# NLP Project Report - Team 2

This report was made as a part of the **CS 6370** - Natural Language Processing course taught at IIT Madras by Professor Sutanu Chakraborti. It details the knowledge that team 2 gained by reading a multitude of technical papers and carrying out a few experiments in the area of word embeddings for NLP tasks.


| Aravind Srinivas | Sanjay Ganapathy | Sahil Sharma |
|---|---|---|
| EE12B090 | CS12B027 | CS12B060 |

IIT Madras

November 2015

# Table of Contents

# Historical word embeddings

IIT Madras, India

**Abstract.** This chapter contains our understanding of Hinton's paper, [1], titled *Learning distributed representations of concepts*. This paper was one of the first to demonstrate the effectiveness of distributed representations of words/concepts in solving NLP tasks.

## Key Contribution

The key technical contribution of this paper is that it demonstrated that concepts can be represented by a distributed pattern of activity in artificial neural networks.
Another important concept which is demonstrated in this paper is the interpretability of the features learnt by the neural network. It also demonstrates that the network learns to chose these features on its own with minimal supervision.

## Types of Representations

There are two ways in which concepts are usually represented in a neural network. These are the localist and the distribution representations. These representations arise as natural extensions of two different theories of semantics. The structuralist theory argues that concepts are defined in terms of their relations to other concepts and have no internal structure. In contrast, the distributional/componential theory seeks to define concepts in terms of a vector of features, where each feature represents a unique trait.
One advantage of such a distributed representation is that it leads to good generalization to unseen data points. Using data that has a size which is linear in $k$, the number of concepts, we hope to learn good representations for each of the $2^k$ data points (in case of the features being binary).
The disadvantage of such componential theories is that they say nothing about how the concepts can be composed to yield a structural entity, which can be reasoned about.

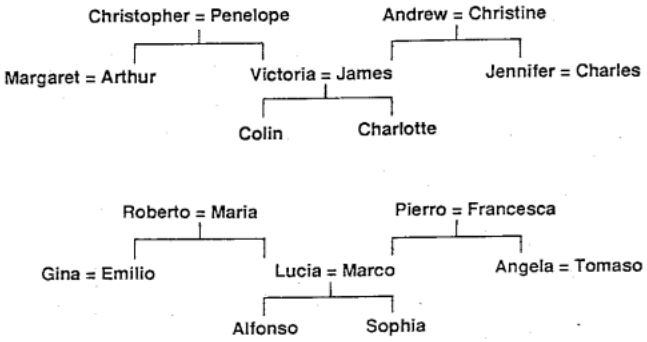
## Choosing representations

The main question which this this paper answers is how to chose patterns to represent concepts (for example, say a person named James), which occur in certain roles (say Husband) in a larger structure (say a relation triple). We could chose

uniformly random representations for these concepts, with the additional constraint that mone of them are too similar to each other. Such representations would indeed be easy to deal with since we could analyze them with methods of expectations and make statements regarding the representations learnt for the hidden layers.
But this eliminates one of the most interesting aspects of distributional representations. The ability to represent a concept in terms of a vector of features and hence capture quantitatively the similarity they might have. This also disallows us from generalizing to unseen input examples since the patterns for representing the concepts would be chosen at random.
To allow the network to *learn* representations for the concepts, the input is presented in a one-hot encoded vector. A dense vector is learnt from this one-hot input by using backpropation algorithm to back-propagate the error. We want the network to use its training data to construct its own representations for the concepts which allow it to perform well at the task at hand.

## The task at hand

This paper uses the relation completion task to demonstrate the novelty of the representations learnt for words. The input relations are represented in the form of triples of the form: (**person1**, **relation**, **person2**). The task is formulated as follows: The **person1** and **relation** are presented as inputs to the neural network as one hot vectors. The output is the **person2** of the relation. The training data comes from the family trees shown below:



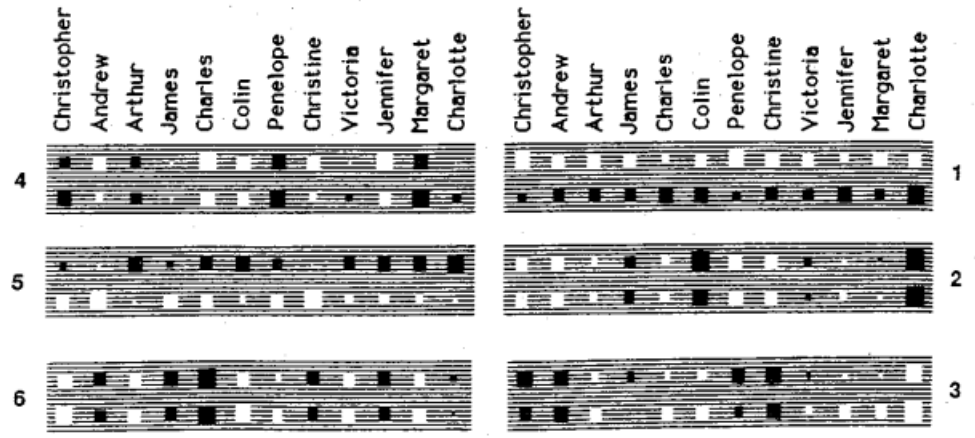

The family trees represented here encode 104 relations. 100 of them were presented as input data and 4 were used as testing points.

## The properties of the vectors learnt

### Vectors for person1

The weights which transform the the 24 input units which encode a one-hot vector for **person1** to a internal distributed representation for **person1** are

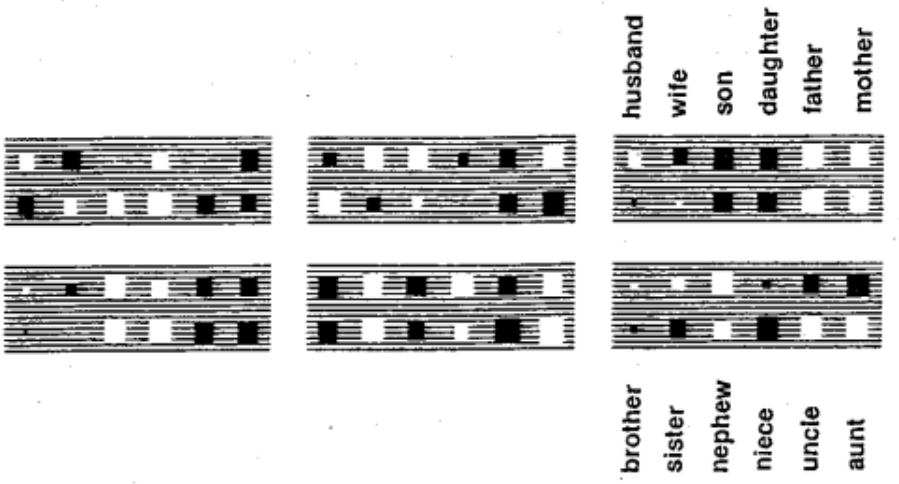
shown below. The white rectangles represent positive weights and the black ones negative weights. The size is proportional to the magnitude of the weight. The weights for the English people are in the row and the bottom row shows the weights learnt for the isomorphic italian person.




These weights form the distributed representation of **person1**. There are several interesting properties about **person1** that these represent.
Unit 1 seems to encode nationality. It can be seen that the isomorphic Italian person has a negative weight which is usually the same magnitude as the English person. Also note that **every** other feature's value has the sign for both the Italian and English person. This seems to suggest that only Unit 1 is concerned with representing the nationality. The representations learnt in terms of the features other than Unit 1 (almost) same for the isomorphic English and Italian counterparts. Hence we expect these representations to be able to allow the network to generalize well to unseen data points.
Unit 2 encodes generation (age). It can be verified that **Charlotte** and **Colin** are the youngest persons in the family tree. Hence they are encoded in terms of the feature having the lowest value. Similarly it can be seen that **Christopher**, **Andrew**, **Penelope** and **Christine** are the eldest and hence encoded in terms of the units having the largest activations.

### Vectors for relation

The weights which connect the 12 input units which encode the one-hot input vector for the relation to the 6 hidden units which form the distributed representation for the relation.




In a distributed representation for a person learnt to predict relationships, one would expect one of the features to encode the gender of the person. But this is not the case. This can be verified by looking at the distributed representations learnt for **person1**. This is probably because the gender of the **person1** is completely determined by the relationship term used in the data-set.
It is hence not surprising to find that one of the units which represents the relationship actually encodes the gender of the person. In the matrix of features above, the unit at position (1,1) (assuming the numbering starts from 0 and goes top-to-down and left-to-right) encodes gender completely. Each of the units with a negative activation imply that **person1** is a male and similarly for female.

## Conclusions

This paper served as one of the first proofs for the fact that distributed representations can be learnt for words and can be helpful for doing the original task as well as interpretable by humans. In the two random initializations that the authors of the paper ran their code, they were able to predict all the 4 test data points correctly in one instance and predicted the correct output person2 3 out of 4 times in the other instance.

# Word2Vec Models for Efficiently Learning Distributional Representation for Words

IIT Madras, India

**Abstract.** We review the work of Mikolov et al, distributed over 2 seminal papers, which introduced an efficient architecture to learn distributional representation of words. We will mainly focus on explaining the skip-gram and continuous bag of words architecture proposed in paper, *Efficient estimation of word representations in vector space.* We'll also explain the three major optimizations to this model proposed in the paper, *Distributed Representations of Words and Phrases and their Compositionality.* namely sub-sampling frequent words and negative sampling as an alternative to the hierarchical soft max function.

Word2Vec is class of neural network models that, given an unlabelled training corpus, produce a vector for each word in the corpus that encodes its semantic information at low computation cost. item These vectors are useful for 2 main reasons

- Semantic similarity between words can be measured using the cosine similarity between the vectors corresponding to the vectors
- Word vectors can be used as features for other supervised NLP tasks such as sentiment analysis and document classification. he semantic information that is contained in these vectors make them powerful features for these tasks.

These vectors adhere surprisingly well to our conventional notion of semantic relations. For example, synonyms tend to have high cosine similarity measure. Consider the analogy Woman is to Queen as Man is to King. It turns out that the learned vectors for these 4 words, namely $v_{queen}, v_{king}, v_{man}, v_{woman}$ are related in the following sense:

$$v_{queen} - v_{woman} + v_{man} \approx v_{king}$$

There are two main symmetric architectures proposed to learn these representations. They are the Skip-gram model and the Continuous Bag-of-Words model. They are both simple one hidden layer neural networks that simply try to maximise the probability of the context given the word and the word given the context respectively.

# 1 Skip-gram Model

## 1.1 Objective function

The goal here is to learn embeddings for the words, which are learnt from the given training data, by maximising the probability of the context words given the word. Formalising this notion the objective function we want to maximise is as follows:

Given a sequence of T training words, $w_1, w_2, \ldots, w_T$, we want to come up with a representation for words that maximises

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

For example, consider the sentence "I hit the ball with a bat", assuming it is part of the given data. Here the context words of 'ball' are {I, hit, the, with, a, bat} with a context window size of 6, that for a context window of size C, we consider the C/2 words preceding and the C/2 words succeeding a word as it's context. This is a meta parameter that we need to choose carefully.

## 1.2 Neural Network Model

We define first the vocabulary, V for which we want to find the embeddings. Our original representation for the words are simply one hot encoded vectors of size $|V|$, with the entry corresponding to the index of the word in V 1 and the others 0. The input to the architecture is a single word, the it has C outputs - the context words.

The input to the model is the one hot vector, $x$, corresponding to word $w$, whose index in V is k. We have to learn 2 matrices in this architecture.

- $W$ - $|V| \times N$ weight matrix between input layer and hidden layer - this is also the matrix which contains embeddings for each word
- $W'$ - $N \times |V|$ weight matrix between hidden layer and each of the C output nodes. This can be visualized as containing columns of vectors, one corresponding to each word, that transforms the embedding vector back to a one hot vector.

Note here that the matrix $W'$ is replicated for each output vector. Basically, the number of nodes in the hidden layer fixes the number of dimensions of our embedding, here it is N. Once we have trained the network on the entire input, each row of $W$ will correspond to the embedding of the word with that index in V.

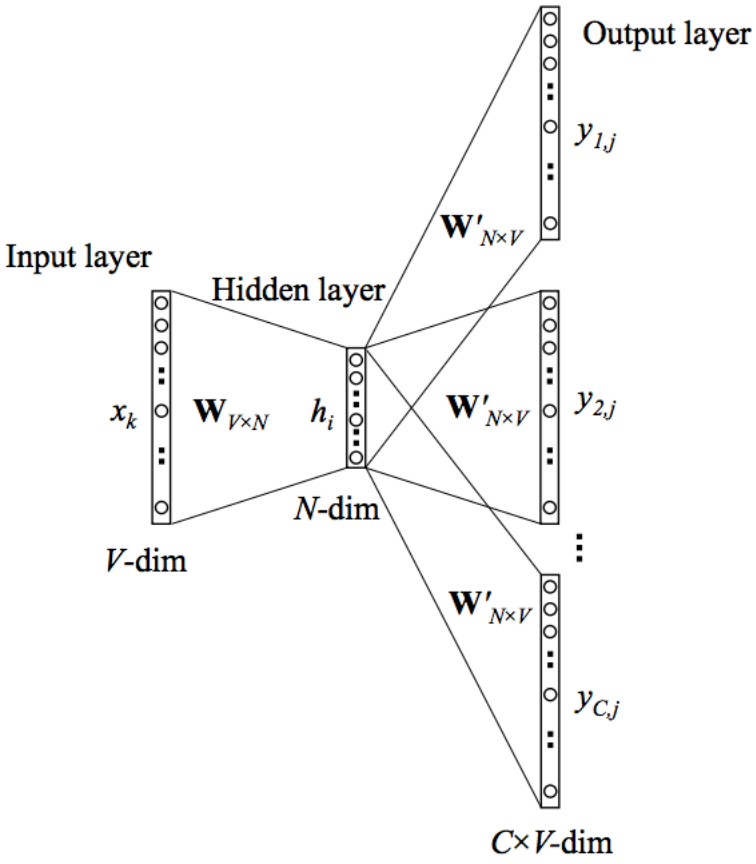



**Fig. 1.** Skip gram architecture from Alex Minnar Word2Vec Tutorial

### 1.3 Mathematical Overview

Let's now analyse what the network is doing for a single training instance. The input to hidden layer is x and so the output of the hidden layer is $x^T W$. Since x is one hot encoded, this will just be the $kth$ row of W, where $w$ is the kth word in V. So, all the first layer does is to choose the correct representation from the matrix $W$ using the index of the word. Let's call this $w_k^{(1)}$.

Now, for the output layer, input to jth node of cth output vector is $u_{c,j} = (w_j^{(2)})^T * w_k^{(1)}$. We observe that it takes the column vector corresponding to jth index in the matrix $W'$ and multiplies it with the embedding vector to get a scalar value. The final output of each node in output layer is arrived at by taking a softmax function for each output vector.

$$y_{c,j} = \frac{e^{u_{c,j}}}{\sum_{i=1}^{|V|} e^{u_i}}$$

Using softmax can help us obtain a well-defined probabilistic (multinomial) distribution among words. The loss function for this model is cross-entropy. Theoretically we can also use squared sum as the error function.

This is the probability that the output of the jth node in the cth output panel is equal to the actual value of the jth index in cth context word vector. Therefore, the probability

$$p(w_c = w_j | w = \frac{e^{u_{c,j}}}{\sum_{i=1}^{|V|} e^{u_i}}$$

where $w_j$ is the expected output vector.

Therefore, the loss function that we have to minimise in order to maximise the objective function is

$$E = -log(p(w_1, w_2, \ldots, w_C | w))$$

$$= -log \prod_{c=1}^{C} \frac{e^{u_{c,j_c^*}}}{\sum_{i=1}^{|V|} e^{u_i}}$$

where $j_c^*$ corresponds to the index of the actual $c^{th}$ context word in $V$. We assume independence conditioned on the word for computing the joint probability of the context words. The parameters, that is the matrices $W$ and $W'$ are learnt using back propagation with the above loss function. Thus, $W'$ will be suitably modified depending on the loss for each output vector in the context.

## 2 Continuous Bag-of-Word Model

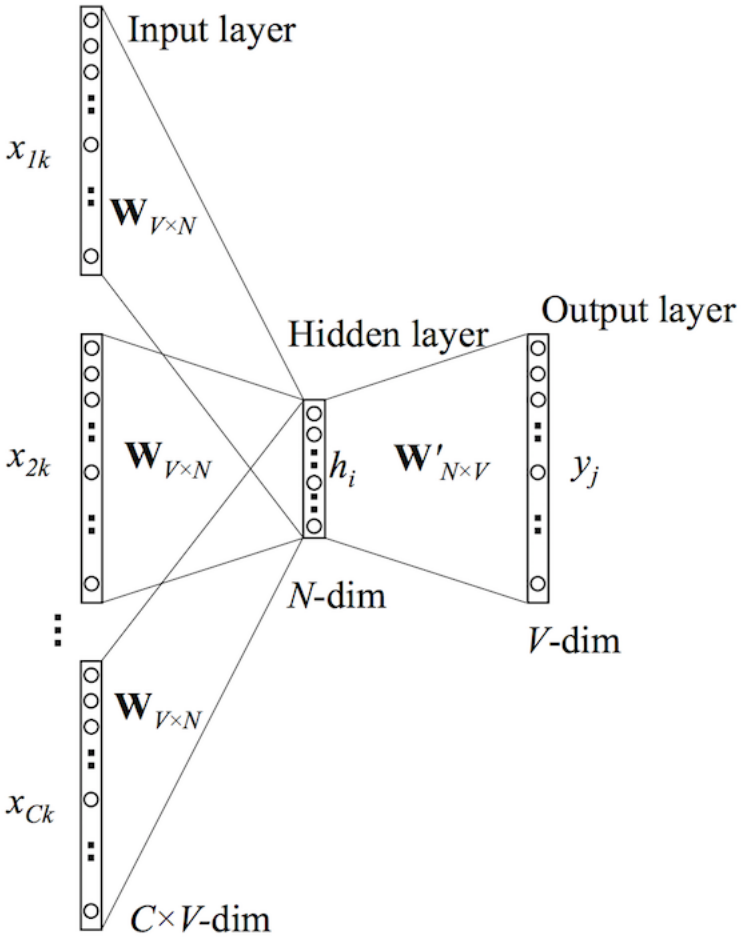



**Fig. 2.** CBOW architecture from Alex Minnar Word2Vec Tutorial

The CBOW is a mirror image of the skip gram model. Similar to skip-gram, we have to learn the parameters of $W'$ and $W$, where $W$ is the matrix of encodings. The main difference is that the C context words of a word are the inputs (one hot encodings) and the one hot vector for the word is the desired output. The soft-max output tries to model the probability of a word given the context words.

The output of the hidden layer is obtained as the average over the context words:

$$h = \frac{1}{C} \sum_{i=1}^{C} x_i^T * W$$

The output of the hidden layer is passed through a softmax function to yield the probability of a word given these context vectors

$$y_j = p(w_j|\text{context words}) = \frac{e^{u_j}}{\sum_{i=1}^{|V|} e^{u_i}}$$

The loss function is defined similarly as skip gram, but in the opposite sense

$$E = -log(p(w|w_1, w_2, \ldots, w_C))$$

## 3 Proposed Optimizations to the Skip-gram Model

### 3.1 Hierarchical Softmax

The idea was introduced in the context of neural network language models by Morin and Bengio. This was proposed to deal with the computational issue of having to iterate over the entire vocabulary to compute the probability of a word given the context, in the skip-gram model.

The words are organized as the leaves of a complete binary tree. Thus there are V-1 internal nodes in the tree, each of which has an output vector associated with it. The basic idea is to find the probability of reaching the target output word from the root given the input word. To reach the output word, we can go either left or right at each internal node in the path. This probability is defined for an internal node n to be

$$p(n, \text{going left}) = \sigma(v_n'^T * h)$$

where $\sigma$ is the sigmoid function and $v_n'^T$ is the output vector associated with internal node n. So, therefore the probability of a path is simply the product of probability of taking the right turn at each node in the path. The length of a path in a complete binary tree with $|V|$ nodes does not exceed $log|V|$ and so we have

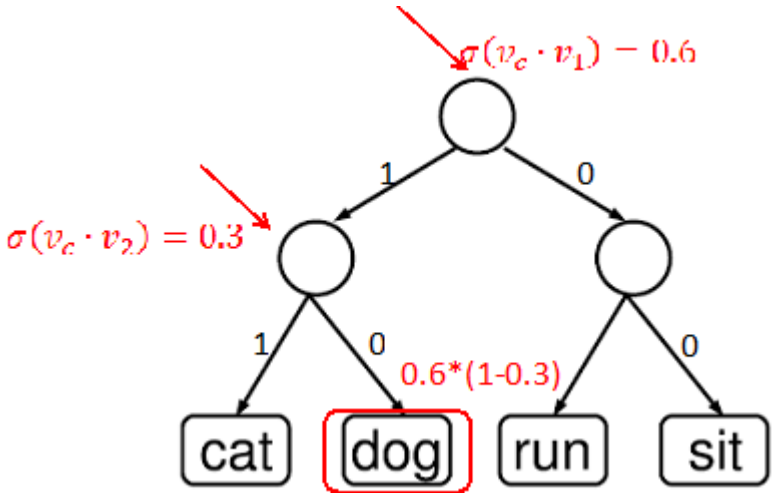



**Fig. 3.** An example hierarchical softmax tree

manged to bring down the order to logarithmic in size of the vocabulary. Given the intuition behind the formulation, let's now define the probability formally

$$p(w) = \prod_{j=1}^{L(w)-1} \sigma(\ \alpha(n(w,j+1) = lc(n(w,j)))\ \ v'^{T}_{n(w,j)} * h)$$

where the probability of a word, w being the output word is simply the product of probabilities of taking the correct turn at each internal node in the path, from root to w.
$\alpha$ is a special function which takes value 1 if the argument is true and -1 otherwise. lc(n(w,j)) represents the left child of the $j^{th}$ node in the path from root to w. The vectors associated with each internal node and the matrix $W$ are updated using back propagation, for each training instance, with the same loss function defined in the skip gram model.

### 3.2 Negative Sampling

This idea is proposed in the paper as an alternative to the hierarchical softmax. Instead of computing the softmax using all words the vocabulary, we instead sample a few words. We obviously need to keep the target word in this list. For sampling the so-called "negative sample", the authors have proposed that a suitable distribution can be empirically determined. The implementation is as follows. We modify the loss function to be minimised as

$$E = -log(\sigma(v'^{T}_{w} h)) - \sigma \sum_{i=1}^{K} log(\sigma(-v'^{T}_{w_i} * h))$$

where w is the required output word and K words, $\{w_1, w_2, \ldots, w_K\}$ are sampled from a probability distribution. K is once a meta parameter that the authors have fixed. They suggest a value of 5-20 for small training data and a value of 2-5 for large training data. The probability distribution used in the paper for sampling negative words is the unigram distribution $U(w)$ raised to the $(3/4)^{th}$ power.

### 3.3 Subsampling of Frequent Words

This is to deal with the issue that observing the occurrence of frequent words in our context, such as 'the', 'of', 'in', etc is not as useful as the occurrence of a rare word in the context. For example, co-occurrence of 'France' and 'Paris' is much more important than the co-occurrence of 'the' and 'France'. The latter may appear many times in any corpus, whereas the former is unlikely to. To counter the imbalance in the effect of frequent and rare words, we use the simple idea of discarding frequent words, that is subsampling them.

Each word $w_i$ is discarded with probability

$$p(\text{discarding } w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where f is the frequency of the word $w_i$ in the corpus and t is meta parameter. The authors have chosen this formula heuristically and have fixed to be around $10^{-5}$, but found that it accelerates learning quite well.

## 4 Evaluation

The task used to evaluate the learned representations is the analogical reasoning task data-set, once again introduced by Mikolov et al. in another paper. The task consists of analogies such as Germany : Berlin :: France : ?, which are solved by finding a vector $x$ such that $vec(x)$ is closest to $vec$(Berlin) - $vec$(Germany) + $vec$(France) according to the cosine distance (we discard the input words from the search). This specific example is considered to have been answered correctly if x is Paris. The task has two broad categories: the syntactic analogies (such as quick : quickly :: slow : slowly) and the semantic analogies, such as the country to capital city relationship.

For training the Skip-gram models, the authors have used a large data-set consisting of various news articles (an internal Google data-set with one billion words).

It is observed that negative sampling and sub-sampling of frequent words outperforms all the other models, both in terms of accuracy and time. It can be argued that the linearity of the skip-gram model makes its vectors more suitable for such linear analogical reasoning.

| | | Accuracy(%) | | |
|---|---|---|---|---|
| Method | Time(mins) | Syntactic | Semantic | Total |
| NEG-5 | 38 | 63 | 54 | 59 |
| NEG-15 | 97 | 63 | 58 | 61 |
| HS | 41 | 53 | 40 | 47 |
| NEG-5 with SS | 14 | 61 | 58 | 60 |
| NEG-15 with SS | 36 | 61 | 61 | 61 |

**Table 1.** Performance of Skip-gram 300 dimension model, NEG-k - Negative sampling with k samples, HS - Hierarchical softmax, SS - sub-sampling with t to be $1p^{-5}$

# Skip Gram as Implicit Matrix Factorization:

IIT Madras, India

**Abstract.** In this chapter, we go through the paper *Neural Word Embedding as Implicit Matrix Factorization* by Levy and Goldberg, NIPS 2014, where they claim and prove that the skip-gram model of Mikolov is *implicitly* factorizing a word-context matrix, whose cells have the PMI (Point-wise Mutual Information) between word and context pairs, shifted by a global constant.

We have seen how the skip-gram model of Mikolov is trained. Even though it is clear that the hidden layer units represent inferred semantic and syntactic properties of the text from the distributional hypothesis, it doesn't explain to us, why exactly a particular semantic property is learnt. For example, on performing a word analogy task for *man : woman :: king: ?* , a skip-gram trained model would retrieve *queen* as the best choice. This happens because the words *king* and *queen* encode high values in directions representing royalty, and reasonably high values in directions representing their corresponding gender (male and female). The encoded values in the royalty directions would be much higher, because the more dominating characteristic about a *king* or a *queen* would be his/her royalty than their corresponding gender. So, when we do a retrieval task, we would do:

$$\text{argmax}_w \text{sim}(\text{king}, w) + \text{sim}(\text{woman}, w) - \text{sim}(\text{man}, w)$$

The first term makes sure the royalty similarity is ensured. But only royalty is not sufficient, since nouns with royalty contexts like *crown, wealth*, etc would also be apt. That is where, similarity with *woman* and being contrasting to *man* plays a role. That ensures that retrieved term is human, and also female. But still, how can we be sure that *princess* is not retrieved with higher score than *queen*? This is inherently ensured because the level of royalty encoded for *queen* should logically be higher. Secondly, contrast to *man* and similarity to *woman* would ensure semantic property of being a *girl*, would be lower. From this example, we understand that a good embedding (that does well on word-analogy tasks like the above example) would have captured all these nice semantic properties in its directions. However, it is not clear how this is being done, in the Mikolov's skip-gram model, where we obtain dense lower dimensional embeddings for the words in the vocabulary, purely by obeying the distributional hypothesis of trying to reconstruct the context words of a specific window length around the target word. Even though we could say that the context words would help capture what kind of syntactic structure the word has (in the case of a king, consider example properties as being male due to being surrounded by pronouns like *His* , in contexts like *His Majesty*), and what kind of semantics characterize the word (like *ruler*), we do not know how exactly these terms/senses got embedded.

On the other hand, if we had a sparse and distributed representation as the one in a distributional context matrix (using PPMI (positive point wise mutual information between a word and its context window word feature/direction), we could easily interpret why the actual syntactic or semantic properties arise, using the sparse high dimensional vector for a word (that is, say, for a *king*, its value on context direction *royal, male* and so on). If it is possible to do the same (encode semantics and syntactic properties in features) in a lower dimensional dense embedding, the immediate question is, whether it is possible to relate the two embeddings. That is, if we were to transform the high dimensional sparse distributed representation where the directions are directly interpretable vocabulary terms, to a lower dimensional dense representation, how would it relate to the skip-gram model embedding?

This question, was addressed by Levy and Goldberg, NIPS 2014, in their paper *Neural Word Embedding as Implicit Matrix Factorization*, where they claim and prove that the skip-gram model of Mikolov is *implicitly* factorizing a word-context matrix, whose cells have the PMI (Pointwise Mutual Information) between word and context pairs, shifted by a global constant. They also prove that another embedding method, NCE (Noise Contrastive Elimination) is also implicitly factorizing a similar matrix, whose contents are the conditional probabilities of the word given the context word, shifted by a global constant. They claim that exact SVD (Singular Value Decomposition) of sparse distributed PMI + Global constant based word-context matrix, leads to dense embeddings that perform as good as the skip-gram model of Mikolov on word similarity tasks, but poorer than the skip-gram model for word analogy tasks.

Briefly explained below, is what the implicit matrix factorization means. Consider the skip gram model, where we use vocabulary $V_W$ for target words $w$, while we use a vocabulary $V_C$ for the context words. Let $|V_W|$ and $|V_C|$ represent their respective cardinalities. The skip gram model takes input of dimensionality $|V_W|$ (one-hot encoding), compresses to a hidden layer embedding of dimensionality $d$ (say), and reconstructs (either several one-hots or a single bag of words) of $|V_C|$ dimensional vector(s). We have the two sets of neural network weights as matrices $W$ and $C^T$ such that their dimensionalities are $|V_W| \times d$ and $|V_C| \times d$, and the neural network basically performs $WC^T$ matrix multiplication to get to the output layer.

The idea is that, we can imagine a matrix $M = WC^T$, of dimensionality $|V_W| \times |V_C|$, which clearly represents the strength of association of each word-term with a context-term, since the element $M_{ij} = V_{w_i}^T V_{c_j}$, where both $V_{w_i}$ and $V_{c_j}$ are of dimensionality $d$. Thus, we are *implicitly* decomposing a matrix $M$ (which can be thought of as a word-context matrix, similar to the one used in distributional similarity), into two matrices $W$ and $C$ to get the respective word and context vectors in lower dimensional dense embeddings. So, what is this matrix $M$, in terms of properties? In distributional hypothesis, we used PPMI or PMI for representing the strength of association between a word $w_i$ and context $c_j$ for the entry $M_{ij}$. The authors of the paper claim that the matrix

$M = WC^T$ is the PMI matrix of word-contexts, shifted by a global constant, that is a hyper-parameter for the Negative sampling method in Skip-Gram model.

To be more specific, let us look at the loss function of the Skip-Gram model, and explain what $M$ matrix is:

$M_{ij} = PMI_{ij} - \log(k)$ where $k$ is used in the Negative Sampling Loss function as shown below:

$$l = \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c)(\log \sigma(\boldsymbol{w}^T \boldsymbol{c}) + k\mathbb{E}_{c \in P_D} \log(-\sigma(\boldsymbol{w}^T \boldsymbol{c})))$$

The constant $k$ basically represents the importance given to the negative sampling in the skip-gram model training, and is therefore a hyper-parameter. The authors show experimental results that the SVD of such a distributed matrix using $-\log(k) + PMI_{ij}$ performs better on the word similarity correlation task than skip gram model itself, but fails to perform as well, on word-analogy tasks. The authors attribute this to the fact that skip gram model does a weighted matrix factorization, whereas all the cells are treated uniformly in the SVD method.

However, the main take-back lesson from this, is that, we have finally understood why skip gram model has been able to discover such nice semantic/syntactic property encoding directions in its embedding. This is because it is implicitly factoring a word-contexts matrix, whose features/directions are context window terms, which carry explicit meaning and sense, and help encode the syntactic properties and semantics for the word terms through the distributional hypothesis, in a way we can understand (as explained for the examples in the introduction).

# Semantic Word Embeddings

IIT Madras, India

**Abstract.** This chapter contains our understanding of the [4], a paper concerned with Learning Semantic Word Embeddings based on Ordinal Knowledge Constraints.

## Key Contribution

This paper presents a generic framework for incorporating semantic world knowledge into the word2vec model, proposed by [3]. In this framework, semantic world knowledge is represented in the form of ordinal inequalities and these are incorporated into the word2vec model by modifying the loss function by adding a regularization term corresponding to these ordinal inequalities.

## Motivation for this work

This paper represents semantic world knowledge a collection of word ordinal ranking inequalities. Moreover, these inequalities are cast as semantic constraints in the optimization process to learn semantically sensible word embeddings. This method has several advantages. Many different types of semantic world knowledge can be represented as such ranking inequalities, such as synonym-antonym, hyponym-hypernym etc. Also, these inequalities can be easily extracted from many existing knowledge resources, such as Thesaurus, WordNet and knowledge graphs. The reason for using ranking inequalities is that they can also be manually generated by human annotation because ranking orders is much easier for human annotators than assigning specific scores. By solving a constrained optimization problem using the stochastic gradient descent algorithm, we can obtain semantic word embedding enhanced by the ordinal knowledge constraints.

## Representing Knowledge By Ranking

Many forms of world knowledge can be represented in the form of ordinal ranking inequalities. The generic form used for such inequalities is:

$$sim(w_i, w_j) > sim(w_i, w_k)$$

where $w_i$ and $w_j$ are words which we expect to be more similar than the pair $w_i$ and $w_k$. For example the $w_i, w_j$ pair could be a synonym pair and $w_i, w_k$ could be an antonym pair. This encodes the human semantic understanding that usually a word is more similar to its synonym than its antonym. Similarly we also

expect a word to be more similar to another word in its WordNet hierarchy, than a word not in its WordNet hierarchy.
Corresponding to each of these inequalities we have corresponding constraints in the word embedding space represented as:

$$sim(w_i^{(1)}, w_j^{(1)}) > sim(w_i^{(1)}, w_k^{(1)})$$

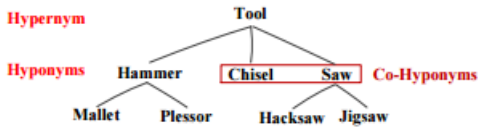
where $w_i^{(1)}$ is the embedding of $w_i$ in the vector space.
This paper uses three types of constraints to regularize the vectors learnt by word2vec:

1. Synonym Antonym Rule: Similarity between a word and its synonym is always larger than similarities between the word and its antonymous words. For example, the similarity between foolish and stupid is expected to be bigger than the similarity between foolish and clever, i.e.,

$$similarity(foolish, stupid) > similarity(foolish, clever)$$

.

2. Semantic Category Rule: Similarity between words that belong to the same semantic category would be larger than similarity between words that belong to different categories. A semantic category may be defined as a synset in WordNet or a hypernym in a semantic hierarchy.




This represents an example of hyponyms in the WordNet hierarchy. We would expect the following similarity inequality:

$$similarity(Mallet, Plessor) > similarity(Mallet, Hacksaw)$$

3. Semantic Hierarchy Rule: Similarities between words that have shorter distances in a semantic hierarchy should be larger than similarities of words that have longer distances. In this work, the semantic hierarchy refers to the hypernym and hyponym structure in WordNet. From Figure 1, this rule may suggest several inequalities like:

$$similarity(Mallet, Hammer) > similarity(Mallet, Tool)$$

## Implementation

The Regularization function chosen by this paper is the **Hinge Loss** function. The overall objective function represented by

$$Q` = Q + \beta D$$

where $D$ is the regularization term consisting of sum of the hinge-loss is defined as:

$$D = \sum_i \max(sim(w_i, w_{a_i}) - sim(w_i, w_{s_i}), 0)$$

where we want $w_i$ to be more similar to $w_{s_i}$ than $w_{a_i}$.
In this work, they used WordNet as the resource to collect ordinal semantic knowledge. WordNet is a large semantic lexicon database of English words (Miller, 1995), where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (usually called synsets). Each synset represents a distinct semantic concept. All synsets in WordNet are interlinked by means of conceptual-semantic and/or lexical relations such as synonyms and antonyms, hypernyms and hyponyms. It is these hierarchies that this paper exploits in the way that has been described in the above sections.

## Conclusions

This paper demonstrates how we can incorporate semantic knowledge sources into the word2vec hierarchy. The idea of using constraints as a regularization term to guide the search for parameters of the neural net is a novel one and is able to incorporate semantic knowledge into the word embeddings, as is demonstrated in the following chapter.

# Retrofitting Word Vectors to Semantic Lexicons

IIT Madras, India

**Abstract.** In this chapter, we go through the Retrofitting Word Vectors to Semantic Lexicons, by Farqui, Dodge, Jauhar ,Dyer, Hovy and Smith, CMU Language Technologies Institute. This paper talks about a new method of improving a given word embedding by incorporating semantic knowledge from lexical resources. Conventional methods of word embedding make use of the *distributional hypothesis* which identifies semantic and syntactic attributes. Such an *attributional* similarity captures semantic *relatedness* but not necessarily semantic similarity. This provides the motivation to incorporating prior knowledge about lexical semantics from resources like WordNet. However, all approaches in this direction have attempted to incorporate the knowledge *during* the learning process of the word-embeddings. This paper, is the first attempt, to incorporate the knowledge after obtained *learned* embeddings, which is why we call it as *Retrofitting*. An important advantage to this approach is that, the *Retrofitting* process is agnostic to the embedding hypothesis, which offers scope for lesser quality in our initial embedding.

## Introduction

### Why should we improve word embeddings like skip-gram?

In lexical semantics, our aim is to discover *properties* to characterize the word such that its meaning is essentially conveyed. This is a hard problem, when you are considering only the distributional semantic and syntactic attributes. For example, it is possible that the words *good* and *bad* are easily substitutable (interchangeable) when given a context window of restricted length, like the ones used to discover these embeddings. In such a case, the *attributional similarity* between *good* and *bad* could be higher than necessary. Thus, it is clear that, two words in these attribute concept spaces, might point in similar directions, but may be of different meanings, worst case, even opposite meanings. This is because, the attributes might not reflect finer properties that may depend on long range dependencies. In the specific example, like *good* and *bad*, maybe the context from the previous sentence(s) is needed to judge what is the meaning conveyed. Also, there is ambiguity as to what noun the adjectives are used for, because of which the sense conveyed could also be hard to gauge from a shorter context. The attributes needed might be something like *bad in the sense of character*, *bad in the sense of quality*, etc. So, the nouns to which these adjectives attach to, will matter, and it is hard to capture such finer syntactic dependencies with short context windows.

So, how do we actually try to address these issues? One might think of using parse tree features, out of say, dependency parsing. But it is not clear how to incorporate the information, and also it is more of syntactic dependencies, which might still not address the lexical semantic issues we have.

This is where, the lexical resources manually constructed, such as WordNet and FreeBase come handy. They consist of *synset* nodes linked by semantic relations like *hypernymy, hyponymy, antonymy, synonymy*. Since WordNet is encoded based on *synset* notion (which means set of *synonyms*), there is a structural information on lexical semantics that we can use to our advantage as prior information. One can also think about this, as trying to encode *relational similarity* into *attributional similarity*.

The point where this paper *diverges* from other papers that have attempted to incorporate *relational semantic knowledge* available in lexical resources, is in *the manner* of incorporating the knowledge. Prior attempts have focused on encoding semantic relational constraints into the word embeddings, as part of the training process itself, in the form of relational inequalities, or direct regularizer terms. Such attempts, are dependent on the training process of the embeddings. However, this paper incorporates the additional *prior lexical knowledge* as a post-processing step and, is also, agnostic to the *training process used to discover the initial embeddings*. This makes it applicable to any given set of embeddings. Another important advantage of this work is the flexibility in how the constraints are added. The constraints are encoded in the form of a graph (*dependency network*), and the edges of the graph are decided on the basis of lexical resources organized in a graph like structure, such as WordNet and FreeBase. We shall see in the next section on the algorithm used to *retrofit* the graph dependency knowledge into the initial embeddings.

## Retrofitting with Semantic Lexicons - Algorithm & Details:

Consider a vocabulary $V$ with words $w_1, w_2, ..., w_n$ and an ontology $O$ that specifies lexical semantic relations as a graph $G$ in the form of $(V, E)$ where $V$ represents the nodes (same as vocabulary $V$) and the edges $E$ represent the semantic relations like *hypernymy, antonymy*, etc.

Let us say that we have an initial embedding $\hat{Q}$ obtained from data-driven techniques like word2vec. Let the matrix be $n \times d$ in shape, which means that each word $w_i$ is represented as a vector $\hat{q_i}$ of dimension $d$.

Our goal is to learn a new embedding $Q$ such that each $q_i$ (for $i$ from 1 to $n$) is close to adjacent vectors $q_j$ in ontology $O$, and is also not so different from the original data driven representation of $\hat{Q}$ ($\hat{q_i}$), so that we preserve the distributional semantic/syntactic properties from the context around the word $w_i$.

The below figure, accurately describes the above explanation:

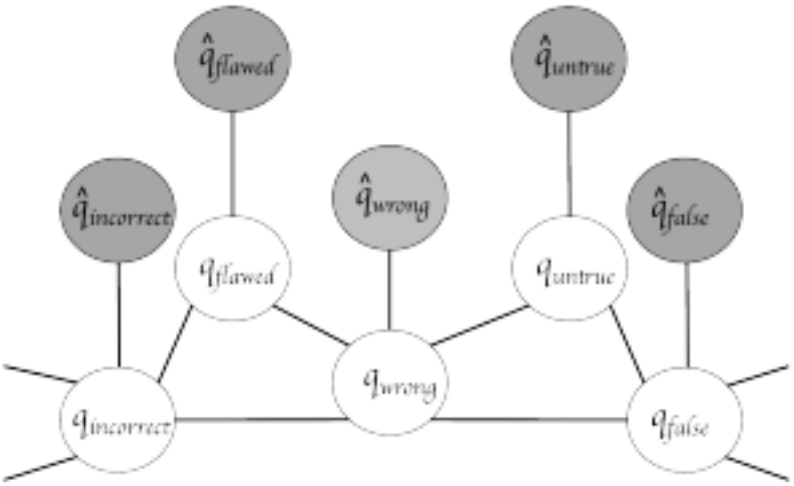



**Fig. 1.** Observed (white) and Inferred (white) Word vector representations. The edges among the white nodes represent the lexical word graph knowledge.

We summarize this in a one line loss function:

$$L(Q) = \sum_{i=1}^{N} \left[ \alpha_i ||q_i - \hat{q}_i||^2 + \sum_{(i,j)\in E} \beta_{ij} ||q_i - q_j||^2 \right]$$

It is clear that we want $q_i$ to be close (in euclidean sense) to neighbors in word graph but at the same time, the effect of the dependency graph knowledge, not to disturb the distributional context structure already present in $\hat{q}_i$.

It is clear that $L(Q)$ is a convex problem in $Q$. In the paper, they use the technique of Alternating Least Squares Minimization for solving this problem iteratively. Even though it is convex, it is not a quadratic in $Q$. But, it is quadratic in $q_i$ assuming that $Q \backslash q_i$ are *known and constants*. Therefore, if we assume the rest of $q_j$ for $j \neq i$ are constant, then $L(Q)$ becomes equivalent to $L(q_i)$ which is a simple quadratic problem, for which we can take the first derivative to be zero to find the minima.

We update

$$q_i = \frac{\sum_{j:(i,j)\in E} \beta_{ij} q_j + \alpha_i \hat{q}_i}{\sum_{j:(i,j)\in E} \beta_{ij} + \alpha_i}$$

.

At each iterative step, over $i = 1$ to $n$, we assume the other $q_j : j \neq i$ are constant, and update $q_i$.

We repeat this sweep over 1 to $N$ several times till $Q$ converges. Convergence is guaranteed because of convexity.

This leads us to a new retrofit $Q$ matrix containing retrofit word embeddings $q_i$ from the original distributional word embedding learnt in a data-driven fashion, $\hat{Q}$.

The implementation details, that are used in the paper are as follows:

- 10 sweeps, over the vocabulary is done, till convergence, where an Euclidean distance error of $10^{-2}$ is obtained, between adjacent nodes.
- Convergence time is 5 seconds for $100,000$ words, with word dimension $d$ as 300.
- Run time length is independent of the model used for word embedding $\hat{Q}$.

The above details, clearly, show that, the retrofitting model is an extremely quick post-processing step, with good flexibility.

## Other approaches to incorporating semantic knowledge:

In the above section, we saw in detail about the retrofitting algorithm. It is also important to understand what have been other approaches to incorporating semantic knowledge. As already pointed out, other approaches, have attempted to incorporate the knowledge as *prior* knowledge during the training algorithm itself, in the form of regularization.

The learning objective of the $\hat{Q}$ model is altered to incorporate the prior. We can think about $Q$ as a posterior semantic embedding we want to learn, and the original loss function governing models like word2vec as a likelihood model, with the semantic resource knowledge playing the role of a *prior* on $Q$ we discover.

Ideally, we want to have an additional term in our loss function as a regularizer placing a constraint on the distance between nodes connected by edges in a relational word-graph like WordNet. That, is if the loss function for the model of $Q$ was $L(Q)$, we want a new modified $\hat{L}(\hat{Q})$ in the following form:

$$\hat{L}(Q) = L(Q) - \gamma \sum_{i=1}^{n} \sum_{j:(i,j)\in E} \beta_{ij} ||q_i - q_j||^2$$

$\gamma$ is a hyper-parameter that denotes how much importance to give to the relational semantic constraints, while $\beta ij$ represent the per-word-pair inequality importance, which must be decided based on the graph structure and the nature of the semantic relation.

**Why is this equivalent to MAP?**

We can think about the additional term added to $L(Q)$ as the prior and the whole loss as a **MAP** (**M**aximum **A- P**osteriori) estimate. Assuming that we want to maximize the log of the posterior probability of $Q$ ($\log(p(Q))$), we can think of the $L(Q)$ term (from the word embedding model) as the log likelihood of $Q$ and the $-\left(\gamma \sum_{i=1}^{n} \sum_{j:(i,j)\in E} \beta_{ij} ||q_i - q_j||^2\right)$ as the log prior. That is, we have a negative exponential prior on $Q$ as follows:

$$p(Q) \propto \exp\left(-\gamma \sum_{i=1}^{n} \sum_{j:(i,j)\in E} \beta_{ij} ||q_i - q_j||^2\right)$$

$$p(Q) = \exp\left(L(Q)\right) \exp\left(-\gamma \sum_{i=1}^{n} \sum_{j:(i,j)\in E} \beta_{ij} ||q_i - q_j||^2\right)$$

**Optimization Problem:**

$$\text{argmax}_Q \log(p(Q) \leftrightarrow \text{argmax}_Q L(Q) - \left(\gamma \sum_{i=1}^{n} \sum_{j:(i,j)\in E} \beta_{ij} ||q_i - q_j||^2\right)$$

There is no closed form solution to the above method, if the training model for $L(Q)$ is non-convex (which it is, for word2vec like models and other neural embedding models). Therefore, we basically need to maximize our $log(P(Q)$ iteratively, which can be done in two methods:

- Gradient descent using the sum of $L(Q)$ (log-likelihood) and log prior from semantic graph using lazy updates.
- Stochastic gradient ascent on log-likelihood, with periodic retrofitting update equations.

I did not understand the motivation behind the second method of updating. It is something interesting, because the authors say it is equivalent to the first. This is something I wish to think about in more detail, as it could lead to new optimization ideas.

## Representation of Word-Graph Knowledge:

We have talked about the semantic relational constraints represented as $\left(\sum_{i=1}^{n}\sum_{j:(i,j)\in E}\beta_{ij}||q_i - q_j||^2\right)$. However, we haven't yet seen how to choose $\beta_{ij}$. For the paper, Farqui *etal* have used a node-degree based parametrization. That is, $\beta_{ij}$ is set as $\frac{1}{\text{degree}(i)}$. All the $\alpha_i$ representing the weight to differing from the distributional representation, are set to 1 for the experiments. This parametrization is done, for experiments all the resources they use, namely, **WordNet, PPDB, FrameNet**.

It is clear that this method of graph representation is not sophisticated, and it may be possible to tune the parameters based on the task for which you are *retrofitting*. This is an avenue which I think is interesting to explore.

## Brief overview of experimental results:

They consider three semantic resources **PPDB, WordNet, FrameNet**. They also use 4 benchmark tasks, **Word Similarity, Syntactic Relations, Synonym Selection, Sentiment Analysis**.

For the sake of brevity, we do not go into the full detail of the experiments, but present the main take-back inferences of their experimental results.

As for the initial embeddings, $\hat{Q}$, the paper considers **Glove** vectors (Pennington *etal*), and **Skip-Gram word2vec** (*Mikolov*) , **Global context vectors** (based on recursive neural networks - Huang *etal*), and **Multilingual vectors** (based on multilingual corpora by using SVD and Canonical Correlation Analysis (CCA) - Faruqui and Dyer (2014)).

We do not go into the details of how these embeddings work, or about the lexical resources used.

- All the three lexical resources offer improvement in word similarity tasks.
- Considerable improvement on TOEFL task except when using FrameNet
- Statistically significant improvement in Extrinsic Sentiment Analysis
- Almost no improvement at all, on Syntactic-Relations task, especially with higher baselines. This is reasonable since we *retrofit semantic* constraints and probably, lose some syntactic constraints from distributional representation too.
- PPDB offers best improvement maximum number of times, aggregated over different initial embeddings, followed by WordNet.
- One reason why FrameNet is not offering improvement, when compared to the other two resources, is that it frames group words based on very abstract concepts and often, words with seemingly distantly related meanings like *push* and *growth* could evoke the same frame.

## Comparisons and Further Analysis:

### Skip Gram vs Retrofitted Skip Gram:

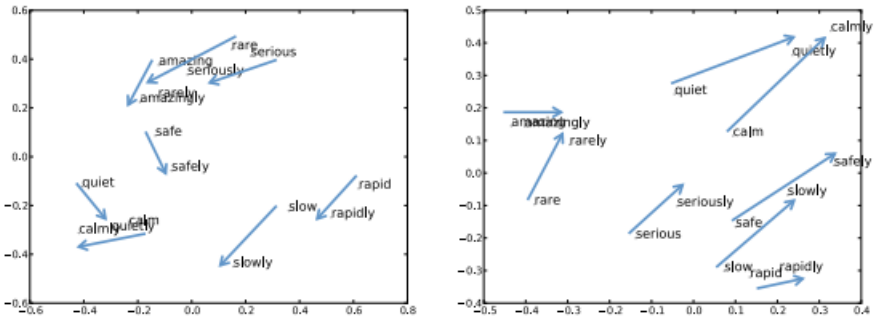



**Fig. 2.** 2 dimensional PCA projection of 100 dimensional word embeddings from Skip-Gram (left) and Retrofitted Skip Gram (right)

We clearly see from the above figure, that retrofitting helps introduce more regularity in the word embedding. The Skip-Gram model, had relatively poor Adjective-Adverb relationship, whereas after retrofitting, it is roughly consistent as a vector translation for all the pairs.

The authors also provide results showing that for both CBOW and Skip-gram architectures of Mikolov, and in comparison to the work by Yu and Dredze on incorporating semantic knowledge, they have better performance on most of the benchmark tasks, showing that retrofitting is indeed a very good solution.

### Performance vs Vector Length:

There has also been analysis on how the performance improves, versus word embedding dimensionality. Intuitively, we would expect that larger dimension embeddings would have already captured comparatively more syntactic and semantic information from distributional hypothesis, and retrofitting might only be an incremental improvement. However, it turns out to be not true.

Below, is a graph that shows, for the Skip Gram model, the performance curves of SG (Skip Gram) and SG + R (Skip Gram + Retrofitting) on word similarity task.

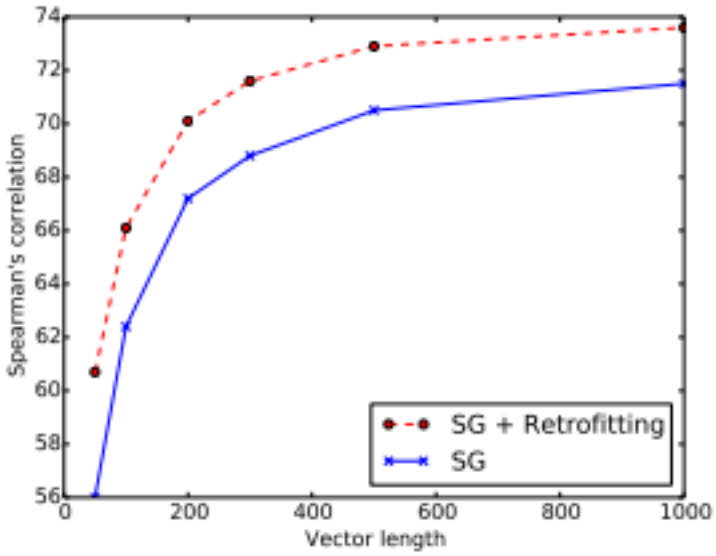



**Fig. 3.** Comparison of Skip Gram vs Skip Gram + Retrofitting, for different vector lengths, on word similarity task

**Multilingual experiments:**

The authors have also carried out experiments in multiple languages, to see if this retrofitting technique is language specific or not, in terms of its success. They have used **Universal WordNet** as the resource for the same. They observe that retrofitting is applicable to other languages too, but report performance only on Word-Similarity domain, since datasets for benchmark tasks in other languages, for other tasks are hard to get.

**Related Work:**

Incorporating lexical semantic knowledge into word embeddings as a regularization prior in the form of a **MAP** estimate, has been approached by Yu and Dredze, 2014 for the first time. Relational knowledge has been used to improve word2vec embeddings by learning based on the constraints that relations are translations in lower dimensions (RCC-Net). This was explored by Xu *etal*, 2014, where both relational and constraint based knowledge from Knowledge Graphs are incorporated in Skip-Gram model training. Recently, Quan Liu has incorporated WordNet knowledge as relational inequalities using hinge loss regularization prior, which we saw in the previous chapter.

The approach proposed by the authors in this paper, is a type of belief propagation algorithm to propagate semantic information among nodes. There has

also been work done to induce POS tags by using graph based belief propagation. Graph based semi-supervised learning has been explored for Machine Translation too. There are other applications where semantic concept propagation has been explored, like Sentiment Analysis, Language Generation, etc.

**References:**

In addition to this paper, and the papers referred to by it, I also referred to the following recent papers :


- Learning Semantic Word Embeddings using Ordinal Inequalities - Quan Liu *etal* - ACL 2015
- Rcnet: A general framework for incorporating knowledge into word representations - Chang Xu *etal* In Proceedings of CIKM, pages 12191228. ACM.

# Experiments

IIT Madras, India

**Abstract.** This chapter of the report documents the experiments we tried out to help us learn about word embeddings better.

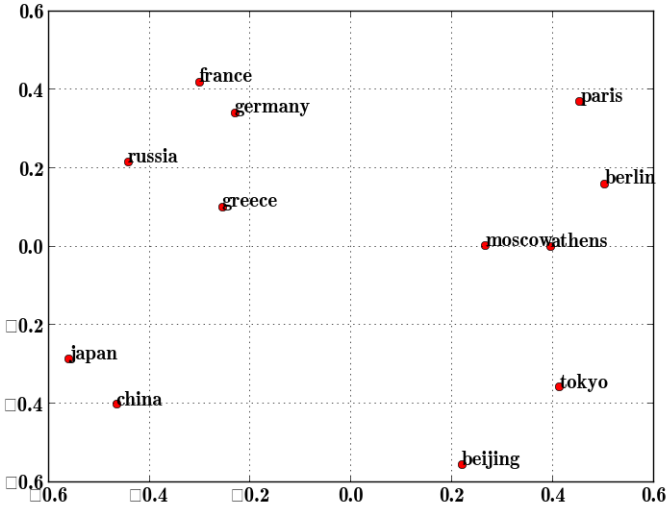
## Experiment

The main experiment we tried out was to try out a different set of in-equations to see whether we could learn better vector representations. This different set was the set of **antonyms**. We generated a fresh list of word antonym pairs using the WordNet as the knowledge source and modified the regularization term defined to the following:

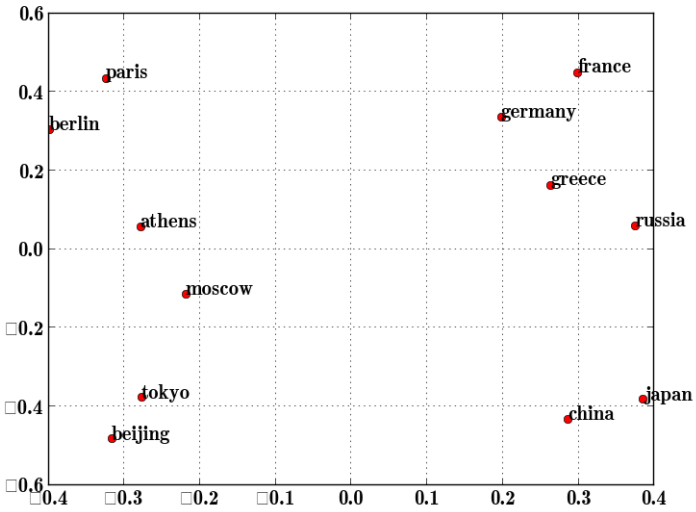
$$D = \sum_i sim(w_i, w_{a_i})$$

where we want $w_i$ to be as dissimilar to $w_{a_i}$ as possible. The word embeddings learnt were brought down to two dimensions using PCA and a few countries and their capitals were visualized to see whether the regularity in the translation which takes country to capital could be increased.
The two main meta parameters in the SWE model are the **number of iterations** we chose to run the code for and the $\beta$ parameter which decides the relative importance given to the semantic constraints, defined in the previous section. The plots obtained were as follows:

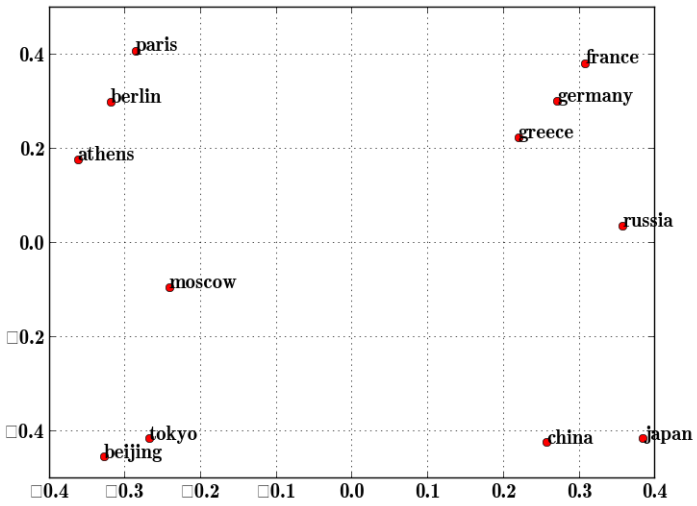
Word2vec Embeddings



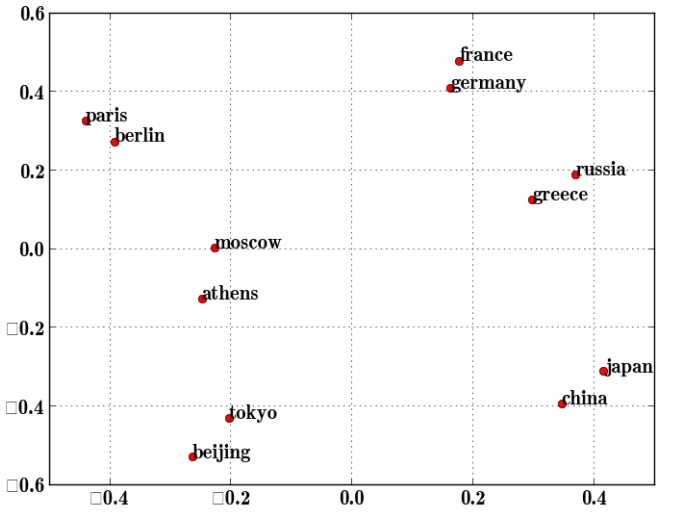

SWE embeddings



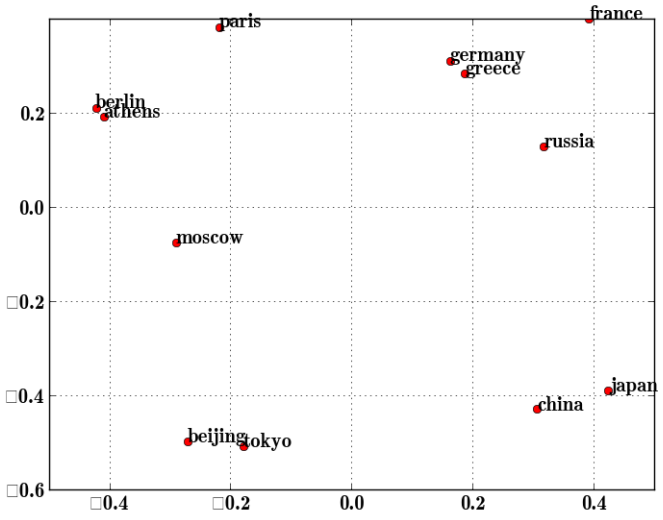
Our experiment - 100 iterations $\beta = 0.1$




Our experiment - 5 iterations $\beta = 0.1$



Our experiment - 100 iterations $\beta = 0.5$




The figures represent an ad-hoc analysis of the word-embeddings. It is good for visualization, but we need a more extensive analysis on how the word embeddings perform for tasks requiring understanding of different semantic and syntactic relations. Country-capital is one type of semantic relation. Just a few pairs plotted, that too, on first two principal components, can only give us a ad-hoc understanding of what is happening, but can be no judge to decide which is the better embedding. We therefore, decided to test the embeddings on a word analogy relations tasks. We use the GOOGLE dataset used by Mikolov 2013, where there are 13 relations, 5 semantic and 8 syntactic. The relations are enumerated in the table. We ignore the out-of-vocabulary words in the tasks.

Since we focused only on antonyms for the change to the Semantic Word Embedding model, a logical thing to try, would be to, incorporate retrofitting to the semantic word embedding learned using regularization on antonym pair similarities, with WordNet graph edges for synonymic, hypernympic and hyponymic edges. This *might* help to give an overall semantic incorporation (with all relations). However, it is also possible that it is counter-productive and performs worse than the embedding before retrofitting. We would like to compare this final embedding obtained (from SWE with Antonyms + Retrofitting with WordNet synonym, hypernym and hyponym knowledge) with the SWE obtained by training along with ordinal inequality constraints from all relations (synonymy , antonymy, hypernymy and hyponymy). We see the performance of the retrofit embeddings and the SWE on the 13 tasks. However, we also compare the original SWE models obtained with antonym regularization, but without retrofitting too.

SWE-1, SWE-2, SWE-3, SWE-4 represent our 4 semantic word embeddings obtained using antonym relation regularization, under 4 hyper parameter settings as pointed out in the figures for the country-capital semantic. R+SWE-$i$ represents the retrofit embeddings of SWE-$i$. SWE represents the original embedding found in the SWE paper by Qian Liu 2015.

We first present the results for the word analogy tasks, for semantic relations, based on SWE, R-SWE$i$ for $i$ from 1 to 4, followed by the results table for syntactic relations word analogy tasks.

The values represent the percentage of questions in that task, that were satisfied with the correct retrieval.

The first two tables show the semantic and syntactic word analogy tasks for the SWE when compared to R-SWE$i$ for $i$ from 1 to 4. The next two tables show the semantic and syntactic word analogy tasks for the SWE when compared to SWE$i$ for $i$ from 1 to 4.

| | Accuracy(%) | | | | |
|---|---|---|---|---|---|
| Datasets | SWE | R-SWE-1 | R-SWE-2 | R-SWE-3 | R-SWE-4 |
| capital-common-countries | 58.7 | 60.1 | 40.1 | 55.9 | 55.1 |
| capital-world | 23.4 | 32.0 | 16.3 | 24.7 | 30.6 |
| currency | 10.2 | 10.4 | 7.9 | 12.1 | 7.9 |
| city-in-state | 14.6 | 13.6 | 5.6 | 8.7 | 12.4 |
| family | 42.1 | 32.6 | 35.7 | 20.7 | 11.7 |

**Table 1.** Semantic Analogy Task

| | Accuracy(%) | | | | |
|---|---|---|---|---|---|
| Datasets | SWE | R-SWE-1 | R-SWE-2 | R-SWE-3 | R-SWE-4 |
| adjective-to-adverb | 9.1 | 8.2 | 7.1 | 3.7 | 2.5 |
| comparative | 49.0 | 38.1 | 41.1 | 12.3 | 2.8 |
| superlative | 13.8 | 16.1 | 13.6 | 4.0 | 1.3 |
| present-participle | 29.3 | 31.8 | 20.7 | 16.8 | 24.1 |
| nationality-adjective | 64.2 | 55.3 | 48.7 | 53.4 | 60.9 |
| past-tense | 22.8 | 22.8 | 15.2 | 15.8 | 13.1 |
| plural | 32.9 | 29.3 | 16.7 | 17.5 | 25.8 |
| plural-verbs | 24.8 | 26.1 | 14.6 | 16.1 | 11.0 |

**Table 2.** Syntactic Analogy Task

We clearly see that, without retrofitting itself, the obtained word embeddings are able to perform really well on most of the word analogy relations. This is quite surprising given that we just ensured regularization from antonym lexical knowledge. We also observe that retrofitting reduces the performance on majority of the tasks, when compared to just antonym regularization. It is possible

| | Accuracy(%) | | | | |
|---|---|---|---|---|---|
| Datasets | SWE | SWE-1 | SWE-2 | SWE-3 | SWE-4 |
| capital-common-countries | 58.7 | 78.3 | 54.2 | 71.7 | 71.5 |
| capital-world | 23.4 | 39.9 | 21.6 | 30.6 | 37.0 |
| currency | 10.2 | 11.1 | 11.6 | 16.9 | 7.9 |
| city-in-state | 14.6 | 29.2 | 14.98 | 21.5 | 28.7 |
| family | 42.1 | 40.9 | 45.23 | 22.9 | 10.5 |

**Table 3.** Semantic Analogy Task

| | Accuracy(%) | | | | |
|---|---|---|---|---|---|
| Datasets | SWE | SWE-1 | SWE-2 | SWE-3 | SWE-4 |
| adjective-to-adverb | 9.1 | 12.6 | 9.47 | 5.24 | 2.11 |
| comparative | 49.0 | 48.34 | 53.6 | 13.36 | 3.37 |
| superlative | 13.8 | 20.5 | 17.3 | 4.4 | 1.7 |
| present-participle | 29.3 | 31.3 | 25.9 | 16.4 | 22.7 |
| nationality-adjective | 64.2 | 74.6 | 64.2 | 72 | 79.8 |
| past-tense | 22.8 | 28 | 23.5 | 21.5 | 14.2 |
| plural | 32.9 | 44.7 | 35.9 | 31.7 | 37.8 |
| plural-verbs | 24.8 | 29.2 | 22.6 | 21.7 | 12.1 |

**Table 4.** Syntactic Analogy Task

that retrofitting is not suited to regularized embeddings. We also observe that when compared to original SWE, many of retrofit antonym regularized embeddings and most of the antonym regularized embeddings get better scores in the word analogy tasks of different kinds. We also observe the first row values for the embeddings without retrofitting, but just regularization. The performance is really good for the country-capital semantic relation, which now proves why our ad-hoc analysis using the first 2 PCs projection showed such regular embeddings. It is interesting to note that just antonym contrasting helped us to discover richer syntactic structure and semantic knowledge.

## References

Geoffrey E Hinton *Learning distributed representations of concepts.* Cognitive science society, 1986.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean *Efficient estimation of word representations in vector space* . In Proceedings of Workshop at ICLR, 2013.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffery Dean *Distributed Representations of Words and Phrases and their Compositionality* In Proceedings of NIPS, 2013.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling ,Yu Hu *Learning Semantic Word Embedding based on Ordinal Knowledge Constraints* Association for Computational Linguistics 2015.