Generalized Tuning of Distributional Word Vectors for Monolingual and Cross-Lingual Lexical Entailment

Goran Glavaš  
University of Mannheim  
Data and Web Science Group  
B6, 26, DE-68161 Mannheim, Germany  
goran@informatik.uni-mannheim.de  

Ivan Vulič  
PolyAI Ltd.  
144A Clerkenwell Road  
London, United Kingdom  
ivan@poly-ai.com  

Abstract

Lexical entailment (LE; also known as hyponymy-hypernymy or is-a relation) is a core asymmetric lexical relation that supports tasks like taxonomy induction and text generation. In this work, we propose a simple and effective method for fine-tuning distributional word vectors for LE. Our Generalized Lexical Entailment model (GLEN) is decoupled from the word embedding model and applicable to any distributional vector space. Yet – unlike existing retrofitting models – it captures a general specialization function allowing for LE-tuning of the entire distributional space and not only the vectors of words seen in lexical constraints. Coupled with a multilingual embedding space, GLEN seamlessly enables cross-lingual LE detection. We demonstrate the effectiveness of GLEN in graded LE and report large improvements (over 20% in accuracy) over state-of-the-art in cross-lingual LE detection.

1 Background and Motivation

Lexical entailment (LE; hyponymy-hypernymy or is-a relation), is a fundamental asymmetric lexico-semantic relation (Collins and Quillian, 1972; Beckwith et al., 1991) and a key building block of lexico-semantic networks and knowledge bases (Fellbaum, 1998; Navigli and Ponzetto, 2012). Reasoning about word-level entailment supports a multitude of tasks such as taxonomy induction (Snow et al., 2006; Navigli et al., 2011; Gupta et al., 2017), natural language inference (Dagan et al., 2013; Bowman et al., 2015; Williams et al., 2018), metaphor detection (Mohler et al., 2013), and text generation (Biran and McKeown, 2013).

Due to their distributional nature (Harris, 1954), embedding models (Mikolov et al., 2013; Levy and Goldberg, 2014; Pennington et al., 2014; Melamud et al., 2016; Bojanowski et al., 2017; Peters et al., 2018, inter alia) conflate paradigmatic relations (e.g., synonymy, antonymy, LE, meronymy) and the broader topical (i.e., syntagmatic) relatedness (Schwartz et al., 2015; Mrkšić et al., 2017). Consequently, distributional vectors (i.e., embeddings) cannot be directly used to reliably detect LE.

Embedding specialization methods remedy for the semantic vagueness of distributional spaces, forcing the vectors to conform to external linguistic constraints (e.g., synonymy or LE word pairs) in order to emphasize the lexico-semantic relation of interest (e.g., semantic similarity of LE) and diminish the contributions of other types of semantic association. Lexical specialization models generally belong to one of the two families: (1) joint optimization models and (2) retrofitting (also known as fine-tuning or post-processing) models. Joint models incorporate linguistic constraints directly into the objective of an embedding model, e.g., Skip-Gram (Mikolov et al., 2013), by modifying the prior or regularization of the objective (Yu and Dredze, 2014; Xu et al., 2014; Kiela et al., 2015) or by augmenting the objective with additional factors reflecting linguistic constraints (Ono et al., 2015; Osborne et al., 2016; Nguyen et al., 2017). Joint models are tightly coupled to a concrete embedding model – any modification to the underlying embedding models warrants a modification of the whole joint model, along with the expensive retraining. Conversely, retrofitting models (Faruqui et al., 2015; Wieting et al., 2015; Nguyen et al., 2016; Mrkšić et al., 2017; Vulić and Mrkšić, 2018, inter alia) change the distributional spaces post-hoc, by fine-tuning word vectors so that they conform to external linguistic constraints. Advantageously, this makes retrofitting models more flexible, as they can be applied to any pre-trained distributional space. On the downside, retrofitting models specialize only the vectors of words seen in constraints, leaving vectors of unseen words unchanged.

In this work, we propose an LE-specialization framework that combines the strengths of both
model families: unlike joint models, our generalized LE specialization (dubbed GLEN) is easily applicable to any embedding space. Yet, unlike the retrofitting models, it LE-specializes the entire distributional space and not just the vectors of words from external constraints. GLEN utilizes linguistic constraints as training examples in order to learn a general LE-specialization function (instantiated simply as a feed-forward neural net), which can then be applied to the entire distributional space. The difference between LE-retrofitting and GLEN is illustrated in Figure 1. Moreover, with GLEN’s ability to LE-specialize unseen words we can seamlessly LE-specialize word vectors of another language \(L_2\), assuming we previously project them to the distributional space of \(L_1\) for which we had learned the specialization function. To this end, we can leverage any from the plethora of resource-lean methods for learning the cross-lingual projection (function \(g\) in Figure 1) between monolingual distributional vector spaces (Smith et al., 2017; Conneau et al., 2018; Artetxe et al., 2018, *inter alia*).

Conceptually, GLEN is similar to the explicit retrofitting model of Glavaš and Vulić (2018), who focus on the symmetric semantic similarity relation. In contrast, GLEN has to account for the asymmetric nature of the LE relation. Besides joint (Nguyen et al., 2017) and retrofitting (Vulić and Mrkšić, 2018) models for LE, there is a number of supervised LE detection models that employ distributional vectors as input features (Tuan et al., 2016; Shwartz et al., 2016; Glavaš and Ponzetto, 2017; Rei et al., 2018). These models, however, predict LE for pairs of words, but do not produce LE-specialized word vectors, which are directly pluggable into downstream models.

### 2 Generalized Lexical Entailment

Following LEAR (Vulić and Mrkšić, 2018), the state-of-the-art LE-retrofitting model, we use three types of linguistic constraints to learn the general specialization function \(f\): synonyms, antonyms, and LE (i.e., hyponym-hypernym) pairs. Similarity-focused specialization models tune only the direction of distributional vectors (Mrkšić et al., 2017; Glavaš and Vulić, 2018; Ponti et al., 2018). In LE-specialization we need to emphasize similarities but also reflect the hierarchy of concepts offered by LE relations (e.g., *car* should be similar to both *Ferrari* and *vehicle* but is a hyponym only of *vehicle*). GLEN learns a specialization function \(f\) that rescales vector norms in order to reflect the hierarchical LE relation. To this end, we use the following asymmetric distance between vectors defined in terms of their Euclidean norms:

\[
d_N(x_1, x_2) = \frac{\|x_1\| - \|x_2\|}{\|x_1\| + \|x_2\|}
\]

Simultaneously, GLEN aims to bring closer together in direction vectors for synonyms and LE pairs and to push vectors of antonyms further apart. We use the cosine distance \(d_C\) as a symmetric measure of direction (dis)similarity between vectors. We combine the asymmetric distance \(d_N\) and symmetric \(d_C\) in different objective functions that we optimize to learn the LE-specialization function \(f\).

#### Lexical Constraints as Training Instances

For each constraint type – synonyms, antonyms, and LE pairs – we create separate batches of training instances. Let \(\{x_1^E, x_2^E\}_K\), \(\{x_1^S, x_2^S\}_K\), and \(\{x_1^A, x_2^A\}_K\) be the batches of \(K\) LE, synonymy, and antonymy pairs, respectively. For each constraint \((x_1, x_2)\) we create a pair of negative vectors \((y_1, y_2)\) such that \(y_1\) is the vector within the batch (except \(x_2\)), closest to \(x_1\) and \(y_2\) the vector closest to \(x_2\) (but not \(x_1\)) in terms of some distance or similarity metric. For LE constraints, we find \(y_1\) and \(y_2\) that minimize \(d_N(x_1, y_1) + d_C(x_1, y_1)\) and \(d_N(y_2, x_2) + d_C(x_2, y_2)\), respectively. Intuitively, we want our model to predict a smaller LE distance \(d_N + d_C\) for a positive LE pair \((x_1, x_2)\) than for negative pairs \((x_1, y_1)\) and \((x_2, y_2)\) in the specialized space. By choosing the most-challenging negative pairs, i.e., \(y_1\) and \(y_2\) that are respectively closest

---

1See (Ruder et al., 2018b; Glavaš et al., 2019) for a comprehensive overview of models for inducing cross-lingual word embedding spaces.
to \( x_1 \) and \( x_2 \) in terms of LE distance in the distributional space, we force our model to learn a more robust LE specialization function (this is further elaborated in the description of the objective function). Analogously, for positive synonym pairs, \( y_1 \) and \( y_2 \) are the vectors closest to \( x_1 \) and \( x_2 \), respectively, but in terms of only the (symmetric) cosine distance \( d_C \). Finally, for antonyms, \( y_1 \) is the vector maximizing \( d_C(x_1, y_1) \) and \( y_2 \) the vector that maximizes \( d_C(x_2, y_2) \). In this case, we want the vectors of antonyms \( x_1 \) and \( x_2 \) after specialization to be further apart from one another (according to \( d_C \)) than from, respectively, the vectors \( y_1 \) and \( y_2 \) that are most distant to them in the original distributional space. A training batch, with \( K \) entailment \((E)\), synonymy \((S)\), or antonymy \((A)\) instances, is obtained by coupling constraints \((x_1, x_2)\) with their negative vectors \((y_1, y_2): \{x_1, x_2, y_1, y_2\}^K\).

**Specialization Function.** The parameterized specialization function \( f(x; \theta) : \mathbb{R}^d \rightarrow \mathbb{R}^d \) (with \( d \) being the embedding size), transforms the distributional space to the space that better captures the LE relation. Once we learn the specialization function \( f \) (i.e., we tune the parameters \( \theta \)), we can LE-specialize the entire distributional embedding space \( X \) (i.e., the vectors of all vocabulary words): \( X' = f(X; \theta) \). For simplicity, we define \( f \) to be a (fully-connected) feed-forward net with \( H \) hidden layers of size \( d_h \) and non-linear activation \( \psi \). The \( i \)-th hidden layer \((i \in \{1, \ldots, H\})\) is parametrized by the weight matrix \( W^i \) and the bias vector \( b^i \):

\[
h^i(x; \theta) = \psi\left(h^{i-1}(x; \theta_{i-1})W^i + b^i\right)
\]  

(2)

**Objectives and Training.** We define four losses which we combine into training objectives for different constraint types \((E, S, \text{ and } A)\). The asymmetric loss \( l_a \) forces the asymmetric margin-based distance \( d_N \) to be larger for negative pairs \((x_1, y_1)\) and \((y_2, x_2)\) than for the positive (true LE) pair \((x_1, x_2)\) by at least the margin \( \delta_a \):

\[
l_a = \sum_{k=1}^{K} \tau\left(\delta_a - d_N(f(x^k_1), f(y^k_1)) + d_N(f(x^k_1), f(x^k_2))\right) + \tau\left(\delta_a - d_N(f(y^k_2), f(x^k_2)) + d_N(f(x^k_1), f(x^k_2))\right)
\]  

(3)

where \( \tau(x) = \max(0, x) \) is the ramp function. The similarity loss \( l_s \) pushes the vectors \( x_1 \) and \( x_2 \) to be direction-wise closer to each other than to negative vectors \( y_1 \) and \( y_2 \), by margin \( \delta_s \):

\[
l_s = \sum_{k=1}^{K} \tau\left(\delta_s - d_C(f(x^k_1), f(y^k_1)) + d_C(f(x^k_1), f(x^k_2))\right) + \tau\left(\delta_s - d_C(f(x^k_2), f(y^k_2)) + d_C(f(x^k_1), f(x^k_2))\right)
\]  

(4)

The dissimilarity loss \( l_d \) pushes vectors \( x_1 \) and \( x_2 \) further away from each other than from respective negative vectors \( y_1 \) and \( y_2 \), by the margin \( \delta_d \):

\[
l_d = \sum_{k=1}^{K} \tau\left(\delta_d - d_C(f(x^k_1), f(x^k_2)) + d_C(f(x^k_1), f(y^k_1))\right) + \tau\left(\delta_d - d_C(f(x^k_2), f(x^k_1)) + d_C(f(x^k_1), f(y^k_1))\right)
\]  

(5)

We also define the regularization loss \( l_r \), preventing \( f \) from destroying the useful semantic information contained in distributional vectors:

\[
l_r = \sum_{k=1}^{K} d_C\left(x^k_1, f(x^k_1)\right) + d_C\left(x^k_2, f(x^k_2)\right)
\]  

\[
+ d_C\left(y^k_1, f(y^k_1)\right) + d_C\left(y^k_2, f(y^k_2)\right)
\]  

(6)

Finally, we define different objectives for different constraints types \((E, S, \text{ and } A)\):

\[
J_E = l_s(E) + \lambda_a \cdot l_a(E) + \lambda_r \cdot l_r(E);
\]

\[
J_S = l_s(S) + \lambda_r \cdot l_r(S);
\]

\[
J_A = l_a(A) + \lambda_r \cdot l_r(A),
\]

(7)

where \( \lambda_a \) and \( \lambda_r \) scale the contributions of the asymmetric and regularization losses, respectively. \( J_E \) pushes LE vectors to be similar in direction (loss \( l_s \)) and different in norm (loss \( l_a \)) after specialization. \( J_S \) forces vectors of synonyms to be closer together (loss \( l_s \)) and \( J_A \) vectors of antonyms to be further apart (loss \( l_a \)) in direction after specialization, both without affecting vector norms. We tune hyperparameters \((\delta_a, \delta_s, \delta_d, \lambda_a, \text{ and } \lambda_r)\) via cross-validation, with train and validation portions containing randomly shuffled \( E, S, \text{ and } A \) batches.

**Inference.** We infer the strength of the LE relation between vectors \( x^k_1 = f(x_1) \) and \( x^k_2 = f(x_2) \) with an asymmetric LE distance combining \( d_C \) and \( d_N \): \( I_{LE}(x^k_1, x^k_2) = d_C(x^k_1, x^k_2) + d_N(x^k_1, x^k_2) \). True LE pairs should have a small \( d_C \) and negative \( d_N \). We thus rank LE candidate word pairs according to their \( I_{LE} \) scores, from smallest to largest. For the binary LE detection, \( I_{LE} \) is binarized via threshold \( t \): if \( I_{LE} < t \), we predict that LE holds.

**Cross-Lingual (CL) LE Specialization.** After learning the generalized LE-specialization function \( f \), we can apply it to specialize any vector that comes from the same distributional vector space.
that we used in training. Let \( L_1 \) be the language for which we have the linguistic constraints and let \( X_{L_1} \) be its corresponding distributional space. Let \( X_{L_2} \) be the distributional space of another language \( L_2 \). Assuming a function \( g : \mathbb{R}^{d_{L_2}} \rightarrow \mathbb{R}^{d_{L_1}} \) that projects vectors from \( X_{L_2} \) to \( X_{L_1} \), we can straightforwardly LE-specialize the distributional space of \( L_2 \) by composing functions \( f \) and \( g \): \( X_{L_2} = f(g(X_{L_2})) \). Recently, a large number of projection-based models have been proposed for inducing bilingual word embedding spaces (Smith et al., 2017; Conneau et al., 2018; Artetxe et al., 2018; Ruder et al., 2018a; Joulin et al., 2018, inter alia), most of them requiring limited (word-level) or no bilingual supervision. Based on a few thousand (manually created or automatically induced) word-translation pairs, these models learn a linear projection \( W \) that projects the vectors from \( X_{L_2} \) to the space \( X_{L_1} \): \( g(X_{L_2}) = X_{L_2} W \). The cross-lingual space is then given as: \( X_{L_1} \cup X_{L_2} W \).

Due to simplicity and robust downstream performance,\(^3\) we opt for the simple supervised learning method. We use a dropout (keep rate 0.5) to each hidden layer of \( f \). We train in mini-batches of \( K = 50 \) constraints and learn with the Adam algorithm (Kingma and Ba, 2015): initial learning rate \( 10^{-4} \).

\(^3\)For a comprehensive downstream comparison of different cross-lingual embedding models, see (Glavaš et al., 2019).

### 3 Evaluation

#### Experimental Setup.

We work with Wikipedia-trained FASTText embeddings (Bojanowski et al., 2017). We take English constraints from previous work – synonyms and antonyms were created from WordNet and Roget’s Thesaurus (Zhang et al., 2014; Ono et al., 2015); LE constraints were collected from WordNet by Vulić and Mrkšić (2018) and contain both direct and transitively obtained LE pairs. We retain the constraints for which both words exist in the trimmed (200K) FASTText vocabulary, resulting in a total of 1,493,686 LE, 521,037 synonym, and 141,311 antonym pairs. We reserve 4,000 constraints (E: 2k, S: 1k, A: 1k) for validation and use the rest for training. We identify the following best hyperparameter configuration via grid search: \( H = 5 \), \( d_h = 300 \), \( \psi = \text{tanh} \), \( \delta_0 = 1 \), \( \delta_0 = \delta_1 = 0.5 \), \( \lambda_a = 2 \), and \( \lambda_r = 1 \).

We apply a dropout (keep rate 0.5) to each hidden layer of \( f \). We train in mini-batches of \( K = 50 \) constraints and learn with the Adam algorithm (Kingma and Ba, 2015): initial learning rate \( 10^{-4} \).

#### Results and Discussion.

The graded LE performance is shown in Table 1 for all seven setups. Graded LE results suggest that GLEN is robust and generalizes well to unseen words: the drop in performance between the 0% and 100% setups is mere 4% for GLEN (compared to a 50% drop for LEAR). Results in the 0% setting, in which GLEN improves over the distributional space by more than 30 points most clearly demonstrate its effectiveness.\(^5\) GLEN, however, lags behind LEAR in setups where LEAR has seen 70% or more of test words. This is intuitive: LEAR specializes the vector of each particular word using only the constraints containing that word; this gives LEAR higher specialization flexibility at the expense of generalization ability. In contrast, GLEN’s specialization function is affected by all constraints and has to work for all words; GLEN trades the effectiveness of LEAR’s word-specific updates for seen words, for the ability to generalize over unseen words. In a sense, there is a trade-off between the ability to generalize the

### Table 1: Spearman correlation for GLEN, compared with LEAR (Vulić and Mrkšić, 2018), on HyperLex, for different word coverage settings (i.e., percentages of HyperLex words seen in constraints in training).

<table>
<thead>
<tr>
<th>Setup</th>
<th>0%</th>
<th>10%</th>
<th>30%</th>
<th>50%</th>
<th>70%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEAR</td>
<td>.481</td>
<td>.485</td>
<td>.478</td>
<td>.474</td>
<td>.470</td>
<td>.506</td>
<td>.504</td>
</tr>
<tr>
<td>GLEN</td>
<td>.481</td>
<td>.485</td>
<td>.478</td>
<td>.474</td>
<td>.470</td>
<td>.506</td>
<td>.504</td>
</tr>
</tbody>
</table>

\(^4\)In the 0% setting we remove all constraints containing any HyperLex word; in the 100% we use all constraints. The full set of constraints contains 99.8% of all HyperLex words.

\(^5\)LEAR’s performance in the 0% setup corresponds to the performance of input distributional vectors.
LE-specialization over unseen words and the performance for seen words. Put differently, by learning a general specialization function—i.e., by using linguistic constraints merely as training instances—GLEN is prevented from “overfitting” to seen words. Evaluation settings like our 90% or 100% settings, in which GLEN is outperformed by a pure retrofitting model, are however unrealistic in view of downstream tasks: for any concrete downstream task (e.g., textual entailment or taxonomy induction), it is highly unlikely that the LE-specialization model will have seen almost all of the test words (words for which LE inference is required) in its training linguistic constraints; this is why GLEN’s ability to generalize LE-specialization to unseen words (as indicated by 0%-50% settings) is particularly important.

3.2 Cross-Lingual LE Detection

Neither joint (Nguyen et al., 2017) nor retrofitting models (Vulić and Mrkšić, 2018) can predict LE across languages in a straightforward fashion. Coupled with a CL space, GLEN can seamlessly predict LE across language boundaries.

Experimental Setup. We evaluate GLEN on datasets from Upadhyay et al. (2018), encompassing two binary cross-lingual LE detection tasks: (1) HYPO task test model’s ability to determine the direction of the LE relation, i.e., to discern hyponym-hypernym pairs from hypernym-hyponym pairs; (2) COHYP tasks tests whether the models are able to discern true LE pairs from cohyponyms (e.g., car and boat, cohyponyms of vehicle). We report results for three language pairs: English (EN) – {French (FR), Russian (RU), Arabic (AR)}. Upadhyay et al. (2018) divided each dataset into train (400-500 word pairs) and test portions (900-1000 word pairs): we use the train portions to tune the threshold \( t \) that binarizes GLEN’s predictions \( I_{LE} \).

We induce the CL embeddings (i.e., learn the projections \( W_g \), see Section §2) by projecting AR, FR, and RU embeddings to the EN space in a supervised fashion, by finding the optimal solution to the Procrustes problem for given 5K word translation pairs (for each language pair). \(^6\) We compare GLEN with more complex models from (Upadhyay et al., 2018): they couple two methods for inducing syntactic CL embeddings – CL-DEP (Vulić, 2017) and B1-SPARSE (Vyas and Carpuat, 2016) – with an LE scorer based on the distributional inclusion hypothesis (Geffet and Dagan, 2005).

Results. GLEN’s cross-lingual LE detection performance is shown in Table 2. GLEN dramatically outperforms CL LE detection models from (Upadhyay et al., 2018), with an average edge of 24% on HYPO datasets and 16% on the COHYP datasets.\(^7\) This accentuates GLEN’s generalization ability: it robustly predicts CL LE, although trained only on EN constraints. GLEN performs better for EN-AR and EN-RU than for EN-FR: we believe this to merely be an artifact of the (rather small) test sets. We find GLEN’s CL performance for more distant language pairs (EN-AR, EN-RU) especially encouraging as it holds promise of successful transfer of LE-specialization to resource-lean languages lacking external linguistic resources.

4 Conclusion

We presented GLEN, a general framework for specializing word embeddings for lexical entailment. Unlike existing LE-specialization models (Nguyen et al., 2017; Vulić and Mrkšić, 2018), GLEN learns an explicit specialization function using linguistic constraints as training examples. The learned LE-specialization function is then applied to vectors of words (1) unseen in constraints and (2) from different languages. GLEN displays robust graded LE performance and yields massive improvements over state-of-the-art in cross-lingual LE detection. We next plan to evaluate GLEN on multilingual and cross-lingual graded LE datasets (Vulić et al., 2019) and release a large multilingual repository of LE-specialized embeddings. We make GLEN (code and resources) available at: https://github.com/codogogo/glen.

Acknowledgments

The work of the first author was supported by the Eliteprogramm of the Baden-Württemberg Stiftung, within the scope of the AGREE grant.

---

\(^6\)We automatically translated 5K most frequent EN words to AR, FR, and RU with Google Translate.

\(^7\)All differences are statistically significant at \( \alpha = 0.01 \), according to the non-parametric shuffling test (Yeh, 2000)

Table 2: CL LE detection results (accuracy) on CL datasets (HYPO, COHYP) (Upadhyay et al., 2018).

<table>
<thead>
<tr>
<th>Model</th>
<th>HYPO</th>
<th>COHYP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-FR</td>
<td>EN-RU</td>
<td>EN-AR</td>
<td>Avg</td>
</tr>
<tr>
<td>HYPO</td>
<td>CL-DEP</td>
<td>.538</td>
<td>.602</td>
<td>.567</td>
</tr>
<tr>
<td></td>
<td>BI-SPARSE</td>
<td>.566</td>
<td>.590</td>
<td>.526</td>
</tr>
<tr>
<td></td>
<td>GLEN</td>
<td>.792</td>
<td>.811</td>
<td>.816</td>
</tr>
<tr>
<td>COHYP</td>
<td>CL-DEP</td>
<td>.610</td>
<td>.562</td>
<td>.631</td>
</tr>
<tr>
<td></td>
<td>BI-SPARSE</td>
<td>.667</td>
<td>.636</td>
<td>.668</td>
</tr>
<tr>
<td></td>
<td>GLEN</td>
<td>.779</td>
<td>.849</td>
<td>.821</td>
</tr>
</tbody>
</table>
References


