Policy Shaping and Generalized Update Equations for Semantic Parsing from Denotations

Dipendra Misra[⋆], **Ming-Wei Chang**[†], **Xiaodong He**[◇], **Wen-tau Yih**[‡]
[⋆]Cornell University, [†]Google AI Language, [◇]JD AI Research
[‡]Allen Institute for Artificial Intelligence
dkm@cs.cornell.edu, mingweichang@google.com
xiaodong.he@jd.com, scottyih@allenai.org

Abstract

Semantic parsing from denotations faces two key challenges in model training: (1) given only the denotations (e.g., answers), search for good candidate semantic parses, and (2) choose the best model update algorithm. We propose effective and general solutions to each of them. Using *policy shaping*, we bias the search procedure towards semantic parses that are more compatible to the text, which provide better supervision signals for training. In addition, we propose an update equation that generalizes three different families of learning algorithms, which enables fast model exploration. When experimented on a recently proposed sequential question answering dataset, our framework leads to a new state-of-the-art model that outperforms previous work by 5.0% absolute on exact match accuracy.

1 Introduction

Semantic parsing from denotations (SpFD) is the problem of mapping text to executable formal representations (or *program*) in a situated environment and executing them to generate denotations (or *answer*), in the absence of access to correct representations. Several problems have been handled within this framework, including question answering (Berant et al., 2013; Iyyer et al., 2017) and instructions for robots (Artzi and Zettlemoyer, 2013; Misra et al., 2015).

Consider the example in Figure 1. Given the question and a table environment, a semantic parser maps the question to an executable program, in this case a SQL query, and then executes the query on the environment to generate the answer *England*. In the SpFD setting, the training data does not contain the correct programs. Thus, the existing learning approaches for SpFD perform two steps for every training example, a search step that explores the space of programs and finds suitable candidates, and an update step that uses these programs to update the model. Figure 2 shows the two step training procedure for the above example.

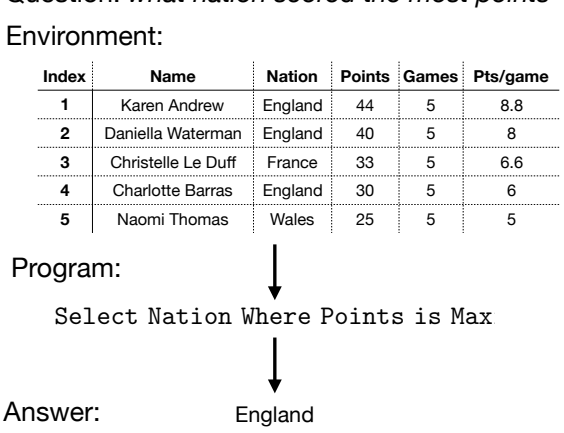

Figure 1: An example of semantic parsing from denotations. Given the table environment, map the question to an executable program that evaluates to the answer.

In this paper, we address two key challenges in model training for SpFD by proposing a novel learning framework, improving both the *search* and *update* steps. The first challenge, the existence of *spurious programs*, lies in the search step. More specifically, while the success of the search step relies on its ability to find programs that are semantically correct, we can only verify if the program can generate correct answers, given that no gold programs are presented. The search step is complicated by *spurious programs*, which happen to evaluate to the correct answer but do not represent accurately the meaning of the natural language question. For example, for the environment in Figure 1, the program `Select Nation Where Name = Karen Andrew` is spurious. Selecting spurious programs as positive examples can greatly affect the performance of semantic parsers as these programs generally do not gen-

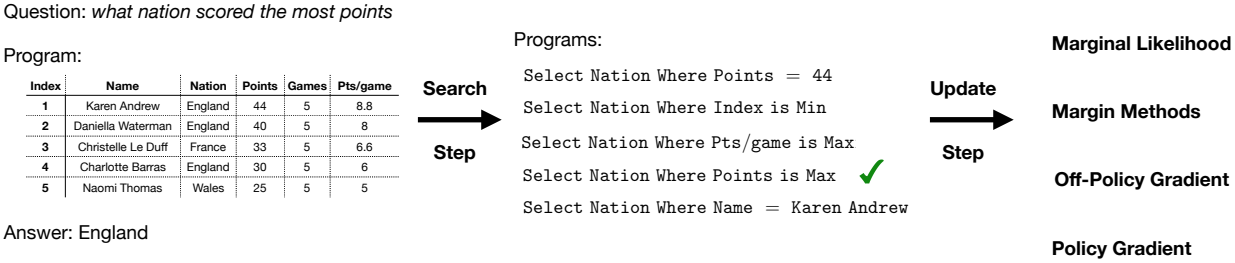

Figure 2: An example of semantic parsing from denotation. Given the question and the table environment, there are several programs which are spurious.

eralize to unseen questions and environments.

The second challenge, *choosing a learning algorithm*, lies in the update step. Because of the unique *indirect supervision* setting of SpFD, the quality of the learned semantic parser is dictated by the choice of how to update the model parameters, often determined empirically. As a result, several families of learning methods, including maximum marginal likelihood, reinforcement learning and margin based methods have been used. How to effectively explore different model choices could be crucial in practice.

Our contributions in this work are twofold. To address the first challenge, we propose a policy shaping (Griffith et al., 2013) method that incorporates simple, lightweight domain knowledge, such as a small set of lexical pairs of tokens in the question and program, in the form of a *critique policy* (§ 3). This helps bias the search towards the correct program, an important step to improve supervision signals, which benefits learning regardless of the choice of algorithm. To address the second challenge, we prove that the parameter update step in several algorithms are similar and can be viewed as special cases of a generalized update equation (§ 4). The equation contains two variable terms that govern the update behavior. Changing these two terms effectively defines an infinite class of learning algorithms where different values lead to significantly different results. We study this effect and propose a novel learning framework that improves over existing methods.

We evaluate our methods using the sequential question answering (SQA) dataset (Iyyer et al., 2017), and show that our proposed improvements to the search and update steps consistently enhance existing approaches. The proposed algorithm achieves new state-of-the-art and outperforms existing parsers by 5.0%.

2 Background

We give a formal problem definition of the semantic parsing task, followed by the general learning framework for solving it.

2.1 The Semantic Parsing Task

The problem discussed in this paper can be formally defined as follows. Let $\mathcal{X}$ be the set of all possible questions, $\mathcal{Y}$ programs (e.g., SQL-like queries), $\mathcal{T}$ tables (i.e., the structured data in this work) and $\mathcal{Z}$ answers. We further assume access to an executor $\Phi : \mathcal{Y} \times \mathcal{T} \to \mathcal{Z}$, that given a program $y \in \mathcal{Y}$ and a table $t \in \mathcal{T}$, generates an answer $\Phi(y, t) \in \mathcal{Z}$. We assume that the executor and all tables are deterministic and the executor can be called as many times as possible. To facilitate discussion in the following sections, we define an environment function $e_t : \mathcal{Y} \to \mathcal{Z}$, by applying the executor to the program as $e_t(y) = \Phi(y, t)$.

Given a question x and an environment e_t, our aim is to *generate* a program $y^* \in \mathcal{Y}$ and then execute it to produce the answer $e_t(y^*)$. Assume that for any $y \in \mathcal{Y}$, the score of y being a correct program for x is $\texttt{score}_\theta(y, x, t)$, parameterized by θ. The *inference* task is thus:

$$y^* = \arg\max_{y \in \mathcal{Y}} \texttt{score}_\theta(y, x, t) \qquad (1)$$

As the size of $\mathcal{Y}$ is exponential to the length of the program, a generic *search* procedure is typically employed for Eq. (1), as efficient dynamic algorithms typically do not exist. These search procedures generally maintain a beam of program states sorted according to some scoring function, where each program state represents an incomplete program. The search then generates a new program state from an existing state by performing an action. Each action adds a set of tokens (e.g., `Nation`) and keyword (e.g., `Select`) to a

program state. For example, in order to generate the program in Figure 1, the DynSP parser (Iyyer et al., 2017) will take the first action as adding the SQL expression `Select Nation`. Notice that $\texttt{score}_\theta$ can be used in either probabilistic or non-probabilistic models. For probabilistic models, we assume that it is a Boltzmann policy, meaning that $p_\theta(y \mid x, t) \propto \exp\{\texttt{score}_\theta(y, x, t)\}$.

2.2 Learning

Learning a semantic parser is equivalent to learning the parameters θ in the scoring function, which is a structured learning problem, due to the large, structured output space $\mathcal{Y}$. Structured learning algorithms generally consist of two major components: *search* and *update*. When the gold programs are available during training, the search procedure finds a set of high-scoring incorrect programs. These programs are used by the update step to derive loss for updating parameters. For example, these programs are used for approximating the partition-function in maximum-likelihood objective (Liang et al., 2011) and finding set of programs causing margin violation in margin based methods (Daumé III and Marcu, 2005). Depending on the exact algorithm being used, these two components are not necessarily separated into isolated steps. For instance, parameters can be updated in the middle of search (e.g., Huang et al., 2012).

For learning semantic parsers from denotations, where we assume only answers are available in a training set $\{(x_i, t_i, z_i)\}_{i=1}^N$ of N examples, the basic construction of the learning algorithms remains the same. However, the problems that search needs to handle in SpFD is more challenging. In addition to finding a set of high-scoring incorrect programs, the search procedure also needs to *guess* the correct program(s) evaluating to the gold answer z_i. This problem is further complicated by the presence of spurious programs, which generate the correct answer but are semantically incompatible with the question. For example, although all programs in Figure 2 evaluate to the same answer, only one of them is correct. The issue of the spurious programs also affects the design of model update. For instance, maximum marginal likelihood methods treat all the programs that evaluate to the gold answer equally, while maximum margin reward networks use model score to break tie and pick one of the programs as the correct reference.

3 Addressing Spurious Programs: Policy Shaping

Given a training example (x, t, z), the aim of the search step is to find a set $\mathcal{K}(\mathtt{x}, \mathtt{t}, \mathtt{z})$ of programs consisting of correct programs that evaluate to z and high-scoring incorrect programs. The search step should avoid picking up spurious programs for learning since such programs typically do not generalize. For example, in Figure 2, the spurious program `Select Nation Where Index is Min` will evaluate to an incorrect answer if the indices of the first two rows are swapped[1]. This problem is challenging since among the programs that evaluate to the correct answer, most of them are spurious.

The search step can be viewed as following an exploration policy $b_\theta(y|x, t, z)$ to explore the set of programs $\mathcal{Y}$. This exploration is often performed by beam search and at each step, we either sample from b_θ or take the top scoring programs. The set $\mathcal{K}(\mathtt{x}, \mathtt{t}, \mathtt{z})$ is then used by the update step for parameter update. Most search strategies use an exploration policy which is based on the score function, for example $b_\theta(y|x, t, z) \propto \exp\{\texttt{score}_\theta(y, t)\}$. However, this approach can suffer from a *divergence* phenomenon whereby the score of spurious programs picked up by the search in the first epoch increases, making it more likely for the search to pick them up in the future. Such *divergence* issues are common with latent-variable learning and often require careful initialization to overcome (Rose, 1998). Unfortunately such initialization schemes are not applicable for deep neural networks which form the model of most successful semantic parsers today (Jia and Liang, 2016; Misra and Artzi, 2016; Iyyer et al., 2017). Prior work, such as ϵ-greedy exploration (Guu et al., 2017), has reduced the severity of this problem by introducing random noise in the search procedure to avoid saturating the search on high-scoring spurious programs. However, random noise need not bias the search towards the correct program(s). In this paper, we introduce a simple policy-shaping method to guide the search. This approach allows incorporating prior knowledge in the exploration policy and can bias the search away from spurious programs.

[1] This transformation preserves the answer of the question.

Algorithm 1 Learning a semantic parser from denotation using generalized updates.

Input: Training set $\{(x_i, t_i, z_i\}_{i=1}^N$ (see Section 2), learning rate μ and stopping epoch T(see Section 4).
Definitions: $\texttt{score}_\theta(y, x, t)$ is a semantic parsing model parameterized by θ. $p_s(y \mid x, t)$ is the policy used for exploration and search(θ, x, t, z, p_s) generates candidate programs for updating parameters (see Section 3). Δ is the generalized update (see Section 4).
Output: Model parameters θ.

```
» Iterate over the training data.
for t = 1 to T, i = 1 to N do
    » Find candidate programs using the shaped policy.
    K = search(θ, x_i, t_i, z_i, p_s)
    » Compute generalized gradient updates
    θ = θ + μΔ(K)
return θ
```

Policy Shaping Policy shaping is a method to introduce prior knowledge into a policy (Griffith et al., 2013). Formally, let the current behavior policy be $b_\theta(y|x, t, z)$ and a predefined critique policy, the prior knowledge, be $p_c(y|x, t)$. Policy shaping defines a new *shaped behavior policy* $p_b(y|x, t)$ given by:

$$p_b(y|x, t) = \frac{b_\theta(y|x, t, z)p_c(y|x, t)}{\sum_{y' \in \mathcal{Y}} b_\theta(y'|x, t, z)p_c(y'|x, t)}. \quad (2)$$

Using the shaped policy for exploration biases the search towards the critique policy's preference. We next describe a simple critique policy that we use in this paper.

Lexical Policy Shaping We qualitatively observed that correct programs often contains tokens which are also present in the question. For example, the correct program in Figure 2 contains the token *Points*, which is also present in the question. We therefore, define a simple surface form similarity feature $\texttt{match}(x, y)$ that computes the ratio of number of non-keyword tokens in the program y that are also present in the question x.

However, surface-form similarity is often not enough. For example, both the first and fourth program in Figure 2 contain the token *Points* but only the fourth program is correct. Therefore, we also use a simple co-occurrence feature that triggers on frequently co-occurring pairs of tokens in the program and instruction. For example, the token *most* is highly likely to co-occur with a correct program containing the keyword `Max`. This happens for the example in Figure 2. Similarly the token *not* may co-occur with the keyword `NotEqual`. We assume access to a lexicon $\Lambda = \{(w_j, \omega_j)\}_{j=1}^k$ containing k lexical pairs of tokens and keywords. Each lexical pair (w, ω) maps the token w in a text to a keyword ω in a program. For a given program y and question x, we define a co-occurrence score as $\texttt{co\_occur}(y, x) = \sum_{(w,\omega)\in\Lambda} \mathbb{1}\{w \in x \wedge \omega \in y\}\}$. We define critique score $\texttt{critique}(y, x)$ as the sum of the `match` and `co_occur` scores. The critique policy is given by:

$$p_c(y|x, t) \propto \exp\left(\eta * \texttt{critique}(y, x)\right), \quad (3)$$

where η is a single scalar hyper-parameter denoting the confidence in the critique policy.

4 Addressing Update Strategy Selection: Generalized Update Equation

Given the set of programs generated by the search step, one can use many objectives to update the parameters. For example, previous work have utilized maximum marginal likelihood (Krishnamurthy et al., 2017; Guu et al., 2017), reinforcement learning (Zhong et al., 2017; Guu et al., 2017) and margin based methods (Iyyer et al., 2017). It could be difficult to choose the suitable algorithm from these options.

In this section, we propose a principle and general update equation such that previous update algorithms can be considered as special cases to this equation. Having a general update is important for the following reasons. First, it allows us to understand existing algorithms better by examining their basic properties. Second, the generalized update equation also makes it easy to implement and experiment with various different algorithms. Moreover, it provides a framework that enables the development of new variations or extensions of existing learning methods.

In the following, we describe how the commonly used algorithms are in fact very similar – their update rules can all be viewed as special cases of the proposed generalized update equation. Algorithm 1 shows the meta-learning framework. For every training example, we first find a set of candidates using an exploration policy (line 4). We use the program candidates to update the parameters (line 6).

4.1 Commonly Used Learning Algorithms

We briefly describe three algorithms: *maximum marginalized likelihood*, *policy gradient* and *maximum margin reward*.

Maximum Marginalized Likelihood The maximum marginalized likelihood method maximizes the log-likelihood of the training data by marginalizing over the set of programs.

$$\begin{aligned} J_{MML} &= \log p(z_i|x_i, t_i) \\ &= \log \sum_{y \in \mathcal{Y}} p(z_i|y, t_i) p(y|x_i, t_i) \quad (4) \end{aligned}$$

Because an answer is deterministically computed given a program and a table, we define $p(z \mid y, t)$ as 1 or 0 depending upon whether the y evaluates to z given t, or not. Let $\texttt{Gen}(\texttt{z}, \texttt{t}) \subseteq \mathcal{Y}$ be the set of compatible programs that evaluate to z given the table t. The objective can then be expressed as:

$$J_{MML} = \log \sum_{y \in \texttt{Gen}(\texttt{z}_\texttt{i}, \texttt{t}_\texttt{i})} p(y|x_i, t_i) \quad (5)$$

In practice, the summation over $\texttt{Gen}(.)$ is approximated by only using the compatible programs in the set $\mathcal{K}$ generated by the search step.

Policy Gradient Methods Most reinforcement learning approaches for semantic parsing assume access to a reward function $R : \mathcal{Y} \times \mathcal{X} \times \mathcal{Z} \rightarrow \mathbb{R}$, giving a scalar reward $R(y, z)$ for a given program y and the correct answer z.[2] We can further assume without loss of generality that the reward is always in $[0, 1]$. Reinforcement learning approaches maximize the expected reward J_{RL}:

$$J_{RL} = \sum_{y \in \mathcal{Y}} p(y|x_i, t_i) R(y, z_i) \quad (6)$$

J_{RL} is hard to approximate using numerical integration since the reward for all programs may not be known a priori. Policy gradient methods solve this by approximating the derivative using a sample from the policy. When the search space is large, the policy may fail to sample a correct program, which can greatly slow down the learning. Therefore, off-policy methods are sometimes introduced to bias the sampling towards high-reward yielding programs. In those methods, an additional exploration policy $u(y|x_i, t_i, z_i)$ is used to improve sampling. Importance weights are used to make the gradient unbiased (see Appendix for derivation).

[2]This is essentially a contextual bandit setting. Guu et al. (2017) also used this setting. A general reinforcement learning setting requires taking a sequence of actions and receiving a reward for each action. For example, a program can be viewed as a sequence of parsing actions, where each action can get a reward. We do not consider the general setting here.

Maximum Margin Reward For every training example (x_i, t_i, z_i), the maximum margin reward method finds the highest scoring program y_i that evaluates to z_i, as the *reference* program, from the set $\mathcal{K}$ of programs generated by the search. With a margin function $\delta : \mathcal{Y} \times \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}$ and reference program y, the set of programs $\mathcal{V}$ that violate the margin constraint can thus be defined as:

$$\begin{aligned} \mathcal{V} = \{y' \mid y' \in \mathcal{Y} \textbf{ and } &\texttt{score}_\theta(y, x, t) \\ &\leq \texttt{score}_\theta(y', x, t) + \delta(y, y', z)\}, \quad (7) \end{aligned}$$

where $\delta(y, y', z) = R(y, z) - R(y', z)$. Similarly, the program that most violates the constraint can be written as:

$$\begin{aligned} \bar{y} = \arg\max_{y' \in \mathcal{Y}} \{&\texttt{score}_\theta(y', x, t) + \delta(y, y', z) \\ &- \texttt{score}_\theta(y, x, t)\} \quad (8) \end{aligned}$$

The most-violation margin objective (negative margin loss) is thus defined as:

$$\begin{aligned} J_{MMR} = &- \max\{0, \texttt{score}_\theta(\bar{y}, x_i, t_i) \\ &- \texttt{score}_\theta(y_i, x_i, t_i) + \delta(y_i, \bar{y}, z_i)\} \end{aligned}$$

Unlike the previous two learning algorithms, margin methods only update the score of the reference program and the program that violates the margin.

4.2 Generalized Update Equation

Although the algorithms described in §4.1 seem very different on the surface, the gradients of their loss functions can in fact be described in the same generalized form, given in Eq. (9)[3]. In addition to the gradient of the model scoring function, this equation has two variable terms, $w(\cdot)$, $q(\cdot)$. We call the first term $w(y, x, t, z)$ *intensity*, which is a positive scalar value and the second term $q(y|x, t)$ the *competing distribution*, which is a probability distribution over programs. Varying them makes the equation equivalent to the update rule of the algorithms we discussed, as shown in Table 1. We also consider meritocratic update policy which uses a hyperparameter β to sharpen or smooth the intensity of maximum marginal likelihood (Guu et al., 2017).

Intuitively, $w(y, x, t, z)$ defines the positive part of the update equation, which defines how aggressively the update favors program y. Likewise, $q(y|x, t)$ defines the negative part of the learning

[3]See Appendix for the detailed derivation.

Generalized Update Equation:

$$\Delta(\mathcal{K}) = \sum_{y \in \mathcal{K}} w(y, x, t, z) \left(\nabla_\theta \texttt{score}_\theta(y, x, t) - \sum_{y' \in \mathcal{Y}} q(y'|x, t) \nabla_\theta \texttt{score}_\theta(y', x, t) \right) \quad (9)$$

Learning Algorithm	**Intensity** $w(y, x, t, z)$	**Competing Distribution** $q(y\|x, t)$
Maximum Margin Likelihood	$\frac{p(z\|y)p(y\|x)}{\sum_{y'} p(z\|y')p(y'\|x)}$	$p(y\|x)$
Meritocratic(β)	$\frac{(p(z\|y)p(y\|x))^\beta}{\sum_{y'}(p(z\|y')p(y'\|x))^\beta}$	$p(y\|x)$
REINFORCE	$\mathbb{1}\{y = \hat{y}\} R(y, z)$	$p(y\|x)$
Off-Policy Policy Gradient	$\mathbb{1}\{y = \hat{y}\}\, R(y, z) \frac{p(y\|x)}{u(y\|x,z)}$	$p(y\|x)$
Maximum Margin Reward (MMR)	$\mathbb{1}\{y = y^*\}$	$\mathbb{1}\{y = \bar{y}\}$
Maximum Margin Avg. Violation Reward (MAVER)	$\mathbb{1}\{y = y^*\}$	$1/\lvert\mathcal{V}\rvert \mathbb{1}\{y \in \mathcal{V}\}$

Table 1: Parameter updates for various learning algorithms are special cases of Eq. (9), with different choices of intensity w and competing distribution q. We do not show dependence upon table t for brevity. For off-policy policy gradient, u is the exploration policy. For margin methods, y^* is the reference program (see §4.1), $\mathcal{V}$ is the set of programs that violate the margin constraint (cf. Eq. (7)) and $\bar{y}$ is the most violating program (cf. Eq. (8)). For REINFORCE, $\hat{y}$ is sampled from $\mathcal{K}$ using $p(.)$ whereas for Off-Policy Policy Gradient, $\hat{y}$ is sampled using $u(.)$.

algorithm, namely how aggressively the update penalizes the members of the program set.

The generalized update equation provides a tool for better understanding individual algorithm, and helps shed some light on when a particular method may perform better.

Intensity versus Search Quality In SpFD, the effectiveness of the algorithms for SpFD is closely related to the quality of the search results given that the gold program is not available. Intuitively, if the search quality is good, the update algorithm could be aggressive on updating the model parameters. When the search quality is poor, the algorithm should be conservative.

The intensity $w(\cdot)$ is closely related to the aggressiveness of the algorithm. For example, the maximum marginal likelihood is less aggressive given that it produces a non-zero intensity over all programs in the program set $\mathcal{K}$ that evaluate to the correct answer. The intensity for a particular correct program y is proportional to its probability $p(y|x, t)$. Further, meritocratic update becomes more aggressive as β becomes larger.

In contrast, REINFORCE and maximum margin reward both have a non-zero intensity only on a single program in $\mathcal{K}$. This value is 1.0 for maximum margin reward, while for reinforcement learning, this value is the reward. Maximum margin reward therefore updates most aggressively in favor of its selection while maximum marginal likelihood tends to hedge its bet. Therefore, the maximum margin methods should benefit the most when the search quality improves.

Stability The general equation also allows us to investigate the stability of a model update algorithm. In general, the variance of update direction can be high, hence less stable, if the model update algorithm has peaky competing distribution, or it puts all of its intensity on a single program. For example, REINFORCE only samples one program and puts non-zero intensity only on that program, so it could be unstable depending on the sampling results.

The competing distribution affects the stability of the algorithm. For example, maximum margin reward penalizes only the most violating program and is benign to other incorrect programs. Therefore, the MMR algorithm could be unstable during training.

New Model Update Algorithm The general equation provides a framework that enables the development of new variations or extensions of existing learning methods. For example, in order to improve the stability of the MMR algorithm, we propose a simple variant of maximum margin reward, which penalizes all violating programs instead of only the most violating one. We call this approach *maximum margin average violation reward* (MAVER), which is included in Table 1 as well. Given that MAVER effectively considers

more negative examples during each update, we expect that it is more stable compared to the MMR algorithm.

5 Experiments

We describe the setup in §5.1 and results in §5.2.

5.1 Setup

Dataset We use the sequential question answering (SQA) dataset (Iyyer et al., 2017) for our experiments. SQA contains 6,066 sequences and each sequence contains up to 3 questions, with 17,553 questions in total. The data is partitioned into training (83%) and test (17%) splits. We use 4/5 of the original train split as our training set and the remaining 1/5 as the dev set. We evaluate using exact match on answer. Previous state-of-the-art result on the SQA dataset is 44.7% accuracy, using maximum margin reward learning.

Semantic Parser Our semantic parser is based on DynSP (Iyyer et al., 2017), which contains a set of SQL actions, such as adding a clause (e.g., `Select Column`) or adding an operator (e.g., `Max`). Each action has an associated neural network module that generates the score for the action based on the instruction, the table and the list of past actions. The score of the entire program is given by the sum of scores of all actions.

We modified DynSP to improve its representational capacity. We refer to the new parser as DynSP++. Most notably, we included new features and introduced two additional parser actions. See Appendix 8.2 for more details. While these improvements help us achieve state-of-the-art results, the majority of the gain comes from the learning contributions described in this paper.

Hyperparameters For each experiment, we train the model for 30 epochs. We find the optimal stopping epoch by evaluating the model on the dev set. We then train on train+dev set till the stopping epoch and evaluate the model on the held-out test set. Model parameters are trained using stochastic gradient descent with learning rate of 0.1. We set the hyperparameter η for policy shaping to 5. All hyperparameters were tuned on the dev set. We use 40 lexical pairs for defining the `co-occur` score. We used common English superlatives (e.g., *highest*, *most*) and comparators (e.g., *more*, *larger*) and did not fit the lexical pairs based on the dataset.

Given the model parameter θ, we use a base exploration policy defined in (Iyyer et al., 2017). This exploration policy is given by $b_\theta(y \mid x, t, z) \propto \exp(\lambda \cdot R(y, z) + \texttt{score}_\theta(y, \theta, z))$. $R(y, z)$ is the reward function of the incomplete program y, given the answer z. We use a reward function $R(y, z)$ given by the Jaccard similarity of the gold answer z and the answer generated by the program y. The value of λ is set to infinity, which essentially is equivalent to sorting the programs based on the reward and using the current model score for tie breaking. Further, we prune all syntactically invalid programs. For more details, we refer the reader to (Iyyer et al., 2017).

5.2 Results

Table 2 contains the dev and test results when using our algorithm on the SQA dataset. We observe that margin based methods perform better than maximum likelihood methods and policy gradient in our experiment. Policy shaping in general improves the performance across different algorithms. Our best test results outperform previous SOTA by 5.0%.

Policy Gradient vs Off-Policy Gradient REINFORCE, a simple policy gradient method, achieved extremely poor performance. This likely due to the problem of exploration and having to sample from a large space of programs. This is further corroborated from observing the much superior performance of off-policy policy gradient methods. Thus, the sampling policy is an important factor to consider for policy gradient methods.

The Effect of Policy Shaping We observe that the improvement due to policy shaping is 6.0% on the SQA dataset for MAVER and only 1.3% for maximum marginal likelihood. We also observe that as β increases, the improvement due to policy shaping for meritocratic update increases. This supports our hypothesis that aggressive updates of margin based methods is beneficial when the search method is more accurate as compared to maximum marginal likelihood which hedges its bet between all programs that evaluate to the right answer.

Stability of MMR In Section 4, the general update equation helps us point out that MMR could be unstable due to the peaky competing distribution. MAVER was proposed to increase the stability of the algorithm. To measure stability, we cal-

Algorithm	Dev		Test	
	w.o. Shaping	w. Shaping	w.o. Shaping	w. Shaping
Maximum Margin Likelihood	33.2	32.5	31.0	32.3
Meritocratic ($\beta = 0$)	27.1	28.1	31.3	30.1
Meritocratic ($\beta = 0.5$)	28.3	28.7	31.7	32.0
Meritocratic ($\beta = \infty$)	39.3	41.6	41.6	45.2
REINFORCE	10.2	11.8	2.4	4.0
Off-Policy Policy Gradient	36.6	38.6	42.6	44.1
MMR	38.4	40.7	43.2	46.9
MAVER	39.6	44.1	43.7	**49.7**

Table 2: Experimental results on different model update algorithms, with and without policy shaping.

w	q	Dev
MMR	MML	**41.9**
Off-Policy Policy Gradient	MMR	37.0
MMR	MMR	40.7

Table 3: The dev set results on the new variations of the update algorithms.

culate the mean absolute difference of the development set accuracy between successive epochs during training, as it indicates how much an algorithm's performance fluctuates during training. With this metric, we found mean difference for MAVER is 0.57% where the mean difference for MMR is 0.9%. This indicates that MAVER is in fact more stable than MMR.

Other variations We also analyze other possible novel learning algorithms that are made possible due to generalized update equations. Table 3 reports development results using these algorithms. By mixing different intensity scalars and competing distribution from different algorithms, we can create new variations of the model update algorithm. In Table 3, we show that by mixing the MMR's intensity and MML's competing distribution, we can create an algorithm that outperform MMR on the development set.

Policy Shaping helps against Spurious Programs In order to better understand if policy shaping helps bias the search away from spurious programs, we analyze 100 training examples. We look at the highest scoring program in the beam at the end of training using MAVER. Without policy shaping, we found that 53 programs were spurious while using policy shaping this number came down to 23. We list few examples of spurious program errors corrected by policy shaping in Table 4.

Policy Shaping vs Model Shaping Critique policy contains useful information that can bias the search away from spurious programs. Therefore, one can also consider making the critique policy as part of the model. We call this model shaping. We define our model to be the shaped policy and train and test using the new model. Using MAVER updates, we found that the dev accuracy dropped to 37.1%. We conjecture that the strong prior in the critique policy can hinder generalization in model shaping.

6 Related Work

Semantic Parsing from Denotation Mapping natural language text to formal meaning representation was first studied by Montague (1970). Early work on learning semantic parsers rely on labeled formal representations as the supervision signals (Zettlemoyer and Collins, 2005, 2007; Zelle and Mooney, 1993). However, because getting access to gold formal representation generally requires expensive annotations by an expert, distant supervision approaches, where semantic parsers are learned from denotation only, have become the main learning paradigm (e.g., Clarke et al., 2010; Liang et al., 2011; Artzi and Zettlemoyer, 2013; Berant et al., 2013; Iyyer et al., 2017; Krishnamurthy et al., 2017). Guu et al. (2017) studied the problem of spurious programs and considered adding noise to diversify the search procedure and introduced meritocratic updates.

Reinforcement Learning Algorithms Reinforcement learning algorithms have been applied to various NLP problems including dialogue (Li et al., 2016), text-based games (Narasimhan et al., 2015), information extraction (Narasimhan et al., 2016), coreference resolution (Clark and Man-

Question	without policy shaping	with policy shaping
"of these teams, which had more than 21 losses?"	`SELECT Club WHERE Losses = ROW 15`	`SELECT Club WHERE Losses > 21`
"of the remaining, which earned the most bronze medals?"	`SELECT Nation WHERE Rank = ROW 1`	`FollowUp WHERE Bronze is Max`
"of those competitors from germany, which was not paul sievert?"	`SELECT Name WHERE Time (hand) = ROW 3`	`FollowUp WHERE Name != ROW 5`

Table 4: Training examples and the highest ranked program in the beam search, scored according to the shaped policy, after training with MAVER. Using policy shaping, we can recover from failures due to spurious programs in the search step for these examples.

ning, 2016), semantic parsing (Guu et al., 2017) and instruction following (Misra et al., 2017). Guu et al. (2017) show that policy gradient methods underperform maximum marginal likelihood approaches. Our result on the SQA dataset supports their observation. However, we show that using off-policy sampling, policy gradient methods can provide superior performance to maximum marginal likelihood methods.

Margin-based Learning Margin-based methods have been considered in the context of SVM learning. In the NLP literature, margin based learning has been applied to parsing (Taskar et al., 2004; McDonald et al., 2005), text classification (Taskar et al., 2003), machine translation (Watanabe et al., 2007) and semantic parsing (Iyyer et al., 2017). Kummerfeld et al. (2015) found that max-margin based methods generally outperform likelihood maximization on a range of tasks. Previous work have studied connections between margin based method and likelihood maximization for supervised learning setting. We show them as special cases of our unified update equation for distant supervision learning. Similar to this work, Lee et al. (2016) also found that in the context of supervised learning, margin-based algorithms which update all violated examples perform better than the one that only updates the most violated example.

Latent Variable Modeling Learning semantic parsers from denotation can be viewed as a latent variable modeling problem, where the program is the latent variable. Probabilistic latent variable models have been studied using EM-algorithm and its variant (Dempster et al., 1977). The graphical model literature has studied latent variable learning on margin-based methods (Yu and Joachims, 2009) and probabilistic models (Quattoni et al., 2007). Samdani et al. (2012) studied various variants of EM algorithm and showed that all of them are special cases of a unified framework. Our generalized update framework is similar in spirit.

7 Conclusion

In this paper, we propose a general update equation from semantic parsing from denotation and propose a policy shaping method for addressing the spurious program challenge. For the future, we plan to apply the proposed learning framework to more semantic parsing tasks and consider new methods for policy shaping.

8 Acknowledgements

We thank Ryan Benmalek, Alane Suhr, Yoav Artzi, Claire Cardie, Chris Quirk, Michel Galley and members of the Cornell NLP group for their valuable comments. We are also grateful to Allen Institute for Artificial Intelligence for the computing resource support. This work was initially started when the first author interned at Microsoft Research.

References

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

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

Kevin Clark and D. Christopher Manning. 2016. Deep reinforcement learning for mention-ranking coreference models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the

world's response. In *Proceedings of the Conference on Computational Natural Language Learning*.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 169–176. ACM.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Lee Isbell, and Andrea Lockerd Thomaz. 2013. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062. Association for Computational Linguistics.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526.

Jonathan K. Kummerfeld, Taylor Berg-Kirkpatrick, and Dan Klein. 2015. An empirical analysis of optimization for max-margin nlp. In *EMNLP*.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2366–2376.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Percy Liang, Michael I Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Ryan T. McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *ACL*.

Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. Arxiv preprint: https://arxiv.org/abs/1704.08795.

Dipendra K. Misra and Yoav Artzi. 2016. Neural shift-reduce CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kumar Dipendra Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. 2015. Environment-driven lexicon induction for high-level instructions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Richard Montague. 1970. English as a formal language.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. 2016. Improving information extraction by acquiring external evidence with reinforcement learning. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Morency Collins, and Trevor Darrell. 2007. Hidden conditional random fields. *IEEE transactions on pattern analysis and machine intelligence*, 29(10).

Kenneth Rose. 1998. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239.

Rajhans Samdani, Ming-Wei Chang, and Dan Roth. 2012. Unified expectation maximization. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, pages 688–698. Association for Computational Linguistics.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max-margin markov networks. In *NIPS*.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *EMNLP-CoNLL*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8.

Chun-Nam John Yu and Thorsten Joachims. 2009. Learning structural svms with latent variables. In *Proceedings of the 26th annual international conference on machine learning*, pages 1169–1176. ACM.

John M Zelle and Raymond J Mooney. 1993. Learning semantic grammars with constructive inductive logic programming. In *AAAI*, pages 817–822.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Appendix

8.1 Deriving the updates of common algorithms

Below we derive the gradient of various learning algorithms. We assume access to a training data $\{(x_i, t_i, z_i)\}_{i=1}^N$ with N examples. Given an input instruction x and table t, we model the score of a program using a score function $\texttt{score}_\theta(y, x, z)$ with parameters θ. When the model is probabilistic, we assume it is a Boltzmann distribution given by $p(y \mid x, t) \propto \exp\{\texttt{score}_\theta(y, x, t)\}$.

In our result, we will be using.

$$\nabla_\theta \log p(y \mid x, t) = \nabla_\theta \texttt{score}_\theta(y, x, t) - \sum_{y' \in \mathcal{Y}} p(y' \mid x, t) \nabla_\theta \texttt{score}_\theta(y', x, t) \tag{10}$$

Maximum Marginal Likelihood The maximum marginal objective J_{MML} can be expressed as:

$$J_{MML} = \sum_{i=1}^{N} \log \sum_{y \in \texttt{Gen}(t_i, z_i)} p(y \mid x_i, t_i)$$

where $\texttt{Gen}(t, z)$ is the set of all programs from $\mathcal{Y}$ that generate the answer z on table t. Taking the derivative gives us:

$$\begin{aligned} \nabla_\theta J_{MML} &= \sum_{i=1}^{N} \nabla_\theta \log \sum_{y \in \texttt{Gen}(t_i, z_i)} p(y \mid x_i, t_i) \\ &= \sum_{i=1}^{N} \frac{\sum_{y \in \texttt{Gen}(t_i, z_i)} \nabla_\theta p(y \mid x_i, t_i)}{\sum_{y \in \texttt{Gen}(t_i, z_i)} p(y \mid x_i, t_i)} \end{aligned}$$

Then using Equation 10, we get:

$$\nabla_\theta J_{MML} = \sum_{i=1}^{N} \sum_{y \in \texttt{Gen}(t_i, z_i)} w(y \mid x_i, t_i) \left\{ \nabla_\theta \texttt{score}_\theta(y, x, t) - \sum_{y' \in \mathcal{Y}} p(y' \mid x, t) \nabla_\theta \texttt{score}_\theta(y', x, t) \right\} \tag{11}$$

where

$$w(y, x, t) = \frac{p(y \mid x, t)}{\sum_{y' \in \texttt{Gen}(t, z)} p(y' \mid x, t)}$$

Policy Gradient Methods Reinforcement learning based approaches maximize the expected reward objective.

$$J_{RL} = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} p(y \mid x_i, t_i) R(y, z_i) \tag{12}$$

We can then compute the derivate of this objective as:

$$\nabla_\theta J_{RL} = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \nabla_\theta p(y \mid x_i, t_i) R(y, z_i) \tag{13}$$

The above summation can be expressed as expectation (Williams, 1992).

$$\nabla_\theta J_{RL} = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} p(y \mid x_i, t_i) \nabla_\theta \log p(y \mid x_i, t_i) R(y, z_i) \tag{14}$$

For every example i, we sample a program y_i from $\mathcal{Y}$ using the policy $p(. \mid x_i, t_i)$. In practice this sampling is done over the output programs of the search step.

$$\begin{aligned}
\nabla_\theta J_{RL} &\approx \sum_{i=1}^{N} \nabla_\theta \log p(y_i \mid x_i, t_i) R(y_i, z_i) \\
&\quad \text{using gradient of } \log p(. \mid .) \\
&\approx \sum_{i=1}^{N} R(y_i, z_i) \left\{ \nabla_\theta \texttt{score}_\theta(y_i, x_i, t) - \sum_{y' \in \mathcal{Y}} p(y' \mid x_i, t_i) \nabla_\theta \texttt{score}_\theta(y', x_i, t_i) \right\}
\end{aligned}$$

Off-Policy Policy Gradient Methods In off-policy policy gradient method, instead of sampling a program using the current policy $p(. \mid .)$, we use a separate exploration policy $u(. \mid .)$. For the i^{th} training example, we sample a program y_i from the exploration policy $u(. \mid x_i, t_i, z_i)$. Thus the gradient of expected reward objective from previous paragraph can be expressed as:

$$\begin{aligned}
\nabla_\theta J_{RL} &= \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \nabla_\theta p(y \mid x_i, t_i) R(y, z_i) \\
&= \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} u(y \mid x_i, t_i, z_i) \frac{p(y \mid x_i, t_i)}{u(y \mid x_i, t_i, z_i)} \nabla_\theta \log p(y \mid x_i, t_i) R(y, z_i) \\
&\quad \text{using, for every } i \; y_i \sim u(. \mid x_i, t_i, z_i) \\
&\approx \sum_{i=1}^{N} \frac{p(y \mid x_i, t_i)}{u(y \mid x_i, t_i, z_i)} \nabla_\theta \log p(y \mid x_i, t_i) R(y, z_i)
\end{aligned}$$

the ratio of $\frac{p(y|x,t)}{u(y|x,t,z)}$ is the importance weight correction. In practice, we sample a program from the output of the search step.

Maximum Margin Reward (MMR) For the i^{th} training example, let $\mathcal{K}(x_i, t_i, z_i)$ be the set of programs produced by the search step. Then MMR finds the highest scoring program in this set, which evaluates to the correct answer. Let this program be y_i. MMR optimizes the parameter to satisfy the following constraint:

$$\texttt{score}_\theta(y_i, x_i, t_i) \geq \texttt{score}_\theta(y', x_i, t_i) + \delta(y_i, y', z_i) \;\; y' \in \mathcal{Y} \tag{15}$$

where the margin $\delta(y_i, y', z_i)$ is given by $R(y_i, z_i) - R(y', z_i)$. Let $\mathcal{V}$ be the set of violations given by: $\mathcal{V} = \{\texttt{score}_\theta(y', x_i, t_i) - \texttt{score}_\theta(y_i, x_i, t_i) + \delta(y_i, y', z_i) > 0 \mid y \in \mathcal{Y}\}$.

At each training step, MMR only considers the program which is most violating the constraint. When $|\mathcal{V}| > 0$ then let y^* be the most violating program given by:

$$\begin{aligned}
\bar{y} &= \arg\max_{y' \in \mathcal{Y}} \left\{ \texttt{score}_\theta(y', x_i, t_i) - \texttt{score}_\theta(y_i, x_i, t_i) + R(y_i, z_i) - R(y', z_i) \right\} \\
&= \arg\max_{y' \in \mathcal{Y}} \left\{ \texttt{score}_\theta(y', x_i, t_i) - R(y', z_i) \right\}
\end{aligned}$$

Using the most violation approximation, the objective for MMR can be expressed as negative of hinge loss:

$$J_{MMR} = -\max\{0, \texttt{score}_\theta(\bar{y}, x_i, t_i) - \texttt{score}_\theta(y_i, x_i, t_i) + R(y_i, z_i) - R(\bar{y}, z_i)\} \tag{16}$$

Our definition of y^* allows us to write the above objective as:

$$J_{MMR} = -\mathbb{1}\{\mathcal{V} > 0\}\{\texttt{score}_\theta(\bar{y}, x_i, t_i) - \texttt{score}_\theta(y_i, x_i, t_i) + R(y_i, z_i) - R(\bar{y}, z_i)\} \quad (17)$$

the gradient is then given by:

$$\nabla_\theta J_{MMR} = -\mathbb{1}\{\mathcal{V} > 0\}\{\nabla_\theta \texttt{score}_\theta(\bar{y}, x_i, t_i) - \nabla_\theta \texttt{score}_\theta(y_i, x_i, t_i)\} \quad (18)$$

Maximum Margin Average Violation Reward (MAVER) Given a training example, MAVER considers the same constraints and margin as MMR. However instead of considering only the most violated program, it considers all violations. Formally, for every example (x_i, t_i, z_i) we compute the ideal program y_i as in MMR. We then optimize the average negative hinge loss error over all violations:

$$J_{MAVER} = -\frac{1}{\mathcal{V}} \sum_{y' \in \mathcal{V}} \{\texttt{score}_\theta(y', x_i, t_i) - \texttt{score}_\theta(y_i, x_i, t_i) + R(y_i, z_i) - R(y', z_i)\} \quad (19)$$

Taking the derivative we get:

$$\begin{aligned} \nabla_\theta J_{MAVER} &= -\frac{1}{\mathcal{V}} \sum_{y' \in \mathcal{V}} \{\nabla_\theta \texttt{score}_\theta(y', x_i, t_i) - \nabla_\theta \texttt{score}_\theta(y_i, x_i, t_i))\} \\ &= \nabla_\theta \texttt{score}_\theta(y_i, x_i, t_i)) - \sum_{y' \in \mathcal{V}} \frac{1}{|\mathcal{V}|} \nabla_\theta \texttt{score}_\theta(y', x_i, t_i) \end{aligned}$$

8.2 Changes to DynSP Parser

We make following 3 changes to the DynSP parser to increase its representational power. The new parser is called DynSP++. We describe these three changes below:

1. We add two new actions: disjunction (`OR`) and follow-up cell (`FpCell`). The disjunction operation is used to describe multiple conditions together example:

   Question: *what is the population of USA or China?*
   Program: `Select Population Where Name = China OR Name = USA`

   Follow-up cell is only used for a question which is following another question and whose answer is a single cell in the table. Follow-up cell is used to select values for another column corresponding to this cell.

   Question: *and who scored that point?*
   Program: `Select Name Follow-Up Cell`

2. We add surface form features in the model for column and cell. These features trigger on token match between an entity in the table (column name or cell value) and a question. We consider two tokens: exact match and overlap. The exact match is 1.0 when every token in the entity is present in the question and 0 otherwise. Overlap feature is 1.0 when atleast one token in the entity is present in the question and 0 otherwise. We also consider related-column features that were considered by Krishnamurthy et al. (2017).

3. We also add recall features which measure how many tokens in the question that are also present in the table are covered by a given program. To compute this feature, we first compute the set $\mathcal{E}_1$ of all tokens in the question that are also present in the table. We then find a set of non-keyword tokens $\mathcal{E}_2$ that are present in the program. The recall score is then given by $w * \frac{|\mathcal{E}_1 - \mathcal{E}_2|}{|\mathcal{E}_1|}$, where w is a learned parameter.