Modeling Order in Neural Word Embeddings at Scale

Andrew Trask ANDREW.TRASK@DIGITALREASONING.COM
Digital Reasoning Systems, Inc., Nashville, TN USA

David Gilmore DAVID.GILMORE@DIGITALREASONING.COM
Digital Reasoning Systems, Inc., Nashville, TN USA

Matthew Russell MATTHEW.RUSSELL@DIGITALREASONING.COM
Digital Reasoning Systems, Inc., Nashville, TN USA

Abstract

Natural Language Processing (NLP) systems commonly leverage bag-of-words co-occurrence techniques to capture semantic and syntactic word relationships. The resulting word-level distributed representations often ignore morphological information, though character-level embeddings have proven valuable to NLP tasks. We propose a new neural language model incorporating both word order and character order in its embedding. The model produces several vector spaces with meaningful substructure, as evidenced by its performance of 85.8% on a recent word-analogy task, exceeding best published syntactic word-analogy scores by a 58% error margin (Pennington et al., 2014). Furthermore, the model includes several parallel training methods, most notably allowing a skip-gram network with 160 billion parameters to be trained overnight on 3 multi-core CPUs, 14x larger than the previous largest neural network (Coates et al., 2013).

1. Introduction

NLP systems seek to automate the extraction of useful information from sequences of symbols in human language. These systems encounter difficulty due to the complexity and sparsity in natural language. Traditional systems have represented words as atomic units with success in a variety of tasks (Katz, 1987). This approach is limited by the curse of dimensionality and has been outperformed by neural network language models (NNLM) in a variety of tasks (Bengio et al., 2003; Morin & Bengio, 2005; Mnih & Hinton, 2009). NNLMs overcome the curse of dimensionality by learning distributed representations for words (G.E. Hinton, 1986; Bengio et al., 2003). Specifically, neural language models embed a vocabulary into a smaller dimensional linear space that models "the probability function for word sequences, expressed in terms of these representations" (Bengio et al., 2003). The result is a vector space model (Maas & Ng, 2010) that encodes semantic and syntactic relationships and has defined a new standard for feature generation in NLP (Manning et al., 2008; Sebastiani, 2002; Turian et al., 2010).

NNLMs generate word embeddings by training a symbol prediction task over a moving local-context window such as predicting a word given its surrounding context (Mikolov et al., 2013a;b). This work follows from the distributional hypothesis: words that appear in similar contexts have similar meaning (Harris). Words that appear in similar contexts will experience similar training examples, training outcomes, and converge to similar weights. The ordered set of weights associated with each word becomes that word's dense vector embedding. These distributed representations encode shades of meaning across their dimensions, allowing for two words to have multiple, real-valued relationships encoded in a single representation (Liang & Potts, 2015).

(Mikolov et al., 2013c) introduced a new property of word embeddings based on word analogies such that vector operations between words mirror their semantic and syntactic relationships. The analogy "king is to queen as man is to woman" can be encoded in vector space by the equation king - queen = man - woman. A dataset of these analogies, the Google Analogy Dataset [1], is divided into two broad categories, semantic queries and syntactic queries. Semantic queries idenfity relationships such as "France is to Paris

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

[1] http://word2vec.googlecode.com/svn/trunk/

as England is to London" whereas syntactic queries identify relationships such as "running is to run as pruning is to prune". This is a standard by which distributed word embeddings may be evaluated.

Until recently, NNLMs have ignored morphology and word shape. However, including information about word structure in word representations has proven valuable for part of speech analysis (Santos & Zadrozny, 2014), word similarity (Luong et al., 2013), and information extraction (Qi et al., 2014).

We propose a neural network architecture that explicitly encodes order in a sequence of symbols and use this architecture to embed both word-level and character-level representations. When these two representations are concatenated, the resulting representations exceed best published results in both the semantic and syntactic evaluations of the Google Analogy Dataset.

2. Related Work

2.1. Word-level Representations (Word2vec)

Our technique is inspired by recent work in learning vector representations of words, phrases, and sentences using neural networks (Mikolov et al., 2013a;b; Le & Mikolov, 2014). In the CBOW configuration of the negative sampling training method by (Mikolov et al., 2013a), each word is represented by a row-vector in matrix syn_0 and is concatenated, summed, or averaged with other word vectors in a context window. The resulting vector is used in a classifier syn_1 to predict the existence of the whole context with the the focus term (positive training) or absence of other randomly sampled words in the window (negative sampling). The scalar output is passed through a sigmoid function ($\sigma(z) = (1 + e^{(-z)})$), returning the network's probability that the removed word exists in the middle of the window, without stipulation on the order of the context words. This optimizes the following objective:

$$\arg\max_{\theta} \prod_{(w,C)\in d} p(w=1|C;\theta) \prod_{(w,C)\in d'} p(w=0|C;\theta)$$

where d represents the document as a collection of context-word pairs (w, C) and C is an unordered group of words in a context window. d' is a set of random (w, C) pairs. θ will be adjusted such that $p(w = 1, C; \theta) = 1$ for context-word pairs that exist in d, and 0 for random context-word pairs that do not exist in d'. In the skip-gram negative sampling work by (Mikolov et al., 2013a;b), each word in a context is trained in succession. This optimizes the following objective:

$$\arg\max_{\theta} \prod_{(w,c)\in d} p(w=1|c;\theta) \prod_{(w,c)\in d'} p(w=0|c;\theta)$$

where d represents the document as a collection of context-word pairs (w, c) and c represents a single word in the context. Modeling an element-wise probability that a word occurs given another word in the context, the element-wise nature of this probability allows (2) to be an equivalent objective to the skip-gram objective outlined in (Mikolov et al., 2013b; Goldberg & Levy, 2014).

Reducing the window size under these models constrains the probabilities to be more localized, as the probability that two words co-occur will reduce when the window reduces which can be advantageous for words subject to short-windowed statistical significance. For example, currency symbols often co-occur with numbers within a small window. Outside of a small window, currency symbols and numbers are not likely to co-occur. Thus, reducing the window size reduces noise in the prediction. Words such as city names, however, prefer wider windows to encode broader co-occurrence statistics with other words such as landmarks, street-names, and cultural words which could be farther away in the document.

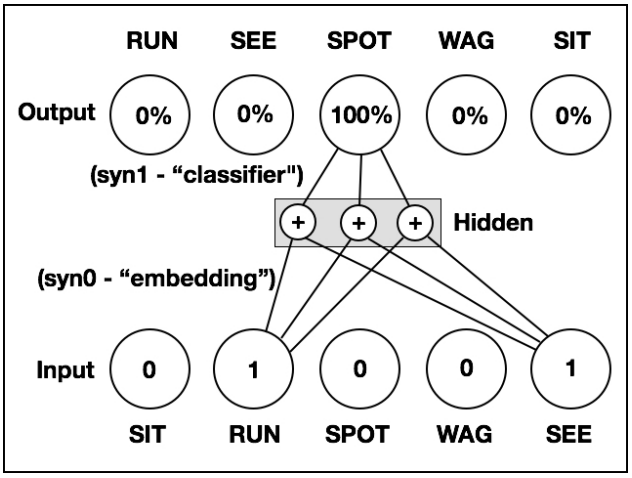

Figure 1. Diagram of word2vec's Continuous Bag of Words training method over the sentence "SEE SPOT RUN". Embeddings for "SEE" and "RUN" are summed into a third vector that is used to predict the probability that the middle word is "SPOT".

Neither skip-gram nor CBOW explicitly preserve word order in their word embeddings (Mikolov et al., 2013a;b; Le & Mikolov, 2014). Ordered concatenation of syn_0 vectors does embed order in syn_1, but this is obfuscated by the fact that the same embedding for each word must be linearly compatible with the feature detectors in every window position. In addition to changing the objective function, this has the effect of cancelling out features that are unique to

only one window position by those in other window positions that are attempting to be encoded in the same feature detector dimension. This effect prevents word embeddings from preserving order based features. The other methods (sum, average, and skip-gram) ignore all order completely in their modeling and model only co-occurrence based probability in their embeddings.

2.2. Character-level Representations

Recent work has explored techniques to embed word shape and morphology features into word embeddings. The resulting embeddings have proven useful for a variety of NLP tasks.

2.2.1. Deep Neural Network

(Santos & Zadrozny, 2014) proposed a Deep Neural Network (DNN) that "learns character-level representation[s] of words and associate[s] them with usual word representations to perform POS tagging." The resulting embeddings were used to produce state-of-the-art POS taggers for both English and Portuguese data sets. The network architecture leverages the convolutional approach introduced in (Waibel et al., 1990) to produce local features around each character of the word and then combines them into a fixed-sized character-level embedding of the word. The character-level word embedding is then concatenated with a word-level embedding learned using word2vec. Using only these embeddings, (Santos & Zadrozny, 2014) achieves state-of-the-art results in POS tagging without the use of hand-engineered features.

2.2.2. Recursive Neural Network

(Luong et al., 2013) proposed a "novel model that is capable of building representations for morphologically complex words from their morphemes." The model leverages a recursive neural network (RNN) (Socher et al., 2011) to model morphology in a word embedding. Words are decomposed into morphemes using a morphological segmenter (Creutz & Lagus, 2007). Using the "morphemic vectors", word-level representations are constructed for complex words. In the experiments performed by (Luong et al., 2013), word embeddings were borrowed from (Huang et al., 2012) and (Collobert et al., 2011). After conducting a morphemic segmentation, complex words were then enhanced with morphological feature embeddings by using the morphemic vectors in the RNN to compute word representations "on the fly". The resulting model outperforms existing embeddings on word similarity tasks accross several data sets.

3. The Partitioned Embedding Neural Network Model (PENN)

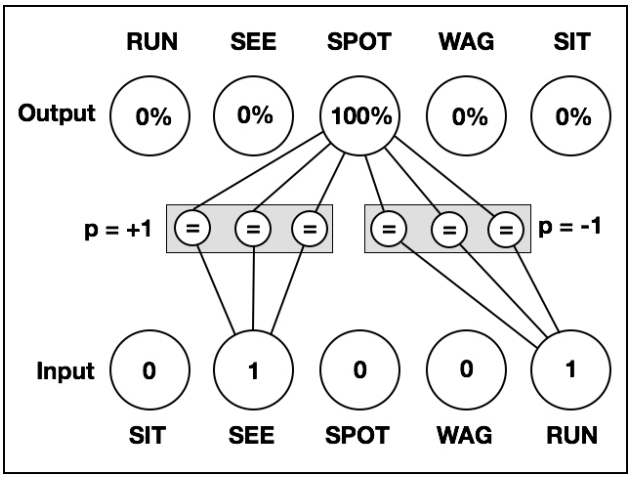

Figure 2. The Windowed configuration of PENN when using the CLOW training method modeling "SEE SPOT RUN".

We propose a new neural language model called a Partitioned Embedding Neural Network (PENN). PENN improves upon word2vec by modeling the order in which words occur. It models order by partitioning both the embedding and classifier layers. There are two styles of training corresponding to the CBOW negative sampling and skip-gram negative sampling methods in word2vec, although they differ in key areas.

The first property of PENN is that each word embedding is partitioned. Each partition is trained differently from each other partition based on word order, such that each partition models a different probability distribution. These different probability distributions model different perspectives on the same word. The second property of PENN is that the classifier has different inputs for words from different window positions. The classifier is partitioned with equal partition dimensionality as the embedding. It is possible to have fewer partitions in the classifier than the embedding, such that a greater number of word embeddings are summed/averaged into fewer classifier partitions. This configuration has better performance when using smaller dimensionality feature vectors with large windows as it balances the (embedding partition size) / (window size) ratio. The following subsection presents the two opposite configurations under the PENN framework.

3.1. Plausible Configurations

3.1.1. Windowed

The simplest configuration of a PENN architecture is the *windowed* configuration, where each partition corresponds to a unique window position in which a word occurs. As

illustrated in Figure 2, if there are two window positions (one on each side of the focus term), then each embedding would have two partitions. When a word is in partition p = +1 (the word before the focus term), the partition corresponding to that position is propagated forward, and subsequently back propagated into, with the p = -1 partition remaining unchanged.

3.1.2. DIRECTIONAL

The opposite configuration to windowed PENN is the *directional* configuration. Instead of each partition corresponding to a window position, there are only two partitions. One partition corresponds to every positive, forward predicting window position (left of the focus term) and the other partition corresponds to every negative, backward predicting window position (right of the focus term). For each partition, all embeddings corresponding to that partition are summed or averaged when being propagated forward.

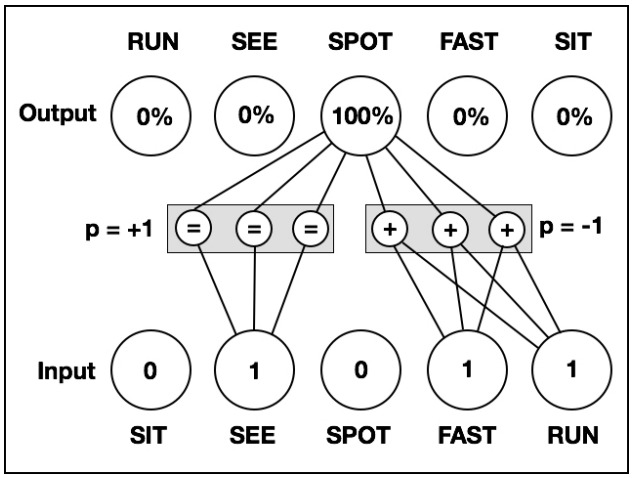

Figure 3. The Directional configuration of PENN when using the CLOW training method. It is modeling the sentence "SEE SPOT RUN FAST".

3.2. Training Styles

3.2.1. CONTINUOUS LIST OF WORDS (CLOW)

The Continuous List of Words (CLOW) training style under the PENN framework optimizes the following objective function:

$$\arg\max_{\theta}\Big(\prod_{(w,C)\in d}\prod_{-c\le j\le c, j\neq 0} p(w=1|c_j^j;\theta)$$

$$\prod_{(w,C)\in d'}\prod_{-c\le j\le c, j\neq 0} p(w=0|c_j^j;\theta)\Big)$$

where c_j^j is the *location specific representation* (partition ^{j}) for the word at window position j relative to the focus word w. Closely related to the CBOW training method, the CLOW method models the probability that in an ordered list of words, a specific word is present in the middle of the list, given the presence and location of the other words. For each training example out of a windowed sequence of words, the middle focus term is removed. Then, a partition is selected from each remaining word's embedding based on that word's position relative to the focus term. These partitions are concatenated and propagated through the classifier layer. All weights are updated to model the probability that the presence of the focus term is 100% (positive training) and other randomly sampled words 0% (negative sampling).

3.2.2. SKIP-GRAM

The skip-gram training style under the PENN framework optimizes the following objective function

$$\arg\max_{\theta}\Big(\prod_{(w,C)\in d}\sum_{-c\le j\le c, j\neq 0} p(w_j=1|c_j^j;\theta)$$

$$\prod_{(w,C)\in d'}\sum_{-c\le j\le c, j\neq 0} p(w_j=0|c_j^j;\theta)\Big)$$

where, like CLOW, c_j^j is the *location specific representation* (partition ^{j}) for the word at window position j relative to the focus word w. w_j is the relative location specific probability (partition) of the focus term. PENN skip-gram is almost identical to the CLOW method with one key difference. Instead of each partition of a word being concatenated with partitions from neighboring words, each partition is fed forward and back propagated in isolation. This models the probability that, given a single word, the focus term is present a relative number of words away in a given direction. This captures information lost in the word2vec skip-gram architecture by modeling based on the relative location of a context word in the window as opposed to an arbitrary location within the window.

The intuition behind modeling w and c based on j at the same time becomes clear when considering the neural architecture of these embeddings. Partitioning the context word into j partitions gives a *location specific representation* for a word's relative position. Location specific representations are important even for words with singular meanings. Consider the word "going", a word of singular meaning. This word's effect on a task predicting a word immediately before it is completely different than predicting a word immediately after it. The phrase "am going" is a plausible phrase. The phrase "going am" is not. Thus, forcing this word to have a consistent embedding across these

tasks forces it to convey identical information optimizing for nonidentical problems.

Partitioning the classifier incorporates this same principle with respect to the focus word. The focus word will read features presented to it in a different light with a different weighting given its position. For example, "dollars" is far more likely to be predicted accurately based on the word before it; whereas, it is not likely to be predicted correctly by a word ten window positions after. Thus, the classifier responsible for looking for features indicating that "dollars" is next should not have to be the same classifier that looks for features ten window positions into the future. Training separate classifier partitions based on window position avoids this phenomenon.

3.3. Distributed Training Optimizations

3.3.1. SKIP-GRAM

When skip-gram is used to model ordered sets of words under the PENN framework each classifier partition and its associated embedding partitions may be trained in full-parallel (with no inter-communication) and reach the exact same state as if they were not distributed. A special case of this is the *windowed* embedding configuration, where every window position can be trained in full parallel and concatenated (embeddings and classifiers) at the end of training. This allows very large, rich embeddings to be trained on relatively small, inexpensive machines in a small amount of time with each machine optimizing a part of the overall objective function. Given machine j, training skip-gram under the *windowed* embedding configuration optimizes the following objective function:

$$\arg\max_{\theta}\Big(\prod_{(w,C)\in d} p(w_j = 1|c_j^j;\theta)$$

$$\prod_{(w,C)\in d'} p(w_j = 0|c_j^j;\theta)\Big)$$

Concatenation of the weight matrices syn_0 and syn_1 then incorporates the sum over j back into the PENN skip-gram objective function during the forward propagation process, yielding identical training results as a network trained in a single-threaded, single-model PENN skip-gram fashion. This training style achieves parity training results with current state-of-the-art methods while training in parallel over as many as j separate machines.

3.3.2. CLOW

The CLOW method is an excellent candidate for the ALOPEX distributed training algorithm (Unnikrishnan & Venugopal, 1994) because it trains on very few (often single) output probabilities at a time. Different classifier partitions may be trained on different machines, with each training example sending a short list of floats per machine across the network. They all share the same global error and continue on to the next iteration.

A second, nontrivial optimization is found in the strong performance of the *directional* CLOW implementation with very small window sizes (pictured below with a window size of 1). *Directional* CLOW is able to achieve a parity score using a window size of 1, contrasted with word2vec using a window size of 10 when all other parameters are equal, reducing the overall training time by a factor of 10.

4. Dense Interpolated Embedding Model

char similarity			
a	**A**	**1**	**s**
o	E	5	p
e	O	7	h
i	I	4	x
u	!	8	d

Table 1. A focus character and the 4 closest characteres ordered by cosine similarity.

SEMANTIC		**SYNTACTIC**	
"general" - similarity			
secretary	0.619	gneral	0.986
elections	0.563	genral	0.978
motors	0.535	generally	0.954
undersecretary	0.534	generation	0.944
"sees" - "see" + "bank" ≅			
firestone	0.580	banks	0.970
yard	0.545	bank	0.939
peres	0.506	balks	0.914
c.c	0.500	bans	0.895

Table 2. An example of syntactic vs semantic embeddings on the cosine similarity and word-analogy tasks.

We propose a second new neural language model called a Dense Interpolated Embedding Model (DIEM). DIEM uses neural embeddings learned at the character level to generate a fixed-length syntactic embedding at the world level useful for syntactic word-analogy tasks, leveraging patterns in the characters as a human might when detecting syntactic features such as plurality.

4.1. Method

Generating syntactic embeddings begins by generating character embeddings. Character embeddings are generated using vanilla word2vec by predicting a focus charac-

Algorithm 1 Dense Interpolated Embedding Pseudocode

Input: wordlength I, list char embeddings (e.g. the word) $char_i$, multiple M, char dim C, vector v_m
for $i = 0$ **to** $I - 1$ **do**
  s =M * i/l
  for $m = 0$ **to** $M - 1$ **do**
    d = pow(1 - (abs(s - m)) / M,2)
    v_m = v_m + d * $char_i$
  end for
end for

ter given its context. This clusters characters in an intuitive way, vowels with vowels, numbers with numbers, and capitals with capitals. In this way, character embeddings represent morphological building blocks that are more or less similar to each other, based on how they have been used.

Once character embeddings have been generated, interpolation may begin over a word of length I. The final embedding size must be selected as a multiple M of the character embedding dimensionality C. For each character in a word, its index i is first scaled linearly with the size of the final "syntactic" embedding such that $s = M * i / l$. Then, for each length C position m (out of M positions) in the final word embedding v_m, a squared distance is calculated relative to the scaled index such that distance $d = pow$(1-(abs(s - j)) / M,2). The character vector for the character at position i in the word is then scaled by d and added elementwise into position m of vector v.

A more efficient form of this process caches a set of transformation matrices, which are cached values of $d_{i,m}$ for words of varying size. These matrices are used to transform variable length concatenated character vectors into fixed length word embeddings via vector-matrix multiplication.

These embeddings are useful for a variety of tasks, including syntactic word-analogy queries. Furthermore, they are useful for syntactic query expansion, mapping sparse edge cases of a word (typos, odd capitalization, etc.) to a more common word and its semantic embedding.

4.2. Distributed Use and Storage Optimizations

Syntactic vectors also provide significant scaling and generalization advantages over semantic vectors. New syntactic vectors may be inexpensively generated for words never before seen, giving loss-less generalization to any word from initial character training, assuming only that the word is made up of characters that have been seen. Syntactic embeddings can be generated in a fully distributed fashion and only require a small vector concatenation and vector-matrix multiplication per word. Secondly, the character vectors (typically length 32) and transformation matrices (at most 20 or so of them) can be stored very efficiently relative to the semantic vocabularies, which can be several million vectors of dimensionality 1000 or more. Despite their significant positive impact on quality, DIEM optimally performs using 6+ orders of magnitude less storage space, and 5+ orders of magnitude fewer training examples than word-level semantic embeddings.

5. Experiments

5.1. Evaluation Methods

We conduct experiments on the word-analogy task of (Mikolov et al., 2013a). It is made up of a variety of word similarity tasks, as described in (Luong et al., 2013). Known as the Google Analogy Dataset, it contains 19,544 questions asking "a is to b as c is to _" and is split into semantic and syntactic sections. Both sections are further divided into subcategories based on analogy type, as indicated in the results tables below.

All training occurs over the dataset available from the Google word2vec website[2], using the packaged word-analogy evaluation script. The dataset contains approximately 8 billion words collected from English News Crawl, 1-Billion-Word Benchmark, UMBC Webbase, and English Wikipedia. The dataset used leverages the default data-phrase2.txt normalization in all training, which includes both single tokens and phrases. Unless otherwise specified, all parameters for training and evaluating are identical to the default parameters specified in the default word2vec big model, which is freely available online.

5.2. Embedding Partition Relative Evaluation

Figure 4 displays the relative accuracy of each partition in a PENN model as judged by row-relative word-analogy scores. Other experiments indicated that the pattern present in the heat-map is consistent across parameter tunings. There is a clear quality difference between window positions that predict forward (left side of the figure) and window positions that predict backward (right side of the figure). "currency" achieves most of its predictive power in short range predictions, whereas "capital-common countries" is a much smoother gradient over the window. These patterns support the intuition that different window positions play different roles in different tasks.

5.3. Evaluation of CLOW and CBOW

Table 3 shows the performance of the default CBOW implementation of word2vec relative to CLOW and DIEM when configured to 2000 dimensional embeddings. Between tables 3 and 4, we see that increasing dimension-

[2]https://code.google.com/p/word2vec/

Figure 4. Green represents the highest quality partition. Red indicates the lowest. Gray indicates the gradient performance between red and green. Two greens in the same row indicates a tie within a 1% margin.

ality of baseline CBOW word2vec past 500 achieves sub-optimal performance. Thus, a fair comparison of two models should be between optimal (as opposed to just identical) parameterization for each model. This is especially important given that PENN models are modeling a much richer probability distribution, given that order is being preserved. Thus, optimal parameter settings often require larger dimensionality. Unlike the original CBOW word2vec, we have found that bigger window size is not always better. Larger windows tend to create slightly more semantic embeddings, whereas smaller window sizes tend to create slightly more syntactic embeddings. This follows the intuition that syntax plays a huge role in grammar, which is dictated by rules about which words make sense to occur immediately next to each other. Words that are +5 words apart cluster based on subject matter and semantics as opposed to grammar. With respect to window size and overall quality, because partitions slice up the global vector for a word, increasing the window size decreases the size of each partition in the window if the global vector size remains constant. Since each embedding is attempting to model a very complex (hundreds of thousands of words) probability distribution, the partition size in each partition must remain high enough to model this distribution. Thus, modeling large windows for semantic embeddings is optimal when using either the *directional* embedding model, which has a fixed partition size of 2, or a large global vector size. The *directional* model with optimal parameters has slightly less quality than the *windowed* model with optimal parameters due to the vector averaging occurring in each window pane.

5.4. Evaluation of DIEM Syntactic Vectors on Syntactic Tasks

Semantic Architecture	CBOW	CLOW	DIEM
Semantic Vector Dim.	500	500	500
SEMANTIC TOTAL	81.02	80.19	80.19
adjective-to-adverb	37.70	35.08	**94.55**
opposite	36.21	40.15	**74.60**
comparative	86.71	87.31	**92.49**
superlative	80.12	82.00	**87.61**
present-participle	77.27	80.78	**93.27**
nationality-adjective	**90.43**	90.18	71.04
past-tense	72.37	**73.40**	47.56
plural	80.18	81.83	**93.69**
plural-verbs	58.51	63.68	**95.97**
SYNTACTIC TOTAL	72.04	73.45	**81.53**
COMBINED SCORE	76.08	76.49	**80.93**

Table 4. Above we see can observe the boost that syntactic based DIEM feature vectors gives our unsupervised semantic models, relative to both word2vec-CBOW and CLOW

Table 4 documents the change in syntactic analogy query quality as a result of the interpolated DIEM vectors. For

Configuration Style	W2V	Window	(see tbl. 5)
Training Style	CBOW	CLOW	ENSEM
Word Vector Size	**2000**	**2000**	**7820**
Partition Size	2000	500	(see tbl. 5)
Window Size	10	2	(see tbl. 5)
capital-common	85.18	**98.81**	95.65
capital-world	75.38	90.01	**93.90**
currency	0.40	16.89	**17.32**
city-in-state	65.18	78.31	**78.88**
family	49.01	84.39	**85.35**
SEMANTIC	65.11	80.62	**82.70**
adjective-to-adverb	15.62	30.04	**90.73**
opposite	8.50	38.55	**73.15**
comparative	51.95	94.37	**99.70**
superlative	33.87	79.77	**91.89**
present-participle	45.45	81.82	**93.66**
nationality-adjective	88.56	89.38	**91.43**
past-tense	55.19	**76.99**	60.01
plural	73.05	83.93	**97.90**
plural-verbs	28.74	73.33	**95.86**
SYNTACTIC	49.42	75.11	**88.29**
TOTAL	56.49	77.59	**85.77**

Table 3. Comparison between Word2vec, CLOW, and Penn-DIEM Ensemble

Conf. Training Style	Window Size	Dimensionality
Windowed	10	500
Directional	5	500
Windowed	2	2000
Directional	5	2000
Directional	10	2000
Directional	1	500
DIEM	x	320

Table 5. Concatenated Model Configurations

the DIEM experiment, each analogy query was first performed by running the query on CLOW and DIEM independently, and selecting the top thousand CLOW cosine similarities. We summed the squared cosine similarity of each of these top thousand with each associated cosine similarity returned by the DIEM and resorted. This was found to be an efficient estimation of concatenation that did not reduce quality.

Table 5 documents the parameter selection for a combined neural network partitioned according to several training styles and dimensionalities. As in the experiments of Table 3, each analogy query was first performed by running the query on each model independently, selecting the top thousand cosine similarities. We summed the cosine similarity of each of these top thousand entries across all models (excluding DIEM for semantic queries) and resorted. (For normalization purposes, DIEM scores were raised to the power of 10 and all other scores were raised to the power of 0.1 before summing).

5.5. High Level Comparisons

Algorithm	**GloVe**	**Word2Vec**		**PENN+D**	
Config	x	CBOW	SG	SG	ENS
Params	x	7.6 B	7.6 B	**40B**	**59B**
Sem. Dims	300	500	500	5000	7820
Semantic	81.9	81.0	82.2	69.6	**82.7**
Syntactic	69.3	72.0	71.3	80.0	**88.3**
Combined	75.0	76.1	76.2	75.3	**85.8**

Table 6. Scores reflect best published results in each category, semantic, syntactic, and combined when parameters are tuned optimally for each individual category.

Our final results show a lift in quality and size over previous models with a 58% syntactic lift over the best published syntactic result, and a 40% overall lift over the best published overall result (Pennington et al., 2014). Table 5 also includes the highest word2vec scores we could achieve through better parameterization (which also exceeds the best published word2vec scores). Within PENN models, there exists a speed vs. performance tradeoff between SG-DIEM and CLOW-DIEM. In this case, we achieve a 20x level of parallelism in SG-DIEM relative to CLOW, with each model training partitions of 250 dimensions (250 * 20 = 5000 final dimensionality). A 160 billion parameter network was also trained overnight on 3 multi-core CPUs, however it yielded 20000 dimensional vectors for each word and subsequently overfit the training data. This is because a dataset of 8 billion tokens with a negative sampling parameter of 10 has 80 billion training examples. Having more parameters than training examples overfits a dataset, whereas 40 billion performs at parity with current state of the art, as pictured in Table 5. Future work will experiment with larger datasets and vocabularies. The previous largest neural network contained 11.2 billion parameters (Coates et al., 2013), whereas CLOW and the largest SG contain 16 billion (trained all together) and 160 billion (trained across a cluster) parameters respectively as measured by the number of weights.

6. Conclusion and Future Work

Encoding both word and character order in neural word embeddings is beneficial for word-analogy tasks, particularly syntactic tasks. These findings are based upon the intuition that order matters in human language and has been validated through the methods above. Future work will further investigate the scalability of these word embeddings to larger datasets with reduced runtimes.

References

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pascal, and Janvin, Christian. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003. ISSN 1532-4435.

Coates, Adam, Huval, Brody, Wang, Tao, Wu, David, Catanzaro, Bryan, and Andrew, Ng. Deep learning with cots hpc systems. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1337–1345, 2013.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel P. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL http://arxiv.org/abs/1103.0398.

Creutz, Mathias and Lagus, Krista. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4(1):3, 2007.

G.E. Hinton, J.L. McClelland, D.E. Rumelhart. Distributed representations. *Parallel dis-tributed processing: Explorations in the microstructure of cognition*, 1(3):77–109, 1986.

Goldberg, Yoav and Levy, Omer. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014. URL http://arxiv.org/abs/1402.3722.

Harris, Zellig. Distributional structure. *Word*, 10(23):146–162, 1954.

Huang, Eric H, Socher, Richard, Manning, Christopher D, and Ng, Andrew Y. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pp. 873–882. Association for Computational Linguistics, 2012.

Katz, Slava M. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Singal processing*, volume ASSP-35, pp. 400–401, March 1987.

Le, Quoc V. and Mikolov, Tomas. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. URL http://arxiv.org/abs/1405.4053.

Liang, P. and Potts, C. Bringing machine learning and compositional semantics together. *Annual Reviews of Linguistics*, 1(1):355–376, 2015.

Luong, Minh-Thang, Socher, Richard, and Manning, Christopher D. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104, 2013.

Maas, Andrew L and Ng, Andrew Y. A probabilistic model for semantic word vectors. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Manning, Christopher D, Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL http://arxiv.org/abs/1301.3781.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc., 2013b.

Mikolov, Tomas, tau Yih, Wen, and Zweig, Geoffrey. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pp. 746–751. The Association for Computational Linguistics, 2013c.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L. (eds.), *Advances in Neural Information Processing Systems 21*, pp. 1081–1088. Curran Associates, Inc., 2009.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pp. 246–252. Citeseer, 2005.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1162.

Qi, Yanjun, Das, Sujatha G, Collobert, Ronan, and Weston, Jason. Deep learning for character-based information extraction. In *Advances in Information Retrieval*, pp. 668–674. Springer, 2014.

Santos, Cicero D. and Zadrozny, Bianca. Learning character-level representations for part-of-speech tagging. In Jebara, Tony and Xing, Eric P. (eds.), *Proceedings of the 31st International Conference on Machine*

Learning (ICML-14), pp. 1818–1826. JMLR Workshop and Conference Proceedings, 2014.

Sebastiani, Fabrizio. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002. ISSN 0360-0300. doi: 10.1145/505282.505283. URL http://doi.acm.org/10.1145/505282.505283.

Socher, Richard, Lin, Cliff C, Manning, Chris, and Ng, Andrew Y. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136, 2011.

Turian, Joseph, Ratinov, Lev, and Bengio, Yoshua. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 384–394. Association for Computational Linguistics, 2010.

Unnikrishnan, KP and Venugopal, Kootala P. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6 (3):469–490, 1994.

Waibel, Alexander, Hanazawa, Toshiyuki, Hinton, Geofrey, Shikano, Kiyohiro, and Lang, Kevin J. Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pp. 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-124-4.