Search-based Neural Structured Learning for Sequential Question Answering

Mohit Iyyer*
Department of Computer Science and UMIACS
University of Maryland, College Park
miyyer@umd.edu

Wen-tau Yih, Ming-Wei Chang
Microsoft Research
Redmond, WA 98052
{scottyih,minchang}@microsoft.com

Abstract

Recent work in semantic parsing for question answering has focused on long and complicated questions, many of which would seem unnatural if asked in a normal conversation between two humans. In an effort to explore a conversational QA setting, we present a more realistic task: answering sequences of simple but inter-related questions. We collect a dataset of 6,066 question sequences that inquire about semi-structured tables from Wikipedia, with 17,553 question-answer pairs in total. To solve this sequential question answering task, we propose a novel dynamic neural semantic parsing framework trained using a weakly supervised reward-guided search. Our model effectively leverages the sequential context to outperform state-of-the-art QA systems that are designed to answer highly complex questions.

1 Introduction

Semantic parsing, which maps natural language text to meaning representations in formal logic, has emerged as a key technical component for building question answering systems (Liang, 2016). Once a natural language question has been mapped to a formal query, its answer can be retrieved by executing the query on a back-end structured database.

One of the main focuses of semantic parsing research is how to address *compositionality* in language, and complicated questions have been specifically targeted in the design of a recently-released QA dataset (Pasupat and Liang, 2015). Take for example the following question: "*of those actresses who won a Tony after 1960, which one took the most amount of years after winning the Tony to win an Oscar?*" The corresponding logical form is highly compositional; in order to answer it, many sub-questions must be implicitly answered in the process (e.g., *"who won a Tony after 1960?"*).

While we agree that semantic parsers *should* be able to answer very complicated questions, in reality these questions are rarely issued by users.[1] Because users can interact with a QA system repeatedly, there is no need to assume a single-turn QA setting where the exact *question intent* has to be captured with just one complex question. The same intent can be more naturally expressed through a sequence of simpler questions, as shown below:

1. *What actresses won a Tony after 1960?*
2. *Of those, who later won an Oscar?*
3. *Who had the biggest gap between their two award wins?*

Decomposing complicated intents into multiple related but simpler questions is arguably a more effective strategy to explore a topic of interest, and it reduces the cognitive burden on both the person who asks the question and the one who answers it.[2]

In this work, we study semantic parsing for answering *sequences* of simple related questions. We collect a dataset of question sequences called SequentialQA (SQA; Section 2)[3] by asking crowdsourced workers to decompose complicated questions sampled from the WikiTableQuestions dataset (Pasupat and Liang, 2015) into multiple easier ones. SQA, which contains 6,066 question sequences with 17,553 total question-answer pairs, is to the best of our knowledge the first semantic parsing dataset for sequential question answering. Section 3 describes our novel dynamic neural semantic parsing framework (DynSP), a weakly su-

*Work done during an internship at Microsoft Research

[1] For instance, there are only 3.75% questions with more than 15 words in WikiAnswers (Fader et al., 2014).

[2] Studies have shown increased sentence complexity links to longer reading times (Hale, 2006; Levy, 2008; Frank, 2013).

[3] Available at http://aka.ms/sqa

Original intent:
What super hero from Earth appeared most recently?

1. Who are all of the super heroes?

2. Which of them come from Earth?

3. Of those, who appeared most recently?

Legion of Super Heroes Post-*Infinite Crisis*

Character	*First Appeared*	*Home World*	*Powers*
Night Girl	2007	Kathoon	Super strength
Dragonwing	2010	Earth	Fire breath
Gates	2009	Vyrga	Teleporting
XS	2009	Aarok	Super speed
Harmonia	2011	Earth	Elemental

Figure 1: An example question sequence created from a compositional question intent. Workers must write questions whose answers are subsets of cells in the table.

pervised structured-output learning approach based on reward-guided search that is designed for solving sequential QA. We demonstrate in Section 4 that DynSP achieves higher accuracies than existing systems on SQA, and we offer a qualitative analysis of question types that our method answers effectively, as well as those on which it struggles.

2 A Dataset of Question Sequences

We collect the SequentialQA (SQA) dataset via crowdsourcing by leveraging WikiTableQuestions (Pasupat and Liang, 2015, henceforth WTQ), which contains highly compositional questions associated with HTML tables from Wikipedia. Each crowdsourcing task contains a long, complex question originally from WTQ as the question *intent*. The workers are asked to compose a sequence of simpler questions that lead to the final intent; an example of this process is shown in Figure 1.

To simplify the task for workers, we only use questions from WTQ whose answers are cells in the table, which excludes those involving arithmetic and counting. We likewise also restrict the questions our workers can write to those answerable by only table cells. These restrictions speed the annotation process because workers can just click on the table to answer their question. They also allow us to collect answer coordinates (row and column in the table) as opposed to answer text, which removes many normalization issues for answer string matching in evaluation. Finally, we only use long questions that contain nine or more words as intents; shorter questions tend to be simpler and are thus less amenable to decomposition.

2.1 Properties of SQA

In total, we used 2,022 question intents from the train and test folds of the WTQ for decomposition. We had three workers decompose each intent, resulting in 6,066 unique questions sequences containing 17,553 total question-answer pairs (for an average of 2.9 questions per sequence). We divide the dataset into train and test using the original WTQ folds, resulting in an 83/17 train/test split. Importantly, just like in WTQ, none of the tables in the test set are seen in the training set.

We identify three frequently-occurring question classes: *column selection*, *subset selection*, and *row selection*.[4] In column selection questions, the answer is an entire column of the table; these questions account for 23% of all questions in SQA. Subset and row selection are more complicated than column selection, as they usually contain coreferences to the previous question's answer. In subset selections, the answer is a subset of the previous question's answer; similarly, the answers to row selections occur in the same row(s) as the previous answer but in a different column. Subset selections make up 27% of SQA, while row selections are an additional 19%. The remaining 31% contains more complex combinations of these three types.

We also observe dramatic differences in the types of questions that are asked at each position of the sequence. For example, 51% of the first questions in the sequences are column selections (e.g., "*what are all of the teams?*"). This number dwindles to just 18% when we look at the second question of each sequence, which indicates that the collected sequences start with general questions and progress to more specific ones.

3 Dynamic Neural Semantic Parsing

The unique setting of SQA provides both opportunities and challenges. On the one hand, it contains short questions with less compositionality, which in theory should reduce the difficulty of the semantic parsing problem; on the other hand, the additional contextual dependencies of the preceding questions and their answers increase modeling complexity. These observations lead us to propose a dynamic neural semantic parsing framework (DynSP) trained using a reward-guided search pro-

[4]In the example sequence "*what are all of the tournaments? in which one did he score the least points? on what date was that?*", the first question is a column selection, the second is a subset selection, and the last one is a row selection.

cedure for solving SQA.

Given a question (optionally along with previous questions and answers) and a table, `DynSP` formulates the semantic parsing problem as a state–action search problem. Each state represents a complete or partial parse, while each action corresponds to an operation to extend a parse. The goal during inference is to find an end state with the highest score as the predicted parse.

The quality of the induced semantic parse obviously depends on the scoring function. In our design, the score of a state is determined by the scores of actions taken from the initial state to the target state, which are predicted by different neural network modules based on action type. By leveraging a margin-based objective function, the model learning procedure resembles several structured-output learning algorithms such as structured SVMs (Tsochantaridis et al., 2005), but can take either strong or weak supervision seamlessly.

`DynSP` is inspired by STAGG, a search-based semantic parser (Yih et al., 2015), as well as the dynamic neural module network (DNMN) of Andreas et al. (2016). Much like STAGG, `DynSP` chains together different modules as search progresses; however, these modules are implemented as neural networks, which enables end-to-end training as in DNMN. The key difference between `DynSP` and DNMN is that in `DynSP` the network structure of an example is not predetermined. Instead, different network structures are constructed dynamically as our learning procedure explores the state space. It is straightforward to answer *sequential* questions using our framework: we allow the model to take the previous question and its answers as input, with a slightly modified action space to reflect a *dependent* semantic parse. The same search / learning procedure is then able to effortlessly adapt to the new setting. In this section, we first describe the formal language underlying `DynSP`, followed by the model formulation and learning algorithm.

3.1 Semantic parse language

Because tables are used as the data source to answer questions in SQA, we decide to form our semantic parses in an SQL-like language[5]. Our parses consist of two parts: a ***select*** statement and conjunctions of zero or more ***conditions***.

[5]Our framework is not restricted to the formal language we use in this work. In addition, the structured query can be straightforwardly represented in other formal languages, such as the lambda DCS logic used in (Pasupat and Liang, 2015).

A ***select*** statement is associated with a column name, which is referred to as the *answer* column. Conditions enforce additional constraints on which cells in the answer column can be chosen; a ***select*** statement without any conditions indicates that an entire column of the table is the answer to the question. In particular, each condition contains a column name as the *condition* column and an operator with zero or more arguments. The operators in this work include: $=, \neq, >, \geq, <, \leq, \arg\min, \arg\max$. A cell in the answer column is only a legitimate answer if the cell of the corresponding row in the condition column satisfies the constraint defined by the operator and its arguments. As a concrete example, suppose the data source is the same table in Fig. 1. The semantic parse of the question "*Which super heroes came from Earth and first appeared after 2009?*" is "Select **Character** Where $\{$**Home World** = *Earth*$\} \wedge \{$**First Appeared** $>$ *2009*$\}$" and the answers are $\{$Dragonwing, Harmonia$\}$.

In order to handle the *sequential* aspect of SQA, we extend the semantic parse language by adding a preamble statement ***subsequent***. A ***subsequent*** statement contains only conditions, as it essentially adds constraints to the semantic parse of the previous question. For instance, if the follow-up question is "*Which of them breathes fire?*", then the corresponding semantic parse is "Subsequent Where $\{$**Powers** $=$ *Fire breath*$\}$". The answer to this question is $\{$Dragonwing$\}$, a subset of the previous answer.

3.2 Model formulation

We start introducing our model design by first defining the state and action space. Let $\mathcal{S}$ be the set of states and $\mathcal{A}$ the set of all actions. A state $s \in \mathcal{S}$ is simply a sequence of variable length of actions $\{a_1, a_2, a_3, \cdots, a_t\}$, where $a_i \in \mathcal{A}$. An empty sequence, $s_0 = \phi$, is a special state used as the starting point of search.

As mentioned earlier, a state represents a (partial) semantic parse of *one* question. Each action is thus a legitimate operation that can be added to *grow* the semantic parse. Our action space design is tied closely to the statements defined by our parse language; in particular, an action *instance* is either a complete or partial statement, and action instances are grouped by *type*. For example, ***select*** and ***subsequent*** operations are two action types. A ***condition*** statement is formed by two different action types:

Id	Type	# Action instances
$\mathcal{A}_1$	Select-column	# columns
$\mathcal{A}_2$	Cond-column	# columns
$\mathcal{A}_3$	Op-Equal ($=$)	# rows
$\mathcal{A}_4$	Op-NotEqual ($\neq$)	# rows
$\mathcal{A}_5$	Op-GT ($>$)	# numbers / datetimes
$\mathcal{A}_6$	Op-GE ($\geq$)	# numbers / datetimes
$\mathcal{A}_7$	Op-LT ($<$)	# numbers / datetimes
$\mathcal{A}_8$	Op-LE ($\leq$)	# numbers / datetimes
$\mathcal{A}_9$	Op-ArgMin	# numbers / datetimes
$\mathcal{A}_{10}$	Op-ArgMax	# numbers / datetimes
$\mathcal{A}_{11}$	Subsequent	1
$\mathcal{A}_{12}$	S-Cond-column	# columns
$\mathcal{A}_{13}$	S-Op-Equal ($=$)	# rows
$\mathcal{A}_{14}$	S-Op-NotEqual ($\neq$)	# rows
$\mathcal{A}_{15}$	S-Op-GT ($>$)	# numbers / datetimes
$\mathcal{A}_{16}$	S-Op-GE ($\geq$)	# numbers / datetimes
$\mathcal{A}_{17}$	S-Op-LT ($<$)	# numbers / datetimes
$\mathcal{A}_{18}$	S-Op-LE ($\leq$)	# numbers / datetimes
$\mathcal{A}_{19}$	S-Op-ArgMin	# numbers / datetimes
$\mathcal{A}_{20}$	S-Op-ArgMax	# numbers / datetimes

Table 1: Types of actions and the number of action instances in each type. Numbers / datetimes are the mentions discovered in the question (plus the previous question if it is a subsequent condition).

(1) selection of the condition column, and (2) the comparison operator. The instances of each action type differ in their arguments (e.g., column names, or specific cells in a column). Because conditions in a ***subsequent*** parse rely on previous questions and answers, they belong to different action types from regular conditions. Table 1 summarizes the action space defined in this work.

Any state that represents a complete and legitimate parse is an end state. Notice that search does not necessarily need to stop at an end state, because adding more actions (e.g., condition statements) can lead to another end state. Take the same example question from before: "*Which super heroes came from Earth and first appeared after 2009?*". One action sequence that represents the parse is $\{(\mathcal{A}_1)$ select-column **Character**, $(\mathcal{A}_2)$ cond-column **Home World**, $(\mathcal{A}_3)$ op-equal *Earth*, $(\mathcal{A}_2)$ cond-column **First Appeared**, $(\mathcal{A}_5)$ op-gt *2009*$\}$.

Notice that many states represent semantically equivalent parses (e.g., those with the same actions ordered differently, or states with repeated conditions). To prune the search space, we introduce the function $Act(s) \subset \mathcal{A}$, which defines the actions that can be taken when given a state s. Borrowing the idea of *staged* state generation in (Yih et al., 2015), we choose a default ordering of actions based on their types, dictating that a *select* action must be picked first and that a *condition-column* needs to be determined before the *operator* is chosen. The full transition diagram is presented in Fig. 2. Note that to implement this transition order, we only need to check the last action in the state. In addition, we also disallow adding duplicates of actions that already exist in the state.

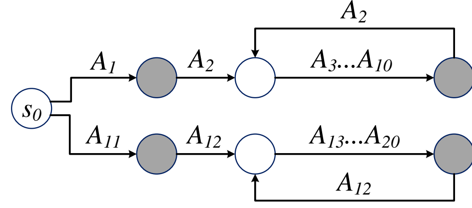

Figure 2: Possible action transitions based on their types (see Table 1). Shaded circles are end states.

We use beam search to find an end state with the highest score for inference. Let s_t be a state consisting of a sequence of actions $a_1, a_2, \cdots, a_t$. The state value function V is defined recursively as $V(s_t) = V(s_{t-1}) + \pi(s_{t-1}, a_t), V(s_0) = 0$, where the *policy* function $\pi(s, a)$ scores an action $a \in Act(s)$ given the current state.

3.3 Policy function

The intuition behind the policy function can be summarized as follows. Halfway through the construction of a semantic parse, the policy function measures the quality of an immediate action that can be taken next given the current state (i.e., the question and actions that have previously been chosen). To enable integrated, end-to-end learning, the policy function in our framework is parameterized using neural networks. Because each action type has very different semantics, we design different network structures (i.e., modules) accordingly.

Most of our network structures encourage learning semantic matching functions between the words in the question and table (either the column names or cells). Here we illustrate the design using the *select-column* action type ($\mathcal{A}_1$). Conceptually, the corresponding module is a combination of various matching scores. Let W_Q be the embeddings of words in the question and W_C be the embeddings of words in the target column name. The component matching functions are:

$$f_{max} = \frac{1}{|W_C|} \sum_{w_c \in W_C} \max_{w_q \in W_Q} w_q^T w_c$$

$$f_{avg} = \left(\frac{1}{|W_C|} \sum_{w_c \in W_C} w_c \right)^T \left(\frac{1}{|W_Q|} \sum_{w_q \in W_Q} w_q \right)$$

Essentially, for each word in the column name, f_{max} finds the highest matching question word and outputs the average score. Conversely, f_{avg} simply uses the average word vectors of the question and column name and returns their inner product. In another variant of f_{avg}, we replace the question representation with the output of a bi-directional LSTM model. These matching component functions are combined by a 2-layer feed-forward neural network, which outputs a scalar value as the action score. Details of the neural module design for other action types can be found in Appendix A.

3.4 Model learning

Because the state value function V is defined recursively as the sum of scores of actions in the sequence, the goal of model optimization is to learn the parameters in the neural networks behind the policy function. Let θ be the collection of all the model parameters. Then the state value function can be written as: $V_\theta(s_t) = \sum_{i=1}^{t} \pi_\theta(s_{i-1}, a_i)$.

In a fully supervised setting where the correct semantic parse of each question is available, learning the policy function can be reduced to a sequence prediction problem. However, while having full supervision leads to a better semantic parser, collecting the correct parses requires a much more sophisticated UI design (Yih et al., 2016). In many scenarios, such as the one in the SQA dataset, it is often the case that only the answers to the questions are available. Adapting a learning algorithm to this *weakly* supervised setting is thus critical.

Generally speaking, weakly supervised semantic parsers operate on one assumption — a candidate semantic parse is treated as a correct one if it results in answers that are identical to the gold answers. Therefore, a straightforward modification of existing structured learning algorithms in our setting is to use any semantic parse found to evaluate to the correct answers during beam search as a *reference* parse, and then update the model parameters accordingly. In practice, however, this approach is often problematic: the search space can grow enormously, and when coupled with poor model performance early during training, this leads to beams that contain no parses evaluating to the correct answer. As a result, learning becomes inefficient and takes a long time to converge.

In this work, we propose a conceptually simple learning algorithm for weakly supervised training that sidesteps the inefficient learning problem. Our key insight is to conduct inference using a beam search procedure guided by an *approximate reward* function. The search procedure is executed *twice* for each training example, one for finding the best possible reference semantic parse and the other for finding the predicted semantic parse to update the model. Our framework is suitable for learning from either implicit or explicit supervision, and is detailed in a companion paper (Peng et al., 2017). Below we describe how we adapt it to the semantic parsing problem in this work.

Approximate reward Let $A(s)$ be the answers retrieved by executing the semantic parse represented by state s, and let A^* be the set of gold answers of a given question. We define the *reward* $R(s; A^*) = \mathbb{1}[A(s) = A^*]$, or the accuracy of the retrieved answers. We use $R(s)$ as the abbreviation for $R(s; A^*)$. A state s with $R(s) = 1$ is called a goal state. Directly using this reward function in search of goal states can be difficult, as rewards of most states are 0. However, even when the answers from a semantic parse are not completely correct, some overlap with the gold answers can still hint that the state is close to a goal state, thus providing useful information to guide search. To formalize this idea, we define an *approximated reward* $\tilde{R}(s)$ in this work using the Jaccard coefficient ($\tilde{R}(s) = |A(s) \cap A^*|/|A(s) \cup A^*|$). If s is a goal state, then obviously $\tilde{R}(s) = R(s) = 1$. Also because our actions effectively add additional constraints to exclude some table cells, any succeeding states of s' with $\tilde{R}(s') = 0$ will also have 0 approximate reward and can be pruned from search immediately.

We use the approximate reward $\tilde{R}$ to guide our beam search to find the reference parses (i.e., goal states). Some variations of the approximate reward can be used to make learning more efficient. For instance, we use the model score for tie-breaking, effectively making the approximate reward function depend on the model parameters:

$$\tilde{R}_\theta(s) = |A(s) \cap A^*|/|A(s) \cup A^*| + \epsilon V_\theta(s), \quad (1)$$

where ϵ is a small constant. When a goal state is not found, the state with the highest approximate reward can still be used as a surrogate reference.

Updating parameters The model parameters are updated by first finding the most *violated* state $\hat{s}$ and then comparing $\hat{s}$ with a reference state s^* to compute a loss. The idea of finding the most violated state comes from Taskar et al. (2004), with the

Algorithm 1 Model parameter updates

1: **for** pick a labeled data (x, A^*) **do**
2: $\quad s^* \leftarrow \arg\max_{s \in \mathcal{E}(x)} \tilde{R}(s; A^*)$
3: $\quad \hat{s} \leftarrow \arg\max_{s \in \mathcal{E}(x)} V_\theta(s) - \tilde{R}(s; A^*)$
4: $\quad$ update θ by minimizing $\max(\mathcal{L}(s), 0)$
5: **end for**

intuition that the learning algorithm should make the state value function behave similarly to the reward. Formally, for every state s, we would like the value function to satisfy the following constraint:

$$V_\theta(s^*) - V_\theta(s) \geq R(s^*) - R(s) \quad (2)$$

$R(s^*) - R(s)$ is thus the margin. As discussed above, we use approximate reward function $\tilde{R}_\theta$ instead of the true reward. We want to update the model parameters θ to make sure that the constraint is satisfied. When the constraint is violated, the degree of violation can be written as:

$$\mathcal{L}(s) = V_\theta(s) - V_\theta(s^*) - \tilde{R}_\theta(s) + \tilde{R}_\theta(s^*) \quad (3)$$

In the algorithm, we want to find the state such that the corresponding constraint is most violated. Finding the most violated state is then equivalent to finding the state with the highest value of $V_\theta(s) - \tilde{R}_\theta(s)$ as the other two terms are constant.

Algorithm 1 sketches the key steps of our method in each iteration. It first picks a training instance (x and y), where x represents the table and the question, and y is the gold answer set. The approximate reward function $\tilde{R}$ is defined by y, while $\mathcal{E}(x)$ is the set of end states for this instance. Line 2 finds the best reference and Line 3 finds the most violated state, both relying on beam search for approximate inference. Line 4 computes the gradient of the loss in Eq. (3), which is then used in backpropagation to update the model parameters.

4 Experiments

Since the questions in SQA are decomposed from those in WTQ, we compare our method, `DynSP`, to two existing semantic parsers designed for WTQ: (1) the *floating parser* (`FP`) of Pasupat and Liang (2015), and (2) the *neural programmer* (`NP`) of Neelakantan et al. (2017). We describe below each system's configurations in more detail and qualitatively compare and contrast their performance on SQA.

Floating parser: The floating parser (Pasupat and Liang, 2015) maps questions to logical forms and then executes them on the table to retrieve the answers. It was designed specifically for the WTQ task (achieving 37.0% accuracy on the WTQ test set) and differs from other semantic parsers by not anchoring predicates to tokens in the question, relying instead on typing constraints to reduce the search space. Using `FP` as-is results in poor performance on SQA because the system is configured for questions with single answers, while SQA contains many questions with multiple-cell answers. We address this issue by removing a pruning hyperparameter (*tooManyValues*) and features that add bias on the denotation size.

Neural programmer: The neural programmer proposed by Neelakantan et al. (2017) has shown promising results on WTQ, achieving accuracies on par with those of `FP`. Similar to our method, `NP` contains specialized neural modules that perform discrete operations such as argmax and argmin, and it is able to chain together multiple modules to answer a single question. However, module selection in `NP` is computed via soft attention (Cho et al., 2014), and information is propagated from one module to the next using a recurrent neural network. Since module selection is not tied to a pre-defined parse language like `DynSP`, `NP` simply runs for a fixed number of recurrent timesteps per question rather than growing a parse until it is complete.

Comparing the baseline systems: `FP` and `NP` exemplify two very different paradigms for designing a semantic parsing system to answer questions using structured data. `FP` is a feature-rich system that aims to output the correct semantic parse (in a logical parse language) for a given question. On the other hand, the end-to-end neural network of `NP` relies on its modular architectures to output a probability distribution over cells in a table given a question. While `NP` can learn more powerful neural matching functions between questions and tables than `FP`'s simpler feature-based matching, `NP` cannot produce a complete, discrete semantic parse, which means that its actions can only be interpreted coarsely by looking at the order of the modules selected at each timestep.[6] Furthermore, `FP`'s design theoretically allows it to operate on partial tables

[6]Since `NP` uses a fixed number of timesteps for each question, the module order is not guaranteed to correspond to a complete parse.

indirectly through an API, which is necessary if tables are large and stored in a backend database, while NP requires upfront access to the full tables to facilitate end-to-end model differentiability.[7]

Even though FP and NP are powerful systems designed for the more difficult, compositional questions in WTQ, our method outperforms both systems on SQA when we consider all questions within a sequence independently of each other (a fair comparison), demonstrating the power of our search-based semantic parsing framework. More interestingly, when we leverage the sequential information by including the ***subsequent*** action, our method improves almost 3% in absolute accuracy.

DynSP combines the best parts of both FP and NP. Given a question, we try to generate its correct semantic parse in a formal language that can be predefined by the choice of structured data source (e.g., SQL). However, we push the burden of feature engineering to neural networks as in NP. Our framework is easier to extend to the sequential setting of SQA than either baseline system, requiring just the additional ***subsequent*** action. FP's reliance on a hand-designed grammar necessitates extra rules that operate over partial tables from the previous question, which if added would blow up the search space. Meanwhile, modifying NP to handle sequential QA is non-trivial due to soft module and answer selection; it is not immediately clear how to constrain predictions for one question based on the probability distribution over table cells from the previous question in the sequence.

To more fairly compare DynSP to the baseline systems, we also experiment with a "concatenated questions" setting, which allows the baselines to access sequential context. Here, we treat concatenated question prefixes of a sequence as additional training examples, where a question prefix includes all questions prior to the current question in the sequence.

For example, suppose the question sequence is: *1. what are all of the teams? 2. of those, which won championships?* For the second question, in addition to the original question–answer pair, we add the concatenated question sequence *"what are all of the teams? of those, which won championships?"* paired with the second question's answer. We refer to these concatenated question baselines as FP$^+$ and NP$^+$.

[7] In fact, NP is restricted during training to only questions whose associated tables have fewer than a certain threshold of rows and columns due to computational constraints.

4.1 DynSP implementation details

Unlike previous dynamic neural network frameworks (Andreas et al., 2016; Looks et al., 2017), where each example can have different but *predetermined* structure, DynSP needs to dynamically explores and constructs different neural network structures for each question. Therefore, we choose DyNet (Neubig et al., 2017) as our implementation platform for its flexibility in composing computation graphs. We optimize our model parameters using standard stochastic gradient descent. The word embeddings are initialized with 100-d pretrained GloVe vectors (Pennington et al., 2014) and fine-tuned during training with dropout rate 0.5. For follow-up questions, we choose uniformly at random to use either gold answers to the previous question or the model's previous predictions.[8] We constrain the maximum length of actions to 3 for computational efficiency and set the beam size to 15 in our reported models, as accuracy gains are negligible with larger beam sizes. We train our model for 30 epochs, although the best model on the validation set is usually found within the first 20 epochs. Only CPU is used in model training, and each epoch in the beam size 15 setting takes about 30 minutes to complete.

4.2 Results & Analysis

Table 2 shows the results of the baseline systems as well as our method on SQA's test set. For each system, we show both the overall accuracy, the sequence accuracy (the percentage of sequences for which every question was answered correctly), and the accuracy at each position in the sequence. Our method without any sequential information (DynSP) outperforms the standard baselines, and when the *subsequent* action is added (DynSP*), we improve both overall and sequence accuracy over the concatenated-question baselines.

With that said, all of the systems struggle to answer all questions within a sequence correctly, despite the fact that each individual question is simpler on average than those in WTQ. Most of the errors made by our system are due to either semantic matching challenges or limitations of the underlying parse language. In the middle example of Figure 3, the first question asks for a list of super heroes; from the model's point of view, *Real name* is a more relevant column than *Character*, although the latter is correct. The second question also con-

[8] Only predicted answers are used at test time.

Model	All	Seq	Pos 1	Pos 2	Pos 3
FP	34.1	7.2	52.6	25.6	**25.9**
NP	39.4	10.8	58.9	35.9	24.6
DynSP	42.0	10.2	**70.9**	35.8	20.1
FP$^+$	33.2	7.7	51.4	22.2	22.3
NP$^+$	40.2	11.8	60.0	35.9	25.5
DynSP*	**44.7**	**12.8**	70.4	**41.1**	23.6

Table 2: Accuracies of all systems on SQA; the models in the first half of the table treat questions independently, while those in the second half consider sequential context. Our method outperforms existing ones both in terms of overall accuracy as well as sequence accuracy.

1. Which nations competed in the FINA women's water polo cup?
SELECT Nation
2. Of these nations, which ones took home at least one gold medal?
SUBSEQUENT WHERE Gold != 0
3. Of those, which ranked in the top 2 positions?
SUBSEQUENT WHERE Rank <= 2

1. Who are all of the super heroes?
SELECT Real name Character
2. Which of those does not have a home world listed?
SUBSEQUENT WHERE Home world != Vyrga Unknown

1. How many naturalizations did Maghreb have in 2000?
SELECT 2000 WHERE ...Origin = Maghreb
2. How many naturalizations did North America have in 2000?
SELECT 2000 WHERE ...Origin = North America
3. Which had more?
SUBSEQUENT WHERE ...Origin = North America
SELECT ...Origin WHERE 2000 =
MAX SUBSEQUENT 1 SUBSEQUENT 2

Figure 3: Parses computed by DynSP for three test sequences (actions in blue boxes, values from table in white boxes). *Top*: all three questions are parsed correctly. *Middle*: semantic matching errors cause the model to select incorrect columns and conditions. *Bottom*: The final question is unanswerable due to limitations of our parse language.

tains a challenging matching problem where the unlisted home worlds referred to in the question are marked as *Unknown* in the table. Many of these matching issues are resolved by humans using common sense, which for computers requires far more data than is available in SQA to learn.

Even when there are no tricky discrepancies between question and table text, questions are often complex enough that their semantic parses cannot be expressed in our parse language. Although trivial on the surface, the final question in the bottom sequence of Figure 3 is one such example; the correct semantic parse requires access to the answers of both the first and second question, actions that we have not currently implemented in our language due to concerns with the search space size. Increasing the number of complex actions requires designing smarter optimization procedures, which we leave to future work.

5 Related Work

Previous work on conversational QA has focused on small, single-domain datasets. Perhaps most related to our task is the context-dependent sentence analysis described in (Zettlemoyer and Collins, 2009), where conversations between customers and travel agents are mapped to logical forms after resolving referential expressions. Another dataset of travel booking conversations is used by Artzi and Zettlemoyer (2011) to learn a semantic parser for complicated queries given user clarifications. More recently, Long et al. (2016) collect three contextual semantic parsing datasets (from synthetic domains) that contain coreferences to entities and actions. We differentiate ourselves from these prior works in two significant ways: first, our dataset is not restricted to a particular domain, and second, a major goal of our work is to analyze the different types of sequence progressions people create when they are trying to express a complicated intent.

Complex, interactive QA tasks have also been proposed in the information retrieval community, where the data source is a corpus of newswire text (Kelly and Lin, 2007). We also build on aspects of some existing interactive question-answering systems. For example, the system of Harabagiu et al. (2005) includes a module that predicts what a user will ask next given their current question.

Other than FP and NP, the work of Neural Symbolic Machines (NSM) (Liang et al., 2017) is perhaps the closest to ours. NSM aims to generate formal semantic parses of questions that can be executed on Freebase to retrieve answers, and is trained using the REINFORCE algorithm (Williams, 1992) augmented with approximate gold parses found in a separate curriculum learning stage. In comparison, finding reference parses is an integral part of our algorithm. Our non-

probabilistic, margin-based objective function also helps avoid the need for empirical tricks to handle normalization and proper sampling, which are crucial when applying REINFORCE in practice.

6 Conclusion & Future Work

In this work we move towards a conversational, multi-turn QA scenario in which systems must rely on prior context to answer the user's current question. To this end, we introduce SQA, a dataset that consists of 6,066 unique sequences of inter-related questions about Wikipedia tables, with 17,553 questions-answer pairs in total. To the best of our knowledge, SQA is the first semantic parsing dataset that addresses sequential question answering. We propose `DynSP`, a dynamic neural semantic parsing framework, for solving SQA. By formulating semantic parsing as a state–action search problem, our method learns modular neural network models through reward-guided search. `DynSP` outperforms existing state-of-the-art systems designed for answering complex questions when applied to SQA, and increases the gain after incorporating the subsequent actions.

In the future, we plan to investigate several interesting research questions triggered by this work. For instance, although our current formal language design covers most question types in SQA, it is nevertheless important to extend it further to make the semantic parser more robust (e.g., by including UNION or allowing comparison of multiple previous answers). Practically, allowing a more complicated semantic parse structure—either by increasing the number of primitive statements or the length of the parse—poses serious computational challenges in both model learning and inference. Because of the dynamic nature of our framework, it is not trivial to leverage the computational capabilities of GPUs using minibatched training; we plan to investigate ways to take full advantage of modern computing machinery in the near future. Finally, better resolution of semantic matching errors is a top priority, and unsupervised learning from large external corpora is one way to make progress in this direction.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We are also grateful to Panupong Pasupat for his help in configuring the floating parser baseline, and to Arvind Neelakantan for his help with the neural programmer model.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of Empirical Methods in Natural Language Processing*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of Empirical Methods in Natural Language Processing*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pages 1156–1165.

Stefan L Frank. 2013. Uncertainty reduction as a measure of cognitive load in sentence comprehension. *Topics in Cognitive Science* 5(3).

John Hale. 2006. Uncertainty about the rest of the sentence. *Cognitive Science* 30(4).

Sanda Harabagiu, Andrew Hickl, John Lehmann, and Dan Moldovan. 2005. Experiments with interactive question-answering. In *Proceedings of the Association for Computational Linguistics*.

Diane Kelly and Jimmy Lin. 2007. Overview of the trec 2006 ciqa task. In *ACM SIGIR Forum*. ACM, volume 41, pages 107–116.

Roger Levy. 2008. Expectation-based syntactic comprehension. *Cognition* 106(3).

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Vancouver, Canada.

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM* 59(9):68–76.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the Association for Computational Linguistics*.

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. 2017. Deep learning with dynamic computation graphs. In *Proceedings of the International Conference on Learning Representations*.

Arvind Neelakantan, Quoc Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. In *Proceedings of the International Conference on Learning Representations*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Association for Computational Linguistics*.

Haoruo Peng, Ming-Wei Chang, and Wen-tau Yih. 2017. Maximum margin reward networks for learning from explicit and implicit supervision. Manuscript Submitted for Publication.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing*.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004. Max-margin Markov networks. In *Proceedings of Advances in Neural Information Processing Systems*.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of machine learning research* 6(Sep):1453–1484.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 1321–1331.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. Berlin, Germany, pages 201–206.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics*.

A Action Neural Module Design

We describe here the neural module design for each action. As most actions try to match question text to column names or table entries, the neural network architectures are essentially various kinds of semantic similarity matching functions.

$\mathcal{A}_1$ **Select-column** Conceptually, the corresponding module is a combination of various matching scores. Let W_Q be the embeddings of words in the question and W_C be the embeddings of words in the target column name. The component matching functions are:

$$f_{max} = \frac{1}{|W_C|} \sum_{w_c \in W_C} \max_{w_q \in W_Q} w_q^T w_c$$

$$f_{avg} = \left(\frac{1}{|W_C|} \sum_{w_c \in W_C} w_c \right)^T \left(\frac{1}{|W_Q|} \sum_{w_q \in W_Q} w_q \right)$$

Essentially, for each word in the column name, f_{max} finds the highest matching question word and outputs the average score. Conversely, f_{avg} simply uses the average word vectors of the question and column name and returns their inner product. In another variant of f_{avg}, we replace the question representation with the output of a bidirectional LSTM model. These matching component functions are combined by a 2-layer feed-forward neural network, which outputs a scalar value as the action score.

$\mathcal{A}_2$ **Cond-column** Because this action also tries to find the correct column (but for conditions), we use the same matching scoring functions as in $\mathcal{A}_1$ module. However, a different 2-layer feed-forward neural network is used to combine the scores, as well as two binary features that indicate whether all the cells in this column are numeric values or not.

$\mathcal{A}_3$ **Op-Equal** This action checks whether a particular column value matches the question text. Suppose the average of the word vectors of the particular cell is w_x and the question word vectors are W_Q. Here the matching function is:

$$f_{max} = \max_{w_q \in W_Q} w_q^T w_x$$

$\mathcal{A}_4$ **Op-NotEqual** The neural module for this action extends the design for $\mathcal{A}_3$. It first uses a $\max$ function similar to f_{max} in $\mathcal{A}_3$ to compare the vector of the negation word "not", and the question words. This score is combined with the f_{max} score in $\mathcal{A}_3$ using a 2-layer feed-forward neural network as the final module score.

$\mathcal{A}_5$-$\mathcal{A}_8$ **Op-GT, Op-GE, Op-LT, Op-LE** The arguments of these comparison operations are extracted from question in advance. Therefore, the action modules just need to decide whether such relations are indeed used in the question. We take a simple strategy by initialing a special word vector that tries to capture the semantics of the relation. Take op-gt, *greater than*, for example. We use the average of the vectors of words like *more*, *greater* and *larger* to initialize the special word vector, denoted as w_{gt}. Let w_{arg} be the averaged vectors of words within a $[-2, +2]$ window centered at the argument in the question. The inner product of w_{gt} and w_{arg} is then used as the scoring function.

$\mathcal{A}_9$-$\mathcal{A}_{10}$ **Op-ArgMin, Op-ArgMax** We handle ArgMin and ArgMax similarly to the comparison operations. The difference is that we compare the special word vector to the averaged vector of all the question words, instead of a short subsequence of words.

Subsequent actions The modules in subsequent actions use basically the same design as their counterparts in the independent question setting. The main difference is that we extend the question representation to words from not just the target question, but also the question that immediately precedes it.