NEURAL PROGRAM PLANNER FOR STRUCTURED PREDICTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the problem of weakly supervised structured prediction (SP) with reinforcement learning (RL) – for example, given a database table and a question, perform a sequence of computation actions on the table, which generates a response and receives a binary success-failure reward. This line of research has been successful by leveraging RL to directly optimizes the desired metrics of the tasks – for example, the accuracy in question answering or BLEU score in machine translation. However, different from the common RL settings, the environment dynamics is deterministic and known in SP, which hasn’t been fully utilized by the model-free RL methods that are usually applied. We propose to apply model-based RL methods, which rely on planning as a primary model component. We demonstrate the effectiveness of planning-based SP with a Neural Program Planner (NPP), which, given a set of candidate programs from a pretrained search policy, decides which program is the most promising considering all the information generated from executing these programs. We evaluate NPP on weakly supervised program synthesis natural language (semantic parsing) by stacked learning a planning module based on pretrained search policies. On the WIKITABLEQUESTIONS benchmark, NPP achieves a new state-of-the-art of 47.2% accuracy.

1 INTRODUCTION

Numerous results from natural language processing tasks have shown that Structured Prediction (SP) can be cast as into the reinforcement learning (RL) framework, and known RL techniques can give formal performance bounds on the SP tasks (Daume et al., 2009; Ross et al., 2011; Bengio et al., 2015). RL also directly optimizes the task metrics, for example, the accuracy in question answering or BLEU score in machine translation, and avoids the exposure bias problem comparing to maximum likelihood training that is commonly used in SP (Ross et al., 2011; Ranzato et al., 2016).

However, previous works on applying RL to SP problems often use model-free RL algorithms (e.g., REINFORCE or actor-critic) and fail to leverage the characteristics of SP, which are different than typical RL tasks, e.g., playing video games (Mnih et al., 2015) or the game of Go (Silver et al., 2016b, 2017). In most SP problems, conditioned on the input x, the environment dynamics, except for the reward signal, is known, deterministic, reversible, and therefore can be searched. This means that there is a perfect model of the environment, which can be used to apply model-based RL methods that utilize planning as a primary model component.

Take semantic parsing (Berant et al., 2013; Pasupat & Liang, 2015) as an example, semantic parsers trained by RL such as Neural Semantic Machine (Liang et al., 2017, 2018) typically rely on beam search for inference – the program with the highest probability in beam is used for execution and generating answer. However, the policy, which is used for beam search, may not be able to assign the highest probability to the correct program. This limitation is due to the policy predicting a locally normalized probability for each possible action based on the partially generated program, and the probability of a program is a product of these local probabilities.

For example, when applied to the WebQUESTIONS SP task, NSM made mistakes with two common patterns: (1) the program would ignore important information in the context; (2) the
Table 1: Wikipedia table for question nt-3516: Which programming is played the most?

<table>
<thead>
<tr>
<th>Call-sign</th>
<th>Location</th>
<th>RF</th>
<th>PSIP</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIVM-LD</td>
<td>Canton</td>
<td>39</td>
<td>39.1</td>
<td>RTV</td>
</tr>
<tr>
<td>WIVM-LD</td>
<td>Canton</td>
<td>39</td>
<td>39.2</td>
<td>TV</td>
</tr>
<tr>
<td>WIVM-LD</td>
<td>Canton</td>
<td>39</td>
<td>39.3</td>
<td>PBJ</td>
</tr>
<tr>
<td>WIVM-LD</td>
<td>Canton</td>
<td>39</td>
<td>39.4</td>
<td>Faith Ministries Radio ...</td>
</tr>
<tr>
<td>WIVN-LD</td>
<td>Newcomerstown</td>
<td>29</td>
<td>29.1</td>
<td>RTV (WIVM-LD Simulcast)</td>
</tr>
<tr>
<td>WIVN-LD</td>
<td>Newcomerstown</td>
<td>29</td>
<td>29.2</td>
<td>PBJ</td>
</tr>
<tr>
<td>WIVN-LD</td>
<td>Newcomerstown</td>
<td>29</td>
<td>29.3</td>
<td>Tuff TV</td>
</tr>
<tr>
<td>WIVN-LD</td>
<td>Newcomerstown</td>
<td>29</td>
<td>29.4</td>
<td>Faith Ministries Radio ...</td>
</tr>
<tr>
<td>WIVX-LD</td>
<td>Loudonville</td>
<td>51</td>
<td>51.1</td>
<td>RTV (WIVM-LD Simulcast)</td>
</tr>
<tr>
<td>WIVD-LD</td>
<td>Newcomerstown</td>
<td>22</td>
<td>22.1</td>
<td>RTV (WIVM-LD Simulcast)</td>
</tr>
<tr>
<td>W27DG-D</td>
<td>Millersburg</td>
<td>27</td>
<td>27.1</td>
<td>RTV (WIVM-LD Simulcast)</td>
</tr>
</tbody>
</table>

generated program does not execute to a reasonable output, but still receives high probability. Resolving this issue requires using the information of the full program and its execution output to further evaluate its quality based on the context, which can be seen as planning. This can be observed in Figure 4 where the model is asked a question “Which programming is played the most?”. The full context of the input table (shown in Table 1) contains a television station programming. The top program generated by a search policy produces the wrong answer by querying against the incorrect column name. If provided the correct context, and if allowed to evaluate the full program forward and backward through time, a planning model would be able to better evaluate which program would produce the correct answer.

To handle errors related to contexts, we propose to train a value function to compute the utility of each token in a program. We evaluate the utility of a token by considering the its program and token probability as well as the attention mask generated by the Seq2Seq model for the policy. We further incorporate features of the question context by including features that represent question/program overlap as well as features that represent program-program overlap such as how many programs share a token at a given timestep.

In the experiments, we found that a planner that uses the learned value function to re-rank the candidates in the beam can significantly and consistantly improve the accuracy. On the WIKITABLEQUESTIONS benchmark, we improve the state-of-the-art by 0.9%, achieving an accuracy of 47.2%.

2 Background

2.1 WIKITABLEQUESTIONS

WIKITABLEQUESTIONS (Pasupat & Liang, 2015) 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 split into train/dev/test set. Each table can be converted into a directed graph that can be queried, where rows and cells are converted to graph nodes while column names become labeled directed edges. For the questions, we use string match to identify phrases that appear in the table. We also identify numbers and dates using the CoreNLP annotation released with the dataset. The task is challenging in several aspects. First, the tables are taken from Wikipedia and cover a wide range of topics. Second, at test time, new tables that contain unseen column names appear. Third, the table contents are not normalized as in knowledge-bases like Freebase, so there are noises and ambiguities in the table annotation. Last, the semantics are more complex comparing to previous datasets like WEBQUESTIONSSP (Yih et al., 2016). It requires multiple-step reasoning using a large set of functions, including comparisons, superlatives, aggregations, and arithmetic operations (Pasupat & Liang, 2015). See (Liang et al., 2018) for more details about the functions.

2.2 NEURAL SEMANTIC MACHINE

We adopt the NSM framework (Liang et al., 2017, 2018), which combines (1) a neural “programmer”, which is a seq2seq model augmented by a key-variable memory that can translate a natural language
We now introduce NPP by first describing the architecture of which considers the program, question and beam context based on its importance and position which goes over candidate program answer pairs and the final score of a candidate program is simply which produces a set of candidate programs with \( y \) (Section 4.2). Finally, we describe a training procedure that is based on max-margin/ranking objective (Liang et al., 2018) we consider the problem of contextual program synthesis \( \hat{a} \) being the expected correct answer. The reward function evaluates a program by executing it on a real environment and comparing the emitted output against the correct answer. For example, the reward may be binary that is 1 when the output equals the answer and 0 otherwise. We assume that the context \( x \) 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, \( \{(x_i, y_i)\}_{i=1}^{N} \), the goal is to find an optimal parameter \( \theta^* \) that parameterizes a mapping of \( x \rightarrow \hat{a} \) with maximum empirical return on a holdout test set.

In this study we will further decompose the policy \( f_{\theta}(x) \) into the stacking of a search model \( s_{\phi}(x) \), which produces a set of candidate programs \( B \) given an environment \( x \), and a value model \( v_{\omega}(a; x, B) \), which assigns a score \( s \) to program \( a \) given the environment and all the candidate programs. Therefore, \( \theta = [\phi; \omega] \) and

\[
\hat{a}(x) \approx \argmax_{a \in A_{\phi}(x)} v_{\omega}(a; x, s_{\phi}(x)).
\]

The search model \( s_{\phi} \) can be learnt by optimizing a conditional distribution \( \pi_{\phi}(a \mid x) \) that assigns a probability to each program given the context. That is, \( \pi_{\phi} \) is a distribution over the countable set of all possible programs, denoted \( A \). Thus \( \forall a \in A : \pi_{\phi}(a \mid x) \geq 0 \) and \( \sum_{a \in A} \pi_{\phi}(a \mid x) = 1 \). Then, to synthesize candidate programs for a novel context, one may find the most likely programs under the distribution \( \pi_{\phi} \) via exact or approximate inference such as beam search. \( B = s_{\phi}(x) \approx \argmax_{a \in A} \pi_{\phi}(a \mid x) \). \( \pi_{\phi} \) is typically a autoregressive model such as a recurrent neural network: (e.g. Hochreiter & Schmidhuber [1997a]) \( \pi_{\phi}(a \mid x) = \prod_{l=1}^{\ell(a)} \pi_{\phi}(a_l \mid a_{<l}, x) \), where \( a_{<l} = (a_1, \ldots, a_{l-1}) \) denotes a prefix of the program \( a \). In the absence of ground truth programs, policy gradient techniques (such as REINFORCE [Williams, 1992]) present a way to optimize the parameters of a stochastic policy \( \pi_{\phi} \) via optimization of expected return. Given a training dataset of context-answer pairs, \( \{(x_i, y_i)\}_{i=1}^{N} \), the training objective is \( \mathcal{O}_{RL}(\theta) = \mathbb{E}_{a \sim \pi_{\phi}(a \mid x)} R(a; x, y) \).

Decision-time planning typically relies on value network [Silver et al., 2016a] trained to predict the true reward. In the next section, however, we introduce a max-margin training objective for the value model \( v_{\omega} \), which optimizes to rank rewarded programs higher than non-rewarded programs.

4 Neural Program Planner (NPP)

We now introduce NPP by first describing the architecture of \( v_{\omega} \) – a sequence to sequence model which goes over candidate program answer pairs and the final score of a candidate program is simply the sum of its token scores (Section 4.1). Secondly we describe the program token representation, which considers the program, question and beam context based on its importance and position (Section 4.2). Finally, we describe a training procedure that is based on max-margin/ranking objective on candidate programs given a question (Section 4.3).

4.1 Architecture

Here we introduce the NPP architecture (Figure 1) in the context of semantic parsing, but the framework should be broadly applicable to applying RL in structured predictions. Given a pre-trained
search policy $\pi_\phi$ and environment $x$, NPP first unrolls the policy with beam search to generate candidate programs (plans) $\mathcal{B} = \pi_\phi(x)$. Then it scores each program $a$ considering token $a_t$ at every step and global statistics among all programs $\mathcal{B}$. $a_t$ is represented as a context feature vector $C_t$ (details in Section 4.2).

To capture the sequential inputs the scoring component is a sequence to sequence model which goes over candidate program answer pairs and assigns preference scores to each program/answer token. We implement a forward recurrent network with LSTM [Hochreiter & Schmidhuber (1997b); Greff et al. (2015)] cells as the first layer of our planner $C_{LSTM} = LSTM(C)$. The LSTM hidden state at each step is fed to a one dimensional convolutional layer with kernel size 3, in order to capture inner function scoring as most functions are of size 3-5, as a feature extractor $O_t = CNN(C_{LSTM}^t)$. Finally we calculate the score per token by feeding into a single node hyperbolic tangent activation layer to compute the score per token $v_t = \text{Tanh}(O_t)$ of token $a_t$. The final score of a candidate program $v_\omega(a) = \sum_{t=1}^T v_t$ is simply the sum of its token scores. The choice of simply summing token level scores makes the score very understandable (details in Section 4.2). Figure 2 gives implementation details of the value model.

### 4.2 Program Token Representation

To better score tokens based on the overall context of the environment we represent each token with a set of context features $C_t = [q_t^{tok}, q_t^{attn}, p_t^{prob}, e_t^{prob}, t_t^{agree}]$ as in Table 2. $q_t^{attn}$ is the
Table 2: Program Token Representation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q^\text{tok}$</td>
<td>binary</td>
<td>The program token matches any of the question tokens.</td>
</tr>
<tr>
<td>$q^\text{attn}$</td>
<td>float vector</td>
<td>Softmax attention over query tokens per program token</td>
</tr>
<tr>
<td>$p^\text{prob}$</td>
<td>float</td>
<td>Program probability according to the search policy $\pi_\phi$</td>
</tr>
<tr>
<td>$t^\text{prob}$</td>
<td>float</td>
<td>Program token probability according to the search policy $\pi_\phi$</td>
</tr>
<tr>
<td>$t^\text{agree}$</td>
<td>count</td>
<td>Number of candidate programs having token $a_t$ at position $t$</td>
</tr>
</tbody>
</table>

softmax attention across question tokens, which helps to discern which part of the question is of most importance to the model when generating the current given token. Together $q^\text{tok}$ and $q^\text{attn}$ represents the alignment between program and query. $t^\text{prob}$ and $p^\text{prob}$ are the probability of token $a_t$ and program $a$ assigned by the search policy $\pi_\phi$, which represent the decisions from the search model. $t^\text{agree}$ is the number of candidate programs having token $a_t$ at position $t$. Access to information such as $t^\text{agree}$ is only available to NPP as it can only be used after all the candidate programs have been generated.

4.3 TRAINING NPP

We formulate NPP training as a learning to rank problem – optimizing pairwise ranking among candidate programs $B$. Given a training dataset of context-answer pairs, $\{(x^l, y^l)\}^N_{l=1}$, the training objective is

$$O_{\text{NPP}}(\omega) = \sum_{l=1}^{N} \sum_{1 \leq i \neq j \leq |s_\phi(x^l)|} I[v^{l,i} > v^{l,j}] \log \sigma(v^{l,i} - v^{l,j})$$

(2)

where $\sigma(v) = 1/(1 + e^{-v})$ is the sigmoid function, and $v^{l,i} = v_\omega(a^{l,i}; x^l, s_\phi(x^l))$ is the estimated value of $a^{l,i}$, the $i$-th program generated for context $x^l$.

We compare NPP training in two setups: a single MAPO setup, and a stack learning setup. For the single MAPO setup the queries used to produce a pretrained MAPO model are also used to train the NPP model. The dev and test queries are used for ablation study and final evaluation. Since the NPP training queries are already used to train the MAPO model, the candidate programs are biased towards better reward (compared to those candidate generated for unseen queries). This setup causes NPP to learn from a different distribution as the intended dev/test data. Surprisingly NPP is still able to improve the prediction of MAPO as will be discussed in Section 5.3.

To overcome the issue with single MAPO setup we also generate NPP training data with a stacked learning setup. First the train and dev queries are merged and splitted into $K$ equal portions, and with Leave-One-Out (LOO) scheme they form $K$ train/dev sets. Then $K$ MAPO models are trained on $K$ train sets. Finally we use each of the $K$ MAPO s to produce a stack learning dataset by running these models on their respective dev set. The stack learning dataset is further split for train/dev purposes for NPP. In this way, each NPP training query is decoded by a MAPO model, which has never seen this query before, and therefore avoid the bias in training data generation.

5 EXPERIMENTS

Our empirical study is based on the semantic parsing benchmark, WikiTableQuestions [Pasupat & Liang, 2015]. To focus on studying the planning part of the problem we assume that the search policy is pretrained using MAPO [Liang et al., 2018], and NPP is trained to rescore given candidate programs produced by MAPO. Additionally we show that stacked learning [Cohen & Carvalho, 2005] is helpful in correctly training a planner given pre-trained policy models.

5.1 EXPERIMENTAL SETUP

Datasets. WikiTableQuestions [Pasupat & Liang, 2015] 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 Pasupat & Liang [2015] for converting a table into a directed graph that can be queried, where rows and cells are converted to graph nodes while column names become labeled directed edges. For the questions, we use string
match to identify phrases that appear in the table. We also identify numbers and dates using the CoreNLP annotation released with the dataset.

**Baselines.** We compare NPP to Liang et al. (2018), the current state of the art on the WikiTableQuestions dataset. MAPO relies on beam search to find candidate programs, and uses the program probability according to the policy to determine the program to execute. MAPO manages to achieve 46.3% accuracy on this task when using an ensemble of size 10. We aim to show that NPP can improve on single MAPO as well as the ensemble of MAPO models.

**Training Details.** We set the stacked learning parameter \( K = 5 \) for all our experiments. We set the batch size to be equal to the size of the beam per program. We use Adam optimizer for training with a learning rate \( 10^{-3} \). We choose a size of 64 nodes for the LSTM (which becomes 128 as it is bidirectional). The CNN consists of 32 filters with kernel size 3. All the hyperparameters are tuned on the development set. We train the models for 10 epochs.

**Ensemble Details.** We formulate the ensemble of \( K \) MAPO models with NPP as the sum of normalized NPP scores under different MAPO models: Let \( \Phi = \{\phi^k\}_{k=1}^K \) be \( K \) MAPO models to be ensembled. We define the ensembled score of a program \( a \) under context \( x \) as

\[
v_{\omega, \Phi}(a; x) = \sum_k [v'_{\omega}(a; x, s^k_{\Phi}(x)) - \bar{v}_{\omega}(x)].
\]

where \( \bar{v}_{\omega}(x) \) is the average score for programs in beam \( s^k_{\Phi}(x) \)

\[
\bar{v}_{\omega}(x) = \frac{1}{|s^k_{\Phi}(x)|} \sum_{a \in s^k_{\Phi}(x)} v_{\omega}(a; x, s^k_{\Phi}(x))
\]

and \( v'_{\omega} \) backs-off \( v_{\omega} \) to \( \bar{v}_{\omega} \) whenever \( a \) is not in beam

\[
v'_{\omega}(a; x, s^k_{\Phi}(x)) = \begin{cases} v_{\omega}(a; x, s^k_{\Phi}(x)), & \text{if } a \in s^k_{\Phi}(x) \\ \bar{v}_{\omega}(x), & \text{else} \end{cases}
\]

<table>
<thead>
<tr>
<th>Setting</th>
<th>( \Delta \text{ Acc}(\text{std}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP- ( q^{\text{tok}} )</td>
<td>-0.37(0.04)</td>
</tr>
<tr>
<td>NPP- ( q^{\text{attn}} )</td>
<td>-0.04(0.2)</td>
</tr>
<tr>
<td>NPP- ( p^{\text{prob}} )</td>
<td>-6.04(0.7)</td>
</tr>
<tr>
<td>NPP- ( p^{\text{prob}} )</td>
<td>-0.14(0.03)</td>
</tr>
<tr>
<td>NPP- ( t^{\text{agree}} )</td>
<td>-0.39(0.2)</td>
</tr>
</tbody>
</table>

**5.2 Ablation Study**

In order to evaluate the effectiveness of our proposed programs token representations, we present an feature ablation test in Figure 3. We can see that the program probability \( p^{\text{prob}} \) produced by the search policy is the most important feature, providing the foundation to NPP scoring.

The program agreement feature \( t^{\text{agree}} \) is also very useful. We often observe cases for which beam \( B \) produces program with similar tokens that are not highly valued by the underlying model. By utilizing this feature, we more strongly consider programs which were repeatedly searched by \( s_{\Phi} \). \( t^{\text{agree}} \) also helps to identify the programs that are very similar throughout most of the sequence to learn their divergence and grade their utility.

Question referencing features such as \( q^{\text{tok}} \) provide significant importance in providing the program with query level context, ensuring we are filtering or selecting values based on the query context. While the help from attention feature \( q^{\text{attn}} \) is not significant.
Table 4: Main results. \( ^1 \)Improvements compared to MAPO. \( ^* \)Stacked learning with Leave-One-Out (LOO) data splits. \( ^\circ \)NPP uses 67%-33% train-dev splits from the stacked learning data.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Model</th>
<th>Dev (std)</th>
<th>( \Delta \uparrow )</th>
<th>Test (std)</th>
<th>( \Delta \uparrow )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of MAPOs trained on a single train/dev split</td>
<td>MAPO</td>
<td>41.9(0.3)</td>
<td>-</td>
<td>43.1(0.5)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MAPO + NPP</td>
<td>42.4(0.7)</td>
<td>0.8</td>
<td>43.7(0.6)</td>
<td>0.5</td>
</tr>
<tr>
<td>Mean of MAPOs trained on LOO splits</td>
<td>MAPO</td>
<td>41.7(1.1)</td>
<td>-</td>
<td>42.8(0.5)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MAPO + NPP*</td>
<td>43.0(0.2)</td>
<td>1.3</td>
<td>43.9(0.2)</td>
<td>1.1</td>
</tr>
<tr>
<td>Ensemble of 5 MAPOs trained on LOO splits</td>
<td>MAPO</td>
<td>-</td>
<td>-</td>
<td>45.5</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MAPO + NPP*</td>
<td>-</td>
<td>-</td>
<td>46.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Ensemble of 10 MAPOs trained on LOO splits</td>
<td>MAPO</td>
<td>-</td>
<td>-</td>
<td>46.3(-)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MAPO + NPP*</td>
<td>-</td>
<td>-</td>
<td>47.2(-)</td>
<td>0.9</td>
</tr>
</tbody>
</table>

5.3 MAIN RESULTS

We evaluate NPP’s impact on MAPO under three different settings. NPP consistently improves the precision of MAPO.

First we consider a single MAPO trained on a single train-dev data split. Similar to other RL models in the literature, MAPO training is a procedure with big variances. Even trained on the same data split multiple times with different random seed gives big variances in accuracy of 0.3% (dev) and 0.5% (test). Use a MAPO model to decode on its own train, develop and test data, and uses the result programs in beam to train, develop and test an NPP model. Training and applying NPP this way is able to improve precision, despite of the exposure bias in the NPP training data. However, it does not improve on the variances.

We next consider MAPO models trained on different train/dev splits created with a Leave-One-Out (LOO) scheme. As described in Section 4.3 we also use these splits to generate a stacked learning dataset for NPP to avoid the data bias problem. We can see that with different data splits MAPO has significantly higher variances on the dev set (1.1%), which is a drawback of RL models in general. Stacked learning helps NPP to improve precision more significantly (1.3% for dev and 1.1% for test). It also helps to reduce the variances to 0.2% on both dev and test.

Finally, we consider ensembled MAPO settings, which produces the previous state of the art result. We use the same NPP model trained from the stacked learning setting, and apply it to either the ensemble of these 5 MAPO models, or a 10 MAPO ensemble from \( [\text{Liang et al., } 2018] \). We can see that when applied to 5 MAPO ensemble, NPP can still significantly improve the precision by 1.1%. When applied to 10 MAPO ensemble, NPP can improves the precision by 0.9%.

5.4 EXPLAINABILITY ANALYSIS

Since the score of a program is the sum of its token scores, it is easy to visualize how NPP plan and select the correct program to execute. We observed that there are two common situations in which MAPO chooses the wrong program from the beam – selecting a spurious program over the semantically correct program and executing the incorrect table filtering or column selection. NPP aims to overcome these non optimal decisions by taking advantage of the larger time horizon and other programs discovered so far. For example NPP may reward earlier program tokens based on program tokens chosen much later on.

We first investigate how NPP demotes spurious programs. MAPO may produce programs which return the correct answer but are not semantically correct. NPP helps solve this by scoring semantically correct programs higher in the beam. An example is shown in Figure 3 for the question “What venue was the latest match played at?” when referring to a soccer (football) player given a table of his matches. The top program in beam proposed by MAPO was to first filter all rows for the competition string, which is incorrect considering the context of the table is only competitions. NPP is able to reevaluate the program given the full context. Although the first function (\( \text{filter\_in} \)) is typically used to filter the table for the correct row/column. NPP learns that in this situation it is
better to find the last of all rows using the function \texttt{last}. NPP re-evaluates the first function of the new best program as being high in utility, and scores all tokens within this function higher than the tokens in the incorrect program.

We then investigate programs from MAPO which produce wrong answers. An example is shown in Figure 4 which is based on Table 1. MAPO has low probability to the second program because it involves an infrequence column name “programming” while having high probability to the first program. Because NPP knows program token overlap with the query as well as the attention matrix, it is able to better understand the question and grade the program which is more closely related to the question. In addition to this we notice that the convolutional layer grades full functions within their context, given a kernel of size 3 the parenthesis at the beginning of program already receives a higher NPP score.

![Figure 3: Comparison of re-scorings by NPP. (Top) a semantically incorrect program now scored lower by NPP. (Bottom) a semantically correct program now scored higher by NPP.](image)

**Conclusion**

Reinforcement learning applied to structured prediction suffers from limited use of the world model as well as not being able to consider future and past program context when generating a sequence. To overcome these limitations we proposed Neural Program Planner (NPP) which is a planning step after candidate program generation, which can better evaluate overall structure value. We applied NPP to a difficult SP task improving state of the art by 0.9% and analyzed the scoring by NPP per program token in order to understand its decisions.
REFERENCES


A  MORE NPP SCORING DETAILS

Figure 5: MAPO probability per token for nt-10858:  
What venue was the latest match played at?  
The score sequences have the same length (16) because of padding within this query.

Figure 6: NPP scores per token for a set of candidate programs. nt-10858:  
What venue was the latest match played at?  
The score sequences have the same length (19) because of padding over all queries.
Figure 7: MAPO probability per token for nt-3516: Which programming is played the most? The score sequences have the same length (10) because of padding within this query.

Figure 8: NPP scores per token for a set of candidate programs. nt-3516: Which programming is played the most? The score sequences have the same length (19) because of padding over all queries.