred to as *add-one* or *Laplace smoothing*. Also, *backing off* to increasingly shorter contexts when longer contexts aren't available (Katz (1987)). Another strategy which reduces the number of calculations needed and helps with generalization is the clustering of words in so-called *classes* (cf. now famous Brown Clustering Brown et al. (1992)).

Finally, neural networks (Bengio et al. (2003); Bengio and Senécal (2003); Collobert and Weston (2008)) and log-linear models (Mnih and Hinton (2007); Mikolov et al. (2013b,c)) have also been used to train language models (giving rise to so-called *neural* language models), delivering better results, as measured by perplexity.

3 Word Embeddings

As mentioned before, word embeddings are fixed-length vector representations for words. There are multiple ways to obtain such representations, and this section will explore various different approaches to training word embeddings, detailing and they work and where they differ from each other.

Word embeddings are commonly (Baroni et al. (2014); Pennington et al. (2014); Li et al. (2015)) categorized into two types, depending upon the strategies used to induce them. Methods which leverage local data (e.g. a word's context) are called **prediction-based** models, and are generally reminiscent of neural language models. On the other hand, methods that use global information, generally corpus-wide statistics such as word counts and frequencies are called **count-based** models. We describe both types next.

3.1 Prediction-based Models

The history of the development of prediction-based models for embeddings is deeply linked with that of neural language models (NNLMs), because that is how they were initially produced. As mentioned before, a word's *embedding* is just the projection of the raw word vector into the first layer of such models, the so-called *embedding layer*.

The history of NNLMs, which started with the first large neural language model (Bengio et al. (2003)), is mostly one of gradual efficiency gains, occasional insights and trade-offs between complex models and simpler models, which can train on more data.

Much though early results (as measured by perplexity) clearly indicated that neural language models were indeed better at modelling language than their previous n-gram-based counterparts, long training times (sometimes upwards of days and weeks) are frequently cited among the major factors that hindered the development of such models.

Not long after the seminal paper by Bengio et al. (2003), many contributions were made towards increasing efficiency and performance of these models.

Bengio and Senècal (2003) identified that one of the main sources of computational cost was the

Article	Overview of Strategy	Architecture	Notes
Bengio et al. 2003	Embeddings are derived as a by-product of training a neural network language model.	Neural Net	Commonly referred to as the first neural network language model.
Bengio and Senecal 2003	Makes improvements on the previous paper, by using a Monte Carlo method to estimate gradients, bypassing the calculation of costly partition functions.	Neural Net	Decreased training times by a factor of 19 with respect to Bengio et al. 2003.
Morin and Bengio 2005	Full softmax prediction is replaced by a more efficient binary tree approach, where only binary decisions at each node leading to the target word are needed.	Neural Net, Hierarchical Softmax	Report a speed up with respect to Bengio and Senecal 2003 (over three times as fast during training and 100 times as fast during testing), but at a slightly lower score (perplexity).
Mnih and Hinton 2007	Among other models, the log-bilinear model is introduced here. Log-bilinear models are neural networks with a single, linear, hidden layer (Mnih and Hinton (2008)).	Log-linear Model	First appearance of the log-linear model, which is a simpler model, much faster and slightly outscores the model from Bengio et al. (2003).
Mnih and Hinton 2008	Authors train the log-bilinear model using hierarchical softmax, as suggested in Morin and Bengio (2005), but the word tree is learned rather than obtained from external sources.	Log-linear Model, Hierarchical Softmax	Reports being 200 times as fast as previous log-bilinear models.
Collobert and Weston 2008	A multi-task neural net is trained using not only unsupervised data but also supervised data such as SRL and POS annotations. The model jointly optimizes all of those tasks, but the target was only to learn embeddings.	Deep Neural Net, Negative Sampling	First time a model was built primarily to output just embeddings. Semi-supervised model (language model + NLP tasks).
Mikolov et al. 2013b	Introduces new two models, namely CBOW and SG. Both are log-linear models, using the two-step training procedure. CBOW predicts the target word given a context, SG predicts each context word given a target word.	Log-linear Model, Hierarchical Softmax	Trained on DistBelief, which is the precursor to TensorFlow (Abadi et al. (2015)).
Mikolov et al. 2013c	Improvements to CBOW and SG, including negative sampling instead of hierarchical softmax and subsampling of frequent words.	Log-linear Model, Negative Sampling	SGNS (skip-gram with negative sampling), the best performing variant of Word2Vec, was introduced here.
Bojanowski et al. 2016	Embeddings are trained at the n-gram level, in order to help generalization for unseen data, especially for languages where morphology plays an important role.	Log-linear Model, Hierarchical Softmax	Reports better results than SGNS. Embeddings are also reported to be good for composition (into sentence, document embeddings).

Table 1: Overview of strategies for building prediction-based models for embeddings.

partition function or *normalization factor* required by softmax output layers [5], such as those in neural network language models (NNLMs). Using a concept called *importance sampling* (Doucet (2001)), they managed to bypass calculation of the costly normalization factor, estimating instead gradients in the neural net using an auxiliary distribution (e.g. old n-gram language models) and sampling random examples from the vocabulary. They report gains of a factor of 19 in training time, with respect to the previous model, with similar scores (as measured by perplexity).

A little bit later, Morin and Bengio [6] (2005) have suggested yet another approach for speeding up training and testing times, using a Hierarchical Softmax layer. They realized that, if one arranged the output words in a hierarchical binary tree structure, one could use, as a proxy for calculating the full distribution for each word, the probability that, at each node leading to the word, the correct path is chosen. Since the height of a binary tree over a set V of words is $|V|/\log(|V|)$, this could yield exponential speedup. In practice, gains were less pronounced, but they still managed gains of a factor of 3 for training times and 100 for testing times, w.r.t. the model using importance sampling.

Mnih and Hinton (2007) were probably the first authors to suggest the Log-bilinear Model [7] (LBL),

[5] Softmax output layers are used when you train neural networks that need to predict multiple outputs, in this case the probability of each word in the vocabulary being the next word, given the context.

[6] To our knowledge, this is the first time the term *word embedding* was used in this context.

[7] These are special cases of *log-linear* models. See Ap-

which has been very influential in later works as well.

Another article by Mnih and Hinton (2008) can be seen as an extension of the LBL (Mnih and Hinton (2007)) model, using a slightly modified version of the hierarchical softmax scheme proposed by Morin and Bengio (2005), yielding a so-called Hierarchical Log-bilinear Model (HLBL). Whereas Morin and Bengio (2005) used a pre-built word tree from WordNet, Mnih and Hinton (2008) learned such a tree specifically for the task at hand. In addition to other minor optimizations, they reports large gains over previous LBL models (200 times as fast) and conclude that using purpose-built word trees was key to such results.

Somewhat parallel to the works just mentioned, Collobert and Weston (2008) approached the problem from a slightly different angle; they were the first to design model with the specific intent of learning embeddings only. In previous models, embeddings were just treated as an interesting by product of the main task (usually language models). In addition to this, they also introduced two improvements worth mentioning: they used words' full contexts (before and after) to predict the centre word [8]. Perhaps most importantly, they introduced a more clever way of leveraging unlabelled data for producing good embeddings: instead of training a language model (which is not the objective here), they expanded the dataset with *false* or *negative* examples [9] and simply trained a model that could tell positive (actually occurring) from false examples.[10]

Here we should mention two specific contributions by Mikolov et al. (2009; 2010), which have been used in later models. In the first work, (Mikolov et al. (2009)) a two-step method for bootstraping a NNLM was suggested, whereby a first model was trained using a single word as context. Then, the full model (with larger context) was trained, using as initial embeddings those found by the first step.

In (Mikolov et al. (2010)), the idea of using Recurrent Neural Networks (RNNs) to train language models is first suggested; the argument is that RNNs keep *state* in the hidden layers, helping the model remember arbitrarily long contexts, and one would not need to decide, beforehand, how many words to use as context in either side.

In 2012 Mnih and Teh have suggested further efficiency gains to the training of NNLMs. By leveraging Noise-contrastive Estimation (NCE).[11] NCE (Gutmann and Hyvärinen (2010)) is a way of estimating probability distributions by means of binary decisions over true/false examples.[12] This has enabled the authors to further reduce training times for NNLMs. In addition to faster training times, they also report better perplexity score w.r.t. previous neural language models.

It could be said that, in 2013, with Mikolov et al. (2013a; 2013b; 2013c) the NLP community have again (the main other example being Collobert and Weston (2008)) had its attention drawn to word embeddings as a topic worthy of research in and of itself. These authors analyzed the embeddings obtained with the training of a recurrent neural network model (Mikolov et al. (2010)) with an eye to finding possible syntactic regularities possibly encoded in the vectors.

Perhaps surprisingly, event for the authors themselves, they did find not only syntactic but also semantic regularities in the data. Many common relationships such as male-female, singular-plural, etc actually correspond to arithmetical operations one can perform on word vectors (see Figure 1 for an example).

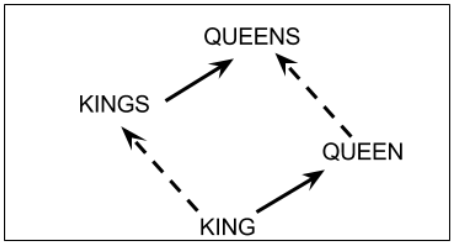

Figure 1: Projection of high dimensional word embeddings (obtained with an RNN language model) in 2D: high-level word embeddings encode multiple relationships between words; here shown: singular-plural (dotted line) and male-female (solid line) relationships. Adapted from Mikolov et al. (2013a).

pendix A for more information.

[8] Previous models focused on building language models, so they just used the left context.

[9] I.e. sequences of words with the actual centre word replaced by a random word from the vocabulary.

[10] This has been (Mikolov et al. (2013c)) called *negative sampling* and speeds up training because one can avoid costly operations such as calculating cross-entropies and softmax terms.

[11] Not to be confused with Contrastive Divergence (Hinton (2002)).

[12] This is somewhat similar to negative sampling, as applied by Collobert and Weston (2008). In fact, negative sampling can be seen as a simplified form of NCE, to be used in cases where you just want to train the model (i.e. obtain embeddings), rather than obtain the full probability distribution over the next word (Mikolov et al. (2013c))

A little later, in 2013b and 2013c, Mikolov et al. have introduced two models for learning embeddings, namely the continuous bag-of-words (**CBOW**) and skip-gram (**SG**) models. Both of these models are log-linear models (as seen in previous works) and use the two-step procedure (Mikolov et al. (2009)) for training. The main difference between CBOW and SG lies in the loss function used to update the model; while CBOW trains a model that aims to predict the centre word based upon its context, in SG the roles are reversed, and the centre word is, instead, used to predict each word appearing in its context.

The first versions of CBOW and SG (Mikolov et al. (2013b)) use hierarchical softmax layers, while the variants[13] suggested in Mikolov et al. (2013c) use negative sampling instead. Furthermore, the variants introduced subsampling of frequent words, to reduce the amount of noise due to overly frequent words and accelerate training. These variants were shown to perform better, with faster training times.

Among the most recent contributions to prediction-based models for word embeddings one can cite the two articles (Bojanowski et al. (2016) and Joulin et al. (2016)) usually cited as the sources of the *FastText*[14] toolkit, made available by Facebook, Inc. They have suggested an improvement over the skip-gram model from Mikolov et al. (2013c), whereby one learns not word embeddings, but *n-gram* embeddings (which can be composed to form words). The rationale behind this decision lies in the fact that languages that rely heavily on morphology and compositional word-building (such as Turkish, Finnish and other highly inflexional languages) have some information encoded in the word parts themselves, which can be used to help generalize to unseen words. They report better results w.r.t. SGNS (skip-gram variant with negative sampling) (Mikolov et al. (2013c)), particularly in languages such as German, French and Spanish.

A structured comparison of prediction-based models for building word embeddings can be seen on Table 1.

3.2 Count-based Models

As mentioned before, count-based models are another way of producing word embeddings, not by training algorithms that predict the next word given its context (as is the case in language modelling) but by leveraging word-context co-occurrence counts globally in a corpus. These are very often represented (Turney and Pantel (2010)) as word-context matrices.

The earliest relevant example of leveraging word-context matrices to produce word embeddings is, of course, *Latent Semantic Analysis* (LSA) (Deerwester et al. (1990)) where SVD is applied to a term-document[15] matrix. This solution was initially envisioned to help with information retrieval. While one is probably more interested in document vectors in IR, it's also possible to obtain word vectors this way; one just needs to look at the rows (rather than columns) of the factorized matrix.

A little later, Lund and Burgess (1996) have introduced the *Hyperspace Analogue to Language* (HAL). Their strategy can be described as follows: for each word in the vocabulary, analyze all *contexts* it appears in and calculate the co-occurrence count between the target word and each context word, inversely proportional to the distance from the context word to the target word. The authors report good results (as measured by analogy tasks), with an optimal context window size of 8.

The original HAL model did not apply any normalization to word co-occurrence counts found. Therefore, very common words like *the* contribute disproportionately to all words that co-occur with them. Rohde et al. (2006) have found this to be a problem, and introduced the *COALS* method, introducing normalization strategies to factor out such frequency differences in words. Instead of using raw counts, they suggest it's better to consider the *conditional* co-occurrence, i.e. how much more more likely a word a is to co-occur with word b than it is to co-occur with a random word from the vocabulary. They report better results than previous methods, using the SVD-factorized variant[16].

A somewhat different alternative was proposed by Dhillon et al. (2011), in which they introduce the *Low Rank Multi-View Learning* (LR-MVL) method. In short, it's an iterative algorithm where embeddings are derived by leveraging Canonical Correlation Analysis (CCA) (Hotelling (1935))

[13] These have been published under the popular *Word2Vec* toolkit (https://code.google.com/archive/p/word2vec/).

[14] https://research.fb.com/projects/fasttext/

[15] Term-document matrices are a subset of word-context matrices Turney and Pantel (2010).

[16] I.e., factorizing the co-occurrence matrix in order to reduce dimensions and improve results.

Article	Overview of Strategy	Notes
Deerwester et al. 1990	LSA is introduced. Singular value decomposition (SVD) is applied on a term-document matrix.	Used mostly for IR, but can be used to build word embeddings.
Lund and Burgess 1996	The HAL method is introduced. Scan the whole corpus one word at a time, with a context window around the word to collect weighted word-word co-occurrence counts, building a word-word co-occurrence matrix.	Reported an optimal context size of 8.
Rohde et al. 2006	Authors introduce the COALS method, which is an improved version of HAL, using normalization procedures to stop very common terms from overly affecting co-occurrence counts.	Optimal variant used SVD factorization. Reports gains over HAL (Lund and Burgess (1996)), LSA (Deerwester et al. (1990)) and other methods.
Dhillon et al. 2011	LR-MVL is introduced. Uses CCA (Canonical Correlation Analysis) between left and right contexts to induce word embeddings.	Reports gains over C&W embeddings (Collobert and Weston (2008)), HLBL (Mnih and Hinton (2008)) and other methods, over many NLP tasks.
Lebret and Collobert 2013	Applied a modified version of Principal Component Analysis (called Hellinger PCA) to the word-context matrix.	Embeddings can be *tuned* before being used in actual NLP tasks. Also reports gains over C&W embeddings, HLBL and other methods, over many NLP tasks.
Pennington et al. 2014	Introduced *GloVe*, a log-linear model trained to encode semantic relationships between words as vector offsets in the learned vector space, using the insight that co-occurrence ratios, rather than raw counts, are the actual conveyors of word meaning.	Reports gains over all previous count-based models and also SGNS (Mikolov et al. (2013c)), in multiple NLP tasks.

Table 2: Overview of strategies for building count-based models for embeddings.

between the left and right contexts of a given word. One interesting feature of this model is that when embeddings are used for downstream NLP tasks, they are concatenated with embeddings for their context words too, yielding better results. Authors report gains over other matrix factorization methods, as well as neural embeddings, over many NLP tasks.

Lebret and Collobert (2013) have also contributed to count-based models by suggesting that a Hellinger PCA[17] transformation be applied to the word-context matrix instead. Results are reported to be better than previous count-based models such as LR-MVL and neural embeddings, such as those by Collobert and Weston (2008) and HLBL Mnih and Hinton (2008).

The last model we will cover in this section is the well-known *GloVe*[18] by Pennington et al. (2014). This model starts at the insight that *ratios* of co-occurrences, rather than raw counts, encode actual semantic information about pair of words. This relationship is used to derive a suitable loss function for a log-linear model, which is then trained to maximize the similarity of every word pair, as measured by the ratios of co-occurrences mentioned earlier. Authors report better results than other count-based models, as well as prediction based models such as SGNS (Mikolov et al. (2013c)), in tasks such as word analogy and NER (named entity recognition).

A structured comparison of count-based models for building word embeddings can be seen on Table 2.

4 Conclusion

Word embeddings have been found to be very useful for many NLP tasks, including but not limited to Chunking (Turian et al. (2010)), Question Answering (Tellex et al. (2003)), Parsing and Sentiment Analysis (Socher et al. (2011)).

We have here outlined some of the main works and approaches used so far to derive these embeddings, both using *prediction-based* models, which model the probability of the next word given a sequence of words (as is the case with language models) and *count-based* models, which leverage global co-occurrence statistics in word-context matrices.

Many of the suggested advances seen in the literature have been incorporated in widely used toolkits, such as *Word2Vec*, *gensim*[19], *FastText*, and *GloVe*, resulting in ever more accurate and

[17] This amounts to minimizing the distance between principal components and actual data, but using the Hellinger distance instead of the more common Euclidean distance.

[18] https://nlp.stanford.edu/projects/glove/

[19] https://radimrehurek.com/gensim/

faster word embeddings, ready to be used in NLP tasks.

5 Further Work

Research on the topic of word representations (and word embeddings in particular) is still active; among the most promising research directions we consider:

5.1 Adapting embeddings for task-specific work

Works such as Maas et al. (2011), Labutov and Lipson (2013) and Lebret and Collobert (2013) have highlighted improved results for NLP tasks when embeddings are *tuned* for specific tasks.

5.2 The link between prediction-based and count-based models

For example, Levy and Goldberg (2014) have suggested that the SGNS model (Mikolov et al. (2013c)) actually is equivalent to using a slightly modified word-context matrix, weighted using PMI (pointwise mutual information) statistics. Insight on what links the two models may yield more advances in both areas.

5.3 Composing word embeddings for higher-level entities

While research on how to compose word vectors to represent higher-level entities such as sentences and documents is not altogether new (generally under the name of *distributional compositionality*), recent works have adapted solutions specifically for neural word embeddings: we can cite here *Paragraph2Vec* (Le and Mikolov (2014)), *Skip-Thought Vectors* by Kiros et al. (2015) and also *FastText* itself (Joulin et al. (2016) and Bojanowski et al. (2016)).

A Log-linear Models and Neural Embeddings

Log-linear models are probabilistic devices which can be used to model conditional probabilities, much like those between word contexts and target words, these being the fundamental parts of language models.

Log linear models subscribe to the following template (Collins) for each output unit:

$$P(y \mid x; v) = \frac{exp(v \cdot f(x, y))}{\sum_{y' \in Y} exp(v \cdot f(x, y'))}$$

As applied to the language modelling task, with neural embeddings: y represents the label, i.e., a target word. x represents a word context, i.e. the words before or around the target word we want to predict. v is a learned parameter, i.e. a single row vector in the shared weight matrix.

It's possible to view the formulation above as a neural network with a single, *linear* [20] hidden layer, linked to a softmax output layer. Furthermore, akin to any neural-network model, this also can be trained with gradient-based methods, be extended to include regularization terms, and so on.

References

Martín Abadi, Ashish Agarwal, Paul Barham, et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Available at http://tensorflow.org/. Software available from tensorflow.org.

L. R. Bahl, F. Jelinek, and R. L. Mercer, March 1983. A maximum likelihood approach to continuous speech recognition.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. June 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Yoshua Bengio and Jean-Sébastien Senécal, 2003. Quick training of probabilistic neural nets by importance sampling.

Yoshua Bengio, Jean Ducharme, Pascal Vincent, and Christian Janvin, March 2003. A neural probabilistic language model. Available at http://dl.acm.org/citation.cfm?id=944919.944966.

William Blacoe and Mirella Lapata. July 2012. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, 2016. Enriching word vec-

[20] The exponential function is only used to force the values to be positive, and can be removed without loss of generality.