UWB at KSAA-RD shared task: Computing the meaning of a gloss

Stephen Taylor
University of West Bohemia
taylor@ntis.zcu.cz

Abstract

To extract the 'meaning' of a gloss phrase, we build a list of sense-IDs for each word in the phrase which is in our vocabulary. We choose one sense-ID from each list so as to maximise similarity of all the IDs in the chosen subset. We take the meaning of the phrase in semantic space to be the weighted sum of the embedding vectors of the IDs.

1 Introduction

The KSAA Reverse Dictionary shared task is to find the embedding vector for a word, given a gloss, or short definition. The two sub-tasks use Arabic and English glosses.

The task is partly inspired by SemEval-2022 Task 1 (Mickus et al., 2022), which provided training data for English, Spanish, French, Italian, and Russion. For that task there was an additional subtask, generating a gloss from a word-embedding, which proved difficult to score; BLEU scores, which are typically used to measure the success of machine translations, seemed to perform poorly to compare the adequacy of generated glosses.

The individual records in the training data for the task provide: an ID, which corresponds to a particular sense of a word; the word; a gloss or definition; and two embedding vectors, a SGNS and an Electra vector. The SGNS vector is a word-based skipgram vector, so that each sense of the word has the same vector; the Electra vector is a transformer context-based vector, and each ID has a different vector.

Thus in all there are four subtasks: to find the SGNS or Electra semantic-vector from the Arabic or English words of the gloss. For neither language are there sufficient training data to completely populate the vectors of the gloss. There are several possible reasons for this problem. Considering just the problems for Arabic gloss phrases:

1. Some of the missing words are particles, which would probably appear in a list of stopwords; for these, the absence of a vector is a feature, not a problem.

2. Some missing words are due to the presence or absence of vowel and gemination marking. A correctly spelled Arabic word can appear with complete vocalization, with partial vocalization, or with no vocalization at all.

3. Several particles, including the prepositions ب , ف , the pronouns ه, ها , هم , كم , and the definite article ال , never stand alone, but always appear affixed to another word.

4. Arabic is an inflected language. Adjectives are inflected for gender, verbs for tense, number, person, and gender, nouns for number and case. Some of these inflections are regular, and others are not.

5. Some words just do not appear in the training data. The test words are unsurprising examples, but of course many others are also absent.

English has similar problems, but we spent more attention on Arabic.

An apparently obvious part of the solution might be to fine-tune a pre-trained transformer on the glosses, and then attempt to generate the word gloss by a transformation on the phrase embedding. This idea was used for baseline, and in the SemEval-2020 task by for example, (Li et al., 2022).

However, in the current task, we are restricted to less than fifty thousand training examples. Training a transformer on so little data seems problematic. Using an externally pre-trained transformer means bringing in external data, not in the spirit of a closed task.

We wanted to try a simple data-processing approach.

Proceedings of the The First Arabic Natural Language Processing Conference (ArabicNLP 2023), pages 461–466

Table 1: Data files and data

File name	# records	# IDs	# # words	mean(max) gloss words
ar.train.json	45200	45058	38498	7.03 (99)
ar.en.train.json	2862	2214	1697	11(65)
en.complete.with.id.json	63596	63596	26068	11.777 (129)

2 Data

We used the data files described in table 1, which were provided by the task organizers. The ar.train.json and the dev and test files are based on the LMF Contemporary Arabic Dictionary(Namly, 2015). The en.complete.with.id.json file is from the SemEval-2022 Task 1 data. Each entry in the train file is a definition for one sense of an Arabic word, with a short definition or gloss. The examples of usage found in the LMF dictionary, and sometimes part of the definition, have been dropped. Each of the IDs mentioned in Table 1 is a single sense. Although some senses are defined several times, only twenty percent of the words have more than one sense appearing in the train file. Although some definitions are quite long, one to three word definitions are quite common.

ar.en.train.json is used only to construct a cross-lingual transform(Artetxe et al., 2018; Brychcín et al., 2019) from the English embeddings of en.complete.with.id.json to the Arabic space of the ar.ae.train.json file. Since every entry in ar.ae.train.json has an enId attribute corresponding to an entry in en.complete.with.id.json, this does not change the amount of English data available for interpreting the gloss.

There are 35224 number of distinct tokens used for the English glosses, while the total English vocabulary of en.complete.with.id.json is only 26068 words. Doing a set subtraction, we see that the larger set contains many capitalized and inflected words, but also a number of words like *chat, majestic, dilemma, xanax, SiO2, inactivate*, that is, both technical terms, and relatively common words which happened not to be included in the dictionary at hand. Doing the subtraction the other way, we see that 15212 of our vocabulary words do not occur in the glosses.

We considered using a separate, larger English embedding, of which there are many available, with a cross-lingual transform, which could be easily prepared for the SGNS vectors based on common vocabulary. But it wasn't clear that 'open' as intended by the organizers included this option, and matching senses for the Electra embedding seems to be exactly the problem on which we are already working.

Similarly extending the Arabic vocabulary has the same problems, except that the organizers used the term 'closed' for subtask 1, which would clearly preclude doing it.

3 System

Our system[1] does not use a neural network. It uses ar.train.json for its Arabic vocabulary, and en.complete.with.it.json for its English vocabulary. It uses ar.en.train.json, which contains both Arabic and English words, in order to build a cross-lingual transform, so that the vectors built in the English space with English glosses can be converted to vectors in one of the Arabic spaces.

In addition to copying the ar.train.json sense dictionary, making a table of all the sense-IDs for each wordform, we also build a dictionary, swords, of derived values which points into that table. This includes several kinds of values:

- *Adjusted* Arabic words, with no vowels, only unmarked alifs ( ), all trailing yaas ( ) as alif maqsura ( ). This follows the convention adopted by Zahran et al. (Zahran et al., 2015a). This discards more information than probably necessary, but it works.

- Inflected verbs. The training files contain a part-of-speech field, and for one-word verbs we build *adjusted* inflected forms. Many of the verbs in the data come with indicated prepositions, given with an object pronoun. For these situations, we inflected the first of the two words in the definition. Sometimes this is *not* the verb, and as a result will never result in a meaningful match. We didn't build

[1] Code available at github.com/StephenETaylor/KSAA-RD

inflected nouns or adjectives during the test phase, nor did we consider one-letter prepositions, conjunctions or possessive or object pronouns.

- Single-word Arabic glosses. For this case we assume that the gloss is a near-synonym, and that the vocabulary is possibly increased by adding it as one.

Each of these substitute words points to one or more words for which we have a least one ID and embedding vector. [Although English offers similar possibilities, we did not build a substitution list for English before the end of the test period.]

3.1 Processing example

The processing loop for Arabic glosses builds a list of lists of possible IDs. For example, the first 'word' in `ar.dev.json` is تحنَّنَ عليه . (Although there are two words here, this is an example of the data file following the dictionary practice of providing the correct preposition to use for this sense of the word.)

The gloss for this word gives three synonyms, 'ترحَّم، تعطَّف عليه ورحمه'. Starting at the beginning, 'ترحَّم ' is not in the vocabulary, but 'ترحم ' is in the swords list, with three possible vocabulary items, ['رحِم يتيمًا ' , 'رحِم اللهُ فلانًا' , ' ترحَّم على صديقه']. The first of these, 'ترحَّم على صديقه' has two IDs, that is, two senses, ['ar.19347', 'ar.19348']. The second has the same two senses, and the third has the single sense ['ar.19344']. We combine these senses into a single list, and move to the second word of the gloss.

The second word is 'تعطَّف ', which has neither a dictionary entry or an swords entry, so we append nothing to our list of gloss IDs.

Continuing to the third word of the gloss, 'عليه ', it is not found in the dictionary, but there is an swords entry, [' عليه دَم ', ' عِلْيَة القوم ', ' عِلِّيَّة '], and these three entries each has a sense-ID, contributing in all ['ar.16839','ar.35831','ar.35683'], so that the possible gloss senses so far are [['ar.19347', 'ar.19348', 'ar.19344'],['ar.16839', 'ar.35831', 'ar.35683']].

The last word of the gloss, and the third synonym, is ' ورحمه ', which doesn't have a vocabulary entry, because it has both an object pronoun and a leading conjunction. It *should* have an swords entry, but we didn't implement those features, so it contributes nothing to the possible gloss IDs.

The next step is to choose the most-compatible IDs. The routine in our system which does this is called `maxids()`, and it is the major bottleneck in processing. For this case, we would need to check only the cosines between nine pairs of possibilities, but our routine can efficiently handle more complex cases. Instead of enumerating all the possible sets of IDs, it randomly chooses a starting point from the cross product of the possibilities, and changing one ID at a time, greedily descends to a local minimum. It repeats this process up to a hundred times, and returns the least of the minima it has encountered, as well as a list of the values each ID contributes to the sum.

For this example, `maxids` returns (['ar.19348', 'ar.16839'], [0.98, 0.98]) so that the angle between the IDs is a bit less than $\pi/3$. (This angle is for the Electra embedding.) Since there are only two IDs, and one angle, both IDs contribute equally. We use the angles to build weights for the vectors, with larger angles getting smaller weights. In this case both weights are equal.

Finally, we add the weighted vectors for ar.19348 and ar.16839 and normalize the result; this is our approximation to the vector for the gloss, and thus the best guess at the vector for the original word. Since this was from the `dev` file, and not the `test` file, we can compare the guess to the correct vector.

4 Results

user	MSE	cos	rank
Subtask 1 SGNS			
BASELINE	0.04922	0.26226	0.50167
bkhmsi	0.029 (1)	0.611 (1)	0.253 (1)
UWB	0.052 (2)	0.375 (3)	0.438 (3)
Ibraham Khurfan	0.065 (3)	0.394 (2)	0.308 (2)
SerrySibaee	- (4)	- (4)	- (4)
Subtask 1 Electra			
BASELINE	0.22105	0.5409	0.36222
bkhmsi	0.150 (1)	0.649 (1)	0.226 (1)
Ibraham Khurfan	0.236 (2)	0.519 (2)	0.281 (2)
SerrySibaee	0.236 (2)	0.519 (2)	0.281 (2)
UWB	0.266 (3)	0.416 (3)	0.466 (3)
Subtask 2 SGNS			
BASELINE	0.04922	0.26226	0.50167
UWB	0.046 (1)	0.217 (2)	0.489 (2)
bkhmsi	0.053 (2)	0.400 (1)	0.320 (1)
Subtask 2 Electra			
BASELINE	0.22105	0.5409	0.36222
bkhmsi	0.170 (1)	0.659 (1)	0.127 (1)
UWB	0.266 (2)	0.479 (2)	0.452 (2)

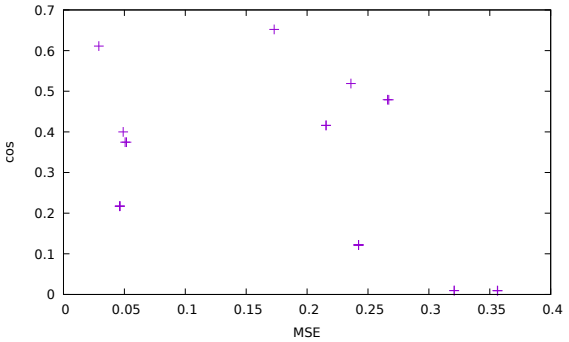

Figure 1: Scatter plot of cos vs MSE

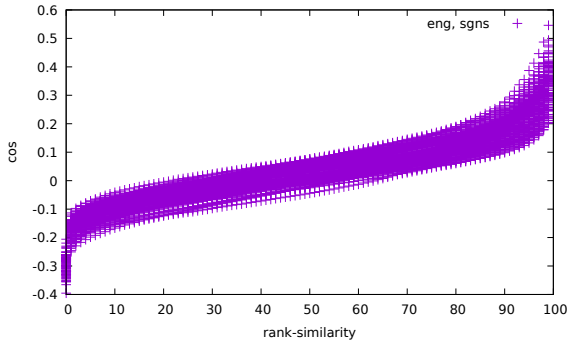

Figure 2: Scatter plot of cos versus rank-simlarity

Our system was second in rank [that is, last] for both SGNS and Electra on sub-task 2, and third and fourth on sub-task 1. It was slightly better than BASELINE on subtask 1, but not on subtask 2. Our Electra results are consistently worse than SGNS.

For individual normalized vectors, there is a straightforward relation between squared error and cosine:

$$SE = 2 - 2cos\theta \tag{1}$$

Where θ is the angle between vectors U and V and

$$SE = \sum_i (u_i - v_i)^2 \tag{2}$$

Neither SGNS nor Electra vectors were presented normalized, but the scoring code(AlShammari, 2023) shows the MSE computed on normalized vectors.

However, it is clear from our limited data that although MSE and mean cosine tend to move in opposite directions, systems with similar MSE can have very different mean cosines. See the graphed values for the leaderboard systems in Figure 1.

It's notable that MSE is generally lower for SGNS; a possible explanation is that the SGNS space has fewer vectors, so getting the sense wrong, but the word right, can happen more often.

Looking over our submissions, we deployed the vector weighting feature on August 20. Those runs were very slightly better than the runs on August 18, but typically only in the fourth or fifth digit of the rank measure.

5 Discussion

5.1 Similarity measures

The central idea in our system is to maximize the similarity of the senses in the gloss. Our measure of (dis)similarity, for each word-sense in the gloss, is the sum of the angles with all the other word-senses in the gloss.

We chose to minimize sum of angles, instead of maximize sum of cosines, because adding angles seems to make more intuitive sense than adding cosines. The best measure, and the one which would relate most closely to the rank-score on which the systems were measured, might be a rank-similarity, a measure of what fraction of the vocabulary is further from word-sense one than word-sense two is. Like cosine, this would have a best value of 1. We guessed that computing that rank would be quite a bit more expensive than computing the arc-cosine, and the maxids() routine which would call it is already the bottleneck in evaluating the gloss.

During the post-evaluation period, we tested precomputing tables of such ranks for each vector. Tables of 100 cosines, one at each percentile of rank, take up about a third of the amount of space used for the table of vectors. This seems like a reasonable amount of space; computing the values, which requires computing the cosine between all pairs of vectors, takes only about 17 minutes on a laptop, and can be done once and the (summarized, condensed) results saved to file. (The unsummarized results for 4.5E4 vectors would be 20E8 cosines or 8 GB as float32 values, an inconvenient size to cache.)

A scatter plot of cosine versus rank-similarity, showing the cached 100 points for 100 sample IDs, is shown in Figure 2. The graph illustrates that each vector has a slightly different curve, (although it seems possible that each curve could be described with a small number of parameters, probably much less than 100) and also hints at the fact that the rank-similarity is not symmetric: the rank-similarity between two vectors depends on the cosine between them, which is symmetric, and which

vector you use for counting the neighbors, which is not symmetric. If we draw a horizontal line at any cosine value in the graph, it is clear that depending on which of the package of curves we stop, we can get a wide range of possible values for the rank-similarity. Averaging rank-similarity measured in both directions *would* give a symmetric measure.

Our results with the rank-similarity scheme were a little worse than with the angle scheme. One apparent problem is that the exciting part of the rank-similarity is in the last percentile, and our implementation uses linear interpolation, which is least accurate at the two edges. (Note the ends of the curves in Figure 2.) A second problem is that, like cosines, but unlike angles, this measure has less relative change as the limit of 1.0 is approached. So a value of 90, corresponding to 4500 close words, is within 10% of 99.9, corresponding to 45 close words, which is a much more interesting value. In contrast, the nearest neighbor is often at an angle of $\pi/6$, while an angle of $\pi/3$ is likely to include about 1% of the closest words. The angle difference is larger exactly where we want it to be.

5.2 English glosses versus Arabic glosses

We spent more effort on Arabic words than English ones. Possibly more effort on English might have improved the `swords` list and given better vocabulary coverage for English glosses. In any case, the Arabic results for our system are currently better than the English ones.

5.3 SGNS vs Electra

Our system performs much better for SGNS than for Electra. An important reason is that all the SGNS vectors for the senses of a word are the same, and so when we encounter a word in a gloss, we automatically get the sense vector right. Our sword scheme adds possible synonyms without regard for the context in which they are encountered. We *could* use the `maxlis()` scheme on glosses to choose sense-IDs before adding any sword entries based on them, if we did this on a second pass over the data. However, one of the benefits of the sword scheme is that those added synonyms give better gloss coverage, so a third pass, etc., might also be indicated.

5.4 Gloss coverage

A primary problem with our approach (and probably for everyone) is dictionary coverage of the glosses. Many words in the glosses are not present in the training data. Our *simplified and substitute words* (`swords`) list tries to deal with this problem by adding inflected forms and some words from the glosses as aliases for training words defined with vectors. Both of these seem like reasonable ideas, but at present we still drop about 50% of the gloss words. A more thorough and systematic approach to adding aliases might have increased our success rate.

Using the development data as extra training data for the test phase would probably also have helped.

6 Conclusion

Our earnest thanks go out to the organizers, who prepared a substantial dataset for this workshop.

Although our approach was not completely successful, it did better than the baseline for Subtask 1, Arabic definitions with SGNS vectors.

We have discussed several variations in Section 5, some of which might improve the system performance on the other three variations of the task.

We look forward to seeing designs of other workshop participants.

References

Carl Allen and Timothy Hospedales. 2019. Analogies explained: Towards understanding word embeddings. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 223–231. PMLR.

Waad AlShammari. 2023. ArReverseDictionary github site.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Tomáš Brychcín, Stephen Taylor, and Lukáš Svoboda. 2019. Cross-lingual word analogies using linear transformations between semantic spaces. *Expert Systems with Applications*, 135:287–295.

Alex Gittens, Dimitris Achlioptas, and Michael W. Mahoney. 2017. Skip-gram - Zipf + uniform = vector additivity. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 69–76, Vancouver, Canada. Association for Computational Linguistics.

Bin Li, Yixuan Weng, Fei Xia, Shizhu He, Bin Sun, and Shutao Li. 2022. LingJing at SemEval-2022 task

1: Multi-task self-supervised pre-training for multilingual reverse dictionary. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 29–35, Seattle, United States. Association for Computational Linguistics.

Timothee Mickus, Kees Van Deemter, Mathieu Constant, and Denis Paperno. 2022. Semeval-2022 task 1: CODWOE – comparing dictionaries and word embeddings. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1–14, Seattle, United States. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*. arXiv1301.3781.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34:1388–1429.

Driss Namly. 2015. LMF contemporary arabic dictionary. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Stephen Taylor and Tomáš Brychcín. 2018. The representation of some phrases in arabic word semantic vector spaces. *Open Computer Science*, 8(1):182–193.

Mohamed A Zahran, Ahmed Magooda, Ashraf Y Mahgoub, Hazem Raafat, Mohsen Rashwan, and Amir Atyia. 2015a. Word representations in vector space and their applications for arabic. In *International Conference on Intelligent Text Processing and Computational Linguistics*, page 430–443. Springer.

Mohamed A. Zahran, Mohsen A. Rashwan, and Hazem Raafat. 2015b. Cross lingual lexical substitution using word representation in vector space. In *FLAIRS*.