TabFact: A Large-scale Dataset for Table-based Fact Verification

Anonymous EMNLP-IJCNLP submission

Abstract

The problem of verifying whether a textual hypothesis holds the truth based on the given evidence, also known as fact verification, plays an important role in the study of natural language understanding and semantic representation. However, existing studies are restricted to dealing with unstructured textual evidence (e.g., sentences and passages, a pool of passages), while verification using structured forms of evidence, such as tables, graphs, and databases, remains unexplored. This paper specifically aims to study fact verification with semi-structured evidence. We construct a large-scale dataset called TABFACT with 16k Wikipedia tables as evidence for 118k human annotated statements. The statements are labeled as either ENTAILED or REFUTED. TABFACT is challenging since it involves both soft linguistic reasoning and hard symbolic reasoning. To address these challenges, we design two different models: Table Verification Network (TVNet) and Latent Program Algorithm (LPA). TVNet leverages the state-of-the-art BERT model to encode linearized tables and statements into continuous vectors for verification. LPA parses statements into LISP-like programs to execute them against the tables for verification. Both methods achieve similar reasonable accuracy which is yet far from human performance. We also perform comprehensive analysis and demonstrate great future opportunities.

1 Introduction

Verifying whether a textual hypothesis is entailed or refuted by the given evidence is a fundamental problem in natural language understanding (Katz and Fodor, 1963; Van Benthem et al., 2008). It has been extensively studied under different natural language tasks such as recognizing textual entailment (RTE) (Dagan et al., 2005), natural language inference (NLI) (Bowman et al., 2015), claim verification (Popat et al., 2017; Hanselowski et al., 2018; Thorne et al., 2018), and commonsense reasoning (Chierchia and McConnell-Ginet, 2000; Zellers et al., 2018). RTE and NLI view a premise sentence as the evidence, whereas claim verification views a passage as the evidence, and commonsense reasoning views a context sentence as the evidence. These problems have been previously addressed using a variety of techniques including logic rules, knowledge bases, and neural networks. Recently large-scale pre-trained language models (Devlin et al., 2019; Peters et al., 2018; Radford et al., 2019) have surged to dominate the other algorithms by achieving extremely strong performances (even approaching human performances) on several textual entailment tasks (Wang et al., 2018).

However, existing related studies are restricted to dealing with unstructured text as the evidence, which would not generalizes to the case where the evidence has a highly structured format. Since such structured evidence (graphs, tables, or databases) are also ubiquitous in real-world applications like database systems, dialog systems, commercial management systems, social networks, etc, we argue that the fact verification problem with structured evidence forms is an equally important yet unexploited problem. Therefore, in this paper, we are specifically interested in studying fact verification with semi-structured Wikipedia tables (Bhagavatula et al., 2013)\(^1\) as evidences owing to its structured and ubiquitous nature (Jauhar et al., 2016; Zhong et al., 2017; Pasupat and Liang, 2015). To this end, we introduce a large-scale dataset called TABFACT, which consists of 118,439 manually annotated statements with regard to 16,621 Wikipedia tables.

\(^1\)In contrast to the database tables, where each column has strong type constraint, the cell records in our semi-structured tables can be string/data/integer/float/phrase/sentences.
tables, their relations are classified as ENTAILED and REFUTED. The entailed and refuted statements are both annotated by human workers. With some examples in Figure 1, we can clearly observe that unlike the previous verification related problems, TABFACT combines two different forms of reasoning in the statements, (i) **Linguistic Reasoning**: the inference requires semantic-level understanding. For example, “John J. Mcfall failed to be re-elected though being unopposed.” requires understanding both the semantics of the statement and the semantics of phrases (sentences) of the table to correctly classify it as a contradictory statement. (ii) **Symbolic Reasoning**: the inference requires symbolic execution on the table structure. “There are three Democratic incumbents” requires condition operation (where condition) and arithmetic operation (count) to be classified as an entailed statement. The two forms of reasoning are interleaved extensively across the statements, which makes it challenging for existing models.

In this paper, we particularly probe two models to deal with such mixed-reasoning: (i) **Table Verification Network**, this model views the verification task completely as an NLI problem by linearizing a table into a template premise sentence $p$, and applies state-of-the-art language understanding model to encode both the linearized table and statements $h$ into distributed representation for classification. This model is expected to excel at the linguistic reasoning aspects but lacks symbolic reasoning skills. (ii) **Latent Program Algorithm**, this model applies lexical matching to find linked entities and then use pre-defined API sets (e.g. argmax, argmin, count, etc) to construct the latent LISP-like program candidates, a discriminator is utilized to select the most “consistent” latent programs. This model excels the symbolic reasoning aspects by leveraging symbolic operations, and it also provides better interpretability. We perform extensive experiments to investigate their performances: the best-achieved accuracy of both models are reasonable, but far below human performance. Thus, we believe that such table-based fact verification can serve as an important new benchmark for the research towards the goal of building powerful AI that can reason over both soft linguistic form and hard symbolic forms. To facilitate future research, we will release all the data, code with its intermediate results. The contributions of the paper are summarized in three folds:

(i) we are the first to investigate the table-based fact verification problem and crowdsource a large-scale dataset to enable this research direction.

(ii) we propose two novel models to handle the mixed reasoning in the proposed dataset.

(iii) we perform in-depth analysis and demonstrate great future research opportunities.

2 Table Fact Verification Dataset

First, we follow the previous Table-based Q&A datasets (Pasupat and Liang, 2015; Zhong et al., 2017) to extract web tables (Bhagavatula et al., 2013) with captions from WikiTables\(^2\). Here we filter out overly complicated and huge tables (e.g. multirows, multicolumns, latex symbol) and obtain 18K relatively clean tables with less than 50 rows and 10 columns.

For crowd-sourcing jobs, we follow the human subject research protocols to hire Amazon Mechanical Turk\(^3\) workers from the native English-speaking countries “US, GB, NZ, CA, AU” with approval rates higher than 95% and more than

\(^{2}\text{http://websail-fe.cs.northwestern.edu/wikitables/about/}\)

\(^{3}\text{https://www.mturk.com/}\)
500 accepted HITs. The workers are paid with compensation above legal requirement. Following WikiTableQuestion (Pasupat and Liang, 2015), we provide the annotators with the corresponding table captions to help them better understand the background. To ensure the annotation quality, we develop a pipeline of “positive two-channel annotation” → “negative statement rewriting” → “verification”, as described below.

### 2.1 Positive Two-Channel Collection

To harvest statements of different difficulty levels, we design a two-channel collection process:

1. **Low-Reward Simple Channel**: the workers are paid 0.45 USD for annotating one Human Intelligent Task (HIT) that requires writing five statements. The workers are encouraged to produce plain statements meeting the requirements: (i) corresponding to a single row/record in the table without involving too much symbolic reasoning. (ii) mention the cell values without dramatic modification. The average annotation time of a HIT is 4.2 min.

2. **High-Reward Complex Channel**: the workers are paid 0.75 USD for annotating a HIT (five statements). They are guided to produce more sophisticated statements to meet the requirements: (i) involving multiple rows in the tables with higher-order semantics like argmax, argmin, count, difference, average, summarize, etc. (ii) rephrase the table records to involve more semantic understanding. The average annotation time of a HIT is 6.8 min.

The data obtained from the complex channel are harder in terms of both linguistic and symbolic reasoning, the goal of the two-channel split is to help us understand how far the proposed models can reach under different levels of difficulty.

### 2.2 Negative Rewriting Strategy

As suggested in (Zellers et al., 2018), there might be annotation artifacts and conditional stylistic patterns such as length and word-preference biases, which can allow shallow models (e.g. bag-of-words) obtain artificially high performance. Therefore, we design a negative rewriting strategy to minimize such linguistic cues or patterns. Instead of letting the annotators write negative statements from scratch, we let them rewrite the collected entailed statements. During the annotation, the workers are explicitly guided to modify the words, phrases or sentence structures but retain the sentence style/length. We disallow naive negations by adding “not, never, etc” to revert the statement polarity to prevent obvious patterns.

### 2.3 Quality Control

To control the quality of the annotation process, we review a randomly sampled statement from each HIT to decide whether the whole annotation job should be rejected during the annotation process. Specifically, a HIT must satisfy the following criteria to be accepted: (i) the statements should contain neither typos nor grammatical errors. (ii) the statements do not contain vague claims like “might”, “few”, etc. (iii) the claims should be explicitly supported or contradicted by the table without requiring additional knowledge, no middle ground is permitted. After the data collection, we re-distribute all the annotated samples to further filter erroneous statements, the workers are paid 0.05 USD per statement to decide whether the statement should be rejected. The criteria we apply is similar: no ambiguity, no typos, explicitly supported or contradictory. Through such post-filtering process, roughly 18% entailed and 27% refuted instances are further abandoned.

### 2.4 Dataset Statistics

**Inter-Annotator Agreement**: After the data collection pipeline, we merged the instances from two different channels to obtain a diverse yet clean dataset for table fact verification. We sample 1000 annotated (table, statement) pairs and re-distribute each to 5 individual workers to re-label them as either **ENTAILED** or **REFUTED**. We follow the previous works (Thorne et al., 2018; Bowman et al., 2015) to adopt the Fleiss Kappa (Fleiss, 1971)
Table 1: Statistics of the data collected from simple/complex channel and the division of Train/Val/Test Split. Pos/Neg are the averaged length of entailed/refuted statements, while Cell is the averaged number of words in each table cell.

| Channel | Sent  | Table | Length |
|---------|-------|-------|--------|
|         | Pos   | Neg   |        |
| Simple  | 50,378| 9,226 | 13.2   |
| Complex | 68,061| 7,395 | 14.2   |
| Total   | 118,439| 16,621| 13.8   |

### Dataset Statistics

As shown in Table 1, the amount of data harvested via the complex channel slightly outnumbers the simple channel, the averaged length of both the positive and negative samples are indistinguishable. More specifically, to analyze to which extent the higher-order operations are included in two channels, we group the common higher-order operations into 8 different categories. As shown in Figure 2, we sample 200 sentences from two different channels to visualize their distribution. We can see that the complex channel overwhelms the simple channel in terms of the higher-order logic, among which, count and superlatives are the most frequent. We split the whole data roughly with 8:1:1 into train, validation, and test splits and show their statistics in Table 1. Each table with an average of 14 rows and 5-6 columns corresponds to 2-20 different statements, while each cell has an average of 2.1 words.

In the training split, the positive instances slightly outnumber the negative instances, while the validation and test split both have rather balanced distributions over positive and negative instances.

### 3 Models

With the collected dataset, we now formally define the table based fact verification task: each instance \((T, S, L)\) consists of a table \(T\), a natural language statement \(S = s_1, \ldots, s_p\) and a verification label \(L \in \{0, 1\}\). The table \(T = \{T_{ij} | i \leq R_T, j \leq C_T\}\) has \(R_T\) rows and \(C_T\) columns with the \(T_{ij}\) being the content of the \((i, j)\)-th cell. \(T_{ij}\) could be a word, a number, a phrase or a natural language sentence. The statement \(S\) describes a fact to be verified against the content in the table \(T\). If it is entailed by \(T\), then \(L = 1\), otherwise the label \(L = 0\). Figure 1 shows some entailed and refuted examples. During training, the model and the learning algorithm are presented with \(K\) instances like \((T, S, L)_{k=1}^{K}\) from the training split. In the testing stage, the model is presented with \((T, S)_{k=1}^{K}\) and supposed to predict the label as \(\hat{L}\). We measure the performance by the prediction accuracy \(Acc = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}(L_k = \hat{L}_k)\) on the test set.

Before building the model, we first perform entity linking to detect all the entities in the statements. Briefly, we first lemmatize the words and search for the longest substring matching pairs between statements and table cells/captions, where the matched phrases are denoted as the linked entities. To focus on statement verification against the table, we do not feed the caption to the model and simply mask the phrases in the statements which links to the caption with placeholders. The details of the entity linker are listed in the Appendix. We describe our two proposed models as follows.

#### 3.1 Table Verification Network (TVNet)

In TVNet, we consider the table verification as a two-sequence binary classification problem like NLI or MPRC (Wang et al., 2018) by linearizing a table \(T\) into a sequence and treating the statement as another sequence. Since the linearized table can be extremely long surpassing the limit of sequence models like LSTM, Transformers, etc. We propose to shrink the sequence by only retaining the columns containing entities linked to the statement to alleviate such memory issue. In order to encode such sub-table as a sequence, we propose two different linearization methods, as is depicted in Figure 3. (i) Serialization: we simply concatenate the table cells with [SEP] tokens in between and restart position counter at the cell boundaries; the column name is fed as another type embedding to the input layer. Such design retains the table information in its machine format. (ii) Naturalization: we adopt simple templates to transform a table into a “somewhat natural” sentences. Taking the horizontal scan as an example, we linearize a table as “row one’s game is 51; the date is February; ... , the score is 3.4 (ot). row 2 is ... “. The isolated cells
are connected with punctuations and copula verbs in a language-like format.

After obtaining the linearized sub-table $\overline{\mathbf{T}}$, we concatenate it with the natural language statement $S$ and prefix a [CLS] token to the sentence to obtain the sequence-level representation $H = f_{BERT}((\overline{\mathbf{T}}, S))$, with $H \in \mathbb{R}^{768}$ from pre-trained BERT (Devlin et al., 2019). The representation is further fed into multi-layer perceptron $f_{MLP}$ to obtain the entailment probability $p_\theta(\overline{\mathbf{T}}, S) = \text{sigmoid}(f_{MLP}(H))$. We finetune the model $\theta$ (including the parameters of BERT and MLP) to minimize the binary cross entropy $CE(p_\theta(\overline{\mathbf{T}}, S), L)$ on the training set. At test time, we use the trained TVNet to compute the matching probability, and classify a (table, statement) pair as ENTAILED when $p_\theta(\overline{\mathbf{T}}, S)$ is greater than 0.5 otherwise as REFUTED.

### 3.2 Latent Program Algorithm (LPA)

In LPA, we formulate the table fact verification as a program synthesis problem, where the latent program algorithm is not given in TABFACT. In consequence, it becomes a weakly supervised learning problem as described in Liang et al. (2017); Lao et al. (2011). Under such a setting, we propose to break down the verification into two steps: (i) latent program search, (ii) discriminator selection. In the first program synthesis step, we adopt aim to parse the statement into LISP-like program format to represent its semantics. We define the plausible API set to include roughly 50 functions like $\text{min, max, count, average, filter, and}$ and realize their interpreter with Python2.7. Each API is defined to take arguments of specific types like number, string, bool and view (e.g. sub-table) to output specific-type variables. During the program execution, we store the generated intermediate variables to different-typed caches $\mathcal{N}, \mathcal{R}, \mathcal{B}, \mathcal{V}$ (Num, Str, Bool, View). At each execution step, the program can fetch the intermediate variable from the caches to achieve semantic compositionality. In order to shrink the search space, we follow NSM (Liang et al., 2017) to use trigger words to prune the plausible API set and accelerate the search speed. The definitions of all API, trigger words can be found in the Appendix. The comprehensive the latent program search procedure is summarized in Algorithm 1, and the searching procedure is illustrated in Figure 4.

After we collected all the potential program candidates $\mathcal{P} = \{(P_1, A_1), \ldots, (P_n, A_n)\}$ for a given statement $S$, we need to learn a discriminator to identify the “appropriate” traces from the set from many erroneous and spurious traces. Since we do not have the ground truth label about such discriminator, we use a weakly supervised training algorithm by viewing all the label-consistent programs as positive instances $\{P_i| (P_i, A_i) : A_i = L\}$ and the label-inconsistent program as negative instances $\{P_i| (P_i, A_i) : A_i \neq L\}$ to minimize the cross-entropy of discriminator $p_\theta(S, P)$ with the weakly supervised label. Specifically, we build our discriminator with a Transformer-based two-way encoder (Vaswani et al., 2017), where the statement encoder encodes the input statement $S$ as a vector $\text{Enc}^S(S) \in \mathbb{R}^{n \times D}$ with dimension $D$, while the program encoder encodes the program $P = p_1, \ldots, p_m$ as another vector $\text{Enc}^P(P) \in \mathbb{R}^{m \times D}$, we concatenate these two vectors and feed it into a linear projection layer to compute $p_\theta(S, P) = \text{sigmoid}(v_p^T[\text{Enc}^S(S); \text{Enc}^P(P)])$ as the matching probability between $S$ and $P$ with weight $v_p \in \mathbb{R}^D$. At test time, we use the discriminator $p_\theta$ to assign confidence $p_\theta(S, P)$ to each candidate $P \in \mathcal{P}$, and then either aggregate the prediction from all hypothesis with the confidence weights or rank the highest-confident hypothesis.

---

**Algorithm 1 Latent Program Search**

1: Initialize Number Cache $\mathcal{N}$, String Cache $\mathcal{R}$, Bool Cache $\mathcal{B}$, View Cache $\mathcal{V} \rightarrow \emptyset$
2: Push linked numbers, strings from the given statement $S$ into $\mathcal{N}, \mathcal{R}, \mathcal{B}$, and push $\overline{\mathbf{T}}$ into $\mathcal{V}$
3: Initialize the result collector $\mathcal{P} \rightarrow \emptyset$
4: Initialize an empty program trace $P = \emptyset$
5: Initialize the Queue $\mathcal{Q} = [(P, \mathcal{N}, \mathcal{R}, \mathcal{B}, \mathcal{V})]$
6: Use trigger words to find plausible function set $\mathcal{F}$
7: while loop over time $t = 1 \rightarrow 7$
   do:
   8: (P, N, R, B, V) = Q.pop() do:
   9: while loop over function $f \in \mathcal{F}$ do:
10: if arguments of $f$ are in the caches then
11: Pop out the required arguments $\text{args}$.
12: Return $A$ by executing $f(\text{args})$.
13: Concatenate the program trace $P$.
14: if Type$(A)$ = Bool then
15: if $\mathcal{N} = S$ = $\mathcal{B} = \emptyset$ then
16: $\mathcal{P}$.push((P, A))
17: else
18: $\mathcal{B}$.push($A$)
19: $\mathcal{Q}$.push((P, N, R, B, V))
20: if Type$(A)$ = $\{$Num, Str, View$\}$ then
21: if $\mathcal{N} = S$ = $\mathcal{B} = \emptyset$ then
22: break
23: else
24: push $A$ into $\mathcal{N}$ or $S$ or $\mathcal{V}$
25: $\mathcal{Q}$.push((P, N, R, B, V))
26: Return the triple $(\mathbf{T}, S, P)$

---

Confidential Review Copy. DO NOT DISTRIBUTE.
and use their outputs as the prediction.

4 Experiments

In this section, we aim to evaluate the proposed methods on TabFact. Besides the standard validation and test sets, we also split the test set into a simple and a complex partition based on the channel from which they were collected. This facilitates analyzing how well the model performs under different levels of difficulty. Additionally, we also hold out a small test set with 2K samples for human evaluation, where we distribute each (table, statement) pair to 5 different workers to approximate human judgments based on their majority voting, the results are reported in Table 2.

TVNet We build TVNet based on the open-source implementation of BERT\(^4\) using the pre-trained model with 12-layer, 768-hidden, 12-heads, and 110M parameters trained on 104 languages. We use the standard BERT tokenizer to break the words in both statements and tables into subwords and join the two sequences with a [SEP] token in between. The representation corresponding to [CLS] is fed into an MLP layer to predict the verification label. We finetune the model on a single TITAN X GPU with a mini-batch size of 6. The best performance is reached after about 3 hours of training (around 10K steps). We implement and compare the following variants of the TVNet model including (i) Serialize vs. Naturalize: whether to use natural language templates during linearization. (ii) Horizontal vs. Vertical: whether to scan the table horizontally or vertically.

LPA We run the latent program search in a distributed fashion on three 88-core machines with Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz and 256GB Memory to generate the latent programs. The search terminates once the buffer has more than 100 traces or the step size is larger than 7. The average search time for each statement is about 2.5s. For the discriminator model, we design two transformer-based encoders (3 layers, 128-dimension hidden embedding, and 4 heads at each layer) to encode the programs and statements, respectively. The variants of LPA models considered include (i) Voting: assign each program with equal weight and vote without the learned discriminator. (ii) Weighted-Voting: compute a weighted-sum to aggregate the predictions of all latent programs with the discriminator con-

---

\(^4\)https://github.com/huggingface/pytorch-pretrained-BERT
Preliminary Evaluation In order to test whether our negative rewriting strategy eliminates the artifacts or shallow cues, we also fine-tune a pre-trained BERT (Devlin et al., 2019) to classify the statement $S$ without any table information. The result is reported as “BERT classifier w/o Table” in Table 2, which is approximately the majority guess and reflects the effectiveness of the rewriting strategy. Before presenting the experiment results, we first perform a preliminary study to evaluate how well the entity linking system, program search, and the statement-program discriminator perform. Since we do not have the ground truth labels for these models, we randomly sample 100 samples from the dev set to perform the human study. For the entity linking, we evaluate the precision of correctly linked entities, and the recall of entities which should be linked. For latent program search, we evaluate whether the “true” programs are included in the candidate set $\mathcal{P}$ and report the recall score. For discriminator, under the cases where the “true” program lies in the candidate set, we use the trained model to select the top K hypothesis and calculate the HITS@K accuracy (the chance of correct program being included in the top K candidates). Please note that the discriminator can also select a spurious program which happens to obtain the same label as ground truth, but this does not count as a hit. These preliminary case study results are reported in Table 3.

Results We report the performance of different methods as well as human performance in Table 2.

| Model                                      | Val | Test | Test (simple) | Test (complex) | Small Test (Mixed) |
|--------------------------------------------|-----|------|---------------|----------------|--------------------|
| Majority Guess                             | 50.7| 50.4 | 50.8          | 50.0           | 50.3               |
| BERT classifier w/o Table                  | 50.9| 50.5 | 51.0          | 50.1           | 50.4               |
| TVNet-Horizontal-F+T-Serialize             | 50.7| 50.4 | 50.8          | 50.0           | 50.3               |
| TVNet-Vertical-F+T-Naturalize              | 56.7| 56.2 | 59.8          | 55.0           | 56.2               |
| TVNet-Vertical-T+T-Naturalize              | 56.7| 57.0 | 60.6          | 54.3           | 55.5               |
| TVNet-Horizontal-F+T-Naturalize            | 66.0| 65.5 | 79.4          | 58.8           | 68.0               |
| TVNet-Horizontal-T+T-Naturalize            | 66.7| 66.1 | 80.2          | 59.7           | 68.2               |
| LPA-Voting w/o Discriminator              | 57.7| 58.2 | 68.5          | 53.2           | 61.5               |
| LPA-Weighted-Voting                        | 62.5| 63.1 | 74.6          | 57.3           | 66.8               |
| LPA-Ranking                                | 64.6| 64.4 | 76.9          | 58.2           | 67.3               |
| Human Performance                          |      |      |               |                | 92.1               |

Table 2: The results of all proposed models, the numbers are reported in percentage. T+F means table followed by fact, while F+T means fact followed by table.

Confidence as the weights. (iii) Ranking: rank all the hypotheses by the discriminator confidence and use the top-rated hypothesis as the output.

First of all, we observe that the naive serialized model fails to learn anything effective (same as the Majority Guess). It reveals the importance of naturalization when using the pre-trained BERT (Devlin et al., 2019) model: the “natural” connection words between individual cells is able to unleash the power of the large pre-trained language model and enable it to perform reasoning on the structured table form. Such behavior is understandable given the fact that BERT is pre-trained on purely natural language corpora. In addition, we also observe that the horizontal scan excels the vertical scan because it better captures the convention of human expression. Among different LPA methods, we found that LPA-Ranking performs the best since it can better suppress the spurious programs than the voting-based algorithm. As suggested in Table 3, the current LPA method is upper bounded by 80% (recall of “true” program hypothesis), but the real accuracy (64%) is still far from that. Diving into specific cases to examine the performance of discriminator, we found that only 28% “true” programs are ranked at the top Table 3. We hypothesize that the weakly supervised learning of the discriminator is the main bottleneck for LPA. By comparing the performance of simple-channel with complex-channel split, we observe a significant accuracy drop (≈ 20%), which reveals the weakness of existing models in dealing with

| Model | HITS@1 | HITS@3 | HITS@5 |
|-------|--------|--------|--------|
| LPA-Ranking | 28     | 33     | 41     |

Table 3: Case Study results on different components.
higher-ordered semantics.

Overall, the TVNet is able to achieve slightly better performance. However, we observe that the model exhibits large prediction variance, as it can miss some very simple cases but hit super hard test cases. The lack of interpretability and stability in TVNet becomes the major weaknesses of this model. In contrast, LPA behaves much more consistently and provides a clear latent rationale for its decision. But, such a pipeline system requires laborious handcrafting of API operations and is also very sensitive to the entity linking accuracy. Both methods have merits and weaknesses; how to combine the strengths of these two models become an exciting research direction.

5 Related Work

Natural Language Inference  Modeling reasoning and inference in human language is a fundamental and challenging problem towards true natural language understanding. There has been extensive research on RTE in the early years (Dagan et al., 2005) and more recently shifted to NLI (Bowman et al., 2015; Williams et al., 2017). NLI seeks to determine whether a natural language hypothesis $h$ can be inferred from a premise $p$. With the surge of deep learning, there have been many powerful algorithms like the Decomposed Model (Parikh et al., 2016), Enhanced-LSTM (Chen et al., 2017) and BERT (Devlin et al., 2019). Our proposed fact verification task is also closely related to NLI, where our semi-structured table can be seen as a collection of “premises” stored in a structured format. Our proposed problem hence could be viewed as the generalization of NLI under the structured domain.

Table Question Answering  Another line of research closely related to our task is the table-based question answering, such as WikiTable-Question (Pasupat and Liang, 2015), Sequential Q&A (Iyyer et al., 2017), and WikiSQL (Zhong et al., 2017), for which approaches have been extended to handle large-scale resources like Wikipedia (Bhagavatula et al., 2013). However, in these Q&A tasks, the question types typically provide strong signals needed for identifying the type of answers, while TABFACT does not provide such specificity. Moreover, TABFACT involves stronger rephrasing from the semi-structured table contents: the statements usually reformatulate the original cell values to fit them smoothly into a human-readable natural language sentence. For example, if the cell contains more than one entries like “john, peter”, people usually translate them into “peter and john”, or “john ... with peter”. Such common rephrasing makes the semantic matching (entity linking) much more challenging than Q&A. Furthermore, TABFACT has One-to-Many mappings; statements might not be explicitly transformed into one Q&A pair. For example, a sentence like “Legace played for the St. Louis Blues against Anaheim at home on February 22nd, in the next year, and the record between the two teams becomes 24/19.” is hard to be parsed into one Q&A pair, rather it can correspond to different possibilities like “Who plays for St. ..... 24-19-?” or “Which team does Legace play for ...?”, etc. These factors greatly increases fact verification’s difficulty in natural language understanding.

Program Synthesis for Q&A  There have also been great interests in using program synthesize or logic forms to solve the table question answering problem. They aim to retrieve answers by synthesizing programs and executing them on the tables, such as in Neural Programmer (Neelakantan et al., 2016, 2017) and Neural Symbolic Machines (Liang et al., 2017, 2018; Agarwal et al., 2019). Compared to the table-based Q&A, our proposed TABFACT exhibits even more challenging spurious programs (Berant et al., 2013; Pasupat and Liang, 2015) (i.e., wrong programs with the true returned answers) issue due to the fact that the program can only return binary values, which can easily misguide the policy searching using standard reinforcement learning. How to resolve such extremely under-specified binary rewards in NLP domain becomes an interesting direction to pursue. Besides, the previously defined API sets are not enough because the verification task requires additional binary-type operations, which also greatly enlarges the search space.

6 Conclusion

This paper investigates a very important yet previously unexplored research problem: semi-structured fact verification. We construct a large-scale dataset and proposed two methods, TVNet and LPA, based on the state-of-the-art BERT model and program synthesis. In the future, we plan to push forward this research direction by inspiring more sophisticated architectures which can perform both linguistic and symbolic reasoning.
References

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to generalize from sparse and underspecified rewards. *International Conference of Machine Learning*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2013. Methods for exploring and mining tables on wikipedia. In *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, pages 18–26. ACM.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced lstm for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668.

Gennaro Chierchia and Sally McConnell-Ginet. 2000. *Meaning and grammar: An introduction to semantics*. MIT press.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*.

Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378.

Andreas Hanselowski, Hao Zhang, Zile Li, Danil Sorokin, Benjamin Schiller, Claudia Schulz, and Iryna Gurevych. 2018. Ukp-athene: Multi-sentence textual entailment for claim verification. *arXiv preprint arXiv:1809.01479*.

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

Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. 2016. Tables as semi-structured knowledge for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 474–483.

Jerrold K Katz and Jerry A Fodor. 1963. The structure of a semantic theory. *language*, 39(2):170–210.

Ni Lao, Tom Mitchell, and William W Cohen. 2011. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 529–539. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *International Conference of Machine Learning*.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems*, pages 9994–10006.

Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. *International Conference on Learning Representation*.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. *International Conference on Learning Representation*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1470–1480.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237.

Kashyap Popat, Subhabrata Mukherjee, Jannik Strögen, and Gerhard Weikum. 2017. Where the truth lies: Explaining the credibility of emerging claims on the web and social media. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 1003–1012. International World Wide Web Conferences Steering Committee.
Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1:8.

James Thorne, Andreas Vlachos, Christos Christoudouloupolous, and Arpit Mittal. 2018. Fever: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 809–819.

Johan Van Benthem et al. 2008. *A brief history of natural logic*. London: College Publications9781904987444.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *EMNLP 2018*, page 353.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. Swag: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104.

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

1