Memory Augmented Policy Optimization for Program Synthesis with Generalization

Chen Liang*
Northwestern University
crazydonkey200@gmail.com

Mohammad Norouzi
Google Brain
mnorouzi@google.com

Jonathan Berant
Tel-Aviv University
joberant@cs.tau.ac.il

Quoc Le
Google Brain
qvl@google.com

Ni Lao*
SayMosaic Inc.
ni.lao@mosaix.ai

Abstract

This paper presents Memory Augmented Policy Optimization (MAPO): a novel policy optimization formulation that incorporates a memory buffer of promising trajectories to reduce the variance of policy gradient estimates for deterministic environments with discrete actions. The formulation expresses the expected return objective as a weighted sum of two terms: an expectation over a memory of trajectories with high rewards, and a separate expectation over the trajectories outside the memory. We propose 3 techniques to make an efficient training algorithm for MAPO: (1) distributed sampling from inside and outside memory with an actor-learner architecture; (2) a marginal likelihood constraint over the memory to accelerate training; (3) systematic exploration to discover high reward trajectories. MAPO improves the sample efficiency and robustness of policy gradient, especially on tasks with a sparse reward. We evaluate MAPO on *weakly supervised* program synthesis from *natural language* with an emphasis on *generalization*. On the WIKITABLEQUESTIONS benchmark we improve the state-of-the-art by 2.5%, achieving an accuracy of 46.2%, and on the WIKISQL benchmark, MAPO achieves an accuracy of 74.9% with only weak supervision, outperforming several strong baselines with full supervision. Our code is open sourced at `https://github.com/crazydonkey200/neural-symbolic-machines`.

1 Introduction

There has been a recent surge of interest in applying policy gradient methods to various application domains including program synthesis [25, 16, 63, 10], dialogue generation [24, 11], architecture search [64, 66], game [48, 29] and continuous control [40, 45]. Simple policy gradient methods like REINFORCE [53] use Monte Carlo samples from the current policy to perform an *on-policy* optimization of the expected return. This often leads to unstable learning dynamics and poor sample efficiency, sometimes underperforming random search [28].

The difficulty of gradient based policy optimization stems from a few sources: (1) policy gradient estimates have a large *variance*; (2) samples from a randomly initialized policy often attain small rewards, resulting in a slow training progress in the initial phase; (3) random policy samples do not explore the search space efficiently because many samples can be repeated. These issues can be especially prohibitive in applications such as program synthesis and robotics [4], which involve a large search space with sparse rewards. In such tasks, a high reward is achieved only after a long sequence of *correct* actions. For instance, in program synthesis, only a few programs in the large

* Part of the work was done while Chen Liang and Ni Lao were at Google.

Preprint. Work in progress.

program space lead to the correct functional form. Unfortunately, this often leads to forgetting a high reward trajectory unless it is re-sampled frequently [25, 3].

This paper presents MAPO: a novel formulation of policy optimization for deterministic environments with discrete actions, which incorporates a memory buffer of promising trajectories within the policy gradient framework. It estimates the policy gradient as a weighted sum of an expectation over the trajectories inside the memory and a separate expectation over those outside the memory. The gradient estimates are unbiased and attain lower variance provided that trajectories in the memory buffer have non-negligible probability. Because high-reward trajectories remain in the memory, it is almost impossible to forget them. To make an efficient algorithm for MAPO, we propose 3 techniques: (1) distributed sampling from inside and outside memory buffer in an actor-learner architecture; (2) a constraint on the marginal likelihood of the high-reward trajectories in the memory buffer to accelerate training at the cost of introducing some bias at the initial training stage;(3) systematic exploration of the search space to efficiently discover the high-reward trajectories.

We assess the effectiveness of MAPO on *weakly supervised* program synthesis from *natural language* (see Section 2). Program synthesis presents a unique opportunity to study *generalization* in the context of policy optimization, while having an impact on a real world application. On the challenging WikiTableQuestions [36] benchmark, MAPO achieves an accuracy of 46.2% on the test set, significantly outperforming the previous state-of-the-art of 43.7% [62]. Interestingly, on the WikiSQL [63] benchmark, MAPO achieves an accuracy of 74.9% without the supervision of gold programs, outperforming several strong *fully supervised* baselines.

2 The Problem of Weakly Supervised Contextual Program Synthesis

Consider the problem of learning to map a natural language question $\mathbf{x}$ to a structured query $\mathbf{a}$ in a programming language such as SQL (*e.g.*, [63]), or converting a textual problem description into a piece of source code, *e.g.*, in python, as in programming competitions (*e.g.*, [5]). We call this family of problems *contextual program synthesis* and aim at tackling such problems in a weakly supervised setting – i.e., no correct action sequence $\mathbf{a}$ is given as part of the training data, and training needs to solve the hard problem of exploring a large program space. Table 1 shows an example question-answer pair. The model needs to first discover the programs that can generate the correct answer in a given context, and then learn to generalize to new contexts.

Year	Venue	Position	Event	Time
2001	Hungary	2nd	400m	47.12
2003	Finland	1st	400m	46.69
2005	Germany	11th	400m	46.62
2007	Thailand	1st	relay	182.05
2008	China	7th	relay	180.32

Table 1: $\mathbf{x}$: Where did the last 1st place finish occur? $\mathbf{y}$: Thailand

We avoid strong supervision at the program level and aim to discover latent programs from weak supervision via trial and error. The reason is that weak supervision, for example, natural language question-answer pairs, can be collected at scale, but annotations of correct programs are harder to collect because they require expertise. And if the programming language changes, the same weak supervision data can still be used while the program annotations will need to be updated. For ambiguous questions, weak supervision also has the benefit of avoiding raters' bias, and directly learns from users' preferences.

We formulate the problem of *weakly supervised* contextual program synthesis as follows: to generate a program by using a parametric mapping function, $\hat{\mathbf{a}} = f(\mathbf{x}; \theta)$, where θ denotes the model parameters. The quality of a generated program $\hat{\mathbf{a}}$ is measured in terms of a scoring or *reward* function $R(\hat{\mathbf{a}} \mid \mathbf{x}, \mathbf{y})$. The reward function may evaluate a program by executing it on a real environment and comparing the emitted output against the correct answer. For example, it is natural to define a binary reward that is 1 when the output equals the answer and 0 otherwise. We assume that the context $\mathbf{x}$ includes both a natural language input and an environment, for example an interpreter or a database, on which the program will be executed. Given a dataset of context-answer pairs, $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, the goal is to find an optimal parameter θ^* that parameterizes a mapping of $\mathbf{x} \to \mathbf{a}$ with maximum empirical return on a *heldout test set*.

One can think of the problem of contextual program synthesis as an instance of *reinforcement learning (RL)* with *sparse terminal rewards* and *deterministic dynamics*, for which *generalization* plays a key role. There has been some recent attempts in the RL community to study generalization to unseen initial conditions (*e.g.* [41, 33]). However, most prior work aims to maximize empirical return on the training environment [6, 9]. The problem of contextual program synthesis presents a natural application of RL for which generalization is the main concern.

3 Optimization of Expected Return via Policy Gradients

To learn a mapping of (context $\mathbf{x}$) $\rightarrow$ (program $\mathbf{a}$), we optimize the parameters of a conditional distribution $\pi_\theta(\mathbf{a} \mid \mathbf{x})$ that assigns a probability to each program given the context. That is, π_θ is a distribution over the *countable* set of all possible programs, denoted $\mathcal{A}$. Thus $\forall \mathbf{a} \in \mathcal{A} : \ \pi_\theta(\mathbf{a} \mid \mathbf{x}) \geqslant 0$ and $\sum_{\mathbf{a} \in \mathcal{A}} \pi_\theta(\mathbf{a} \mid \mathbf{x}) = 1$. Then, to synthesize a program for a novel context, one finds the most likely program under the distribution π_θ via exact or approximate inference $\hat{\mathbf{a}} \approx \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}} \pi_\theta(\mathbf{a} \mid \mathbf{x})$.

Autoregressive models present a tractable family of distributions that estimates the probability of a sequence of tokens, one token at a time, often from left to right. To handle variable sequence length, one includes a special *end-of-sequence* token at the end of the sequences. We express the probability of a program $\mathbf{a}$ given $\mathbf{x}$ as $\pi_\theta(\mathbf{a} \mid \mathbf{x}) \equiv \prod_{i=t}^{|\mathbf{a}|} \pi_\theta(a_t \mid \mathbf{a}_{<t}, \mathbf{x})$,where $\mathbf{a}_{<t} \equiv (a_1, \ldots, a_{t-1})$ denotes a prefix of the program $\mathbf{a}$. One often uses a recurrent neural network (*e.g.* [19]) to predict the probability of each token given the prefix and the context.

In the absence of ground truth programs, policy gradient techniques present a way to optimize the parameters of a stochastic policy π_θ via optimization of *expected return*. Given a training dataset of context-answer pairs, $\mathcal{D} \equiv \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, the objective is expressed as $\mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{x})} \, R(\mathbf{a} \mid \mathbf{x}, \mathbf{y})$. The reward function $R(\mathbf{a} \mid \mathbf{x}, \mathbf{y})$ evaluates a complete program $\mathbf{a}$, based on the context $\mathbf{x}$ and the correct answer $\mathbf{y}$. These assumptions characterize the problem of program synthesis well, but they also apply to many other discrete optimization domains. Note that these assumptions are required for the exact unbiased gradients, but we expect the MAPO algorithm to benefit many discrete RL benchmarks on which these assumptions apply to a big extent.

Simplified notation. In what follows, we simplify the notation by dropping the dependence of the policy and the reward on $\mathbf{x}$ and $\mathbf{y}$. We use a notation of $\pi_\theta(\mathbf{a})$ instead of $\pi_\theta(\mathbf{a} \mid \mathbf{x})$ and $R(\mathbf{a})$ instead of $R(\mathbf{a} \mid \mathbf{x}, \mathbf{y})$, to make the formulation less cluttered, but the equations hold in the general case.

We express the expected return objective in the simplified notation as,

$$\mathcal{O}_{\mathrm{ER}}(\theta) \; = \; \sum_{\mathbf{a} \in \mathcal{A}} \pi_\theta(\mathbf{a}) \, R(\mathbf{a}) \; = \; \mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a})} \, R(\mathbf{a}) \, . \tag{1}$$

The REINFORCE [53] algorithm presents an elegant and convenient way to estimate the gradient of the expected return (1) using Monte Carlo (MC) samples. Using K trajectories sampled *i.i.d.* from the current policy π_θ, denoted $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(K)}\}$, the gradient estimate can be expressed as,

$$\nabla_\theta \mathcal{O}_{\mathrm{ER}}(\theta) \; \approx \; \frac{1}{K} \sum_{k=1}^{K} \nabla \log \pi_\theta(\mathbf{a}^{(k)}) \, [R(\mathbf{a}^{(k)}) - b] \, , \tag{2}$$

where a baseline b is subtracted from the returns to reduce the variance of gradient estimates. A close-to-optimal form of a baseline is the on-policy average of the returns $\mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a})} \, R(\mathbf{a})$, often approximated empirically as $b = \sum_k R(\mathbf{a}^{(k)})/K$. This formulation enables direct optimization of $\mathcal{O}_{\mathrm{ER}}$ via MC sampling from an unknown search space, which also serves the purpose of exploration. To improve such exploration behavior, one often includes the entropy of the policy as an additional term inside the objective to prevent early convergence. However, the key limitation of the formulation stems from the difficulty of estimating the gradients accurately only using a few *fresh* samples.

There has been various attempts to incorporate off-policy samples within the policy gradient framework to improve the sample efficiency of the REINFORCE and actor-critic algorithms (*e.g.*, [12, 52, 46, 15]). Most of these approaches sample from an old policy and resort to truncated importance correction to obtain a low variance, but *biased* estimate of the gradients. Previous work has aimed to incorporate a replay buffer into policy gradient in the general RL setting of stochastic dynamics and possibly continous actions. By contrast, we focus on deterministic environments with discrete actions and develop a formulation resulting in an *unbiased* policy gradient estimate.

4 MAPO: Memory Augmented Policy Optimization

We consider a RL environment with a finite number of discrete actions, deterministic dynamics, and deterministic terminal returns. In other words, the set of all possible action trajectories $\mathcal{A}$ is countable, even though possibly infinite, and re-evaluating the return of a trajectory $R(\mathbf{a})$ twice results in the same value. These assumptions characterize the problem of program synthesis, but also apply to many combinatorial optimization domains (*e.g.*, [7]).

Our goal is to optimize the expected return objective (1) via gradient ascent. We assume the access to a memory buffer of trajectories and their corresponding returns denoted $\mathcal{B} \equiv \{(\mathbf{a}^{(i)}, r^{(i)})\}_{i=1}^{M}$, where $r^{(i)} = R(\mathbf{a}^{(i)})$. Let $\mathcal{B}_a$ and $\mathcal{B}_r$ denote the action trajectories and the rewards stored in the memory independently, *i.e.* $\mathcal{B}_a \equiv \{(\mathbf{a}^{(i)})\}_{i=1}^{M}$ and $\mathcal{B}_r \equiv \{(r^{(i)})\}_{i=1}^{M}$. Our key observation is that one can re-express the expected return objective in terms of a sum of two terms: an enumeration over the memory buffer elements, and a separate enumeration over the unknown trajectories,

$$\mathcal{O}_{\mathrm{ER}}(\theta) \;=\; \sum_{(\mathbf{a},r)\in\mathcal{B}} \pi_\theta(\mathbf{a})\, r \;+\; \sum_{\mathbf{a}\in(\mathcal{A}-\mathcal{B}_a)} \pi_\theta(\mathbf{a})\, R(\mathbf{a}), \tag{3}$$

where $\mathcal{A} - \mathcal{B}_a$ denotes the set of action trajectories not included in the memory. The significance of this decomposition hinges on the fact that one can enumerate the trajectories in the buffer and compute their exact expected reward and its gradient with no variance, and use the budget of MC samples to compute the expectation only in the unexplored part of the space.

Let $\pi_{\mathcal{B}} = \sum_{\mathbf{a}\in\mathcal{B}_a} \pi_\theta(\mathbf{a})$ denote the total probability of the trajectories in the buffer, and let $\pi_\theta^+(\mathbf{a})$ and $\pi_\theta^-(\mathbf{a})$ denote a normalized probability distribution inside and outside of the buffer,

$$\pi_\theta^+(\mathbf{a}) = \begin{cases} \pi_\theta(\mathbf{a})/\pi_{\mathcal{B}} & \text{if } \mathbf{a} \in \mathcal{B}_a \\ 0 & \text{if } \mathbf{a} \notin \mathcal{B}_a \end{cases}, \qquad \pi_\theta^-(\mathbf{a}) = \begin{cases} 0 & \text{if } \mathbf{a} \in \mathcal{B}_a \\ \pi_\theta(\mathbf{a})/(1-\pi_{\mathcal{B}}) & \text{if } \mathbf{a} \notin \mathcal{B}_a \end{cases}. \tag{4}$$

Then, one can re-express the expected return objective as the linear combination of two expectations,

$$\mathcal{O}_{\mathrm{ER}}(\theta) \;=\; \pi_{\mathcal{B}} \underbrace{\mathbb{E}_{\mathbf{a}\sim\pi_\theta^+(\mathbf{a})}\, R(\mathbf{a})}_{\text{enumeration inside } \mathcal{B}} + (1-\pi_{\mathcal{B}}) \underbrace{\mathbb{E}_{\mathbf{a}\sim\pi_\theta^-(\mathbf{a})}\, R(\mathbf{a})}_{\text{sampling outside } \mathcal{B}}\,. \tag{5}$$

The key intuition of the MAPO algorithm is to use enumeration to evaluate the former expectation and $\pi_{\mathcal{B}}$, and MC sampling to compute the latter expectation. To sample from $\pi_\theta^-(\mathbf{a})$, one can resort to rejection sampling by sampling from $\pi_\theta(\mathbf{a})$ and rejecting the sample if $\mathbf{a} \in \mathcal{B}_a$. We get an exact estimate of the first expectation while sampling from a smaller stochastic space of size $(1-\pi_{\mathcal{B}})$ to get an estimate of the latter expectation.

Based on (5), using K trajectories $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(K)}\}$ sampled *i.i.d.* from current π_θ^-, *i.e.* the unexplored region of the space, we formulate an unbiased estimator of the policy gradients as,

$$\nabla_\theta \mathcal{O}_{\mathrm{ER}}(\theta) \;\approx\; \sum_{(\mathbf{a},r)\in\mathcal{B}} \nabla \pi_\theta(\mathbf{a})\, [r - b] \;+\; \frac{1-\pi_{\mathcal{B}}}{K} \sum_{k=1}^{K} \nabla \log \pi_\theta(\mathbf{a}^{(k)})\, [R(\mathbf{a}^{(k)}) - b]\,. \tag{6}$$

Assuming that $\pi_{\mathcal{B}} > 0$, using a full enumeration of the buffer and a budget of K MC samples, the variance of the estimator in (6) is lower than the estimator in (2) because it has less stochasticity. For practical applications in which sample evaluation is expensive, we expect the MAPO estimator in (6) to significantly outperform policy gradients. Note that even if we use sampling to approximate the first term, this can be viewed as a stratified sampling estimator. The variance reduction still holds as long as the trajectories inside and outside $\mathcal{B}$ have different distributions, which is usually true.

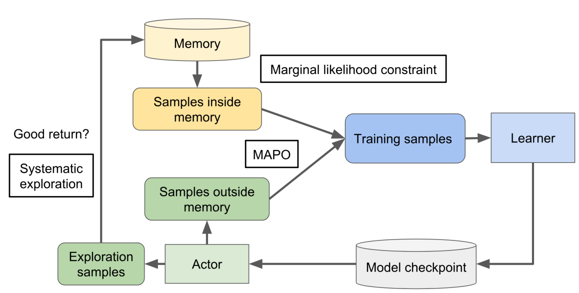

Figure 1: Overview of the MAPO algorithm with systematic exploration and marginal likelihood constraint.

In the following we present 3 techniques to create an efficient algorithm. An overview of the MAPO training algorithm is also shown in Figure 1.

4.1 Distributed Sampling

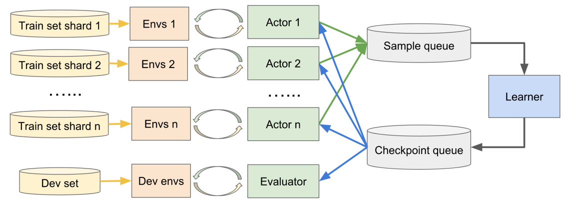

Figure 2: Distributed actor-learner architecture.

An exact computation of the first term on the RHS of (6) involves an enumeration over $\mathcal{B}$, which is efficient only when $|\mathcal{B}|$ is small. If $|\mathcal{B}|$ is large, enumeration can become prohibitive even though it involves no fresh samples. One solution is to simply truncate the memory buffer to keep only a few trajectories with the highest probabilities. However, this simple approach may lead to suboptimal result, because one often benefits from keeping all of the promising programs in the buffer to let model select the ones that generalize well. For example, in program synthesis, there can sometimes be many programs that compute the correct answer for a certain question, but only a handful will generalize and others are spurious (getting the correct answer because of luck, e.g., using $2 + 2$ to answer the question "what is two times two"). The generalizable programs could be discarded during the truncation of the memory buffer, making it hard for the model to overcome the spurious programs.

Alternatively, we can approximate the expectations over the trajectories both inside and outside $\mathcal{B}$ using sampling. As mentioned above, this can be viewed as *stratified sampling* from inside and outside the memory buffer and variance reduction still holds. A key insight is that the cost of sampling can be distributed through an actor-learner architecture depicted in Figure 2 inspired by [15]. The actors uses a stale model checkpoint to sample trajectories from inside the memory buffer through renormalization, and uses rejection sampling to pick trajectories from outside the memory. It also computes the weights for these trajectories using the stale model. These trajectories and their weights are then pushed to a queue of samples. A learner fetches samples from the queue and use them to compute gradient estimates to update the parameters.

4.2 Marginal Likelihood Constraint

Policy gradient methods usually suffer from a cold start problem if the model is randomly initialized. When supervised data in the form of demonstrations $\mathcal{D}$ is available, a reasonable approach (*e.g.*, [55]) incorporates such data by combining the expected return objective with a conditional log likelihood objective via,

$$\mathcal{O}_{\mathrm{AUG}}(\theta) = \lambda \mathcal{O}_{\mathrm{ER}}(\theta) + (1-\lambda) \sum_{\mathbf{a} \in \mathcal{D}} \log \pi_\theta(\mathbf{a}) \tag{7}$$

In our setup, we do not have access to demonstrations, but through exploration we accumulate the memory buffer with various high-reward trajectories. Instead of optimizing $\mathcal{O}_{\mathrm{AUG}}(\theta)$, which introduces some bias, we adopt a clipping mechanism that ensures that the buffer probability is greater or equal to α, *i.e.* $\pi_{\mathcal{B}} \geqslant \alpha$, otherwise clips it to α. Then, the policy gradient estimates are forced to pay a certain amount of attention to the high-reward trajectories in the buffer effectively optimizing

Algorithm 1 Systematic Exploration

Input: context $\mathbf{x}$, policy π, exploration buffer of fully explored subsequences $\mathcal{B}^e$, memory buffer of high-reward sequences $\mathcal{B}$
Initialize: empty sequence $a_{0:0}$
while true **do**
  $V = \{a \mid a_{0:t-1} \| a \notin B^e\}$
  if $V == \varnothing$ **then**
    $\mathcal{B}^e \leftarrow \mathcal{B}^e \cup a_{0:t-1}$
    break
  sample $a_t \sim \pi^V(a|a_{0:t-1})$
  $a_{0:t} \leftarrow a_{0:t-1} \| a_t$
  if a_t == EOS **then**
    if $R(a_{0:t}) > 0$ **then**
      $\mathcal{B} \leftarrow \mathcal{B} \cup a_{0:t}$
    $\mathcal{B}^e \leftarrow \mathcal{B}^e \cup a_{0:t}$
    break

$$\mathcal{O}_{\mathrm{MML}}(\theta) = \frac{1}{N} \sum_i \log \sum_{\mathbf{a} \in \mathcal{B}_i} \pi_\theta(\mathbf{a}) = \frac{1}{N} \log \prod_i \sum_{\mathbf{a} \in \mathcal{B}_i} \pi_\theta(\mathbf{a}). \tag{8}$$

At the beginning of training, the clipping will be active and it will introduce a bias, but will facilitate much faster training. Once the policy is off the ground, the buffer probabilities are almost never clipped given that they are naturally larger than α and the gradients are not biased any more.

4.3 Systematic Exploration

To avoid repeated samples during exploration, we propose to use systematic exploration. More specifically we keep a set $\mathcal{B}^e$ of fully explored partial sequences, which can be efficiently implemented using a bloom filter. Then, we use it to enforce a policy to only take actions that lead to unexplored sequences. Using a bloom filter we can store billions of sequences in $\mathcal{B}^e$ with only several gigabytes of memory. The pseudo code for this approach is shown in Algorithm 1. We warm start the memory buffer using systematic exploration from random policy as it can be trivially parallelized. In parallel to training, we continue the systematic exploration with the current policy to discover new high reward trajectories.

4.4 Final Algorithm

Putting things together, the final training procedure is summarized in Algorithm 2. As mentioned above, we adopt the actor-learner architecture for distributed training. It uses multiple actors to collect training samples asynchronously and one learner for updating the parameters based on the training samples. Each actor interacts with a set of environments to generate new trajectories. For efficiency, an actor uses a stale policy (π_θ^{old}), which is often a few steps behind the policy of the learner and will be synchronized periodically. To apply MAPO, each actor also maintains a memory buffer $\mathcal{B}_i$ to save the high-reward trajectories. To prepare training samples for the learner, the actor picks n_b samples from inside $\mathcal{B}_i$ and also performs rejection sampling with n_o on-policy samples, both according to the actor's policy π_θ^{old}. We then use the actor policy to compute a weight $max(\pi_\theta(\mathcal{B}), \alpha)$ for the samples in the memory buffer, and use $1 - max(\pi_\theta(\mathcal{B}), \alpha)$ for samples outside of the buffer. These samples are pushed to a queue and the learner reads data from the queue to compute gradients and update the parameters.

Algorithm 2 MAPO

```
Input: data {(x_i, y_i)}_{i=1}^N, memory and exploration buffer {(B_i, B_i^e)}_{i=1}^N, constants α, ε, M
repeat                                                  ▷ for all actors
    Initialize training batch D ← ∅
    Get a batch of inputs C
    for (x_i, y_i, B_i^e, B_i) ∈ C do
        SystematicExploration(x_i, π_θ^old, B_i^e, B_i)
        Sample a_i^+ ~ π_θ^old over B_i
        w_i^+ ← max(π_θ^old(B_i), α)
        D ← D ∪ (a_i^+, R(a_i^+), w_i^+)
        Sample a_i ~ π_θ^old
        if a_i ∉ B_i then
            w_i ← (1 − w_i^+)
            D ← D ∪ (a_i, R(a_i), w_i)
    Push D to training queue
until converge
repeat                                                  ▷ for the learner
    Get a batch D from training queue
    for (a_i, R(a_i), w_i) ∈ D do
        dθ ← dθ + w_i R(a_i) ∇ log π_θ(a_i)
    update θ using dθ
    π_θ^old ← π_θ                                       ▷ once every M batches
until converge or a certain number of steps is reached
Output: final parameters θ
```

5 Experiments

We evaluate MAPO on two program synthesis from natural language (also known as *semantic parsing*) benchmarks, WIKITABLEQUESTIONS and WIKISQL, which requires generating programs to query and process data from tables to answer natural language questions. We first compare MAPO to four common baselines to show the advantage of combining samples from inside and outside the memory buffer, and ablate systematic exploration and marginal likelihood constraint to show their utility. Then we compare MAPO to the state-of-the-art on these two benchmarks. On WIKITABLEQUESTIONS, MAPO is the first RL-based approach that significantly outperforms the previous state-of-the-art. On WIKISQL, MAPO trained with weak supervision (question-answer pairs) outperforms several strong models trained with full supervision (question-program pairs).

5.1 Experimental setup

Datasets. WIKITABLEQUESTIONS [36] contains tables extracted from Wikipedia and question-answer pairs about the tables. See Table 1 as an example. There are 2,108 tables and 18,496 question-answer pairs. We follow the construction in [36] 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 [57]. It requires multiple-step reasoning using a large set of functions, including comparisons, superlatives, aggregations, and arithmetic operations [36]. See Supplementary Material A for more details about the functions.

WIKISQL [63] is a recently introduced large scale dataset on learning natural language interfaces for databases. The dataset 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 in the dataset. Comparing to WIKITABLEQUESTIONS, the semantics are simpler because the SQLs use fewer operators (column selection, aggregation, and conditions). We perform similar preprocessing as for WIKITABLEQUESTIONS. Most of the state-of-the-art models relies on question-program pairs for supervised training, while we only use the question-answer pairs for weakly supervised training.

Model architecture. We adopt the Neural Symbolic Machines framework[25], which combines (1) a neural "programmer", which is a seq2seq model augmented by a key-variable memory that can translate a natural language utterance to a program as a sequence of tokens, and (2) a symbolic "computer", which is an Lisp interpreter that implements a domain specific language with built-in functions and provides code assistance by eliminating syntactically or semantically invalid choices.

For the Lisp interpreter, we added functions according to [62, 32] for WIKITABLEQUESTIONS experiments and used the subset of functions equivalent to column selection, aggregation, and conditions for WIKISQL. See the Supplementary Material A for more details about functions used.

We implemented the seq2seq model augmented with key-variable memory from [25] in TensorFlow [1]. Some minor differences are: (1) we used a bi-directional LSTM for the encoder; (2) we used two-layer LSTM with skip-connections in both the encoder and decoder. GloVe [39] embeddings are used for the embedding layer in the encoder and also to create embeddings for column names by averaging the embeddings of the words in a name. Following [32, 23], we also add a binary feature in each step of the encoder, indicating whether this word is found in the table, and an integer feature for a column name counting how many of the words in the column name appear in the question. For the WIKITABLEQUESTIONS dataset, we use the CoreNLP annotation of numbers and dates released with the dataset. For the WIKISQL dataset, only numbers are used, so we use a simple parser to identify and parse the numbers in the questions, and the tables are already preprocessed. The tokens of the numbers and dates are anonymized as two special tokens <NUM> and <DATE>. The hidden size of the encoder and decoder LSTM is 200. We keep the GloVe embeddings fixed during training, but project it to 200 dimensions using a trainable linear transformation. The same architecture is used for both WIKITABLEQUESTIONS and WIKISQL.

Training Details. We first apply systematic exploration using a random policy to discover high-reward programs to warm start the memory buffer of each example. For WIKITABLEQUESTIONS, we generated 10k programs per example using systematic exploration with pruning rules inspired by the grammars from [62]. We apply 0.2 dropout on both encoder and decoder, and 0.01 entropy regularization [29]. Each batch includes samples from 25 examples. For experiments on WIKISQL, we generated 1k programs per example due to computational constraint. Because the dataset is much larger, we don't use any regularization. Each batch includes samples from 125 questions. We use distributed sampling for WIKITABLEQUESTIONS. For WIKISQL, due to computational constraints, we use the simple approach to truncate each memory buffer to top 5 and then enumerate all 5 programs for training. For both experiments, the samples outside memory buffer are drawn using rejection sampling from 1 on policy sample per example. At inference time, we apply beam search of size 5. We evaluate the model periodically on the dev set to select the best model.

We apply a distributed actor-learner architecture for training. The actors use CPUs to generate new trajectories and push the samples into a queue for training. The learner reads batches of data from the queue and uses GPU to accelerate training. We use Adam optimizer for training and the learning rate is 10^{-3}. All the hyperparameters are tuned on the dev set.

5.2 Comparison to baselines implemented within our framework

To assess the effectiveness of MAPO, we compare against the following baselines in a controlled manner by using the same neural architecture and experimental framework.

▸ **REINFORCE:** We use on-policy samples to estimate the gradient of expected return as in (2), not utilizing any form of memory buffer.

▸ **MML:** Maximum Marginal Likelihood maximizes the marginal probability of the memory buffer as in (8). Assuming binary rewards and assuming that the memory buffer contains almost all of the trajectories with a reward of 1, MML optimizes the marginal probability of generating a rewarding program. Note that under these assumptions, expected return can be expressed as $\mathcal{O}_{\mathrm{ER}}(\theta) \approx \frac{1}{N}\sum_i \sum_{\mathbf{a}\in\mathcal{B}_i} \pi_\theta(\mathbf{a})$. Comparing the two objectives, we can see that

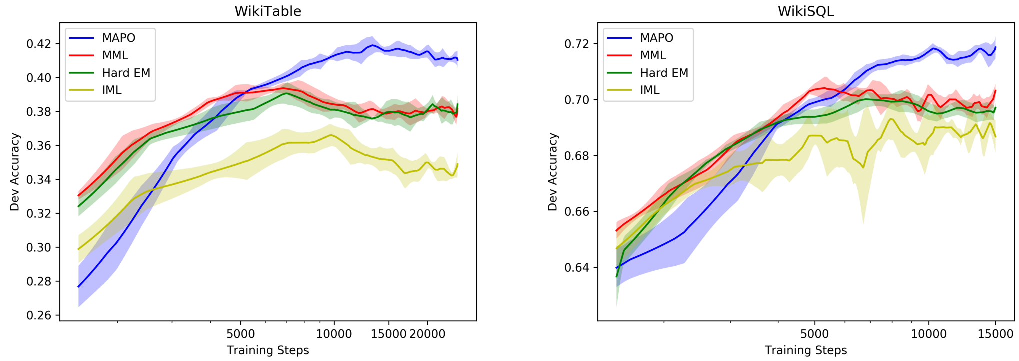

Figure 3: Comparison of MAPO and 3 baselines' dev set accuracy curves. Results on WIKITABLEQUESTIONS is on the left and results on WIKISQL is on the right. The plot is average of 5 runs with a bar of one standard deviation. The horizontal coordinate (training steps) is in log scale.

MML maximizes the product of marginal probabilities, whereas expected return maximizes the sum. More discussion of these two types of objectives can be found in [16, 34, 43].

- **Hard EM:** Expectation-Maximization algorithm is commonly used to optimize the marginal likelihood in the presence of latent variables. Hard EM uses the samples with the highest probability to approximate the gradient to $\mathcal{O}_{\text{MML}}$.
- **IML:** Iterative Maximum Likelihood training [25] uniformly maximizes the likelihood of all the trajectories with the highest rewards $\mathcal{O}_{\text{CLL}}(\theta) = \sum_{\mathbf{a} \in \mathcal{B}} \log \pi_\theta(\mathbf{a})$.

Because the memory buffer is too large to enumerate, we use samples from the buffer to approximate the gradient for MML and IML, and uses samples with highest $\pi_\theta(\mathbf{a})$ for Hard EM.

The results are summarized in Table 2. We show the accuracy curves on the dev set in Figure 3. Removing systematic exploration or the marginal likelihood constraint significantly weaken MAPO possibly because high-reward trajectories are not found or easily forgotton. REINFORCE barely learns anything useful because starting from a random policy, most samples result in a reward of zero. MML and Hard EM converge faster, but the learned models underperform MAPO, which suggests that the expected return is a better objective. IML runs faster than MML and MAPO because it randomly samples from the buffer, but the objective is prone to spurious programs.

Method	WIKITABLE	WIKISQL
REINFORCE	< 10	< 10
MML (Soft EM)	39.7 ± 0.3	70.7 ± 0.1
Hard EM	39.3 ± 0.6	70.2 ± 0.3
IML	36.8 ± 0.5	70.1 ± 0.2
MAPO	$\mathbf{42.3} \pm 0.3$	$\mathbf{72.2} \pm 0.2$
No Systematic Exploration	< 10	< 10
No Marginal Likelihood	< 10	< 10

Table 2: Ablation study on both datasets. We report mean accuracy % $\pm$ standard deviation on dev sets based on 5 runs.

Method	E.S.	Dev.	Test
Pasupat & Liang (2015) [36]	-	37.0	37.1
Neelakantan *et al.* (2017) [32]	1	34.1	34.2
Neelakantan *et al.* (2017) [32]	15	37.5	37.7
Haug *et al.* (2017) [17]	1	-	34.8
Haug *et al.* (2017) [17]	15	-	38.7
Zhang *et al.* (2017) [62]	-	40.4	43.7
MAPO	1	**42.3**	43.8
MAPO (ensembled)	5	-	**46.2**

Table 3: Results on WIKITABLEQUESTIONS. E.S. is the ensemble size, when applicable.

Method	Dev.	Test
Fully supervised		
Zhong *et al.* (2017) [63]	60.8	59.4
Wang *et al.* (2017) [51]	67.1	66.8
Xu *et al.* (2017) [56]	69.8	68.0
Huang *et al.* (2018) [21]	68.3	68.0
Yu *et al.* (2018) [58]	74.5	73.5
Sun *et al.* (2018) [49]	75.1	74.6
Dong & Lapata (2018) [14]	**79.0**	**78.5**
Weakly supervised		
MAPO	**72.2**	72.6
MAPO (ensemble of 5)	-	**74.9**

Table 4: Results on WIKISQL. Unlike other methods, MAPO only uses weak supervision.

5.3 Comparison to state-of-the-art

On WIKITABLEQUESTIONS (Table 3), MAPO is the first RL-based approach that significantly outperforms the previous state-of-the-art by 2.5%. Unlike previous work, MAPO does not require manual feature engineering or additional human annotation[2]. On WIKISQL (Table 4), even

[2]Krishnamurthy *et al.* [23] achieved 45.9 accuracy when trained on the data collected with dynamic programming and pruned with more human annotations [37, 30].

though MAPO does not exploit ground truth programs and only uses weak supervision, it is able to outperform many strong baselines trained using programs as full supervision. The techniques introduced in other models can be incorporated to further improve the result of MAPO, but we leave that as future work. We also qualitatively analyzed a trained model and see that it can generate fairly complex programs. See the Supplementary Material B for some examples of generated programs.

6 Related work

Program synthesis & semantic parsing. There has been a surge of recent interest in applying reinforcement learning to program synthesis [10, 2, 59, 31] and combinatorial optimization [65, 7]. Different from these efforts, we focus on the contextualized program synthesis where generalization to new contexts is important. Semantic parsing [60, 61, 26] maps natural language to executable symbolic representations. Training semantic parsers through weak supervision is challenging because the model must interact with a symbolic interpreter through non-differentiable operations to search over a large space of programs [8, 25]. Previous work [16, 32] reports negative results when applying simple policy gradient methods like REINFORCE [53], which highlights the difficulty of exploration and optimization when applying RL techniques to program synthesis or semantic parsing. MAPO takes advantage of discrete and deterministic nature of program synthesis and significantly improves upon REINFORCE.

Experience replay. An experience replay buffer [27] enables storage and usage of past experiences to improve the sample efficiency of RL algorithms. Prioritized experience replay [44] prioritizes replays based on temporal-difference error for more efficient optimization. Hindsight experience replay [4] incorporates goals into replays to deal with sparse rewards. MAPO also uses past experiences to tackle sparse reward problems, but by storing and reusing high-reward trajectories, similar to [25, 35]. Previous work[25] assigns a fixed weight to the trajectories, which introduces bias into the policy gradient estimates. More importantly, the policy is often trained equally on the trajectories that have the same reward, which is prone to reinforcing spurious programs. By contrast, MAPO uses the trajectories in a principled way to obtain an unbiased low variance gradient estimate.

Variance reduction. Policy optimization via gradient descent is challenging because of: (1) large *variance* in gradient estimates; (2) small gradients in the initial phase of training. Prior variance reduction approaches [54, 53] mainly relied on control variate techniques by introducing a critic model [22, 29, 46]. MAPO takes a different approach to reformulate the gradient as a combination of expectations inside and outside a memory buffer. Standard solutions to the small gradient problem involves supervised pretraining [47, 18, 42] or using supervised data to generate rewarding samples [34, 13], which cannot be applied when supervised data are not available. MAPO solves this problem by a soft constraint on the marginal likelihood of the memory buffer, which accelerates training at the beginning and becomes unbiased once the constraint is satisfied.

Exploration. Recently there has been a lot of work on improving exploration [38, 50, 20] by introducing additional reward based on information gain or pseudo count. For program synthesis [5, 32, 10], the search spaces are enumerable and deterministic. Therefore, we propose to conduct systematic exploration, which ensures that only novel trajectories are generated.

7 Conclusion

We present memory augmented policy optimization (MAPO) that incorporates a memory buffer of promising action trajectories to reduce the variance of policy gradients. We propose 3 techniques to enable an efficient training algorithm for MAPO. (1) distributed sampling from inside and outside memory buffer in an actor-learner architecture; (2) a constraint over the marginal likelihood of the trajectories in the memory buffer to accelerate training; (3) systematic exploration to efficiently discover high-reward trajectories. MAPO is evaluated on real world program synthesis from natural language tasks. On WikiTableQuestions, MAPO is the first RL approach that significantly outperforms previous state-of-the-art; on WikiSQL, MAPO trained with only weak supervision outperforms several strong baselines trained with full supervision.

Acknowledgments

We would like to thank Dan Abolafia, Ankur Taly, Thanapon Noraset, Arvind Neelakantan, Wenyun Zuo, Chenchen Pan and Mia Liang for helpful discussions. Jonathan Berant was partially supported by The Israel Science Foundation grant 942/16.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv:1603.04467*, 2016.

[2] Daniel A Abolafia, Mohammad Norouzi, and Quoc V Le. Neural program synthesis with priority queue training. *arXiv preprint arXiv:1801.03526*, 2018.

[3] Daniel A. Abolafia, Mohammad Norouzi, Jonathan Shen, Rui Zhao, and Quoc V. Le. Neural program synthesis with priority queue training. *arXiv:1801.03526*, 2018.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *NIPS*, 2017.

[5] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. *ICLR*, 2017.

[6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *JMLR*, 2013.

[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv:1611.09940*, 2016.

[8] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. *EMNLP*, 2(5):6, 2013.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv:1606.01540*, 2016.

[10] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *International Conference on Learning Representations*, 2018.

[11] Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. *arXiv:1703.06585*, 2017.

[12] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *ICML*, 2012.

[13] Nan Ding and Radu Soricut. Cold-start reinforcement learning with softmax policy gradient. In *Advances in Neural Information Processing Systems*, pages 2817–2826, 2017.

[14] Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. *CoRR*, abs/1805.04793, 2018.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv:1802.01561*, 2018.

[16] Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. *ACL*, 2017.

[17] Till Haug, Octavian-Eugen Ganea, and Paulina Grnarova. Neural multi-step reasoning for question answering on semi-structured tables. In *ECIR*, 2018.

[18] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. *AAAI*, 2018.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 1997.

[20] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[21] Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen tau Yih, and Xiaodong He. Natural language to structured query generation via meta-learning. *CoRR*, abs/1803.02400, 2018.

[22] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[23] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. *EMNLP*, 2017.

[24] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv:1606.01541*, 2016.

[25] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *ACL*, 2017.

[26] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. *ACL*, 2011.

[27] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[28] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[30] Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. It was the training data pruning too! *arXiv:1803.04579*, 2018.

[31] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2775–2785, 2017.

[32] Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew D McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. *arXiv:1611.08945*, 2016.

[33] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv:1804.03720*, 2018.

[34] Mohammad Norouzi, Samy Bengio, Navdeep Jaitly, Mike Schuster, Yonghui Wu, Dale Schuurmans, et al. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*, pages 1723–1731, 2016.

[35] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *ICML*, 2018.

[36] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *ACL*, 2015.

[37] Panupong Pasupat and Percy Liang. Inferring logical forms from denotations. *ACL*, 2016.

[38] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[39] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. *EMNLP*, 2014.

[40] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. *IROS*, 2006.

[41] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. *NIPS*, 2017.

[42] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *ICLR*, 2016.

[43] Nicolas Le Roux. Tighter bounds lead to improved classifiers. *ICLR*, 2017.

[44] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *ICLR*, 2016.

[45] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. *ICML*, 2015.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[47] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[48] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 2017.

[49] Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. Semantic parsing with syntax-and table-aware sql generation. *arXiv:1804.08338*, 2018.

[50] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2753–2762. Curran Associates, Inc., 2017.

[51] Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. Pointing out SQL queries from text. *ICLR*, 2018.

[52] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *ICLR*, 2017.

[53] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, pages 229–256, 1992.

[54] Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. *ICLR*, 2018.

[55] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*, 2016.

[56] Xiaojun Xu, Chang Liu, and Dawn Song. SQLNet: Generating structured queries from natural language without reinforcement learning. *ICLR*, 2018.

[57] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. *ACL*, 2016.

[58] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv:1804.09769*, 2018.

[59] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *arXiv:1505.00521*, 2015.

[60] M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055, 1996.

[61] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666, 2005.

[62] Yuchen Zhang, Panupong Pasupat, and Percy Liang. Macro grammars and holistic triggering for efficient semantic parsing. *ACL*, 2017.

[63] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, 2017.

[64] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2016.

[65] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.

[66] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.

A Domain Specific Language

We adopt a Lisp-like domain specific language with certain built-in functions. A program C is a list of expressions $(c_1...c_N)$, where each expression is either a special token "*EOS*" indicating the end of the program, or a list of tokens enclosed by parentheses "$(FA_1...A_K)$". F is a function, which takes as input K arguments of specific types. Table A defines the arguments, return value and semantics of each function. In the table domain, there are rows and columns. The value of the table cells can be number, date time or string, so we also categorize the columns into number columns, date time columns and string columns depending on the type of the cell values in the column.

Function	Arguments	Returns	Description
(**hop** v p)	**v**: a list of rows. **p**: a column.	a list of cells.	Select the given column of the given rows.
(**argmax** v p) (**argmin** v p)	**v**: a list of rows. **p**: a number or date column.	a list of rows.	From the given rows, select the ones with the largest / smallest value in the given column.
(**filter**$_>$ v q p) (**filter**$_\geqslant$ v q p) (**filter**$_<$ v q p) (**filter**$_\leqslant$ v q p) (**filter**$_=$ v q p) (**filter**$_\neq$ v q p)	**v**: a list of rows. **q**: a number or date. **p**: a number or date column.	a list of rows.	From the given rows, select the ones whose given column has certain order relation with the given value.
(**filter**$_{in}$ v q p) (**filter**$_{!in}$ v q p)	**v**: a list of rows. **q**: a string. **p**: a string column.	a list of rows.	From the given rows, select the ones whose given column contain / do not contain the given string.
(**first** v) (**last** v)	**v**: a list of rows.	a row.	From the given rows, select the one with the smallest / largest index.
(**previous** v) (**next** v)	**v**: a row.	a row.	Select the row that is above / below the given row.
(**count** v)	**v**: a list of rows.	a number.	Count the number of given rows.
(**max** v p) (**min** v p) (**average** v p) (**sum** v p)	**v**: a list of rows. **p**: a number column.	a number.	Compute the maximum / minimum / average / sum of the given column in the given rows.
(**mode** v p)	**v**: a list of rows. **p**: a column.	a cell.	Get the most common value of the given column in the given rows.
(**same_as** v p)	**v**: a row. **p**: a column.	a list of rows.	Get the rows whose given column is the same as the given row.
(**diff** v0 v1 p)	**v0**: a row. **v1**: a row. **p**: a number column.	a number.	Compute the difference in the given column of the given two rows.

Table 5: Functions used in the experiments.

In the WIKITABLEQUESTIONS experiments, we used all the functions in the table. In the WIKISQL experiments, because the semantics of the questions are simpler, we used a subset of the functions (hop, filter$_=$, filter$_{in}$, filter$_>$, filter$_<$, count, maximum, minimum, average and sum). We created the functions according to [62, 32].[3]

B Examples of Generated Programs

The following table shows examples of several types of programs generated by a trained model.

Statement	Comment
Superlative	
nt-13901: the most points were scored by which player?	
(argmax all_rows r.points-num)	Sort all rows by column 'points' and take the first row.
(hop v0 r.player-str)	Output the value of column 'player' for the rows in v0.
Difference	
nt-457: how many more passengers flew to los angeles than to saskatoon?	
(filter$_{in}$ all_rows ['saskatoon'] r.city-str)	Find the row with 'saskatoon' matched in column 'city'.
(filter$_{in}$ all_rows ['los angeles'] r.city-str)	Find the row with 'los angeles' matched in column 'city'.
(diff v1 v0 r.passengers-num)	Calculate the difference of the values in the column 'passenger' of v0 and v1.
Before / After	
nt-10832: which nation is before peru?	
(filter$_{in}$ all_rows ['peru'] r.nation-str)	Find the row with 'peru' matched in 'nation' column.
(previous v0)	Find the row before v0.
(hop v1 r.nation-str)	Output the value of column 'nation' of v1.
Compare & Count	
nt-647: in how many games did sri lanka score at least 2 goals?	
(filter$_\geqslant$ all_rows [2] r.score-num)	Select the rows whose value in the 'score' column >= 2.
(count v0)	Count the number of rows in v0.
Exclusion	
nt-1133: other than william stuart price, which other businessman was born in tulsa?	
(filter$_{in}$ all_rows ['tulsa'] r.hometown-str)	Find rows with 'tulsa' matched in column 'hometown'.
(filter$_{!in}$ v0 ['william stuart price'] r.name-str)	Drop rows with 'william stuart price' matched in the value of column 'name'.
(hop v1 r.name-str)	Output the value of column 'name' of v1.

Table 6: Example programs generated by a trained model.

C Analysis of Sampling from Inside and Outside Memory Buffer

In the following we give theoretical analysis of the distributed sampling approaches. For the purpose of the analysis we assume binary rewards, and exhaustive exploration, that the buffer $\mathcal{B}^+ \equiv \mathcal{B}$ contains all the high reward samples, and $\mathcal{B}^- \equiv \mathcal{A} - \mathcal{B}_a$ contains all the non-rewarded samples. It provides a general guidance of how examples should be allocated on the experiences and whether to use baseline or not so that the variance of gradient estimations can be minimized. In our work, we take the simpler approach to not use baseline and leave the empirical investigation to future work.

C.1 Variance: baseline vs no baseline

Here we compare baseline strategies based on the variances of gradients when applying stratified sampling estimator to policy gradient.

Assume $\sigma_+^2 = Var_{\mathbf{a}\sim\pi_\theta^+(\mathbf{a})}[\nabla \log \pi(\mathbf{a})]$ and $\sigma_-^2 = Var_{\mathbf{a}\sim\pi_\theta^-(\mathbf{a})}[\nabla \log \pi(\mathbf{a})]$ are the variance of the gradient of the log likelihood inside and outside the buffer. If we don't use a baseline, the the optimal

[3]The only function we have added to capture some complex semantics is the same_as function, but it only appears in 1.2% of the generated programs (among which 0.6% are correct and the other 0.6% are incorrect), so even if we remove it, the significance of the difference in Table 3 will not change.

sampling strategy is to only sample from $\mathcal{B}^+$. The variance of the gradient estimation is

$$\mathrm{Var}[\nabla\mathcal{O}_{\mathrm{ER}}] \approx \pi_\theta(\mathcal{B})^2\sigma_+^2 \tag{9}$$

If we use a baseline $b = \pi_\theta(\mathcal{B})$ and apply optimal sampling allocation (Section C.2), then the variance is

$$\mathrm{Var}[\nabla\mathcal{O}_{\mathrm{ER}}]_b \approx \pi_\theta(\mathcal{B})^2(1-\pi_\theta(\mathcal{B}))^2(\sigma_+^2 + \sigma_-^2) \tag{10}$$

We can prove that both of these estimators reduce the variance for the gradient estimation. To compare the two, we can see that the ratio of the variance with and without baseline is

$$\frac{\mathrm{Var}[\nabla\mathcal{O}_{\mathrm{ER}}]_b}{\mathrm{Var}[\nabla\mathcal{O}_{\mathrm{ER}}]} = (1+\frac{\sigma_-^2}{\sigma_+^2})(1-\pi_\theta(\mathcal{B}))^2 \tag{11}$$

So using baseline provides lower variance when $\pi_\theta(\mathcal{B}) \approx 1.0$, which roughly corresponds to the later stage of training, and when σ_- is not much larger than σ_+; it is better to not use baselines when $\pi_\theta(\mathcal{B})$ is not close to 1.0 or when σ_- is much larger than σ_+.

C.2 Optimal Sample Allocation

Given that we want to apply stratified sampling to estimate the gradient of REINFORCE with baseline 2, here we show that the optimal sampling strategy is to allocate the same number of samples to $\mathcal{B}^+$ and $\mathcal{B}^-$.

Assume that the gradient of log likelihood has similar variance on $\mathcal{B}^+$ and $\mathcal{B}^-$:

$$\mathrm{Var}_{\pi_\theta^+(\mathbf{a})}[\nabla\log\pi_\theta(\mathbf{a})] \approx \mathrm{Var}_{\pi_\theta^-(\mathbf{a})}[\nabla\log\pi_\theta(\mathbf{a})] = \sigma^2 \tag{12}$$

The variance of gradient estimation on $\mathcal{B}^+$ and $\mathcal{B}^-$ are:

$$\begin{aligned}\mathrm{Var}_{\pi_\theta^+(\mathbf{a})}[(1-\pi_\theta(\mathcal{B}^+))\nabla\log\pi_\theta(\mathbf{a})] =& (1-\pi_\theta(\mathcal{B}^+))^2 * \sigma^2 \\ \mathrm{Var}_{\pi_\theta^-(\mathbf{a})}[(-\pi_\theta(\mathcal{B}^+))\nabla\log\pi_\theta(\mathbf{a})] =& \pi_\theta(\mathcal{B}^+)^2 * \sigma^2\end{aligned} \tag{13}$$

When performing stratified sampling, the optimal sample allocation is to let the number of samples from a stratum be proportional to its probability mass times the standard deviation $P_i\sigma_i$ In other words, the more probability mass and the more variance a stratum has, the more samples we should draw from a stratum. So the ratio of the number of samples allocated to each stratum under the optimal sample allocation is

$$\frac{k^+}{k^-} = \frac{\pi_\theta(\mathcal{B}^+)\sqrt{\mathrm{Var}_{\pi_\theta^+(\mathbf{a})}}}{\pi_\theta(\mathcal{B}^-)\sqrt{\mathrm{Var}_{\pi_\theta^-(\mathbf{a})}}} \tag{14}$$

Using equation 13 and $\pi_\theta(\mathcal{B}^-) = 1 - \pi_\theta(\mathcal{B}^+)$, we can see that

$$\frac{k^+}{k^-} = 1 \tag{15}$$

So the optimal strategy is to allocate the same number of samples to $\mathcal{B}^+$ and $\mathcal{B}^-$.