Supplementary Material for Learning to Generalize from Sparse and Underspecified Rewards

1. Semantic Parsing

Our implementation is based on the open source implementation of MAPO (Liang et al., 2018) in Tensorflow (Abadi et al., 2016). We use the same model architecture as MAPO which combines a seq2seq model augmented by a key-variable memory (Liang et al., 2017) with a domain specific language interpreter. We utilized the hyperparameter tuning service (Golovin et al., 2017) provided by Google Cloud for BoRL.

1.1. Datasets

WikiTableQuestions (Pasupat & Liang, 2015) contains tables extracted from Wikipedia and question-answer pairs about the tables. There are 2,108 tables and 18,496 question-answer pairs splitted into train/dev/test set. We follow the construction in (Pasupat & Liang, 2015) for converting a table into a directed graph that can be queried, where rows and cells are converted to graph nodes while column names become labeled directed edges. For the questions, we use string match to identify phrases that appear in the table. We also identify numbers and dates using the CoreNLP annotation released with the dataset.

The task is challenging in several aspects. First, the tables are taken from Wikipedia and cover a wide range of topics. Second, at test time, new tables that contain unseen column names appear. Third, the table contents are not normalized as in knowledge-bases like Freebase, so there are noises and ambiguities in the table annotation. Last, the semantics are more complex comparing to previous datasets like WebQuestionsSP (Yih et al., 2016). It requires multiple-step reasoning using a large set of functions, including comparisons, superlatives, aggregations, and arithmetic operations (Pasupat & Liang, 2015).

Figure 1 shows some natural language queries in WikiTableQuestions for which both the models trained using MAPO and MeRL generated the correct answers despite generating different programs.

WikiSQL (Zhong et al., 2017) is a recent large scale dataset on learning natural language interfaces for databases. It also uses tables extracted from Wikipedia, but is much larger and is annotated with programs (SQL). There are 24,241 tables and 80,654 question-program pairs splitted into train/dev/test set. Comparing to WikiTableQuestions, the semantics are simpler because SQL use fewer operators (column selection, aggregation, and conditions). We perform similar preprocessing as for WikiTableQuestions. We don't use the annotated programs in our experiments.

1.2. Auxiliary Reward Features

In our semantic parsing experiments, we used the same preprocessing as implemented in MAPO. The natural language queries are preprocessed to identify numbers and date-time entities. In addition, phrases in the query that appear in the table entries are converted to string entities and the columns in the table that have a phrase match are assigned a column feature weight based on the match.

We used the following features for our auxiliary reward for both WikiTableQuestions and WikiSQL:

- f_1: Fraction of total entities in the program weighted by the entity length
- f_2, f_3, f_4: Fraction of date-time, string and number entities in the program weighted by the entity length respectively
- f_5: Fraction of total entities in the program
- f_6: Fraction of longest entities in the program
- f_7: Fraction of columns in the program weighted by the column weight
- f_8: Fraction of total columns in the program with non-zero column weight
- f_9: Fraction of columns used in the program with the highest column column weight
- f_{10}: Fractional number of expressions in the program
- f_{11}: Sum of entities and columns weighted by their length and column weight respectively divided by the number of expressions in the program

1.3. Training Details

We used the optimal hyperparameter settings for training the vanilla IML and MAPO provided in the open source implementation of MAPO. One major difference was that we used a single actor for our policy gradient implementation as opposed to the distributed sampling implemented in Memory Augmented Program Synthesis.

Example	Comment
Query nu-1167: **Who was the first oldest living president?** MAPO: v_0 = (first all_rows); $v_{\rm ans}$ = (hop v_0 r.president) MeRL: v_0 = (argmin all_rows r.became_oldest_living_president-date); $v_{\rm ans}$ = (hop v_0 r.president)	Both programs generate the correct answer despite MAPO's program being spurious since it assumes the database table to be sorted based on the *became_oldest_living_president-date* column.
Query nu-346: **What tree is the most dense in India?** MAPO: v_0 = (argmax all_rows r.density); $v_{\rm ans}$ = (hop v_0 r.common_name) MeRL: v_0 = (filter_str_contain_any all_rows [u'india'] r.location); v_1 = (argmax v_0 r.density); $v_{\rm ans}$ = (hop v_1 r.common_name)	MAPO's program generates the correct answer by chance since it finds the tree with most density which also happens to be in India in this specific example.
Query nu-2113: **How many languages has at least 20,000 speakers as of the year 2001?** MeRL: v_0 = (filter_ge all_rows [20000] r.2001_. . . -number); $v_{\rm ans}$ = (count v_0) MAPO: v_0 = (filter_greater all_rows [20000] r.2001_. . . -number); $v_{\rm ans}$ = (count v_0)	Since the query uses "at least", MeRL uses the correct function token *filter_ge* (i.e $\geq$ operator) while MAPO uses *filter_greater* (i.e. $>$ operator) which accidentally gives the right answer in this case. For brevity, *r.2001_. . . -number* refers to *r.2001_census_1_total_population_1_004_59_million-number*.

Figure 1. Example of generated programs from models trained using MAPO and MeRL on WIKITABLEQUESTIONS. Here, v_i correponds to the intermediate variables computed by the generated program while $v_{\rm ans}$ corresponds to the variable containing the executed result of the generated program.

Table 1. MAPOX hyperparameters used for experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	9.86 x 10^{-2}
Learning Rate	4 x 10^{-4}
Dropout	2.5 x 10^{-1}

Table 2. BoRL hyperparameters used in experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	5 x 10^{-2}
Learning Rate	5 x 10^{-3}
Dropout	3 x 10^{-1}

Table 3. MeRL hyperparameters used in experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	4.63 x 10^{-2}
Learning Rate	2.58 x 10^{-2}
Dropout	2.5 x 10^{-1}
Meta-Learning Rate	2.5 x 10^{-3}

Table 4. MAPOX hyperparameters used for experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	5.1 x 10^{-3}
Learning Rate	1.1 x 10^{-3}

Table 5. BoRL hyperparameters used in experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	2 x 10^{-3}
Learning Rate	1 x 10^{-3}

Table 6. MeRL hyperparameters used in experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	6.9 x 10^{-3}
Learning Rate	1.5 x 10^{-3}
Meta-Learning Rate	6.4 x 10^{-4}

For our WIKITABLEQUESTIONS experiments reported in Table 2, we initialized our policy from a pretrained MAPO checkpoint (except for vanilla IML and MAPO) while for all our WIKISQL experiments, we trained the agent's policy starting from random initialization.

For the methods which optimize the validation accuracy using the auxiliary reward, we trained the auxiliary reward parameters for a fixed policy initialization and then evaluated the top K hyperparameter settings 5 times (starting from random initialization for WIKISQL or on 5 different pretrained MAPO checkpoints for WIKITABLEQUESTIONS) and picked the hyperparameter setting with the best average validation accuracy on the 5 runs to avoid the danger of overfitting on the validation set.

We only used a single run of IML for both WIKISQL and WIKITABLEQUESTIONS for collecting the exploration trajectories. For WikiSQL, we used greedy exploration with one exploration sample per context during training. We run the best hyperparameter setting for 10k epochs for both WIKISQL and WIKITABLEQUESTIONS. Similar to MAPO, the ensembling results reported in Table 4, used 10 different training/validation splits of the WIKITABLEQUESTIONS dataset. This required training different IML models on each split to collect the exploration trajectories.

We ran BoRL for 384 trials for WIKISQL and 512 trials for WIKITABLEQUESTIONS respectively. We used random search with 30 different settings to obtain the optimal hyperparameter values for all our experiments. The detailed hyperparameter settings for WIKITABLEQUESTIONS and WIKISQL experiments are listed in Table 1 to Table 3 and Table 4 to Table 6 respectively. Note that we used a dropout value of 0.1 for all our experiments on WIKISQL except MAPO which used the optimal hyperparameters reported by Liang et al. (2018).

2. Instruction Following Task

2.1. Auxiliary Reward Features

In the instruction following task, the auxiliary reward function was computed using the single and pairwise comparison of counts of symbols and actions in the language command $\boldsymbol{x}$ and agent's trajectory $\boldsymbol{a}$ respectively. Specifically, we created a feature vector $\boldsymbol{f}$of size 272 containing binary features of the form $f(a, c) = \#_a(\boldsymbol{x}) == \#_c(\boldsymbol{a})$ and $\boldsymbol{f}(ab, cd) = \#_{ab}(\boldsymbol{x}) == \#_{cd}(\boldsymbol{a})$ where $a, b \in \{$Left, Right, Up, Down$\}$ and $c, d \in \{0, 1, 2, 3\}$ and $\#_i(\boldsymbol{j})$ represents the count of element i in the vector $\boldsymbol{j}$. We learn one weight parameter for each single count comparison feature. The weights for the pairwise features are represented using the weights for single comparison features $\boldsymbol{w}_{(ab,cd)} = \alpha * \boldsymbol{w}_{ac} * \boldsymbol{w}_{bd} + \beta * \boldsymbol{w}_{ad} * \boldsymbol{w}_{bc}$ using the additional weights α and β.

The auxiliary reward is a linear function of the weight parameters (see equation 7). However, in case of MeRL, we also used a softmax transformation of the linear auxiliary reward computed over all the possible trajectories (at most 10) for a given language instruction.

2.2. Training Details

We used the Adam Optimizer (Kingma & Ba, 2014) for all the setups with a replay buffer memory weight clipping of 0.1 and full-batch training. We performed hyperparameter sweeps via random search over the interval (10^{-4}, 10^{-2}) for learning rate and meta-learning rate and the interval (10^{-4}, 10^{-1}) for entropy regularization. For our MeRL setup with auxiliary + underspecified rewards, we initialize the policy network using the MAPO baseline trained with the underspecified rewards. The hyperparameter settings are listed in Table 7 to Table 9. MeRL was trained for 5000 epochs while other setups were trained for 8000 epochs. We used 2064 trials for our BoRL setup which was approximately 20x the number of trials we used to tune hyperparameters for other setups.

Table 7. MAPO hyperparameters used for the setup with Oracle rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	3.39 x 10^{-2}
Learning Rate	5.4 x 10^{-3}

Table 8. MAPO hyperparameters used for the setup with underspecified rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	1.32 x 10^{-2}
Learning Rate	9.3 x 10^{-3}

Table 9. MeRL hyperparameters used for the setup with underspecified + auxiliary rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	2 x 10^{-4}
Learning Rate	4.2 x 10^{-2}
Meta-Learning Rate	1.5 x 10^{-4}
Gradient Clipping	1 x 10^{-2}

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv:1603.04467*, 2016.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liang, C., Berant, J., Le, Q., Forbus, K. D., and Lao, N. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *ACL*, 2017.

Liang, C., Norouzi, M., Berant, J., Le, Q. V., and Lao, N. Memory augmented policy optimization for program synthesis and semantic parsing. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and

Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 9994–10006. 2018.

Pasupat, P. and Liang, P. Compositional semantic parsing on semi-structured tables. *ACL*, 2015.

Yih, W.-t., Richardson, M., Meek, C., Chang, M.-W., and Suh, J. The value of semantic parse labeling for knowledge base question answering. *ACL*, 2016.

Zhong, V., Xiong, C., and Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, 2017.