Context and Embeddings in Language Modelling – an Exploration

© Matthias Nitsche © Marina Tropmann-Frick
Hamburg University of Applied Sciences, Department of Computer Science, Hamburg, Germany
matthias.nitsche@haw-hamburg.de marina.tropmann-frick@haw-hamburg.de

Abstract. Embeddings are a natural way to map text to a latent space, commonly consumed in downstream language tasks, e.g., question- answering, named-entity recognition or neural machine translations. Embeddings typically capture syntactical relations between parts of a sequence and solve semantic problems connected with word-sense-disambiguation (WSD) well. As a result of *WSD*, the curse of dimensionality, out-of-vocabulary words, overfitting to a domain and missing real world knowledge, infering meaning without context is hard. Thus we require two things. First, we need techniques to actively overcome syntactical problems dealing with *WSD* and semantically correlating words/sentences. Second, we require context to reconstruct the intentions and settings of a given text such that it can be understood. This work explores different embedding models, data augmentation techniques and context selection strategies (sub-sampling on the input space) for real world language problems.

1 Introduction

Many NLP applications start with preprocessing pipelines involving stemming, stopword removal, special character extraction and tokenization. When the morphological treatment of text is done the most important step is the representation of text: projection. Language is a high dimensional and multi-sense problem domain dealing with polysemy, synonymy, antonymy and hyponymy. Therefore, we often need to reduce the dimensions of the problem domain, projecting it to a latent space. Classical models project words using WordNet mapping each word to a relation, employ methods from linear algebra like Singular Value Decomposition (SVD) and most famously Latent Semantic Indexing (LSI) [1]. More complicated statistical models involve expectation maximization procedures for which Latent Dirichlet Allocation (LDA) [2] is the standard. Word and subword-level embeddings try to overcome some of the limitations of the former methods using neural networks posing language models as an optimization problem. Word2Vec by [3] was the first successful model that superseded the quality of preceding methods. Embeddings map words, sentences, characters or part of words to a non-linear latent space in $\mathbb{R}^{\boldsymbol{l}}$ where $\boldsymbol{l}$ stands for the amount of dimensions the embedding has. Projects like *fastText*, *spaCy*, *Starspace*, *GloVe* and *Word2Vec Googles News embeddings* offer pretrained language models on vast amounts of data. There are multiple ways to choose a context for embeddings: by window of size $\boldsymbol{k}$ around a center word, by dependency tree around a word or by representing words as probability distributions and discarding unlikely words. [4] generalize context embeddings to models of the exponential family (*ef-emb*). [5] enhance *ef-emb* by creating a very complex selection procedure based on an amortization network and variational inference to drop unimportant items from the context with an indicator vector. In theory context selection works with two functions. The first is a function for selecting what a viable context is, e.g., $\boldsymbol{c} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{X})$, where $\boldsymbol{x}$ is the target item/ center word, $\boldsymbol{X}$ all items/vocabulary. The second for subsampling on the target and context $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})$. The origins of neural language models are based on [6] pro-posing a shallow single layer neural network with a softmax layer. The neural language model computes a conditional probability distribution over words – producing embeddings based on the $\boldsymbol{n}$ preceding words, represented as a vector of dimensions $\boldsymbol{d}$, shared across the entire network in the respective context vectors $\boldsymbol{C}$. The most basic language model computes the conditional probabilities given a word $\boldsymbol{w_t}$ and the preceding words using the chain rule. When the vocabulary grows large the normalization term in the denominator of the softmax becomes more difficult to handle. The model in [6] is intractable and could not be successfully build. The first model that successfully beat state-of-the-art language models was Word2Vec by [3]. Later we will review word and subword-level embeddings.

2 Word Embeddings

At first we briefly review word-level embeddings. Corpora typically consist of words that are part of sentences in documents. Before training, each sentence is tokenized and morphologically altered with stemming or lemmatization. Classical models use the bag of words model, so words are represented as a co-occurrence feature matrix. We start with Word2Vec, since almost every model leveraging embeddings in language take it as a point of reference.

Proceedings of the XX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2018), Moscow, Russia, October 9-12, 2018

2.1 Word2Vec

[3] improved on several aspects of Bengio's model by using the skip-gram window function (an alternative would be CBOW) and a tractable approximation of the softmax called negative sampling/hierarchical softmax.
Word2Vec has become the de facto standard in a lot of language downstream tasks. Google shipped pre-trained Word2Vec skip-gram models on Google News articles for everybody to use. The corpus is large (up to a billion words) and the dimensions of the latent space is large $d = 300$. The training would take weeks up to months on a just a few state-of-the-art GPUs, saving each researcher the time to train them themselves. We will see a great influx of pre-trained language models in the future because OOV words are a real issue and generalization on small sparse domains is highly problematic. While most of the premises of pre-trained models are great, they also introduce biases. [7] have shown that this particular dataset employs gender biases.
Skip-gram predicts the context of a center word w_i over a window c such that $w_{i-c}, \dots, w_i, \dots, w_{i+c}$ is satisfied.

$$\frac{1}{T}\sum_{t-1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t)$$

CBOW does the opposite, given a word context $w_{i-c}, \dots, w_i, \dots, w_{i+c}$ predict the center word w_i that is most likely. Negative sampling speeds up the performance by using the positive samples of the context words $2 * c$ and uses only a few negative samples that are not in its context. The respective objective cost function is

$$\log\sigma(\mathrm{v'}_{w_O}^{T} v_{w_I}) + \sum_{t=1}^{k} \mathbb{E}_{w_i} \sim P_n(w)[\log\sigma(-\mathrm{v'}_{w_O}^{T} v_{w_I})]$$

where σ is the sigmoid function, a binary function, drawing k samples from the negative or noise distribution $P_n(w)$, to distinguish the negative draws $\mathbb{E}_{w_i}$ from the target word w_O drawn from the context of w_I.
The objective of negative sampling is to learn high quality word embeddings by comparing noise (out of context) to words from the context. Another language model building upon Word2Vec is Global Vectors for Word Representation (*GloVe*) by [10], which is trained on an aggregated global word co-occurrence matrix from a corpus. The difference to Word2Vec is that the global statistics are taken into account contrary to Word2Vec, that works on local context windows alone. GloVe typically performs better than Word2Vec skip-gram, especially when the vocabulary is large. *GloVe* is also available on different corpora such as Twitter, Common Crawl or Wikipedia.

2.2 Bag of Tricks - fastText

Another interesting and popular word embedding model is *fastText* by [11]. It bases on a similar idea as Word2Vec. Instead of negative sampling - using the hierarchical softmax, and instead of words - using n-gram features. N-grams build on bag of words, commonly known as a co-occurrence matrix $\boldsymbol{D} \times \boldsymbol{V}$ where documents $\boldsymbol{D}$ are rows and the whole vocabulary $\boldsymbol{V}$ the features assuming i.i.d word order. Given a sequence of words $[\boldsymbol{w_1}, \dots, \boldsymbol{w_k}]$ n-grams take slices of $\boldsymbol{n}$, e.g., $[[\boldsymbol{w_1}, \dots, \boldsymbol{w_n}]_1, \dots, [\boldsymbol{w_{i+1}}, \dots, \boldsymbol{w_{n+1}}]_k]$. *fastText* comes in two flavours: character-level and word-level n-grams. We will review the character-level n-grams later.

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n))$$

The corresponding cost function where $\boldsymbol{f}$ is the hierarchical softmax function, $\boldsymbol{x_n}$ is a document with bag of n-gram feature vectors, $\boldsymbol{A}$ and $\boldsymbol{B}$ are weight matrices and $\boldsymbol{y_n}$ the label given a classification task. The label $\boldsymbol{y}$ in this case is the word. The unsupervised learning task is hierarchical softmax with *CBOW* denoted as $\boldsymbol{f}$ and has the following form:

$$P(y = j|C) = \frac{exp(\beta_j^T \cdot C)}{\sum_{t=1}^{|V|} exp(\beta_t^T \cdot C)}$$

As can be seen instead of finding the surrounding context of a word w we try to find the most probable word given the context C. What is novel about this approach is using n-gram features instead of windows speeding up the training, while still matching state-of-the-art results. *fastText* training time on a sentiment analysis task was 10 seconds compared to the shortest running model of 2-3 hours up to several days. As we see later, this model can be largely improved with character n-grams proposed in [12] and [13].

2.3 CoVe

So far we have investigated shallow neural networks with single layers and therefore only one non-linearity. [14] have found that training an attentional sequence-to-sequence model normally used for neural machine translations helps at enriching word vectors not just on the word-level hierarchy. By training a two-layer, bidirectional long short-term memory [15], on a source language (English) to a target (German), they achieve state-of-the-art performance. All sequences of words $\boldsymbol{w^x}$ are pre-initialized with GloVe($\boldsymbol{w^x}$) where words become sequences of vectors.

$$\begin{gathered} w^x = [w_1^x, \dots, w_n^x] \\ \mathit{GloVe}(w^x) = [\mathit{GloVe}(w_1^x), \dots, \mathit{GloVe}(w_n^x)] \\ w^z = [w_1^z, \dots, w_n^z] \end{gathered}$$

where w^x is a sentence in the source language and w^z a sentence of the target language maximizing the likelyhood of an encoder MT-LSTM h, a decoder LSTM h_t^{dec}

$$\begin{gathered} h = \mathit{MT\text{-}LSTM}(\mathit{GloVe}(w^x)) \\ h_t^{dec} = \mathit{LSTM}([z_{t-1}; \widetilde{h}_{t-1}], h_{t-1}^{dec}) \\ \alpha_t = \mathit{softmax}(H(W_1 h_t^{dec} + b_1)) \\ \widetilde{h}_t = [\tanh(W_2 H^T \alpha_t + b_2); h_t^{dec}] \\ y = \mathit{softmax}(W_{out}\widetilde{h}_t + b_{out}) \end{gathered}$$

The softmax attention α_t over the decoder $\boldsymbol{h_t^{dec}}$ represents the relevance of each step from the encoder h. $\widetilde{h}_t$ then is a hidden state where the softmax and $\boldsymbol{h_t^{dec}}$ are concatenated, possibly to attend to the relevant parts, while not forgetting what was learned during the

decoding. Intuitively we are training a machine translation model where the only interesting part are the learned context vectors for sequences of the MT-LSTM. It was shown that the model performs better, when concatenating GloVe and CoVe into one single vector. The idea behind this is that we can transfer the higher level features learned in sequence-to-sequence tasks to standard downstream tasks like classification. By first using GloVe on the word-level and then the MT-LSTM, we are creating layers of abstractions. Essentially this is a first step towards transfer learning, which is standard practice in computer vision tasks with pre-trained CNNs. The top achiever is a model called Char + CoVe-L with a large *CoVe* model concatenated with a *n-gram* character features model.

2.4 Bias and Critique

Currently is a time producing a lot of different models based on experimentation and educated guesses. It is usually left to the reader trying to find explanations in embeddings for language. What does a word-level embedding like Word2Vec actually represent? While there is still a lot of ground to cover, recent papers focus a little more on the whys instead of the hows. Before going into details about subword embeddings and selection procedures let us discuss some of the problems, challenges and critique gaining a little more insight on why embeddings actually work. Most of the state-of-the-art models evaluate word embeddings with intrinsic evaluations. Intrinsic evaluation is usually qualitative, given a set of semantic word analogy pairs, test if the model connects them correctly. $\overrightarrow{man} - \overrightarrow{woman} \approx \overrightarrow{king} - \overrightarrow{queen}$. The woman/queen vs. man/king is the most famous of all examples. One could deduct that given a large number of such analogy word pairs, testing the presence of synonymy, polysemy and word positioning is sufficient. Intrinsic evaluation shows exactly what works, not what does not work or even what works but should not. Extrinsically it is not possible to use labels testing the precision and recall of our system. And it is easy to see why: What would you expect should a general approximation of a word should look like? Should it be able to learn every possible dimension and therefore interpretation of what we perceive of it? If so, how should it learn to distinguish different domains with a different context? The context of a domain is never explained or given to the models.

Given a reasonable amount of test cases, quality can be ensured to some extent. How good or bad they actually perform is usually tested in downstream language tasks. If the embeddings perform better on that specific task compared to a preceding model, it is declared state-of-the-art. Interestingly, [7] show that even state of the art embeddings display a large amount of bias towards certain topics: $\overrightarrow{man} - \overrightarrow{woman} \approx \overrightarrow{programmer} - \overrightarrow{homemaker}$.

Training real language models on real data yields real bias. The world and its written words are not fair and they incorporate really narrow views and concepts. Gender inequality and racism are two of the most challenging societal problems in the 21st century. Learning embeddings always yields a representation of the input. The bias is statistically significant. The problem is more obvious when considering that the standard Word2Vec model trained on the Googles News Corpus is applied on thousands of downstream language tasks. These kind of biases are not unique to language modelling and can be found in computer vision as well. [7] hints that there are three forms of bias: occupational stereotypes, analogies with stereotypes and indirect gender bias. They also acknowledge that not everything we perceive as bias should be seen as such, e.g. *football* and *footballer* is male dominant for other reasons than just bias. To debias embeddings the answer is quite clear: we need additional knowledge in form of gender specific word lists. [7] suggests to create a reference model g with word vectors that are gender biased words.

While this works for a direct bias, it is much harder with indirect bias to spread across different latent dimensions. Therefore, a debiasing algorithm is suggested with two steps 1.) Identify the gender subspace and 2.) Equalize (factor out gender) or soften (reduce magnitude). What do these models learn? [16] have found that Word2Vec with skip-gram and negative sampling is a PMI matrix. A (P)PMI matrix (extra P for keeping only positive entries) is a high dimensional and sparse context matrix, where each row is a word w from the vocabulary V and each column represents a context c, where it occurs. PPMI matrices are theoretically well known and provide a guiding hand for what Word2Vec actually learns. The problem of PPMI matrices is actually that you need to carefully consider each context for each occurring word, which does not scale up to billions of tokens. The results actually show that Word2Vec skip-gram with negative sampling is still the better choice from a view of precision and scalability. For further exploration of the theoretical aspects of word embeddings see [17]; for an explanation of the additivity of vectors [18], and for a geometric interpretation of Word2Vec skip-gram with negative sampling.

3 Subword Embeddings

Subword embeddings deal with words by slicing them into smaller proportions. This is advantageous due to the fact that single words and their corresponding vectors only match by symbolic comparison. Thus, there are advantages of representing words as vectors of sub-level symbolic representations, that first largely occurred in neural machine translations. The representations range from character CNNs/LSTMs [19] to character n-grams [12][13]. These models typically handle out-of-vocabulary words much better than their corresponding word embeddings. While subword-level embeddings deal better with OOV and relatedness than words, there are dedicated strategies for OOV handling beyond subword embeddings.

3.1 Out-of-vocabulary words

Out-of-vocabulary (OOV) words is a problem in two circumstances. The first is that the amount of OOV

words is large, and second - the dataset is small and deals with niche words, where every word constitutes heavily. Words that do not match any given word vector are mapped to the *UNK* token. There are several strategies on dealing with OOV words ranging from using the context words around OOV words [20], using pre-trained language models to assign their vector to OOV words [21] or retrain character-level language models on pre-trained models [22]. [20] found a few tricks to improve on Word2Vec with their proposed model Nonce2Vec. They use pre-trained word embeddings from Word2Vec and treat OOV words as the sum of their context words. They show that this is applicable on smaller datasets as well. [21] found it effective to use vectors of pre-trained language models, where a word was OOV in their domain. Using the pre-trained vector of a different domain helped them in improving the initialization of their OOV words in comparison to assign a global *UNK* token to their data points. They improved models on reading comprehension considerably especially with OOV words. [22] have shown that generating OOV word embeddings by training a character-level model on a pre-trained dataset. The goal is to re-create the vectors by leveraging character information. With a character-level vector word representation OOV words can be handled based on the sum of character vectors. They have found that this is much better in cases where the dataset is small and pre-trained embeddings are available.

3.2 Character-level

Character-level embedding models typically build on pre-trained word embeddings. Additionally characters based representations of words are itself vectors for each character of a word or vector representation of the n-grams of a word. [23] explore different architectures for language modelling and compare three different models with differing inputs to language models. The three setups, see Figure 1, use an LSTM for the language model and either words as

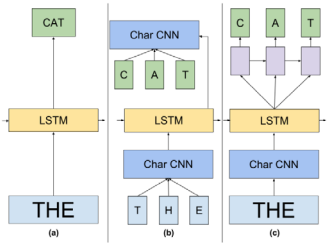


Figure 1: Three different models as depicted in [23]

input and softmax as output, single characters with a CNN as input and output, or a character CNN as input with a softmax output. In the following we will explore different character-level models. [19] presents a model with a character-level convolutional neural network (CNN) with a highway network over characters. Characters are used as an input to a single layer CNN with max-pooling, using a highway network, introduced in [24], similarly to a RNN with a carry mechanism, before applying a LSTM with a softmax for the most likely next word representation. Most interesting in this work is the application of the CNN with the highway network. A few things to note, the vocabulary C over characters and d as usual the embedding size, we deal with $\mathbb{R}^{d\times|C|}$ matrix character embeddings. A word $k \in V$ is decomposed as a sequence of characters $[c_1, \dots, c_l]$, where $l = |k|$, the matrix representation then is $C^k \in \mathbb{R}^{d\times l}$. The columns are character vectors, the rows character dimensions d. The character-level CNN maximizes the following cost function

$$f^k[i] = \tanh(\langle C^k[*, i : i + w - 1], \mathrm{H}\rangle + b)$$
$$y^k = \max_i(f^k[i])$$

where C^k is a filter of width w creating a feature map f^k, indexed by $i \dots i + w - 1$ columns over the filters of C^k. $\langle \dots \rangle$ is the inner product. The convolution or kernel can be seen as a generator for character n-grams. This is then fed to y^k which takes the maximum of the feature map, e.g., applies a max pooling transformation. After this y^k is used as input to a highway network, which is essentially a RNN/LSTM network with different gating mechanisms.

$$z = t \odot g(W_H \cdot y + b_H) + (1 - t) \odot y$$
$$t = \sigma(W_T \cdot y + b_T)$$

The transform gate t maps the input into a different latent space, $(1 - t)$ is the carry gate, deciding, what information will carry on over time. $g(W_H y + b_H)$ is a typical affine transformation with a non-linearity applied. $\odot$ is the entry-wise product or Hadamard Product. Stacking several layers of highway networks allows to carry parts of the input to the output, while combining them in a recurrent fashion. At last, the output z is fit into an LSTM with a softmax to obtain distributions over the next word. [19] manages to reduce parameter size by 60% while achieving state of the art language modelling results. Furthermore, they find that their models learn semantic and orthographic relations from characters, arguing if word-level embeddings seem even necessary. They also successfully deal with OOV words assigning intrinsically chosen words like $looooook$ to the correct word $look$, that word-level models failed to learn.

3.3 Character n-grams

While character-level models work on par with word-level models, recent works focuse on character n-grams. *Charagram* by [13] is an approach to learn character-level compositions, not the statistics of single characters. Given a textual word or sentence, e.g., a sequence of characters x

$$x = \langle x_1, x_2, \dots, x_m\rangle, x_j^i = \langle x_i, x_{i+1}, \dots, x_j\rangle.$$

Charagram produces a character n-gram count vector, where each character n-gram has its own vector $W^{x_j^i}$, if the n-gram $x_j^i \in V$ is part of all n-grams of the model. f

is the indicator function, if $x_j^i \in V$ then 1 else 0.

$$g_{\text{char}}(x) = h(b + \sum_{i=1}^{m+1} \sum_{j=1+i-k}^{i} f(x_j^i \in V) W^{x_j^i})$$

h is a single non-linearity applied over the sum of all n-gram character vectors of x, where k is the maximum length of any character n-gram in the model. V can be initialized by different choices as a model parameter. They achieve state-of-the-art results and further beating LSTM and CNN based models using the spearman's ρ correlation.

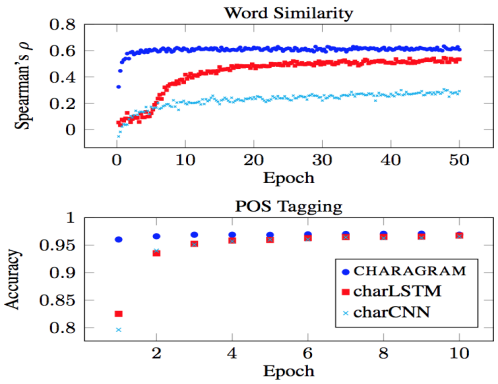


Figure 2: Convergence as depicted in [13]

As we can see in Figure 2, the convergence compared the other models is extremely fast, almost hitting an optimum in the first epochs and much faster than comparable models. It was also found that the models could be trained on far fewer examples while still being comparable to state-of-the-art models. OOV words are handled naturally, because *Charagram* represents words as the sum of characters that even unseen words can be trained and successfully embedded. [13] have shown that character n-grams can be used to beat state of the art models trained on words. [12] proposed an architecture using the Word2Vec skip-gram objective on a bag of words of character n-grams.

The authors describe the skip-gram with negative sampling introduced by [3] and exchange the respective scoring function.Word2Vec takes two vectors u_w and v_w element in $\mathbb{R}^d$, where d is the dimensionality and u_{w_t} is the target word vector with the corresponding context vectors v_{w_c}: $s(w_t, w_c) = u_{w_t}^T \cdot v_{w_c}$.

We would like to represent a word as a character representtation through n-grams, e.g., $where = \langle wh, whe, her, ere, re \rangle$. The above Word2Vec objective can be rewritten to represent each word as a bag of character n-grams vector representation:

$$s(w, c) = \sum_{g \in \mathbb{G}_w} z_g^T v_c$$

where z_g is a vector representation of a single n-gram, from a global set $\mathbb{G}$ with all character n-grams. We are interested in the Word2Vec objective, where each word is now a sum of these character n-gram representations $\mathbb{G}_w \subset 1, \dots, \mathbb{G}$. [12]successfully improve on the analogy task over previous models and deal with OOV words even where the morphemes do not match up. The size of n-grams matter and they suggest above $n > 2$ or $n \geq 5$ for languages like German with many noun compounds.

4 Context Selection

Context selection is about choosing a suitable function over a domain $\mathbb{D}$ that maps a given center and its context to a latent space in $\mathbb{R}^d$ where d is the dimension of a latent column space. In text context selection is narrowly replaced by the surrounding words $c_w \in \mathbb{R}^{k \times d}$ of a target word w, where k is a window size, which is generally known as the skip-gram objective. CBOW on the other hand is the reverse operation, given a context c_w what is its center word. It turns out that context is a much larger topic than just in language modelling. We will first review a couple of concepts applied to general problems of count and real valued data, using exponential family distributions proposed by [4] and [5].

4.1 Generalization of Context selection

Context embeddings are not only useful to textual data, but to sequential data of different shapes and forms as well. [4] presents a general procedure modelling on count and real valued data, using an expectation-maximization (EM) algorithm to approximate exponential family embeddings. The exponential family distributions are distributions with a special form given the natural parameters and sufficient statistics giving rise to the possibility of fitting different kinds of probability distributions to the same problem set. The most famous distributions are Gaussian, Poisson or categorical. [4] propose two example models for Gaussian (real valued) and Poisson (count based) distributions. The general form of exponential families are as follows:

$$x_i | x_{c_i} \sim ExpFam\left(\eta_i(x_{c_i}), t(x_i)\right),$$

where x_i is any data point, for which we like to learn the distribution, x_{c_i} the context of each data point i. $\eta_i(x_{c_i})$ is the natural parameter space that is always convex, e.g., within the bounds of the applicable finite integral of the function and $t(x_i)$ the sufficient statistic, a function that fully summarizes the data x such that there exist no other statistic that provides additional information. The natural parameter has the general form

$$\eta_i(x_{c_i}) = f_i(\rho[i]^T \sum_{j \in c_i} \alpha[j] x_j),$$

where $\rho[i]$ are the embedding parameters for a respective target, $\alpha[j]$ are the context parameters, a probability distribution over context elements, and f is the link function that must be defined for each individual problem, connecting context with a data point. The objective cost function is the sum of log conditional probabilities of each data point which is then optimized using stochastic gradient descent. If the probability distribution is categorical, the objective is almost equivalent to Word2Vec with CBOW.

Given this framework one can construct all kinds of contexts and link functions to solve embeddings for a specific domain. [5] propose an advancement on the *ef-emb* by [4], by considering only a subset of elements in the context, instead of using all of them, naming their

model context selection for exponential family embeddings (CS-EFE). Additionally, CS-EFE depends on three parameters, the embeddings for a target, the context of the target and a hidden binary vector that indicates what the target depends on. The authors leverage amortized variational inference (VI). We will try to describe the work in 3 steps. Why VI? Why blackbox VI? Why amortized VI?

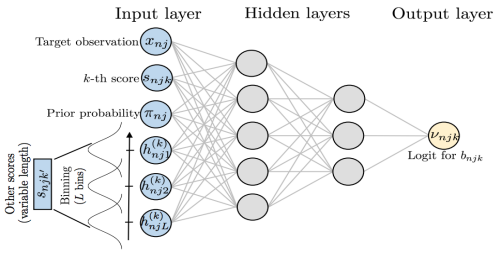
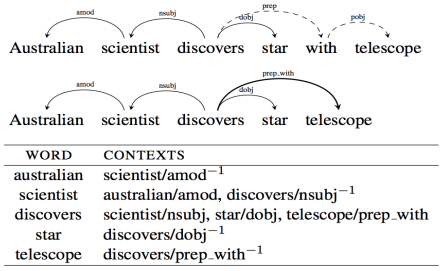


Figure 3: Amortized inference network as depicted in [5]

ef-emb could be easily optimized with gradient descent given the cost function. What has changed is that CS-EFE deploys an additional set of coefficients b that indicate if an element of a context is part of the target word or not. To do this we need to marginalize out this binary vector b. Therefore, we use VI to deposit the functional over the exponential family to approximate the best solution possible. While this is a good starting point, this objective is still intractable and we need to find ways to approximate this even further by the variational lower bound or ELBO and share parameters across the contexts, which VI alone is not able to do. This reduces the runtime and storage complexities considerably and introduces a lower error bound that guarantees errors lower than, but not errors close to. The first problem is that the original VI has no parameter sharing of the context v, which in this case is absolutely needed. Context is shared, that is why an amortization network for parameter sharing is needed, e.g., amortized VI. In Figure 3 we see the amortization network, where x is the target, in language modelling the word w, the score s_{njk} is a score over α_{jk} the context vector, ρ_j the target embedding, π_{nj} are the prior probability parameters of b_{njk} and h^k_{njL} are Gaussian kernels, where each score of each target word except the *kth* is assigned to one of the kernels. The second problem is that we cannot fit the variational distribution $q(b_{nj}; v_{nj})$ to each target individually and hence use blackbox VI, approximating the expectation by Monte Carlo sampling, obtaining noisy gradients of the ELBO. To simplify: Select the correct context from a window using a binary vector as indicator, which cannot be computed, using VI. VI cannot share parameters, which there are plenty of and cannot, even with sharing, estimate the correct gradients given the KLD. Using an indicator vector to select appropriate elements from the context results in variable length context vectors, for which we need a fixed size representation. Instead of this we use Gaussian real valued kernels to estimate mean and variance for each binary vector and assign it. We use Monte Carlo sampling, because we would need to compute every possible setting between the binary vector and context vector, which guarantees an error that is smaller than the evidence lower bound obtaining "tainted" or "noisy" gradients.

4.2 Context selection

Context selection in language models is at this point a well studied task. Word2Vec by [3] uses a context window of surrounding words. While this sounds intuitive, there are a lot of suggestions on improving this. Originally, [3] suggested to use sub-sampling to remove frequently co-occurring words and use context distribution smoothing reducing bias towards rare words. This is very much in conjunction with count based methods that clip off the top/bottom percent of a vocabulary. [25] have found that using dependency based word embeddings has an impact on the quality and quantity of functional similarity tasks such as $cosine$. However, it is to note that on topical similarity tasks the suggested model performs worse. [25] note that mostly a linear context, e.g., windows, is used. Given a corpora and a target word w, with a corresponding sentence (e.g., context) and modifiers of that sentence $m_1, \dots, m_k$ with head h, a dependency tree is created, see Figure 4, with the Stanford Dependency parser.



WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

Figure 4: Context capture as depicted in [25]

The contexts $(m_1, l_1), \dots, (m_k, l_k), (h, l_h^1)$, where l is the dependency relation between head and modifier (e.g., nsubj, dobj, prep with, amod). While l is the forward relation or outgoing relation from the head - the target word - l^1 is the in-going relation or inverse-relation. Given a Word2Vec model with a small window size of $k = 2$ and a larger window size $k = 5$, the dependency based model learns different word relations and minimizes two effects. We can see in Figure 4 that coincidental filtering takes place, because "Australian" is obviously not part of "science" in general, which Word2Vec would take as a context in either model. Secondly, if the window size is small, out-of-reach words like "discover" and "telescope" would have been filtered out. Longer more complex sentences could have several head words, where the context is out-of-reach in larger Word2Vec models as well. In comparison with Word2Vec, the dependency base model has a higher precision and recall on functional similarity tests.

4.3 Dict2Vec

Word2Vec, GloVe and fastText create strong baseline models for word embeddings. Newer trends also incurporate additional information from external data sources, augmenting word vectors. [26] improve on the Word2Vec model by [3] using dictionaries. Dictionaries are records with a word mapping to a definition.

Guitar - a stringed musical instrument, with a fretted fingerboard, typically incurved sides, and six or twelve strings, played by plucking or strumming with the fingers or a plectrum.

The key concept presented in [26] is that each word can be weakly and strongly linked to each other given the definition. For instance, the *Guitar* and *Violin* share the words *stringed musical instrument*, that should strongly tie them together. In the definition of the *Violin* there is no *plucking* or *strumming* and thus is considered a weak pair. Moreover weak pairs are promoted to strong pairs, when they are within the K closest neighbouring words calculated with a cosine distance. The skip-gram objecttive with negative sampling can be rephrased given the definition to positively and negatively couple words. The positive sampling cost function is

$$\mathbf{J_{pos}}(\mathbf{w_t}) = \boldsymbol{\beta_s} \cdot \sum_{w_i \in V_{s(w_t)}} \ell(\boldsymbol{v_t} \cdot \boldsymbol{v_i}) = \boldsymbol{\beta_w} \cdot \sum_{w_j \in V_{w(w_t)}} \ell(\boldsymbol{v_t} \cdot \boldsymbol{v_j}) \ .$$

ℓ is the logistic loss function, $\boldsymbol{w_t}$ is each target word of the corpus with its corresponding vector $\boldsymbol{v_t}$, $\boldsymbol{V_{s(w_t)}}$ are strong pairs, $\boldsymbol{V_{w(w_t)}}$ are weak pairs and $\boldsymbol{v_i}/\boldsymbol{v_j}$ are corresponding strong and weak pair vectors. The hyperparameters $\boldsymbol{\beta_s}$ and $\boldsymbol{\beta_w}$ are chosen to best fit to the learning of strong and weak pairs. Set to zero, the model behaves exactly like Word2Vec. The corresponding negative sampling cost function is

$$\mathbf{J_{neg}}(\mathbf{w_t}) = \sum_{w_i \in \mathcal{F}(w_t), w_i \notin \mathcal{S}(w_t), w_i \notin \mathcal{W}(w_t)} \ell(-\boldsymbol{v_t} \cdot \boldsymbol{v_i}).$$

Where $\boldsymbol{w_i}$ is chosen such that it is randomly chosen from the vocabulary at random without self $\boldsymbol{\mathcal{F}(w_t)}$ and it is not part of strong $\boldsymbol{w_i \notin \mathcal{S}(w_t)}$ or weak $\boldsymbol{w_i \notin \mathcal{W}(w_t)}$ word pairs. Which results in the cost function J from a target word $\boldsymbol{w_t}$ with a context $\boldsymbol{w_c}$:

$$J(\boldsymbol{w_t}, \boldsymbol{w_c}) = \ell(\boldsymbol{v_t} \cdot \boldsymbol{v_c}) + \mathbf{J_{pos}}(\mathbf{w_t}) + \mathbf{J_{neg}}(\mathbf{w_t}).$$

The results show an improvement over state-of-the-art models on word similarity and text classification. They parsed and trained on a last language corpus from Wikipedia comparing a pre-trained Word2Vec model augmented with dictionaries, a retrofitted model using WordNet and a single model on a raw corpus. Dict2Vec showed superior results on the raw corpus and improved the other models by up to 13%.

4.4 Comparison

[27] have found that different downstream and language modelling tasks need different types of context applied. They compare window-based, substitution-based, dependency-based, concatenation and SVD on sub-sampling context for word embeddings. In Figure 5 we can see three kinds of datasets. *WordSim-353-R*, for topical coherence, *WordSim-353-S* and *SimLex999* for functional similarity and *TOEFL* for evenly balanced parts of topical coherence and functional similarity. First to note: substitution based word embeddings performed worse overall in all domains. The idea is to substitute words in sentences, e.g., "I love my job" $[I, ?, my, job]$, substituting for "love" yields $love = [quit\ 0.5,\ love 0.3,\ hate 0.1,\ lost 0.1]$ learned by a language model. What we can immediately see is that typical word embeddings like Word2Vec with window 1, 5 and 10 outperform the other models on topical coherence *WordSim-353-S* and are on par with dependency based models on *SimLex999* and *TOEFL*. Further, dependency-based models perform much better on functional similarly tasks like *WordSim-353-S*. Their results also suggest that concatenating different word embeddings yields the highest results on downstream language tasks such as parsing, ner or sentiment. Unfortunately, *Dict2Vec* is not in the list of curated models as it is still being evaluated and new.

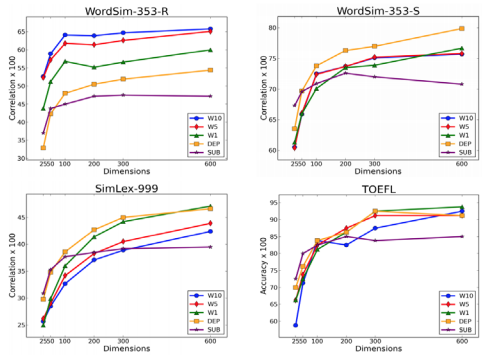


Figure 5: Context results as depicted in [27]

5 Conclusion

In this paper we explored a wide variety of concepts dealing with word-level and subword-level embeddings as well as context selection procedures. All of the suggested methods have assets and drawbacks. However, strategies using pre-trained character n-grams on large datasets with negative sampling/hierarchical softmax on the skip-gram and CBOW objective performs best. That is, they bring all the features of pre-trained word embeddings, while dealing with OOV words and faster training. It would be interesting to see if character-level embeddings could be enhanced with procedures like $Dict2Vec$, $GloVe$ and $CoVe$ to leverage external sources and incorporate global statistics as well. Word2Vec is the basic work-unit behind all current text representation learning tasks. Besides what is covered here, there are multiple research directions open. E.g., statistical models that treat words as a distribution, see [28] and [29]. They treat words as a probability mass functions (pmfs) and can express uncertainty in different dimensions as well as deal with all kind of WSD problems and entailment. [30] goes even further by representing words as hierar-chical pmfs. Instead of changing how the representation is created, they alter the representation to fit certain conditions and features. Another issues are domain adaptation and transfer learning techniques. In the future they will help in dealing

with the asymmetry of data. Given a dataset of a domain that is well known, generalize it to a target domain with fewer samples. This will be particularly helpful in smaller domains and help transpose different ideas beyond the current context. At last there is a desperate need for further theoretical under-standing. It is hard to compare every model and even harder when the evaluation is largely intrinsic and effects can only be indirectly tested in downstream language tasks. Here we will also work on further improvements.

References

[1] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6):391–407, 1990

[2] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. J. Mach. Learn. Res.

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. CoRR, abs/1310.4546, 2013.

[4] M. Rudolph, F. Ruiz, S. Mandt, and D. Blei. Exponential family embeddings. In Advances in Neural Information Processing Systems 29.

[5] L. Liu, F. Ruiz, S. Athey, and D. Blei. Context selection for embedding models. In Advances in Neural Information Processing Systems 30.

[6] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. J. Mach. Learn. Res., 3:1137–1155, March 2003.

[7] T. Bolukbasi, K.W. Chang, J.Y. Zou, V. Saligrama, and A. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. CoRR.

[8] C. Dyer. Notes on noise contrastive estimation and negative sampling. CoRR.

[9] X. Rong. word2vec parameter learning explained. CoRR.

[10] J. Pennington, R. Socher, and C.D. Manning. Glove: Global vectors for word representation. In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, 2014.

[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. CoRR.

[12] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. CoRR.

[13] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Charagram: Embedding words and sentences via character n-grams. CoRR.

[14] B. McCann, J. Bradbury, C. Xiong, and R. Socher. Learned in translation: Contextualized word vectors. CoRR.

[15] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks, pages 5–6, 2005.

[16] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In Advances in Neural Information Processing Systems 27.

[17] A. Gittens, D. Achlioptas, and M.W. Mahoney. Skip-gram - zipf + uniform = vector additivity. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Canada, Volume 1, pages 69–76.

[18] D. Mimno and L. Thompson. The strange geometry of skip-gram with negative sampling. In 2017 Conference on Empirical Methods in Natural Language Processing, September 2017.

[19] Y. Kim, Y. Jernite, D. Sontag, and A.M. Rush. Character-aware neural language models. CoRR.

[20] A. Herbelot and M. Baroni. High-risk learning: acquiring new word vectors from tiny data. CoRR.

[21] B. Dhingra, H. Liu, R. Salakhutdinov, andW.W. Cohen. A comparative study of word embeddings for reading comprehension. CoRR.

[22] Y. Pinter, R. Guthrie, and J. Eisenstein. Mimicking word embeddings using subword rnns. CoRR.

[23] R. J´ozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. CoRR.

[24] R.K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. CoRR.

[25] O. Levy and Y. Goldberg. Dependency-based word embeddings. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, USA, Volume 2, pages 302–308, 2014.

[26] J. Tissier, C. Gravier, and A. Habrard. Dict2vec: Learning word embeddings using lexical dictionaries. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, September 9-11, 2017, pages 254–263, 2017.

[27] O. Melamud, D. McClosky, S. Patwardhan, and M. Bansal. The role of context types and dimensionality in learning word embeddings. CoRR.

[28] L. Vilnis and A. McCallum. Word representations via gaussian embedding. CoRR.

[29] B. Athiwaratkun and A.G. Wilson. Multimodal word distributions. In Conference of the Association for Computational Linguistics (ACL), 2017.

[30] M. Nickel and D. Kiela. Poincar´e embeddings for learning hierarchical representations. CoRR.