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.