Decomposable Attention Model for Weakly-Supervised Semantic Parsing

Tomer Ben Moshe [1] **Dvir Ginzburg** [2] **Vered Zilberstein** [3]

Abstract

We are using a neural network architecture called 'Decomposable Attention Model' that was suggested by (Parikh et al., 2016) for solving a Weakly-Supervised Semantic Parsing problem. The architecture initially tried to solve the problem of testing whether two natural language sentences are a contradiction or whether they entail one another. We are using it for testing whether a logical form (i.e. executable program) and a natural language utterance fit in their meaning. While using the architecture "as is" as described in the paper failed to converge, after some tweaking we were able to achieve accuracy scores of more than 99%.

1. Introduction

We regard semantic parsing as the process of finding the best executable program for an utterance in natural language. An executable program is a command that when executed on a world state, transforms it to a new state. For example, in figure 1, observe an initial world state and an utterance which describes the transition to another world state. Our goal is to find the best executable program to produce that transition.

The term Weakly-Supervised Semantic Parsing means trying to solve the problem of Semantic Parsing without having the actual executable program in the training data. Training data is only comprised of the initial world state, a natural language utterance, and the final world state. The problem lies in finding an executable program that transforms the initial world state into the final world state.

The main advantage of this problem is the abundance of training data since an executable program is harder to reach than natural language utterances and world states. The main difficulty here is that we do not have the correct program when training the model. We need to search randomly in the program space in order to find it.

Natural Language Interface (NLI) refers to the problem of finding entailment or contradiction between two natural language sentences.

For example, in the following sentences:

- Alice is studying
- Alice is asleep

there is a contradiction between the sentences since Alice cannot be studying while she is asleep.

One suggested architecture for the NLI problem is the '**Decomposable Attention Model**' (Parikh et al., 2016). This architecture decomposes the problem into several subproblems that can be solved separately, thus making it parallelizable.

We suggest an implementation of the 'Decomposable Attention Model' architecture targeted towards solving Semantic Parsing. Our implementation has several differences from the original architecture.

First, the original architecture assumes both sentences are in natural language. In our architecture, only the first sentence is in natural language, and the second is an executable command.

Second is the label. The label still means 'contradiction' or 'entailment' between the sentences but with a slight difference in the context. In this context, 'contradiction' means that the program is not in correspondence with the utterance, and 'entailment' means that the program is fitting for the given utterance.

Third is the representation. Our implementation has to consider the different representations of the two sentences. The original architecture had two natural language sentences with similar representation, while we have to contend with two conceptually different sentences. This changes the input size of the network as well as its symmetry. In the NLI model, swapping the sentences yields similar results. In our model, changing their order losses its meaning.

2. The SCONE dataset

We use the 'SCONE' (Long et al., 2016; Guu et al., 2017) dataset, which presents a weakly-supervised semantic parsing problem. The dataset is comprised of examples, and every example has its initial state (w_0), a set of utterances in natural language $(u_1, u_2, ..., u_M)$, and for every utterance

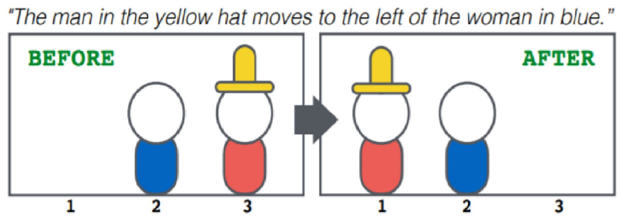
"The man in the yellow hat moves to the left of the woman in blue."
BEFORE
AFTER
1
2
3
1
2
3

Figure 1. Weakly-supervised example in the SCONE dataset

there is the corresponding world-state after its execution (w_i). The problem is parsing the utterance in natural language (NL) to a series of executable programs(commands) with predefined syntax $(z_1, z_2, ..., z_M)$, that when executed on the initial world state, will reach the desired world state $(w_1, w_2, ..., w_M)$ accordingly.

In the original SCONE dataset paper (Long et al., 2016), they use the beam algorithm. For each command in NL, the algorithm finds a set of suggested executable commands. Each executable command is scored with the (locally-normalized) probability for each program given the NL command. The executable commands are arranged in decreasing order of probabilities - the most probable command first, then the second most probable etc.

The 'Decomposable Attention Model' needs as an input both the NL utterance and the executable program. It then produces a score for how much they 'fit'. Since in this dataset we don't have the commands, we use the model as a globally-normalized re-ranker for the beam algorithm published for the SCONE dataset to address this issue.

3. Related Work

Weakly-Supervised Semantic problems has been widely studied (Artzi & Zettlemoyer, 2013; Berant et al., 2013; Bordes et al., 2014; Kwiatkowski et al., 2013).

Our goal is to implement an architecture which was initially suggested for the NLI problem and see if it is capable of solving a simple Weakly-Supervised Semantic problem.

The approach is based on alignment, which is an approach that is rooted in machine translation (Liu et al., 2006; Och & Ney, 2004).

We use this approach as a re-ranker for the beam algorithm. This usage is also common (Goldman et al., 2017; Nisioi et al., 2017; Farkas & Schmid, 2012)

4. Approach

Let u_i be an utterance in natural language. Let z_i be a command in the machine language API, i.e. a sequence of API calls. The problem is converting u_i to the correct corresponding z_i.

Our approach takes the top k API commands generated by the beam, and deploys a re-ranker model to evaluate the best command for the given utterance. The incentive of using the re-ranker model comes from the problem space of the task. The beam searches are done in a greedy manner in an NP-hard complexity problem space, causing 'label bias'. The model encounters two major problems:

1. It is unlikely the model will find the optimal solution (we are not addressing this problem in the paper)
2. Most of the solutions fed to the underlying model for the beam are completely wrong, adding noise to the learning process.

The re-ranker is built on top of the decomposable model, with slight modifications to support the different input types (utterances in NL and commands in machine language). The training data comes in the form of m of labeled pairs $\{(u_i, z_i), y_i\}_{i=1}^{m}$ where y is a binary classification where 1 indicates that the command is a correct representation of the utterance, and 0 otherwise.

We feed our decomposable model with each utterance and its corresponding sequences of API calls (executable programs), and try to maximize the output score for the correct sequences. Upon test time we feed the model with the predicted sequences extracted from the beam algorithm. We then survey the ranker with regard to two indexes.

After training the beam algorithm, we save each suggested program for every NL command to a file. For each pair of utterance and command we also save the gold label. So, for every utterance the file containes $\{(u_i, z_i), y_i\}_{i=1}^{m}$. We prepared two files of this kind - one for training and the other for testing. The new 'Decomposable Attention Model' network is trained and tested with these files.

The reason for storing this information in files is to separate the learning process of the beam algorithm and the learning process of our 'decomposable' model. The learning process of the beam algorithm depends on many random occurrences. Thus, having one deterministic file helps in understanding the learning process of our 'decomposable' model.

5. The Decomposable Attention Model

The model is built as follows:

Input encoder: Each utterance is encoded using GloVe (Pennington et al., 2014). Each API call is encoded via a one-hot vector since there is only a limited amount of possible API calls.

Bi-LSTM encoder: Since an utterance and a command

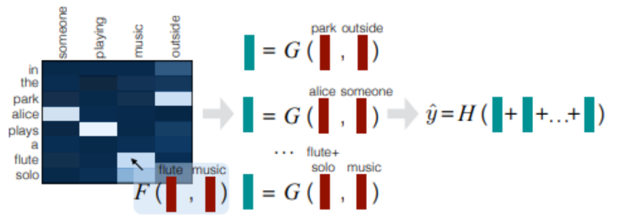

Figure 2. Pictoral overview of the approach, showing the Attend (left), Compare (center) Aggregate and normalization steps. This architecture takes place twice, once for the each word in the utterance with the full command, and once for each API call in the command with the full utterance. Each instance have different weights.

translation have different representations, we start by applying Bi-LSTM twice[1]: for the utterance, and for the command. The Bi-LSTM network is the classic Bidirectional LSTM network with additional dense and dropout layers for each encoded word/API call [2]. We then apply the decomposable attention model.

Attention: Again, due the difference in representation we apply the attention layer twice. For each word we apply 2 pairs of dense-dropout layers with $l2$ regularization and normal initialization, ended by a relu activation. After computing the the output of those layers, we apply matrix multiplication to get the attention weights, avoiding the quadratic complexity needed in the vanilla attention model (we follow the idea as presented in the original decomposable paper), i.e. producing the aligned values for each pair of word-API as follows[3]:

$$e_{ij} := F(u^i, z^j) := F'(u^i)^T F''(z^j)$$

Then, for each output attention vector we apply soft normalization (denote n_z as the length of the command and n_u as the length of the utterance):

$$\beta_i := \sum_{j=1}^{n_z} \frac{exp(e_{ij})}{\sum_{k=1}^{n_z} exp(e_{ik})} z^j$$

$$\alpha_j := \sum_{i=1}^{n_u} \frac{exp(e_{ij})}{\sum_{k=1}^{n_u} exp(e_{kj})} u^i$$

alpha and beta are the soft aligned normalized attention vectors.

Comparison and aggregation: Now, we propagate each aligned vector with the corresponding phrase (β_i with u^i, and α_j with z^j) through a net of two pairs of dropout-dense-relu activation layers(Again, two different instances). This generates a vector output for each phrase-vector pairs (i.e. a total of $n_z + n_u$ vectors). Denote:

$$v_{i,1} = G(\beta_i, u^i)$$

$$v_{j,2} = G'(\alpha_j, z^j)$$

We now aggregate the utterance of length n_u's, and command of length n_z's comparison vectors using both average and max pooling transformation, to form one unified vector for each input[4] and concatenating the results together.

$$v_{U,1} = \frac{\sum v_{i,1}}{n_u},\ v_{U,2} = max(v_{i,1}),$$

$$v_{Z,1} = \frac{\sum v_{j,2}}{n_z},\ v_{Z,2} = max(v_{j,2}),$$

$$v_U = [v_{U,1}, v_{U,2}],\ v_Z = [v_{Z,1}, v_{Z,2}]$$

The brackets [,] denote concatenation.

Re-ranking: Finally, we apply another pair of dropout-dense-relu activation layers, where the inputs are the aggregated results for the command and utterance attention vectors. We apply a softmax layer to output a probability vector of size 2. The first index represents 'negatives' i.e. the probability that the command does not fit the utterance. The second index represents 'positives' i.e. the probability that they fit. The loss is binary cross-entropy.

Learning to rank: We use two types of metrics in order to assess our ranker: the first is 'Area Under the Curve'. This is a form of pairwise approach which essentially measures the expected proportion of positives ranked before a uniformly drawn random negative. The second is a simple listwise approach which measures the percentage of times in which the command with the highest score is a correct translation to a given utterance. These assessments are performed using the test file.

In the next section we show and discuss which configurations (e.g. different learning rates, number of hidden states etc.) show the best results, and compare the trade-offs between the different models.

6. Experiments

Our re-ranker is evaluated on the SCONE dataset as described before. For that we use the beam algorithm's output (i.e. k commands which are possibly a corresponding executable program representation of the utterance in natural language, arranged from most probable to least probable).

[1]Each instance of the Bi-LSTM holds different weights

[2]Extra layers were added to increase expressiveness or address over-fitting

[3]u^i means the i^{th} word in the utterance. z^j means the j^{th} API Call in the command.

[4]While the original paper used summation layer, we found it not to be expressive enough, causing the net to fail to converge

Given an utterance u and a set of k possible commands z, the task is to rank the correct commands [5]with the highest score (i.e the net output for those paths is the highest).

6.1. Implementation Details

The model was implemented in Keras 1.2.2 with a TensorFlow (Abadi et al., 2015) 0.12.0 back engine.

6.2. Data preprocessing

We use the tangrams subset of the SCONE dataset. We only used the first utterance and world state. It contains 3469 train utterances, and 318 test utterances. For each sample we extracted all commands output from the beam algorithm (configurable), on which we trained our re-ranker. For each utterance and each command we zero padded to the maximum length (configurable).

6.3. Embeddings

We use 100 dimensional GloVe embeddings to represent each word in the utterance, and a one-hot vector of size 14 for each API call.

6.4. Model parameters

The following parameters were tweaked during the experiment to find the best configuration for our re-ranker model: learning rate (LR), dropout rate (DROPOUT), optimizer (OPT), number of hidden layers (HL), and training mode. The training mode was either training on all examples (ALL), or training on the best and worst examples for each utterance according to the beam probability output (EDGES).

7. Results

The results of the model were evaluated using both list-wise (LW) and area under the curve (AUC) metrics. These can be seen at table 1.

As introduced in the Abstract section, implementing the model as described in the original article failed to converge, yielding list-wise accuracy close to a complete random choice.

After adding the extra layers, and tuning the parameters of the model we achieved up to 99% accuracy. Another thing to notice is the distinct difference between using adagrad(Duchi et al., 2011) and adam(Kingma & Ba, 2014) as optimizers: while adam is consistently yielding better accuracy, adagrad occasionally doesn't even converge.

Another thing to notice is that a high number of hidden layers doesn't necessarily gives better results - the second configuration used only 15 hidden layer compared to the third configuration which used 300. Both with the same learning rate but different dropout rate. Despite this, both configurations achieved similar results on both metrics. This might imply some responsibility on the dropout rate.

Using the best and worst mode also doesn't ensure better results in the 'AUC' metric. The seconds and fifth runs used the same configurations except for the training mode - the second with 'all' and the fifth with 'edges'. Here, 'all' showed better results. In comparison, the forth and sixth runs also differ in the training mode - the forth with 'edges' and the sixth with 'all'. This time 'edges' showed better results.

Using 'all' mode consistently gives better than using 'best and worst' on the 'LW' metric. The 'LW' metric checks the whole group of commands for each utterance, making the 'all' mode better for this specific metric.

Table 1. Ranking accuracies across all runs

	LR	#HL	DROP	OPT	MODE	AUC	LW
1	0.0001	100	0.2	ADAM	ALL	0.990	0.965
2	0.0001	15	0.05	ADAM	ALL	0.989	0.956
3	0.0001	300	0.2	ADAM	ALL	0.987	0.947
4	0.001	100	0.15	ADAGRAD	EDGES	0.905	0.354
5	0.0001	15	0.05	ADAM	EDGES	0.895	0.344
6	0.001	100	0.15	ADAGRAD	ALL	0.849	0.973
7	0.0001	100	0.2	ADAGRAD	ALL	0.662	0.141
8	0.1	100	0.2	ADAGRAD	ALL	0.555	0.185

Several examples of utterances that most configurations failed on:

- take away the 2nd to last figure
- swap the second figure with the one to its left

It seems that the architecture can't comprehend idioms that needs several words to understand. We believe that using intra-sentence attention would solve most of these types of failures

It can be seen in the appendix some learning graphs. As expected, the graph with the smaller Hidden Layers converges faster.

7.1. Comparison to the original Beam Parser

With just the beam probabilities and without re-ranking, the model achieves an accuracy of 0.891 (for 1 utterance). This is calculated by taking the mean of all the times that the first output of the beam was correct - corresponds to LW. Regarding AUC, the beam algorithm cannot compare two commands separately due to its nature of creating the commands from scratch each time. It is our conclusion

[5]There may be more than one correct output from the beam

that utilizing our re-ranker on the output of the beam will achieve better results.

8. Conclusion

We have shown that the architecture that was proposed for the Natural Language Inference is also useful for semantic parsing. We proved it by examining it on a simple semantic parsing problem.

Our code is made available at `https://github.com/Verose/SPEN-Seq2Seq-SemanticParsing/`

Further research could check the architecture limitations and see if it still holds on more difficult semantic parsing problems.

A possible enhancement can be made by adding the intra-sentence attention mentioned in the decomposable attention paper and checking whether it improves the accuracy of the network.

9. Appendix

Below are some of the graphs generated during the learning process:

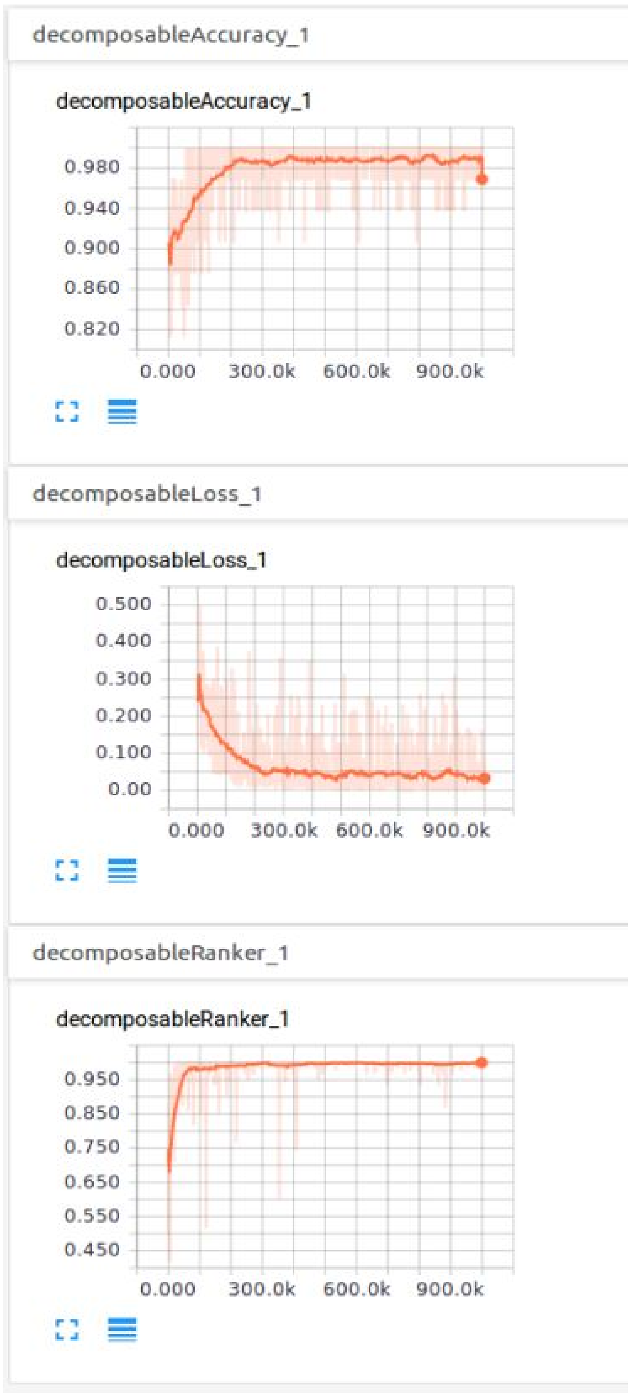

Figure 3. Training graph of the 2nd configuration

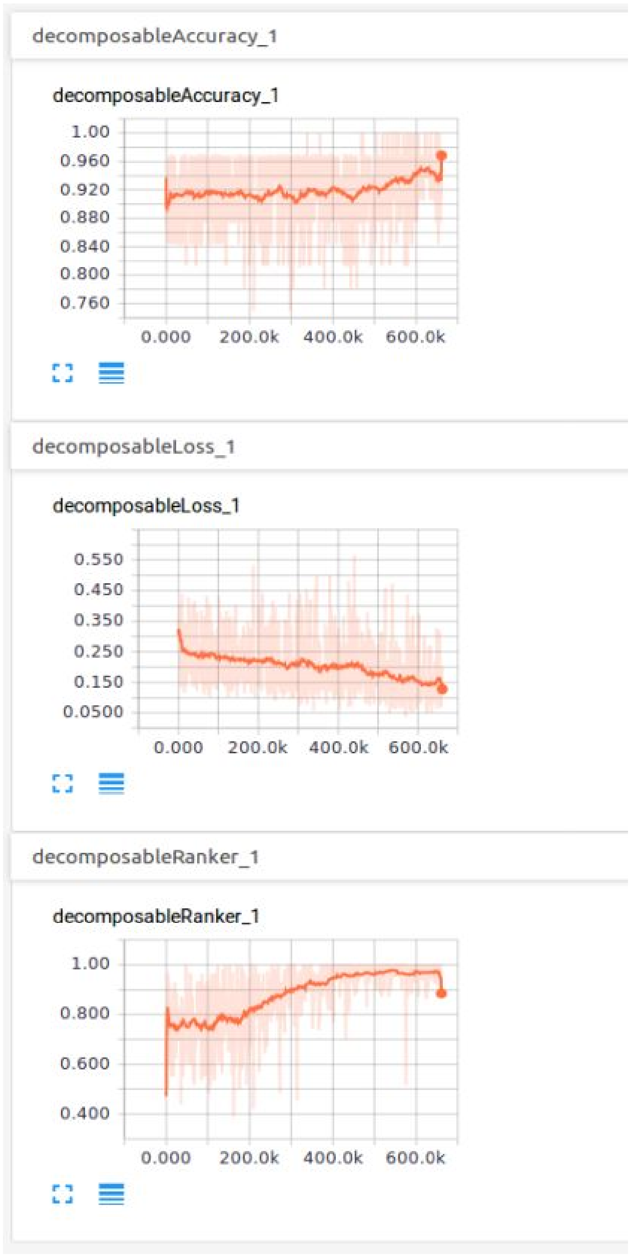

Figure 4. Training graph of the 6th configuration

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dandelion, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Artzi, Yoav and Zettlemoyer, Luke. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62, 2013.

Berant, Jonathan, Chou, Andrew, Frostig, Roy, and Liang, Percy. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, 2013.

Bordes, Antoine, Weston, Jason, and Usunier, Nicolas. Open question answering with weakly supervised embedding models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 165–180. Springer, 2014.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011.

Farkas, Richárd and Schmid, Helmut. Forest reranking through subtree ranking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pp. 1038–1047, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=2390948.2391064`.

Goldman, Omer, Latcinnik, Veronica, Naveh, Udi, Globerson, Amir, and Berant, Jonathan. Weakly-supervised semantic parsing with abstract examples. *CoRR*, abs/1711.05240, 2017. URL `http://arxiv.org/abs/1711.05240`.

Guu, Kelvin, Pasupat, Panupong, Liu, Evan Zheran, and Liang, Percy. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. *CoRR*, abs/1704.07926, 2017. URL `http://arxiv.org/abs/1704.07926`.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Kwiatkowski, Tom, Choi, Eunsol, Artzi, Yoav, and Zettlemoyer, Luke. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 conference* *on empirical methods in natural language processing*, pp. 1545–1556, 2013.

Liu, Yang, Liu, Qun, and Lin, Shouxun. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 609–616. Association for Computational Linguistics, 2006.

Long, Reginald, Pasupat, Panupong, and Liang, Percy. Simpler context-dependent logical forms via model projections. *CoRR*, abs/1606.05378, 2016. URL `http://arxiv.org/abs/1606.05378`.

Nisioi, Sergiu, Štajner, Sanja, Ponzetto, Simone Paolo, and Dinu, Liviu P. Exploring neural text simplification models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pp. 85–91, 2017.

Och, Franz Josef and Ney, Hermann. The alignment template approach to statistical machine translation. *Computational linguistics*, 30(4):417–449, 2004.

Parikh, Ankur P., Täckström, Oscar, Das, Dipanjan, and Uszkoreit, Jakob. A decomposable attention model for natural language inference. *CoRR*, abs/1606.01933, 2016. URL `http://arxiv.org/abs/1606.01933`.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pp. 1532–1543, 2014.