Joint word2vec Networks for Bilingual Semantic Representations

Lior Wolf, Yair Hanani, Kfir Bar, and Nachum Dershowitz

Tel Aviv University

Abstract. We extend the word2vec framework to capture meaning across languages. The input consists of a source text and a word-aligned parallel text in a second language. The joint word2vec tool then represents words in both languages within a common "semantic" vector space. The result can be used to enrich lexicons of under-resourced languages, to identify ambiguities, and to perform clustering and classification. Experiments were conducted on a parallel English-Arabic corpus, as well as on English and Hebrew Biblical texts.

1 Introduction

Semantic models of languages map words, or word senses, in the language under study to some space in which semantic relationships between them can be observed and measured. Within such models, one would expect synonymous words to appear "near" one another. If the model incorporates more than one language, then the translation of a word should also reside nearby.

Word2vec [16] is a recently developed technique for building a neural network that maps words to real-number vectors, with the desideratum that words with similar meanings will map to similar vectors. One problem with a straightforward learning scheme from words and their context is polysemy, since the resultant vector will be an average of the different senses of the word.

Some languages are more ambiguous than others. There are many homographs in English, fewer in Spanish (a more phonetic language than English), and many more in (unvocalized) Hebrew. Sometimes homographs are also homophones, being pronounced the same (e.g., "bank"); other times they are heteronyms that are distinguished in speech (e.g., "dove"). We show how to leverage word-aligned parallel texts to improve the quality of the inferred model.

In the next section, we provide some background on the use of bilingual models for single-language tasks and to aid in machine translation. Section 3 contains an explanation of our bilingual extension to the word2vec model builder. It is followed by a section on experimental results for word-aligned English-Arabic and English-Hebrew corpora and then a brief discussion of the implications.

2 Background

Word sense disambiguation [13] is a difficult problem, partly because of the paucity of sense-tagged training data in most languages. Languages like Ara-

bic and Hebrew pose a bigger challenge because of their unvowelized and undiacritized writing. In English and other European languages, senses are represented, in particular, by WordNet [18]. Although similar WordNet repositories exist for Arabic [5] and Hebrew [21], they are quite limited in scope. The only available sense-tagged corpus in Hebrew is the Bible. For Arabic, there is OntoNotes [12], a relatively large corpus of various genres, tagged for syntax and shallow semantic attributes, including word senses.

It is well known that there is little agreement between dictionaries as to the division of words into their senses. Some distinctions are subtle and often cross language boundaries, so may not be considered distinct senses by other lexicographers; others are substantive and are unlikely to share the same words in multiple languages. Accordingly, it has been argued (e.g., [6,9,8]) that sense distinctions can be derived from co-occurrence statistics across languages. To quote [13]: "Homograph distinctions do not require a lexicographer to locate them, since they are basically those that can be found easily in parallel texts in different languages".

In [17], the authors showed how one can train word2vec independently in two languages, then use some anchor points between the two languages (a limited number of corresponding words in the two languages) to find an affine mapping from one language model to the other. The hope was that the resultant spaces can be used to enrich the dictionary by looking in the model of the second language for the word closest to the position of a word from the first language.

3 word2vec

Word2vec belongs to the class of methods called "neural language models". Using a scheme that is much simpler than previous work in this domain, where neural networks with many hidden units and several non-linear layers were normally constructed (e.g., [3]), word2vec [16] constructs a simple log-linear classification network [19]. Two such networks are proposed: the Skip-gram architecture and the Continuous Bag-of-words (CBOW) architecture. While the Skip-gram architecture is sometimes preferable, in our experience – probably due to the relatively small corpora we use, the CBOW architecture outperforms it. Therefore, we limit our exposition to the latter architecture.

3.1 Continuous Bag-of-Words Architecture

In the CBOW variant of word2vec, the network predicts each word based on its neighborhood – the five words preceding and the five words following that word. An input layer denotes the bag of words representation of the surrounding words, and contains one input element per glossary word. It is projected linearly to the hidden encoding layer. The hidden layer in then mapped to an output Huffman code representation of the given word. Once the network is trained, the projections from each input unit to the middle encoding layer are used to represent the associated glossary word. Interestingly, the resulting encoding not

only captures meaningful word representations, where (unambiguous) words of similar meaning have nearby representations, but also captures, in a surprising manner, pairwise relations through simple arithmetic operations [16].

Next, we formalize the CBOW architecture and introduce the notation used throughout the paper. Word2vec models are learned from an input corpus. Every word that occurs at least three times in the corpus becomes part of the glossary. Let D be the size of the glossary. The goal of the word2vec procedure is to associate each of the D glossary words with a vector representation in $\mathbb{R}^L$. In all of our experiments, L is fixed at a value of 200.

Each training example i is an occurrence of a glossary word $w(i)$ in the corpus. During the learning phase, the CBOW network is trained to map (through a hidden projection layer of size L) binary input vectors B_i^{in} of length D, to the matching output vectors B_i^{out} of length $\lg D$. The input vectors contain one element corresponding to each glossary entry, which denotes the existence of each glossary entry within the small neighborhood of size $2N$ surrounding example i in the text. In our experiments the window-size parameter N is fixed to be 5.

The binary output vector $B_i^{\text{out}} = H_{w(i)}$ is the Huffman code of the associated glossary entry. The Huffman codes H_j, $j = 1 \,..\, D$ of length $\lg D$ are constructed by considering the frequency of each glossary entry j. The use of Huffman codes follows a common practice in neural net language models [15]. We adhere to this practice, although its main advantage over encoding through balanced trees is to reduce evaluation time through an early stopping mechanism. This consideration is not a major concern in our system, where D is relatively small.

As mentioned above, the vector representation V_j of each glossary word, $j = 1 \,..\, D$, is its projection to the hidden layer; that is, it is the array of weights used to project the corresponding bit in the input vector B^{in} to the hidden layer. Intuitively, it captures the contribution of each glossary word to a layer that enables a relatively effective way of predicting the output vector. This is a semantic task, and the representation is therefore semantic. Also, by design, since linear projections are summed to capture local neighborhoods, the representation behaves well under addition.

3.2 Bilingual Extension

Very recently, a method was proposed for employing word2vec in the bilingual context [17]. The proposed method trains two word2vec models independently and then, using a seed list of pairs of words and their translation, the two spaces are aligned.

More specifically, assume that word w_k in the first language is translated as word w'_k in the second, $k = 1 \,..\, t$, where t is size of the seed set. Let the vectors $V_1, V_2, \ldots, V_D$ be the learned word2vec representations in the first language, and $V'_1, V'_2, \ldots, V'_{D'}$ the learned vectors of the second. A transformation T is then learned from the space of the second representation to the space of the first,

such that the following measure is minimized:

$$\sum_{k=1}^{t} \left\| TV'_{w'_k} - V_{w_k} \right\|^2 .$$

This is a linear least squares problem. In [17] stochastic gradient descent was used to solve for the transformation T. In our experiments we employ conventional linear solvers.

4 Joint word2vec

The bilingual method of [17] is motivated by the observation that when visualizing the word vectors using PCA, the vector representations of similar words in different languages were related by a linear transformation. However, since there is no guarantee that the two spaces are indeed related by a linear transformation, it could be suboptimal to learn each space separately and then try to align the two. In addition, if the corpora used to train the two vector spaces are related, it might be beneficial to use one language to improve the representation of the other, for example, in cases of word sense ambiguities.

In our method, *joint word2vec*, two versions of the same text are used for training. One of the versions is set to be the baseline version, and the other is aligned to it using an HMM-based word-alignment model [20]. The result, for a specific text (which will be used in our next examples as well) is illustrated in Fig. 1. In this example, a Hebrew verse from Genesis is used as the baseline text, and the English King James translation is mapped to it. Some of the words in the English text are not mapped to any Hebrew words and vice versa.

The joint word2vec model is a variant of the word2vec model with some significant modifications in both structure and training procedure that allow for simultaneous learning of unified representations for both languages. The input layer of the joint model corresponds to a union of the two dictionaries. Let the glossary of the first (second) language, consisting of frequent-enough words, be of size D (D', respectively). The input layer of the joint architecture is of size $D+D'$, where the first D (last D') elements correspond to the entries of the first (second) glossary. The output layer is of size $\lg(D+D')$, and encodes the words of the two dictionaries based on their frequencies in the bilingual corpora. The hidden layer and the log-linear model structure remain as in the conventional word2vec model.

The training procedure is illustrated in Fig. 2. It consists of four stages for each input occurrence i of the first language. First, as in the conventional word2vec, the neighborhood of i (first language) is used to predict word i. Then, the same process is repeated for the second-language word that is aligned to i, using the second-language words that are aligned with the first-language neighborhood of i. Naturally, this only occurs when there is a second-language word aligned with word i. Also, the aligned neighborhood could be smaller than the first-language neighborhood since not all first-language words are mapped to the second language.

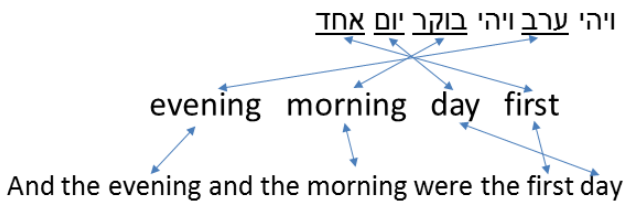

Fig. 1. A short alignment example (Genesis 1:5), which is also used in Fig. 2. The Hebrew verse (top row) is the baseline text to which the English verse at the bottom row is aligned. The words in the middle are those words that were found by the HMM model to match the Hebrew words they are connected to with an arrow.

These first two stages train two models using the same hidden layer. However, the two models are not tied together: each is using only the part of the input vectors B^{in} that corresponds to just one language. The next two stages tie the models together at the semantic level. In these stages, a word from one language is being predicted using the neighborhood in the other language, which helps make the model language-agnostic. First, the aligned word of the second language is predicted based on the neighborhood of word i in the first language. The input vector B^{in} is derived from data in the first language, and the output vector B^{out} is derived from second language data. Lastly, the process is reversed: the aligned second language neighborhood is used to predict and output vector B^{out} that is the Hoffman coding of word i in the first language.

Since training of the joint word2vec is four times slower than training of the conventional method, we modify the training procedure slightly. Whereas the original method employs stochastic gradient descent, which updates the network after observing each training samples, we introduced "mini-batches" and update the weights only after a small number of training examples. In between updates, the gradient is accumulated but not used to perform updates.

5 Evaluation

We compare the performance of the proposed joint word2vec to the performance of conventional word2vec and the performance of the bilingual model of [17] (Sect. 3.2). Given two texts, where the second is aligned to the first, the transformation of this bilingual method is estimated using all the words of the first text and their alignment, when they exist. Each pair is used once, regardless of the number of its occurrences.

Three benchmarks have been devised: (i) a *translation* benchmark that looks to see how close to each other a word and its translation are; (ii) a *homograph* benchmark; and (iii) a single-language *synonym* benchmark. The benchmarks are applied on two datasets, except for the third task, which is only applied to one dataset.

5.1 Datasets

In our experiments, as a second language, we used (transliterated) Arabic and Hebrew – both highly inflected languages. Among other interesting characteristics, Semitic languages in general are based on complicated derivational as well as inflectional morphologies. Furthermore, the lack of short vowels in writing increases the level of ambiguity of the written word. Words are derived from a root and a pattern (template), combined with prefixes and suffixes. The root consists of 3 or 4 consonants and the pattern is a sequence of consonants and variables for root letters. Using the same root with different patterns may yield words with different meanings. Words are then inflected for person, number and gender; proclitics and enclitics are added to indicate definiteness, conjunction, various prepositions, and possessive forms. On account of this morphological complexity, a single Arabic or Hebrew word often translates into several English words; for example, the English translation of the Arabic word *wbbytyn* is "and in his two houses".

Our first benchmark uses Arabic news stories that we aligned on the word level with their English sources. Overall, we have about 4M Arabic words. Due to Arabic's rich morphology, we preprocessed the Arabic text with MADA [11], a context-sensitive morphological analyzer that works on the word level, and then used TOKAN [11] to tokenize the text, additionally separating the definite article (*Al*) from nouns (and modifiers) and the future particle (*s*) from verbs. For example, the word *wbbyth*, "and in his house", gets tokenized like this: *w+ b+ byt +h*, each token respectively translated into: "and", "in", "house", and "his". Alignment proceeds on this token level.

The second dataset was constructed by aligning the Bible, in Hebrew, to its King James translation into English. The two resources are already aligned at the verse level, and contain 23,145 verses. The Hebrew text, like in Arabic, was tokenized using a context-sensitive morphological analyzer and a tokenizer. We used the MILA [14] morphological analyzer, with probabilities obtained using EM-HMM training [1,10]. The Hebrew tokenization scheme is similar to the one for Arabic. Both resources were aligned on the token level using GIZA++ [20], an implementation of the IBM word alignment models [7]. GIZA++ is a popular statistical machine translation toolkit that includes an HMM-based word-alignment model. It uses Baum-Welch training, and includes smoothing techniques for fertility (multiple-word translations of a single word).

For the synonym benchmark, we obtained the list of Biblical synonyms used in [2]. That list was compiled in the following manner: The King James translation of the Bible almost invariably translates synonyms identically. So, one can generally identify Hebrew synonyms by considering their translations. Word senses were derived from Strong's 1890 concordance [22], which lists separately every occurrence of each sense of each root that appears in the Bible. (Some anomalies due to polysemy in English were manually removed from the list by a Bible scholar.) The procedure yielded 529 synonym sets, ranging in size from 2 to 7 ("fear"), for a total of 1595 individual synonyms.

Table 1. The performance of various word2vec networks in the translation benchmark, in which distances of normalized vector representations of word pairs that are uniquely aligned between the two languages are compared. For the Bible (Heb) Dataset, the English text is aligned to the Hebrew one, and vice versa for Bible (Eng). Since the alignment is not symmetric, the pairs of words used for benchmarking differ between the two directions. The baseline algorithm [17] (Sect. 3.2) runs conventional word2vec on each corpus separately and then estimates a linear transformation that minimizes the L2 distance between the vector representations of aligned words. It is evaluated for both alignment directions. The joint word2vec algorithm is applied to both corpora at once. Shown are the mean distance between vector pairs normalized to have a norm of one and the standard error of these distances. A lower distance indicates greater proximity of the vector representations of translated words.

Dataset	Method	Normalized Distance ± Standard Error
News	2×Vanilla W2V (Eng and Arab mapped to Eng space)	1.1534 ± 0.0000
	2×Vanilla W2V (Arab and Eng mapped to Arab space)	1.1505 ± 0.0000
	Joint W2V (Eng+aligned Arab)	1.1644 ± 0.0000
Bible (Heb)	2×Vanilla W2V (Heb and Eng mapped to Heb space)	0.9739 ± 0.0002
	2×Vanilla W2V (Eng and Heb mapped to Eng space)	1.0066 ± 0.0002
	Joint W2V (Heb+aligned Eng)	0.9710 ± 0.0002
Bible (Eng)	2×Vanilla W2V (Eng and Heb mapped to Eng space)	0.8900 ± 0.0005
	2×Vanilla W2V (Heb and Eng mapped to Heb space)	0.8543 ± 0.0005
	Joint W2V (Eng+aligned Heb)	0.8790 ± 0.0006

5.2 Experiments

The translation benchmark. For each of the two datasets we identify pairs of glossary words that are uniquely mapped between the two texts, that is, we extract a list of words in the first text that are consistently aligned by GIZA++ to the same word in the second. Ideally, the vector representations of the words in each pair would be as close as possible; indeed the bilingual method of [17] directly minimizes the distance between the representations of words that are mapped between the two languages.

The distance computed in the first benchmark is measured for our method in the joint space, and in the method of [17] in each one of the spaces, after the computed linear transformation is applied. In word2vec experiments [16], cosine similarity (dot product of the normalized vectors) is often used. We comply with this by computing the distance between the normalized vectors, which also makes the comparison of distances between the various vector spaces valid.

The results of this benchmark are depicted in Table 1. Sometimes the joint word2vec outperforms the baseline algorithm and sometimes it is the other way around. This is remarkable given that the score that the baseline algorithm optimizes is directly related to the score employed in the benchmark and that the baseline algorithm is applied twice.

The homograph benchmark. We now consider the words in one language that are mapped to multiple words in the second language. This can be (i) a result of synonyms in the second language, or, (ii) as is almost always the case when aligning the Hebrew Bible to the King James edition, a result of homographs in the first language. For the first case, we would like to have the vector representation of the word in the first space as close as possible to that of the vector representations in the second space, and the vector representations in the second space as close as possible. In the second case, since the model is additive, it makes sense to expect the vector representation of the word in the first language to be a linear combination of the vector representations of the vectors in the second language, that is, to be spanned by those vectors.

Since we have no means to determine which of the two cases is the source of the multiple glossary word alignments, and since the spanning criteria is also approximately satisfied in case two vectors are similar, this criteria is the basis of our success score. Let V_i be the vector representation of the word in the first language, and let $V_i^{'(k)}$, $k = 1 \ldots n$ be the $n > 1$ words that are aligned to it throughout its occurrences in the dataset. The score employed in this benchmark is the reconstruction error:

$$\left\| \frac{V_i}{\|V_i\|} - \sum_{k=1}^{n} \lambda_k V_i^{'(k)} \right\| ,$$

where the λ_k are the coefficients that minimize this error. The minimization is obtained by solving (least squares) a linear set of $L = 200$ equations in these n unknowns. Note again, that in order to adhere to previous work with word2vec, and to allow a fair comparison between different vector spaces, the normalized version of V_i is used.

Table 2 depicts the results of this benchmark. The joint word2vec method clearly outperforms the baseline method, and the difference in performance is significant at an extremely low p-value in both t-test and Wilcoxon signed rank test. We also report results for another variant of the reconstruction error in which the weights λ_k are constrained to be positive. This is motivated by the nature of the word2vec architectures, in which, during training, projections are added but never subtracted. Naturally, the added positiveness constraint increases the reconstruction error, but this increase seems to be moderate, and in our experiments the order of the scores obtained by the various algorithms does not change.

The synonym benchmark. Lastly, we evaluate the quality of the learned word2vec representation using the Hebrew list of biblical synonyms. Eighty percent of the identified synonym pairs were included in our glossary, which contains only words that appear at least three times. For each of these we employ the distance between the normalized vector representations of the two words as the score.

This is a single language task and joint word2vec would outperform the baseline method only if learning jointly two languages successfully utilizes the information in the second language to improve the vector space of the first. The

Table 2. The performance of various word2vec networks in the homograph benchmark, in which the glossary words in the first language that are aligned to multiple words in the second language are employed. The reconstruction error is the distance between the L2 normalized vector representation in the first language to the linear space spanned by the vector representations in the second language; lower is better. Also shown is the reconstruction error when the combination weights λ_k used to compute the linear combination are constrained to be positive. Due to the additional constraint, the reconstruction errors are aways bigger; however, qualitatively the results remain the same. See text and the caption of Table 1 for more details.

Dataset	Method	Reconstruction Err ± Standard Error	Reconstruction Err ± Standard Error (Positive Weights)
News	2×Vanilla W2V (Eng + transformed Arab)	0.9387 ± 0.0000	0.9469 ± 0.0000
	2×Vanilla W2V (Arab + transformed Eng)	0.9321 ± 0.0000	0.9388 ± 0.0000
	Joint W2V (Eng+aligned Arab)	0.8893 ± 0.0000	0.9000 ± 0.0000
Bible (Heb)	2×Vanilla W2V (Heb and Eng mapped to Heb space)	0.8063 ± 0.0001	0.8477 ± 0.0001
	2×Vanilla W2V (Eng and Heb mapped to Eng space)	0.8410 ± 0.0001	0.8797 ± 0.0000
	Joint W2V (Heb+aligned Eng)	0.7462 ± 0.0001	0.7613 ± 0.0001
Bible (Eng)	2×Vanilla W2V (Eng and Heb mapped to Eng space)	0.9033 ± 0.0002	0.9228 ± 0.0002
	2×Vanilla W2V (Heb and Eng mapped to Heb space)	0.8173 ± 0.0004	0.8696 ± 0.0004
	Joint W2V (Eng+aligned Heb)	0.6610 ± 0.0006	0.6896 ± 0.0006

results in Table 3 suggest that this is indeed the case. A significant improvement is obtained in the joint word2vec method in comparison to the baseline method.

6 Discussion

In the experiments we conducted, both datasets are relatively small compared to the sizes typically employed to learn representations in word2vec models. Larger bilingual datasets are available; however, the processing required by the GIZA++ software for these datasets could not fit into our schedule. Fortunately, the joint word2vec software itself proved efficient enough and scales equally well as the open-source word2vec, upon which code base it is built. The up-side is that our experiments are the first to explore the utilization of word2vec on a scale possible for under-resourced languages, in which digitized texts and translations are scarce. It is interesting that, at least at these scales, joint word2vec representa-

Table 3. The performance of various word2vec networks on the Bible-Synonyms benchmark. The distance between the normalized representations of each pair of Hebrew synonyms is computed. As can be seen, the joint word2vec outperforms the baseline method, although this is a single-language task. The results in the two last rows are added for completeness, as they are expected to be worse. It is interesting to note that even learning joint word2vec with Hebrew aligned to English, which processes the Hebrew text in a way that is likely to disrupt its structure, seems to outperform the baseline method.

Method	Normalized Distance ± Standard Error
Vanilla W2V (Heb)	1.1864 ± 0.0013
Joint W2V (Heb+aligned Eng)	1.0637 ± 0.0015
2×Vanilla W2V (Heb mapped to Eng space)	1.2396 ± 0.0009
Joint W2V (Eng+aligned Heb)	1.1073 ± 0.0015

tions outperform vanilla word2vec learned vectors even at tasks requiring just one language.

While our exposition is focused on the bilingual case, it seems straightforward to generalize it to multiple languages. In fact, with little modifications, the joint word2vec system can employ heterogeneous inputs to learn a unified "Babel fish" semantic vector representation for a union of multiple dictionaries. During training, the weights can be updated from unilingual texts (conventional word2vec), bilingual texts (as done here), or aligned multi-lingual texts, each time updating the projection weights of the current word with accordance to the available data.

As an effective method to learn representations, word2vec is often seen as part of the "deep learning" trend, in which neural networks containing multiple layers and multiple non-linear transformations are used to learn state-of-the-art representations in domains such as computer vision and speech processing. While there is nothing deep in the word2vec architecture, by comparing it to deeper representations, the creators of the original architecture have been able to demonstrate that the simplest log-linear models are "as deep as is necessary", thereby complying with the road plan set forth by [4].

Still, the word2vec representation is wanting in some respects. For example, it would be desirable that homographs would be represented, not by one vector, but by a set of vectors. In the bilingual context, the automatic identification of homographs seems plausible. Recall that words in one language that are aligned with multiple words in the second language are either homographs or are translated to a set of synonyms. As discussed in Sect. 5.2, the set of synonyms should form a tight cluster, which can be easily measured, for example, by computing the mean distance to the centroid of the set of vectors representing the aligned words.

References

1. Adler, M.: Hebrew Morphological Disambiguation: An Unsupervised Stochastic Word-based Approach. PhD thesis, Ben-Gurion University (2007)
2. Akiva, N.: Style-Based Analysis of Natural Language Using Lexical Features. PhD thesis, Dept. Computer Science, Bar-Ilan Univ. (November 2013)
3. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3** (March 2003) 1137–1155
4. Bengio, Y., LeCun, Y.: Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., Weston, J., eds.: Large Scale Kernel Machines. MIT Press (2007)
5. Black, W., Elkateb, S., Vossen, P.: Introducing the Arabic WordNet project. In: Proceedings of the Third International WordNet Conference (GWC-06. (2006)
6. Brown, P.F., Della Pietra, S.A., Della Pietra, V.J., Mercer, R.L.: Word-sense disambiguation using statistical methods. In: Proceedings of the 29th Annual Meeting on Association for Computational Linguistics. ACL '91, Stroudsburg, PA, USA, Association for Computational Linguistics (1991) 264–270
7. Brown, P.F., Della-Pietra, S.A., Della-Pietra, V.J., Mercer, R.L.: The mathematics of statistical machine translation. Computational Linguistics **19**(2) (1993) 263–313
8. Dagan, I., Itai, A.: Word sense disambiguation using a second language monolingual corpus. Computational Linguistics **20**(4) (1994) 563–596
9. Gale, W., Church, K., Yarowsky, D.: Using bilingual materials to develop word sense disambiguation methods. In: Proceedings of the International Conference on Theoretical and Methodological Issues in Machine Translation. (1992) 101–112
10. Goldberg, Y., Adler, M., Elhadad, M.: EM can find pretty good HMM POS-taggers (when given a good start). In: Proceedings of ACL-08: HLT, Columbus, OH, Association for Computational Linguistics (June 2008) 746–754
11. Habash, N., Rambow, O., Roth, R.: MADA+TOKAN: A toolkit for arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization. In Choukri, K., Maegaard, B., eds.: Proceedings of the Second International Conference on Arabic Language Resources and Tools, Cairo, Egypt, The MEDAR Consortium (April 2009)
12. Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., Weischedel, R.: Ontonotes: The 90% solution. In: Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers. NAACL-Short '06, Stroudsburg, PA, USA, Association for Computational Linguistics (2006) 57–60
13. Ide, N., Wilks, Y.: Making sense about sense. In Agirre, E., Edmonds, P., eds.: Word Sense Disambiguation. Volume 33 of Text, Speech and Language Technology. Springer Netherlands (2006) 47–73
14. Itai, A., Wintner, S.: Language resources for Hebrew. Language Resources and Evaluation **42**(1) (March 2008) 75–98
15. Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., Khudanpur, S.: Extensions of recurrent neural network language model. In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. (2011) 5528–5531
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
17. Mikolov, T., Le, Q.V., Sutskever, I.: Exploiting similarities among languages for machine translation. CoRR **abs/1309.4168** (2013)
18. Miller, G.A.: Wordnet: A lexical database for English. Communications of the ACM **38** (1995) 39–41

19. Mnih, A., Hinton, G.: Three new graphical models for statistical language modelling. In: Proceedings of the 24th International Conference on Machine Learning. ICML '07, New York, NY, USA, ACM (2007) 641–648
20. Och, F.J., Ney, H.: A systematic comparison of various statistical alignment models. Computational Linguistics **29**(1) (2003) 19–21
21. Ordan, N., Wintner, S.: Hebrew WordNet: A test case of aligning lexical databases across languages. International Journal of Translation **19**(1) (2007) 39–58
22. Strong, J.: The Exhaustive Concordance of the Bible, Nashville (1890)

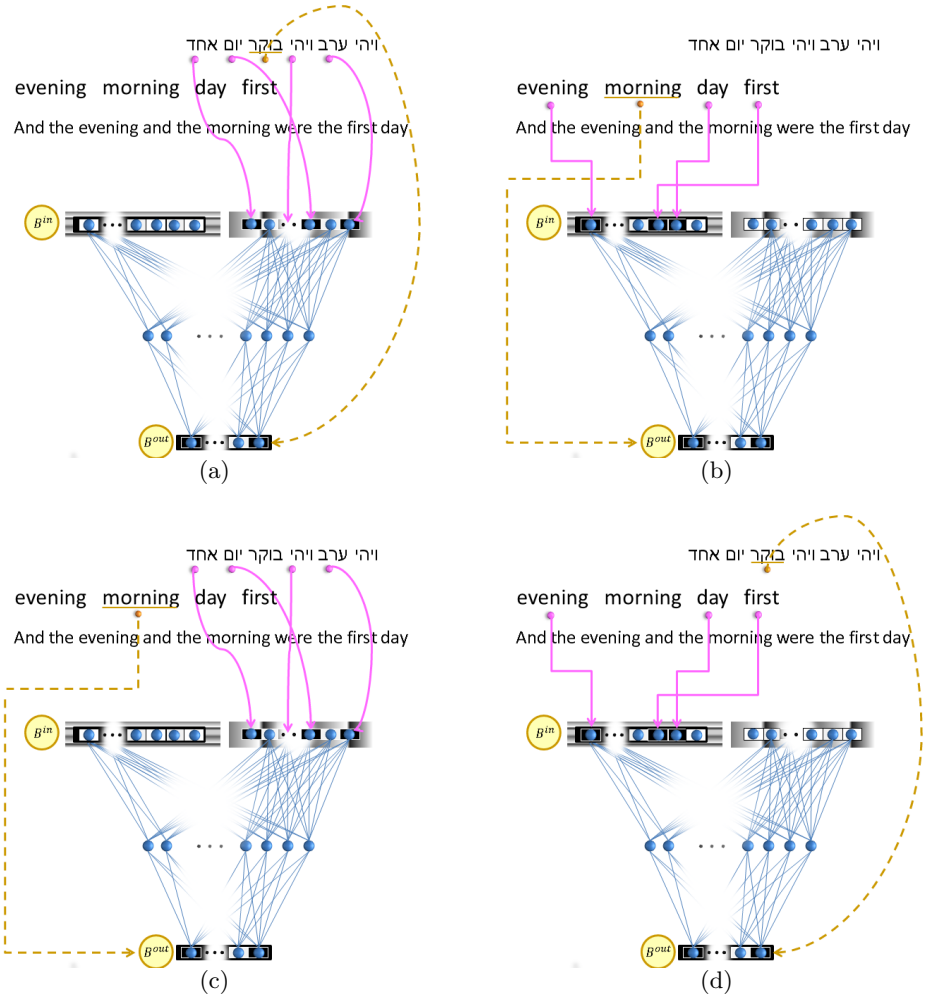

Fig. 2. An illustration of the joint word2vec network applied to a sample Hebrew/English verse during training. The input layer has two parts, corresponding to the two vocabularies. During training there are four steps (a–d): (a) Word i of the Hebrew text is predicted, that is, encoded at the output layer B_i^{out} in accordance to its Huffman encoding $H_{w(i)}$. The Hebrew neighborhood of word i is encoded through a bag-of-words scheme as the input layer B_i^{in}. The hidden layer is a linear projection layer and the entire architecture is a simple log-linear one. (b) Using the same architecture and the same hidden units, the process is repeated for predicting the English word that corresponds to the Hebrew word i from the English words that are mapped to the neighborhood of i. Note that this neighborhood typically differs from the neighborhood in the English text and is often smaller than the Hebrew neighborhood since some Hebrew words are not aligned to any English words. (c) The Hebrew neighborhood is used to predict the English word that matches word i. (d) The English neighborhood (see (b)) is used to predict the Hebrew word i.