University of California
Santa Barbara

Learning Natural Language Interfaces using Deep Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Izzeddin Gur

Committee in charge:

Professor Xifeng Yan, Chair
Professor Ambuj Singh
Professor William Wang

September 2019

The Dissertation of Izzeddin Gur is approved.

Professor Ambuj Singh

Professor William Wang

Professor Xifeng Yan, Committee Chair

June 2019

Learning Natural Language Interfaces using Deep Neural Networks

To my parents, sisters, and my wife for their unconditional love and support.

Acknowledgements

First and foremost, I want to thank my advisor Professor Xifeng Yan. Xifeng supported me throughout my Ph.D. journey in UCSB by giving me invaluable advises in academia, in industry, and in real life. He provided me with excellent mentorship, enabled me to understand and grow myself, guided me towards what is the most valuable not only in conducting research but also in keeping good relationships with my peers. Xifeng inspired me by his passionate and hardworking personality for which and for dedication of his time, experience, and ideas I am endlessly grateful.

I would like to thank my committee members, Professor Ambuj Singh and Professor William Wang, for their great guidance on my research and my professional life after graduation. I am grateful to have them in my committee as they have always encouraged me to understand my potential. I am also very grateful to Professor Amr El Abadi, Professor Shiv Chandrasekaran, Professor Kenneth Rose, Professor Stefano Tessaro, Professor Tevfik Bultan, and Professor Matthew Turk for giving me different perspectives in my Ph.D. career. In addition, I would like to thank to the academic staff: Benji Dunson, Nicole McCoy, Karen van Gool, Simran Singh, who have provided me enormous support every time I needed. I would like to also thanks my lab peers, especially Semih Yavuz, Yu Su, Honglei Liu, Shengqi Yang, Bo Zong, and Fangqiu Han for insightful discussions on machine learning and collaboration. Also, Wenhu Chen, Keqian Li, Xiaoyong Jin, Hanwen Zha, Hongmin Wang, Zhiyu Chen, Xiyou Zhou, Theodore Georgiou, Huan Sun, Shulong Tan, and Dong-Anh Nguyen for all the research discussions and activities. Furthermore, I would like to express my gratitude to all the funding agencies that supported me during my studies including Computer Science Department at UCSB and Army Research Lab.

Additionally, I have had the opportunity to collaborate with a lot of valuable re-

searchers during my internships. I would like to thank Dr. Dilek Hakkani-Tur who has given me invaluable guidance throughout my internships which led to impactful research. I want to express my thanks to Dr. Gokhan Tur from whom I learned how to gain a broader perspective. I would like to thank Dr. Aleksandra Faust who has guided me not only during my research but also after in my industry life. I am also grateful to Ulrich Rueckert who gave me practical advice during my internship. I want to express my thanks to Dr. Daniel Hewlett for introducing me to research at Google. I would like to extend my thanks to David Berthelot, Abhinav Rastogi, Ankur Bapna, Jindong Chen, Nevan Wichers, Raghav Gupta, Amir Fayazi, Aleksandre Lacoste, Thyago Duque, Bing Liu, Ondej Klejch, Rama Kumar Pasumarthi, Shyam Upadhyay, Pararth Shah, and many others for all the research discussions and fun we had together.

This thesis would not have been possible without the support and encouragement of my families. I want to thank my dad and mom, Ebubekir and Kamile, and my sisters Busra, Ayse, and Fatma. Even though, I spent most of my time away from home studying, they always supported me in every decision and aspect of my life and made me who I am today. Lastly, I would like to thank my wife Kubra for her love, energy, comfort, and constant support. She is my biggest motivation, my best friend, and my inspiration and I admire her for her modesty, kindness, and intelligence.

Curriculum Vitæ
Izzeddin Gur

Education

2019	Ph.D. in Computer Science, University of California, Santa Barbara.
2013	M.S. in Computer Science, Bilkent University.
2012	B.S. in Mathematics, TOBB Economy and Technology University.
2011	B.S. in Computer Engineering, TOBB Economy and Technology University.

Publications

1. Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, Dilek Hakkani-Tur, Learning to Navigate the Web, ICLR '2019 (International Conference on Learning Representations).
2. Chantal Nguyen, Kimberly J Schlesinger, Fangqiu Han, Izzeddin Gur, Jean M Carlson, Modeling individual and group evacuation decisions during wildfires, Fire Technology '2019.
3. Izzeddin Gur, Dilek Hakkani-Tur, Gokhan Tur, Pararth Shah, User Modeling for Task Oriented Dialogues, SLT '2018 (IEEE Spoken Language Technology Workshop (SLT))
4. Izzeddin Gur, Semih Yavuz, Yu Su, Xifeng Yan, Dialsql: Dialogue based structured query generation, ACL '2018 (Annual Meeting of the Association for Computational Linguistics)
5. Semih Yavuz, Izzeddin Gur, Yu Su, Xifeng Yan, What It Takes to Achieve 100% Condition Accuracy on WikiSQL, EMNLP '2018 (Conference on Empirical Methods in Natural Language Processing).
6. Yu Su, Honglei Liu, Semih Yavuz, Izzeddin Gur, Huan Sun, Xifeng Yan, Global relation embedding for relation extraction, NAACL '2018 (Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies).
7. Semih Yavuz, Izzeddin Gur, Yu Su, Xifeng Yan, Recovering question answering errors via query revision, EMNLP '2017 (Conference on Empirical Methods in Natural Language Processing).
8. Izzeddin Gur, Daniel Hewlett, Llion Jones, Alexandre Lacoste, Accurate supervised and semi-supervised machine reading for long documents, EMNLP '2017 (Conference on Empirical Methods in Natural Language Processing).
9. Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, Xifeng Yan, On generating characteristic-rich question sets for qa evaluation, EMNLP '2016 (Conference on Empirical Methods in Natural Language Processing).

10. Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, Xifeng Yan, Improving semantic parsing via answer type inference, EMNLP '2016 (Conference on Empirical Methods in Natural Language Processing).
11. Izzeddin Gur, Mehmet Guvercin, Hakan Ferhatosmanoglu, Scaling forecasting algorithms using clustered modeling, VLDB Journal '2015 (The International Journal on Very Large Data Bases).

Abstract

Learning Natural Language Interfaces using Deep Neural Networks

by

Izzeddin Gur

Automating user tasks with natural language utterances, such as answering questions over Wikipedia or booking flight tickets on the Web, is a key component in designing intelligent systems. Natural language is usually preferred as a unified interface for these systems and requires no domain expertise for users; however, understanding wide range of diverse inputs and resolving errors that occur during this process are still open challenges and the topics of this thesis.

Traditional machine learning systems for natural language interfaces usually require large-scale labeled datasets with handcrafted rules to train and evaluate the performances of the respective models. Firstly, the handcrafted design constrains the scope to a limited set of domains and prevents the adaptation to new tasks. Additionally, large-scale labeled data collection is generally domain dependent, costly, and time consuming. These systems further assume that the underlying database, such as Freebase, is accessible and can be queried indefinitely which is prohibitive when learning from constrained user interfaces, such as Web pages. Last but not least, current systems focus on training offline in a closed loop where users are excluded from the system inference process. They lack the capabilities to continuously learn from users.

In this thesis, we address the drawbacks of the existing systems and propose data efficient and user-centric solutions. We classify the natural language inference problem based on two different perspectives: Accessiblity of the system functions – unconstrained or constrained user interfaces, and nature of user involvement during inference – non-

interactive or interactive user interfaces. We first develop neural network based systems for non-interactive and unconstrained users interfaces with different data types (i.e. structured and unstructured). The system is trained to learn a continuous representation of user utterance, generate and rank candidate answers from underlying database using this representation. We augment these systems with an extractive candidate refinement framework by integrating task-oriented human-machine dialogues. Our system is able to understand, point, and refine the error in candidates by asking users validation questions and offering alternatives. We also address the limitations of unconstrained user interfaces and propose reinforcement learning methods to develop policies that are capable of learning from more constrained web interfaces. The policies are trained on a variety of web pages, such as flight booking and social media interaction, with task-based reward signals and no human supervision. We test the performance of our models with simulated as well as real users. Empirical results show that the proposed models are able to learn from limited supervised data and have successful dialogues with users. We observe improvements in answer prediction accuracy, task success rate, and real user ratings.

Contents

Chapter 1

Introduction

1.1 Motivation

Building natural language interfaces that enable users to communicate with complex systems to complete tasks without any domain expertise is one of the most prominent and challenging problems in Artificial Intelligence. Before we delve into tackling these challenges, we need to understand the scope of the problems. Consider the following set of sample tasks that users face frequently.

- **question answering.** Take the following question that a user wants to answer from Wikipedia: *How many states are there in United States?* One way for our system to answer this question is to understand that *United States* points to the Wikipedia document, *states* is the search token, and answer should be a number. By searching the document with the corresponding fields, the answer should be extracted and returned.

- **assistant interaction.** Consider the following instruction that a user issues to an intelligent assistant: *Set up an alarm for 9:30am in the morning.* Our system first needs to detect the actual action *set up alarm* and the time *9:30am*; then, call the corresponding low level functions to set the alarm.

- **web page navigation.** Let's assume that a user wants to *Book a flight ticket from SFO to NYC on June, 24* on an airline website. Extracting the departure and destination airports with the date of the flight is the first job of the system. Next, it is to navigate the airline website with the extracted fields and complete the task.

We argue that **natural language** can serve as a unified interface for all these different tasks. It can simplify the design and complexity of the system by fusing various types of sensory inputs into a single and general form of input. Natural language interfaces offer users a holistic view of the system, reducing the effort to learn and adapt to new systems.

In this thesis, we are interested in learning **natural language interfaces** — understanding a user utterance as a form of representation and reflecting this representation on the underlying system as a sequence of write and read actions. An utterance can be represented as a continuous vector generated from a machine learning model or a symbolic representation such as a LISP program. This representation can be converted into a write or read action to alter the state of the system or return a response to the user. In the web page navigation task above, the utterance can be represented as a list of key and value pairs where each pair reflects an argument, such as {*date: "June, 24"*}, or an action such as {*action: "inform"*}. Guided by the utterance representation, the system can navigate the airline website and reserve a flight ticket. Particularly, we will focus on using deep neural networks, a powerful class of function approximators which are shown to perform superior across a range of tasks, to learn natural language interfaces.

This thesis is centered around the following classification of natural language interfaces — unconstrained interfaces, the database (structured or unstructured) and the querying language are accessible and constrained interfaces, the database can be accessed

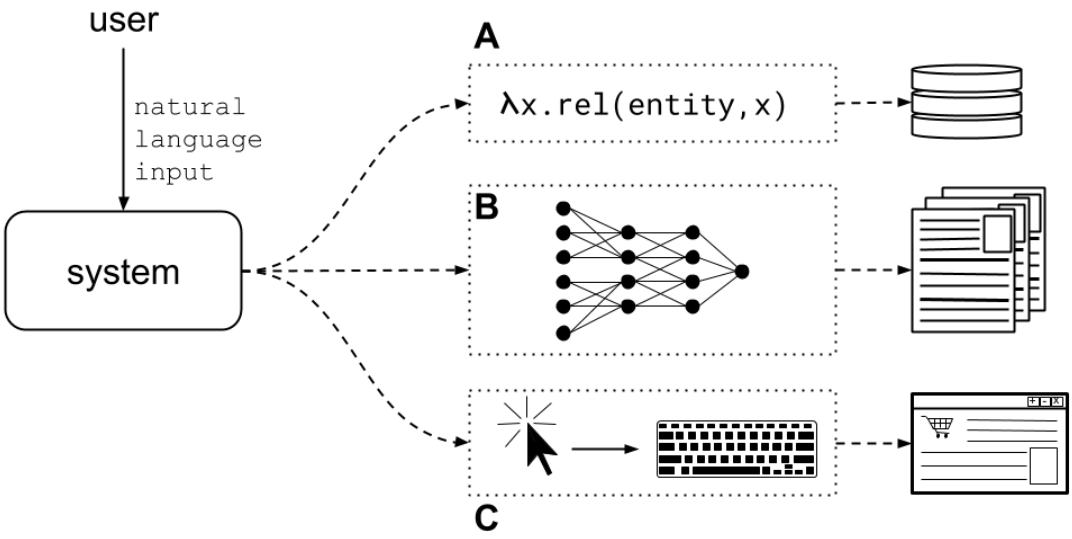


Figure 1.1: An overview of natural language interface types studied in this thesis. (A) denotes a semantic parsing based representation of utterance with structured database, (B) is a neural network based representation with unstructured documents, and (C) is a constrained interface via interaction with a web page by mouse and keyboard.

by the limited functionality of the user interface. Former set of tasks assumes that the underlying database can be observed by executing arbitrary number of queries; while, in the latter, user interface forms a restricted view of the database by executing only a limited set of queries. Setting up an alarm task, where the system is able to insert an entry to the underlying database by forming a query from user specified fields, is an example of unconstrained interfaces. As an example to constrained interfaces, we can give the flight booking task where our system is only able to interact with the database by navigating the website. In addition to the previous perspective of natural language interfaces, we can also classify based on the nature of user interaction — non-interactive interfaces, where system inference process is uninterrupted by user, and interactive interfaces, where users are involved in the process. Our classification of a general natural language interface is illustrated in Figure 1.1.

1.2 Thesis Outline

Based on the above classification, in this thesis, we study non-constrained and constrained natural language interfaces that are non-interactive or interactive.

In Chapter 2, we focus on learning from structured databases, such as Freebase. In Section 2.1, we first give a brief introduction to semantic parsing problem and discuss recent approaches. Next, we present a hierarchical neural model for encoding semantic parse structures and a ranking model. Section 2.2 introduces a new dialogue-based structured query generation problem. We first discuss the labeled data collection bottleneck and give a user simulation approach for the query generation. Then, we propose a hierarchical neural dialogue model that learns to refine candidate SQL queries by multi-turn conversations with humans. To alleviate the first problem, labeled data collection, we propose a range of neural user simulators in Section 2.3. We inject diversity and goal-oriented models by integrating neural variational inference and a new goal regularization approach. This chapter is based on our works [40], and [39].

In Chapter 3, we present neural models for learning natural language interfaces from unstructured databases, such as Wikipedia. We first give an overview of the reading comprehension problem for long document texts. Next, we give supervised neural approaches for long document representations. Finally, we propose novel semi-supervised neural networks that learns better models when the labeled data is scarce. This chapter is based on our work [38].

In Chapter 4, we focus on learning from constrained interfaces, more specifically web interfaces. We first introduce the instruction following on web pages problem and discuss the shortcomings of a baseline reinforcement learning policy. Next, we propose guided policy learning with curriculum learning and meta-learning methods which are built on top of our baseline model and substantially improve the baseline. This chapter is based

on our work [41].

Finally, in chapter 5, we propose a set of models for understanding neural model predictions. We first introduce the neural indicators, the most important input tokens in a sequence based on confidence of the output prediction. Then, we propose two approaches based on statistical uncertainty to rank indicators. These indicators are used to understand neural predictions as well as to regularize the neural network.

1.3 Contributions

In this thesis, we make the following contributions,

- We are one of the first to study the neural semantic parsing and neural reading comprehension problems. In particular, our neural models for reading comprehension achieved state-of-the-art performance and showed superior results when labeled data is very scarce.

- We introduced the dialogue-based structured query generation problem. We developed hierarchical neural models to illustrate that by human-machine conversation, query performance can be improved.

- Our web page navigation works are among the first to pioneer the research in learning from constrained interfaces domain. We proposed a novel meta-learning framework that achieves superior performance and broadly applicable to other domains.

- We made an attempt at understanding neural network predictions. Our statistical uncertainty based approach to the problem showed that token level inputs, the most effective when making predictions, can be successfully detected.

Chapter 2

Learning from Structured Data

In this chapter, we explain deep neural models equipped with the capability to map natural language utterances into structured queries which then can be executed against a given database to retrieve the desired answer.

In Section 2.1, we describe semantic parsing on Freebase problem and develop a neural semantic parsing framework in Section 2.1.2. Our framework consists of a joint utterance and logical form encoder and a ranking module which outputs the highest scoring logical form. We detail our hierarchical logical form encoder that incorporate the structure of the logical forms without handcrafted feature engineering and discuss the scoring function and logical form ranking approach in the end.

Next, in Section 2.2, we introduce our interactive query generation problem. We discuss a rule-based user simulator for structured query generation that interacts with our models to supply labeled dialogue data in Section 2.2.3. A novel hierarchical task-oriented dialogue model that extracts precise errors in candidate queries and refines by validating with users via multiple turns is proposed in Section 2.2.4.

Finally, we tackle modeling users in task-oriented dialogue systems in Section 2.3. In Section 2.3.4, we propose hierarchical neural user simulator models that can generate diverse user responses using variational inference. We develop a new goal regularization approach that penalizes models if the generated user responses diverge from the user

goal.

2.1 Semantic Parsing on Freebase

In recent years, an array of question answering methods have been proposed for knowledge bases. Some derive logical forms from the input question using a grammar [54, 11, 10], while some others directly construct logical forms from a KB, in the form of query graphs [119]. Despite of these progresses, the challenge of factoid question answering remains open, especially for complex questions involving more than one relation. Figure 2.1 illustrates a question requiring multiple KB relations to represent. The path from `ViggoMortensen` to ANSWER is the core relation representing all the characters starred by `ViggoMortensen`, and the other path `Film.LoTR` serves as an auxiliary constraint, limiting the answer to the character that `ViggoMortensen` starred in `LoTR`. For this particular example, AgendaIL [10] is able to correctly identify the core relation, `ViggoMortensen.Film.Character`, but has trouble recognizing the auxiliary constraint.

We aim to develop an end-to-end neural model for factoid question answering, which can capture the *holistic* structure of multi-relation questions and query graphs (logical forms) with minimal feature engineering. Our goal is to embed both the input question and its corresponding query graphs into the same vector space, such that the embedding of the correct query graph is the closest to the embedding of the question. Fig. 2.1 sketches our model, which consists of two encoder neural networks. The first encoder (left), instantiated as a Long Short-Term Memory (LSTM) model, maps a question into a fixed-length vector. The second encoder (right) is a hierarchical LSTM. It first encodes *auxiliary relations* (S) and then combines the local embeddings with the *core relation* (R) to get a global embedding of the entire query graph. By recursively encoding both

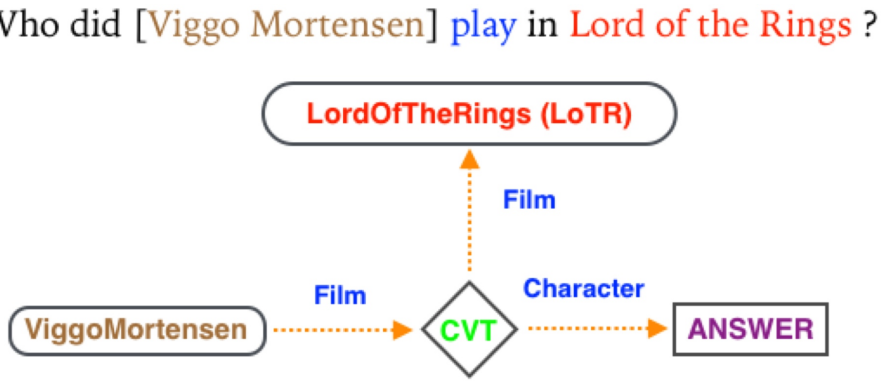


Figure 2.1: An example question and the corresponding query graph. Each node represents a real world entity or a hyperedge called *compound value type* (CVT) connecting multiple entities together.

auxiliary and core relations, our model is able to fully exploit the hierarchy of a query graph. We develop a max-margin objective function to jointly train the two encoders, such that the embedding of the correct query graph is closer to the embedding of the input question than anyone else.

2.1.1 Related Work

Factoid Question Answering. As a promising method for QA, many semantic parsing techniques have been proposed recently, which utilize combinatory categorical grammar (CCG) [13, 54, 80, 18], Dependency-based Compositional Semantics (DCS) [62, 11, 9, 10], and query graphs [7] for aligning questions with entities and relations in knowledge bases. Vector space embedding approaches are also emerging [12, 112], which jointly embed questions and KB subgraphs to measure their semantic similarity.

Our method is mostly related to graph-centric and vector space encoding based approaches. [117] attacks the problem from an IE perpective and learns patterns of question and answer pairs. They utilize a graph perspective by representing both question and candidate answers as graphs based on dependency parse tree of question and Freebase graph structure of answers, respectively. Features from question graph and Freebase

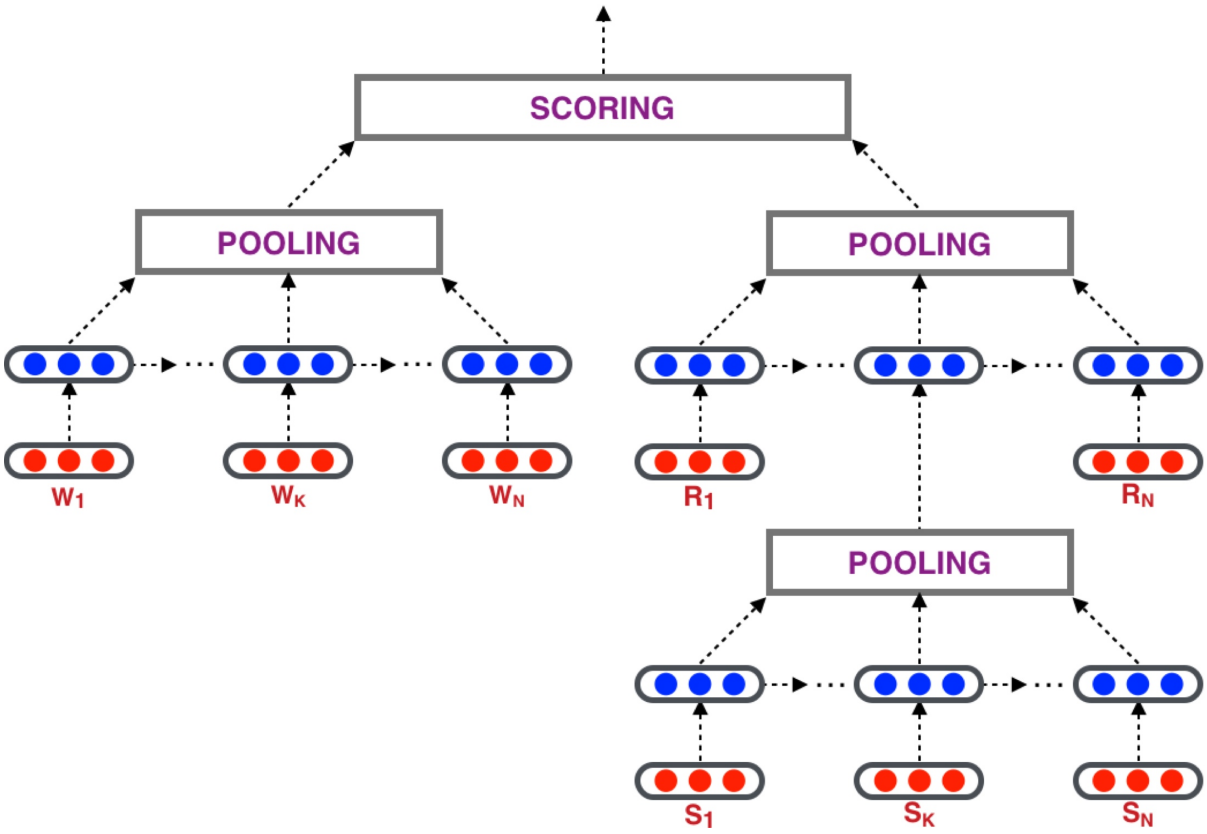


Figure 2.2: Our joint embedding model for question (left) and logical form (right) pairs. W is a question. R and S are the corresponding relation mappings in KB. R is a core relational path and S is an auxiliary path. There could be multiple auxiliary paths.

graph are then combined per node by pairwise concatenation. This captures the associations between question patterns and answer nodes. Similar to [117], [80] converts the question into a graph representation, exploiting CCG parse instead of dependency parse, which is then grounded onto Freebase. This way, the problem reduces to graph matching problem where the objective is to identify which Freebase graph best corresponds to ungrounded NL graph. On the other hand, [12] uses an embedding method to represent both question and candidate answers in the same low dimensional vector space. Similarity between questions and candidate answers are measured in this vector space. Similar to high-level approach in [12], [61] is the first method to use neural network for encoding

questions into the same vector space as embeddings of candidate answers. While embedding of the answer-related Freebase subgraphs resembles the approach in [12], [61] uses multi-column Convolutional Neural Network (CNN) for encoding the natural language question. [109] uses a CNN based approach to extract relations and retrieve the candidate answers from Freebase. In addition, they introduce a validation step that prunes incorrect answers based on evidence from Wikipedia.

The work closest to ours is Yih et al. [119], in which a CNN is used to score question-query graph pairs. Instead of extracting features for compositional semantics, our model takes the holistic structure of query graphs as input. Yih et al. [119] use deterministic rules to augment the core query graph with constraints. In contrast, we systematically augment the query with a large set of candidate relations. We generate much richer logical forms that span a broader range of semantics by leveraging *type constraints*, *auxiliary relations*, *dates*, etc. Our model directly utilizes all the relations in a query graph, which allows to integrate constraints in a more systematic way.

LSTM Models. LSTM models have been shown to succeed at learning sequential patterns in machine translation [65], handwriting recognition [34], and language modeling [76], among others. Our model bears a resemblance to Tree-LSTM models [98], which generalize the LSTM model to tree-structured data. The main distinction is that our model is tailored for a core-auxiliary structure, while Tree-LSTM deals with general tree structures. The information from different layers of a tree is processed using the same set of parameters in Tree-LSTM. In contrast, we use different parameters for core and auxiliary relations, which helps to differentiate core and auxiliary information. Moreover, Tree-LSTM combines the cell states of children to generate a new cell state. In our model, we use auxiliary path embeddings as inputs to the upper layer. The model can then be more focused on the core relation, and changes in auxiliary relations will have a less direct impact on the final cell state. For example, in Figure 2.2, a change on S_k will have

a smaller impact than a change on the core path, $R_1, \ldots, R_N$.

2.1.2 Question Answering with Hierarchical LSTM

Our question answering framework consists of three steps: (1) Detecting and linking entity mentions, (2) generating query graphs, and (3) learning a model to rank query graphs. Yih et al. [119] show that a better entity detection/linking algorithm can improve F1 score by a substantial margin (4.1% absolute improvement in their case). This work is focused on Steps 2 and 3.

Query Graph Generation

Core Relational Path. Given the topic entity T_e identified in $\mathcal{K}$, we generate candidate paths starting at T_e using a breadth-first search (BFS). To generate a manageable set of paths, we restrict the length of the path to 2 if there is a CVT node, otherwise to 1. The core paths will later be augmented with other auxiliary relations. For the question, "Who did Viggo Mortensen play in Lord of the Rings?" the topic entity mention is "Viggo Mortensen", which is linked to `ViggoMortensen` in $\mathcal{K}$. We then generate candidate paths using a BFS, e.g., `Film.Character`, `StarringRoles.Character`, and `Film.Film`. Out of these candidate paths, only `ViggoMortensen.Film.Character` gives the correct answer and captures the main relation in the question. We aim at ranking correct paths higher to produce a set of high quality paths for query augmentation. Before delving into the details of our ranking model, we first describe the augmentation of the initial core paths with auxiliary relations.

Query Augmentation. The initial query path of a given question is likely to capture the core relation and can be executed against a KB to retrieve candidate answers. In our example, the initial query path `ViggoMortensen.Film.Character` is going to return

all the characters that `ViggoMortensen` played in. Although the set of characters will include the answer `Aragorn`, the precision of this set will be small. To further restrict the answer set and interpret the question better, we augment the initial query graph by attaching additional KB paths. Starting from the nodes in the initial query path, we can add other paths reachable in the KB. These new paths are called *auxiliary paths*, related to other relations and constraints mentioned in the question. To reduce the search space of auxiliary paths, we only consider one-hop relations from the CVT and ANSWER nodes. The relation from the ANSWER node will be restricted to `Type` and `Gender`. For instance, in Figure 2.2, our query augmentation process returns the following auxiliary paths {`Film.TheLordOfTheRings`, `Film.TheIndianRunner`} from the CVT node and {`Type.BookCharacter`, `Type.OperaCharacter`} from the ANSWER node. Finally, we rank the resulting augmented candidate queries to attain the correct answer.

Long Short-Term Memory with Pooling

We propose a neural network architecture that learns to encode input questions and query graphs into fixed-length vectors. We extend the traditional LSTM model to our "core-auxiliary" structure. Representations of auxiliary paths are combined with the core relation path to obtain a global embedding of the entire query. In the following, we briefly introduce our implementation of LSTM.

LSTM is a kind of recurrent neural networks. It introduces several gates that control information flow and a memory cell that carries the long-term information throughout the sequence. Given the current input vector x_t, the previous hidden vector h_{t-1}, and

the previous cell state c_{t-1}, the new cell state and hidden vector are estimated as follows,

$$
\begin{aligned}
X_t &= [x_t; h_{t-1}] && (2.1)\\
i_t &= \sigma(W_i X_t + b_i) && (2.2)\\
f_t &= \sigma(W_f X_t + b_f) && (2.3)\\
o_t &= \sigma(W_o X_t + b_o) && (2.4)\\
z_t &= tanh(W_z X_t + b_z) && (2.5)\\
c_t &= f_t * c_{t-1} + i_t * z_t && (2.6)\\
h_t &= o_t * tanh(c_t), && (2.7)
\end{aligned}
$$

where i_t, f_t, o_t, and z_t are input, forget, output, and block input gates, respectively. σ is the sigmoid function which controls the information flow for each dimension. W and b matrices are model parameters to learn. We add a "pooling" layer on top of the hidden vectors :

$$\hat{h} = POOLING(h_1, \ldots, h_l), \tag{2.8}$$

where l is the length of the sequence, and $POOLING$ is a generic function that operates on each dimension independently and returns a single vector $\hat{h}$ as a result. We empirically found that AVERAGE pooling works better for our problem.

Given input vectors $\{x_1, \ldots, x_l\}$ for each word in the question, we represent the resulting output vector $\hat{h}$ as

$$\hat{h} = LSTM_{\theta,E}(x_1, \ldots, x_l), \tag{2.9}$$

where θ is the parameters of the LSTM model and E is the word embedding matrix for

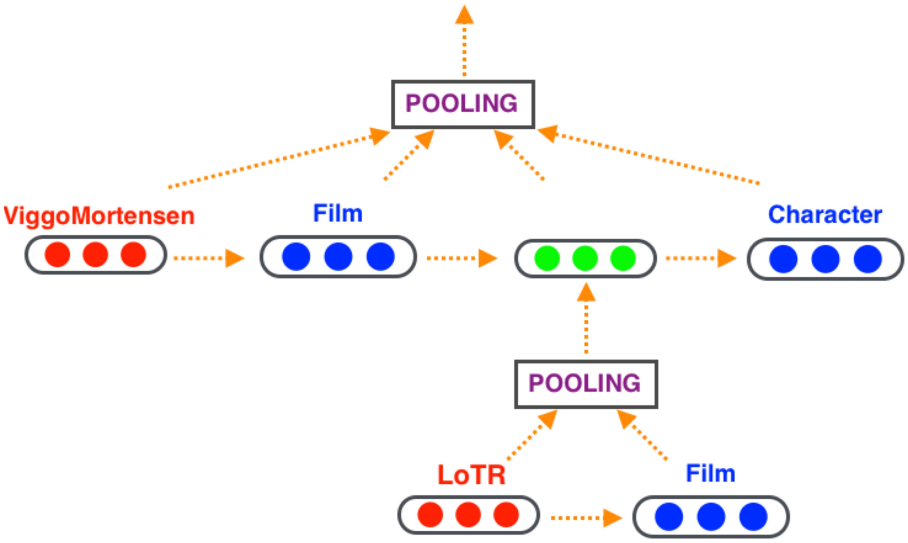


Figure 2.3: Hierarchical LSTM for computing the vector representation of the query graph in Figure 2.2. Input to each node is either generated recursively from child vectors or using its corresponding word vectors.

word vectors. We refer to LSTM with Pooling as the "standard LSTM" model.

Hierarchical LSTM Network

We use the standard LSTM to encode questions and modify the standard LSTM to encode the tree-like query graph where a node in the core path might have a child auxiliary path. Figure 2.3 shows the modified LSTM.

Assume there is a query graph Q defined as a core path $C = (x_1, x_2, \ldots, x_n)$ with an auxiliary path $A = (a_1, a_2, \ldots)$, where A is the child auxiliary path at node x_i. The new cell state and hidden vector at x_i is estimated using the representation of A as input in (2.1),

$$X_i \quad = \quad [LSTM_{\beta,E}(A); h_{i-1}]\,, \tag{2.10}$$

where $LSTM_{\beta,E}(A)$ is the representation of the auxiliary path A and the model parameters β are shared by all the auxiliary paths. The query graph is then encoded as follows,

$$h_Q = LSTM_{\theta,E}(x_1, \ldots, LSTM_{\beta,E}(A), \ldots, x_n). \tag{2.11}$$

In practice, β can be different from θ to allow the model to differentiate core and auxiliary paths. Word embeddings E are shared for question, auxiliary, and core pathes.

The input to each LSTM cell is the output generated by another LSTM if the cell has any child, otherwise the input is the word vector at that node. For the question "Who did Viggo Mortensen play in Lord of the Rings?" we first compute the output of the auxiliary path `Film.LoTR` as $\hat{h}_{aux} = LSTM(x_{Film}, x_{LoTR})$ and use this as input at the CVT node, i.e., $x_{CVT} = \hat{h}_{aux}$. We use the word vectors at other nodes (`ViggoMortensen`, `Film`, and `Character`) as their corresponding inputs. The gate vectors at the CVT cell are estimated using $\hat{h}_{aux}$ and the previous cell state in (2.1), i.e., $X_{CVT} = \left[\hat{h}_{aux}; h_{Film}\right]$. The pooling layer is then applied to obtain the final semantic representation of the query graph.

The intuition to our LSTM model is that, a single core relational path carries the essential information, while it is restricted by several auxiliary relations. These auxiliary relations modify the information that flows on the core path to reflect other facts. The cell state on the core path is affected by auxiliary paths indirectly. For example, if we change the relation `Character` in our example, the hidden vector at that node will be affected, which will in turn affect the representation of the query graph. However, if we change the relation `Film` at the auxiliary path, this will affect the final output via two more steps: hidden vector at auxiliary path, pooling layer at auxiliary path, and then hidden vector at the CVT node. This hierarchical structure has the advantage that the network output will focus on the core query graph more and reflect the effects of the auxiliary paths indirectly.

Ranking Loss Function

Given a question-answer pair (u, y), we generate a set of candidate query graphs Q where each yields an answer set S_q for $q \in Q$. For the training data, let f_q be the F1 score between a candidate answer set S_q and the true answer y.

Let $C(u)$ denote the set of query graphs that have positive F1 score. Given a query graph q, let $C'(u, q)$ be the set of queries q' with $f_{q'} < f_q$. For every question, we aim at ranking query graphs so that queries with higher F1 score appear earlier in the candidate list. The loss function is defined as follows,

$$\mathcal{O}_{\mathcal{U}}(\Theta, E) = \sum_{u \in \mathcal{U}} \sum_{q \in C(u)} \sum_{q' \in C'(u,q)} max(0, \Delta(q, q') - h_q^T h_u + h_{q'}^T h_u), \tag{2.12}$$

where $\mathcal{U}$ is the set of questions, Θ is the parameters of the LSTM models for question and query graph, and E is the word embedding matrix. We use the difference between F1 scores as the separation margin, $\Delta(q, q') = f_q - f_{q'}$. We use inner product between embedding vectors to measure the semantic similarity.

The final loss function also includes an L2 regularization term $\frac{\rho}{2} \|\Theta\|$ to prevent overfitting. We use AdaDelta [122] for optimization of Θ and E.

The above model does not consider the error that could be introduced by topic entity detection and linking. If the topic entity linking is wrong, likely the query graphs and answers derived in the following steps will not be useful. Fortunately, entity linking algorithms often generate a score between an entity mention in the question and the linked entity in the KB. This score shall be combined with the LSTM output in Figure 2.2 to give a final score. We apply a log-linear model to achieve this goal.

#	Template	Example Question
1	Entity linking score	*Who played Luke Skywalker in **Star Wars episode 4** ?*
2	Question Category	***Which** airport to fly into Rome ?*
3	"What" + NOUN	***What state** was theodore roosevelt from?*
4	Temporal Information	*Who does David James play for **2011** ?*

Table 2.1: Lexical features to prune the search space of auxiliary paths.

Initializing Word Embeddings

For each relation description in Freebase, we tokenize it using separators, "." and "_". For example, relation `People.Person.Profession` is tokenized as `People`, `Person`, and `Profession`. The embedding matrix E is initialized with Glove word vectors [75]. As the relations in Freebase can be described very differently from natural language, some words are not found in the Glove vocabulary. For example, in relation `Base.Semanticnames.PersonWithName`, `Semanticnames` is not found. We adopt a prefix-based greedy search approach to solve this problem. If a word is not found in the vocabulary, we greedily search for the largest prefix in the word that can be found. The prefix found is appended to the input sequence, and removed from the word. The process continues until the whole word is consumed or no prefix can be found in the vocabulary. In our example, the word "semanticnames" is tokenized into "semantic" and "names" which both can be found in the vocabulary. For the words that can not be further tokenized, we initialize them with an average word vector.

Template-Based Augmentation

Query augmentation might produce correct auxiliary paths but with ambiguous or incorrect entities. Consider the following example "Who played Luke Skywalker in Star Wars episode 4?" Our model correctly identifies the core relation `PortrayedInFilms.Actor` and auxiliary relation `Film` but several entities such as `StarWarsEpisode5` and

`StarWarsEpisode4` correspond to the same auxiliary path. To tackle this problem, we propose a *template-based* learning approach which searches for lexical clues in the question. A small set of templates (Table 2.1) is designed to advise whether query augmentation shall be done or not.

To resolve the entity disambiguation problem, we extract the mention of the auxiliary entity and use the entity linking score as a feature. In our example, "Star Wars episode 4" will be extracted and will be correctly mapped to `StarWarsEpisode4`. "Which" questions generally ask for a specific type and "Who" questions usually ask for a `Person`. We categorize each question according to its WH word {What, Where, How, Who, Which} and use its category as a feature. Furthermore, we use "What" word and the NOUN next to it as a feature for the "answer type". For example, "what state" and "what city" will be used to augment `State` and `City` types to the query, respectively. We also generate a small set of type triggers (such as "governor", "senator", and "ethnicity") that we observe in the KB but our templates can not capture. The CVT nodes might have relations that define a time interval for an event such as the duration of a contract. We use years in questions to capture this temporal information. To provide some flexibility, we use a window of [-1, +1] for a year as the start and end date of an event might be represented slightly different in Freebase. To capture gender information, we identify a set of words such as "wife," "son," and "daughter" as binary features.

Miscellaneous

Our LSTM model takes question and graph query as input. How to handle entities in the input is an open question. Given an entity, at least we could have four options: Using the entity directly without modification, replacing it with its most common KB type, replacing it with a common token, e.g., #ENTITY#, or remove it. Relations often have type specification. For example, the relation `People.Person.PlaceOfBirth`

Model Parameter	Search Space
Input vector dimension	50 - 100 - **300**
Hidden vector Dimension	50 - 100 - **300** - 500
Dropout rate	**0** - 0.5
L2	**0** - 0.001 - 0.01
Batch size	8 - **16** - 32
Initialization	**Glove** - Random
Relearn embeddings	**True** - False
Pooling operator	**Average** - Max
LSTM parameters	Shared - **Not**

Table 2.2: Hyper-parameter setting for our model. Bold terms represent the final setting.

in Freebase takes a `Person` as subject and `Location` as object. We empirically observe that using the most common type of the entity in the question and removing the entities from query graphs give the best result. Note that the relation representation in query graphs already contain subject/object type information.

2.1.3 Experimental Evaluation

In this section, we evaluate our methods and conduct error analysis.

Dataset and Evaluation Metric

Dataset. We evaluate our model on the popular WEBQUESTIONS dataset [11] which contains 5,810

question/answer pairs. The questions are collected using Google Suggest API and the answers are collected via crowdsourcing using Amazon MTurk. It is split into training (3,778) and testing sets (2,032). The dataset has natural questions and has been widely used in question answering.

Identifying Topic Entities. Stanford NER Tagger [68] is used to detect topic entities in these questions. We use the Freebase Search API to get a ranked list of candidate Freebase entities for each span annotated with the NER tag. The top-2 returned entities

are used, and the entity linking scores are also used as a feature.

Training Data Preparation. Since WEBQUESTIONS dataset only provides question-answer pairs, we need to generate candidate logical forms as training data. We run a BFS search from the identified topic entities for each question to extract candidate query graphs. We first retrieve all 1-hop and 2-hop relations from the annotated topic entity as initial query graphs. We then augment each initial query with auxiliary 1-hop relations. We restrict the 1-hop relations from the ANSWER node to *type* and *gender*. We run each query against Freebase and retrieve candidate answers. Each query is associated with the F1 score of the corresponding answer set. To train our model, we generate positive and negative question/query graph pairs using the training set. Given a training question, the query graphs with a positive F1 score are the positive samples, while the ones taht get a zero F1 score are the negative samples. We develop our system using Theano [99].

Experimental Setup. Table 2.2 gives a list of hyper-parameters tuned in our model. Following [36], we tune each parameter independently. Our empirical results show that initializing word vectors using Glove and updating these vectors during LSTM training gives the best performance. Average pooling performs significantly better than Max pooling. Using separate parameters for the two LSTMs that encode questions and query graphs gives better results than sharing. We tune all the parameters for core relational path ranking and simply apply the same setting to augmentation.

Main Results

Table 2.3 shows the main results in comparison with the existing question answering methods. Our system achieves a very competitive result compared to the existing approaches without utilizing any external corpora. Two recently proposed question answering systems [119] and [109] utilize external corpora (ClueWeb and Wikipedia) and an advanced entity linking system, S-MART [113]. To make the results more directly

Model	Accuracy
[117] (Yao et al., 2014)	33.0
[11] (Berant et al., 2013)	35.7
[7] (Bao et al., 2014)	37.5
[12] (Bordes et al., 2014)	39.2
[9] (Berant et al., 2014)	39.9
[61] (Dong et al., 2015)	40.8
[112] (Yang et al., 2014)	41.3
[116] (Yao, 2015)	44.3
[10] (Berant et al., 2015)	49.7
[81] (Reddy et al.,, 2016)	50.3
[119] (Yih et al., 2015)	52.5
[109] (Xu et al., 2016)	**53.3**
[109] w/o Wikipedia	47.1
[119] w/ Freebase API	48.4
[119] w/o ClueWeb	50.9
Our Approach	**51.0**

Table 2.3: Comparison of our system with several existing works on the WEBQUESTIONS dataset.

Model	Accuracy
Core Path	49.2
Core + Auxiliary Path	51.0

Table 2.4: Performance of (1) using only core relational paths and (2) adding auxiliary paths.

comparable, we also provide the F1 score of [119] in the settings of (1) without ClueWeb (50.9%) or (2) using Freebase Search API (same as ours) for entity linking (48.4%). For [109], if tested without Wikipedia, its F1 score is 47.1% (still using S-MART).

Table 2.4 shows the performance of the main designs. It shows that using core relational path achieves a competitive result. Our LSTM model on augmented query graphs further improves the performance by a substantial margin.

Model	Accuracy
Core Path	49.2
+ WH Word (Which)	49.3
+ What+NN	49.6
+ Secondary Entities	49.7
+ Dates (in years)	49.3
+ Type and Gender Triggers	49.9
All	51.0

Table 2.5: Performance of using each trigger template separately for query augmentation.

Augmentation Ablation

Next, we examine the contribution of each component in the template-based augmentation module.

Table 2.5 shows the results of the template generation module. Out of the 2,032 test questions, 301 of them are augmented by our model. Each template improves the result by a small margin, but together they lead to a substantial improvement. A few questions have an additional type constraint such as "which country" or "which political party." One of the major improvements comes from type constraints as reflected by what+NOUN template. Our system performs better by incorporating the secondary entity information in the question. Temporal triggers give a small improvement and the reason is that in the KB, dates might include different months and days for the same year which makes it hard to pick a specific year.

Error Analysis

A certain amount of errors are caused by the imperfect answer annotation process of WEBQUESTIONS, where crowdsourcing workers are directed to find answers in the Freebase page of a topic entity. One interesting observation is that a considerable number of questions, 288 in training and 139 in testing, have exactly 10 answers, the maximum possible F1 score for 265 and 127 of which are lower than 1.0. We find that for a

single relation, at most 10 results will be displayed on a Freebase page. For example, `JoshHutcherson` has starred in 27 movies, but only 10 of them are shown on his Freebase page. A worker needs to click the "expand" button in order to get the full list. Some workers may not realize that, resulting in incomplete answer sets. Excluding such questions, our F1 score will reach 51.6%. Since this problem is systematic for the WEBQUESTIONS datset, it is not clear how other systems will behave if this problem is corrected.

We also randomly selected 100 testing questions that have an imperfect F1 score (< 1.0) and categorize the errors. 4% of the errors are due to answer annotation process. 17% of the errors are caused by entity linking. Yih et al. Yih et al. [119] show that an entity linking system better than the combination of Stanford NER tagger and Freebase Search API improves F1. 14% of the errors are the result of incorrect or missing labels of the answers. 48% of the errors are due to incorrect core relational path ranking and 16% of the errors come from incorrect augmentations of query graphs. The remaining 1% error is caused by the template set selection approach.

2.2 Dialogue-Based Structured Query Generation on Relational Databases

In the previous section, we described how we can train a hierarchical deep learning model to map user questions into structured logical form representations that can be executed on Freebase. We used a single-turn methodology where the user informed her intent at the beginning and our model executed without any interruption from the user. This form of closed-loop execution of systems is also practiced in recent work on learning natural language interfaces to relational databases (NLIDB).

Natural Language Question :	What are the countries that joined the NATO before 2004 ?
Ground Truth SQL Query :	SELECT country WHERE date of join < 2004
Initial SQL Query :	SELECT count (country) WHERE date of join = 2004

Turn Side	NL Expression using Rule-Based Templates	Response
System:	Could you please confirm if you are asking **total number of the results**?	validate_agg (1, 2)
User:	No, that's not true.	negate()
System:	Could you please select one of the following options : (i) **average number of results**, (ii) **all the results**?	offer_agg (average, no_agg)
User:	It should be **all the results**.	select(2)
System:	Could you please confirm if **date of join is 2004** ?	validate_where_change(6, 11)
User:	That's wrong.	negate()
System:	Could you please select one of the following options : (i) **date of join is before 2004**, (ii) **date of invitation is before 2004**?	offer_where (date of join < 2004, date of invitation < 2004)
User:	**Date of join is before 2004**.	select(1)

Table 2.6: DialSQL model running example. Initial SQL query is generated by running a black box model on the question. Natural language (NL) expressions are generated using a template based method. Substrings in red represent the error spans and substrings in blue represent the choices offered. Each response is accompanied with natural language utterances for clarity.

While taking a single-turn approach could result in faster execution times with less user efforts, methods that achieve the state-of-the-art performance on NLIDB datasets, such as WikiSQL [111, 126, 110, 47], suffer from lack of understanding ambiguous user questions and errors made during inference process. After analyzing the error cases of Seq2SQL [126] and SQLNet, we recognized that many wrong translations cannot be easily corrected without any external knowledge or semantic understanding which is why we aim to integrate humans into the inference process of these systems. Previous human-in-the-loop NLIDBs [56, 111] rely on users to carefully go through a generated SQL query and revise it accordingly, which is infeasible for users without domain expertise. Instead, we resort to a different approach by introducing a goal-oriented dialogue model, DialSQL, that interacts with users to extract and refine potential errors in the generated queries.

2.2.1 Overview

Given a candidate SQL query generated from a natural language question using any black-box query generation system (such as SQLNet), we assume any segment, or span,

of the generated query such as a `WHERE` clause can be potentially erroneous. The goal of DialSQL is to extract the erroneous spans and ask users multi-choice questions to validate and correct these errors. DialSQL is based on a hierarchical encoder-decoder architecture with attention and pointer mechanisms. The model first encodes each turn of interaction and runs a dialogue level RNN network on the dialogue history. The output of the network is then used to predict the error category, i.e., whether it is a selection, projection, or aggregation error. Conditioned on the error category, the output of a second RNN is used to predict the start and end positions of the error span by pointing to the query tokens. Finally, candidate choices are decoded from the error category and span representations. Following previous work [126, 110], we only use column names and do not utilize table values.

How to train and evaluate DialSQL become two challenging issues due to the lack of error data and interaction data. We construct a simulator to generate simulated dialogues, a general approach practiced by many dialogue studies. Inspired by the agenda-based methods for user simulation [86], we keep an agenda of pending actions that are needed to induce the ground truth query. At the start of the dialogue, a new query is carefully synthesized by randomly altering the ground truth query and the agenda is populated by the sequence of altering actions. Each action consists of three sub-actions: (i) Pick an error category and extract a span; (ii) Raise a question; (iii) Update the query by randomly altering the span and remove the action from the agenda. Consider the example in Figure 2.4: Step-1 synthesizes the initial query by randomly altering the `WHERE` clause and `AGGREGATION`; Step-2 generates the simulated dialogue by validating the altered spans and offering the correct choice.

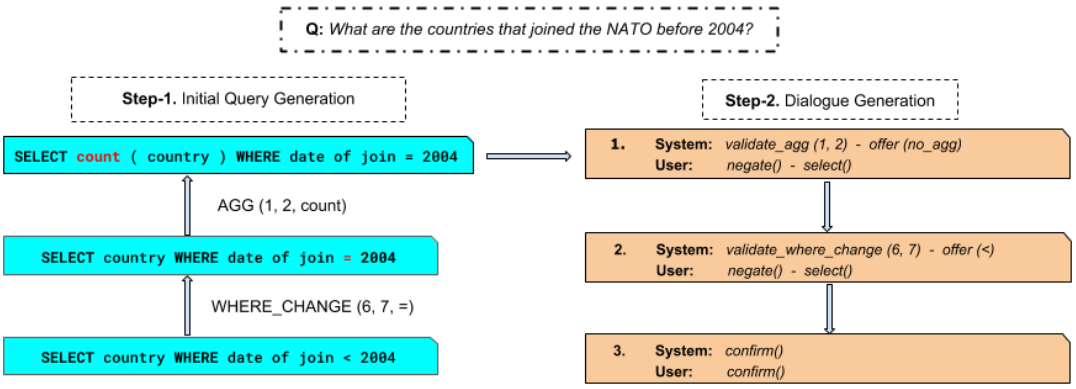


Figure 2.4: An instantiation of our dialogue simulation process. Step-1 synthesizes the initial query (top) by randomly altering the ground truth query (bottom). Step-2 generates the dialogue by validating the sequence of actions populated in Step-1 with the user. Each action is defined by the error category, start and end positions of the error span, and the random replacement, ex. `AGG (1, 2, count)`.

2.2.2 Related Work

Research on natural language interfaces to databases (NLIDBs), or semantic parsing, has spanned several decades. Early rule-based NLIDBs [104, 3, 78] employ carefully designed rules to map natural language questions to formal meaning representations like SQL queries. While having a high precision, rule-based systems are brittle when facing with language variations. The rise of statistical models [125, 49, 11], especially the ongoing wave of neural network models [119, 23, 95, 126, 110, 37, 118], has enabled NLIDBs that are more robust to language variations. Such systems allow users to formulate questions with greater flexibility. However, although state-of-the-art systems have achieved a high accuracy of 80% to 90% [23] on well-curated datasets like GEO [123] and ATIS [124], the best accuracies on datasets with questions formulated by real human users, e.g., WebQuestions [11], GraphQuestions [93], and WikiSQL [126], are still far from enough for real use, typically in the range of 20% to 60%.

Human-in-the-loop systems are a promising paradigm for building practical NLIDBs. A number of recent studies have explored this paradigm with two types of user interaction:

coarse-grained and fine-grained. Iyer et al. [47] and Li et al. [59] incorporate coarse-grained user interaction, i.e., asking the user to verify the correctness of the final results. However, for real-world questions, it may not always be possible for users to verify result correctness, especially in the absence of supporting evidence. Li and Jagadish [56] and Yaghmazadeh et al. [111] have shown that incorporating fine-grained user interaction can greatly improve the accuracy of NLIDBs. However, they require that the users have intimate knowledge of SQL, an assumption that does not hold for general users. Our method also enables fine-grained user interaction for NLIDBs, but we solicit user feedback via a dialogue between the user and the system.

Our model architecture is inspired by recent studies on hierarchical neural network models [91, 87, 38]. Recently, Saha et al. [85] propose a hierarchical encoder-decoder model augmented with key-value memory network for sequential question answering over knowledge graphs. Users ask a series of questions, and their system finds the answers by traversing a knowledge graph and resolves coreferences between questions. Our interactive query generation task significantly differs from their setup in that we aim to explicitly detect and correct the errors in the generated SQL query via a dialogue between our model and the user.

Agenda based user simulations have been investigated in goal-oriented dialogues for model training [86]. Recently, Seq2seq neural network models are proposed for user simulation [5] that utilize additional state tracking signals and encode dialogue turns in a more coarse way. We design a simulation method for the proposed task where we generate dialogues with annotated errors by altering queries and tracking the sequence of alteration steps.

Error Category	Meaning in a dialogue
validate_sel	Validate the select clause
validate_agg	Validate the aggregation operator
validate_where_changed	Validate if a segment of a where clause is incorrect
validate_where_removed	Validate if a new where clause is needed
validate_where_added	Validate if an incorrect where clause exists
no_error	Validate if there is no remaining error

Table 2.7: The list of error categories and their explanations for our interactive query generation task.

2.2.3 Problem Setup and Datasets

We study the problem of building an interactive natural language interface to databases (INLIDB) for synthesizing SQL queries from natural language questions. In particular, our goal is to design a dialogue system to extract and validate potential errors in generated queries by asking users multi-choice questions over multiple turns. We will first define the problem formally and then explain our simulation strategy.

Interactive Query Generation

At the beginning of each dialogue, we are given a question $Q = \{q_1, q_2, \cdots, q_N\}$, a table with column names $T = \{T_1, T_2, \cdots, T_K\}$ where each name is a sequence of words, and an initial SQL query U generated using a black box SQL generation system. Each turn t is represented by a tuple of system and user responses, (S_t, R_t), and augmented with the dialogue history (list of previous turns), H_t. Each system response is a triplet of error category c, error span s, and a set of candidate choices C, i.e., $S_t = (c, s, C)$. An error category (Table 2.7) denotes the type of the error that we seek to correct and an error span is the segment of the current query that indicates the actual error. Candidate choices depend on the error category and range over the following possibilities: (i) a column name, (ii) an aggregation operator, or (iii) a where condition. User responses are represented by either an affirmation or a negation answer and an index c' to identify a

choice. We define the *interactive query generation task* as a list of subtasks: at each turn t, (i) predict c, (ii) extract s from U, and (iii) decode C. The task is supervised and each subtask is annotated with labeled data.

Consider the example dialogue in Table 2.2. We first predict `validate_agg` as the error category and error span ($start = 1, end = 2$) is decoded by pointing to the *aggregation* segment of the query. Candidate choices, `(average, no_agg)`, are decoded using the predicted error category, predicted error span, and dialogue history. We use a template based natural language generation (NLG) component to convert system and user responses into natural language.

Dialogue Simulation for INLIDB

In our work, we evaluate our model on the WikiSQL task. Each example in WikiSQL consists of a natural language question and a table to query from. The task is to generate a SQL query that correctly maps the question to the given table. Unfortunately, the original WikiSQL lacks error data and user interaction data to train and evaluate DialSQL. We work around this problem by designing a simulator to bootstrap training dialogues and evaluate DialSQL on the test questions of WikiSQL.

Inspired by the agenda-based methods [86], we keep an agenda of pending actions that are needed to induce the ground truth query. At the start of the dialogue, we synthesize a new query by randomly altering the ground truth query and populating the agenda by the sequence of altering actions. Each action launches a sequence of sub-actions: (i) Randomly select an error category and extract a related span from the current query, (ii) randomly generate a valid choice for the chosen span, and (iii) update the current query by replacing the span with the choice. The dialogue is initiated with the final query and a rule-based system interacts with a rule-based user simulator to populate the dialogue. The rule-based system follows the sequence of altering actions previously generated and

asks the user simulator a single question at each turn. The user simulator has access to the ground truth query and answers each question by comparing the question (error span and the choice) with the ground truth.

Consider the example in Figure 2.4 where Step-1 synthesizes the initial query and Step-2 simulates a dialogue using the outputs of Step-1. Step-1 first randomly alters the `WHERE` clause; the operator is replaced with a random operator. The updated query is further altered and the final query is passed to Step-2. In Step-2, the system starts with validating the aggregation with the user simulator. In this motivating example, the aggregation is incorrect and the user simulator negates and selects the offered choice. During training, there is only a single choice offered and DialSQL trains to produce this choice; however, during testing, it can offer multiple choices. In the next step, the system validates the `WHERE` clause and generates a `no_error` action to issue the generated query. At the end of this process, we generate a set of labeled dialogues by executing Step-1 and Step-2 consecutively. DialSQL interacts with the same rule-based simulator during testing and the SQL queries obtained at the end of the dialogues are used to evaluate the model.

2.2.4 Dialogue Based SQL Generation

In this section, we present our DialSQL model and describe its operation in a fully supervised setting. DialSQL is composed of three layers linked in a hierarchical structure where each layer solves a different subtask : (i) Predicting error category, (ii) Decoding error span, and (iii) Decoding candidate choices (illustrated in Figure 2.5). Given a (Q, T, U) triplet, the model first encodes Q, each column name $T_i \in T$, and query U into vector representations in parallel using Recurrent Neural Networks (RNN). Next, the first layer of the model encodes the dialogue history with an RNN and predicts the

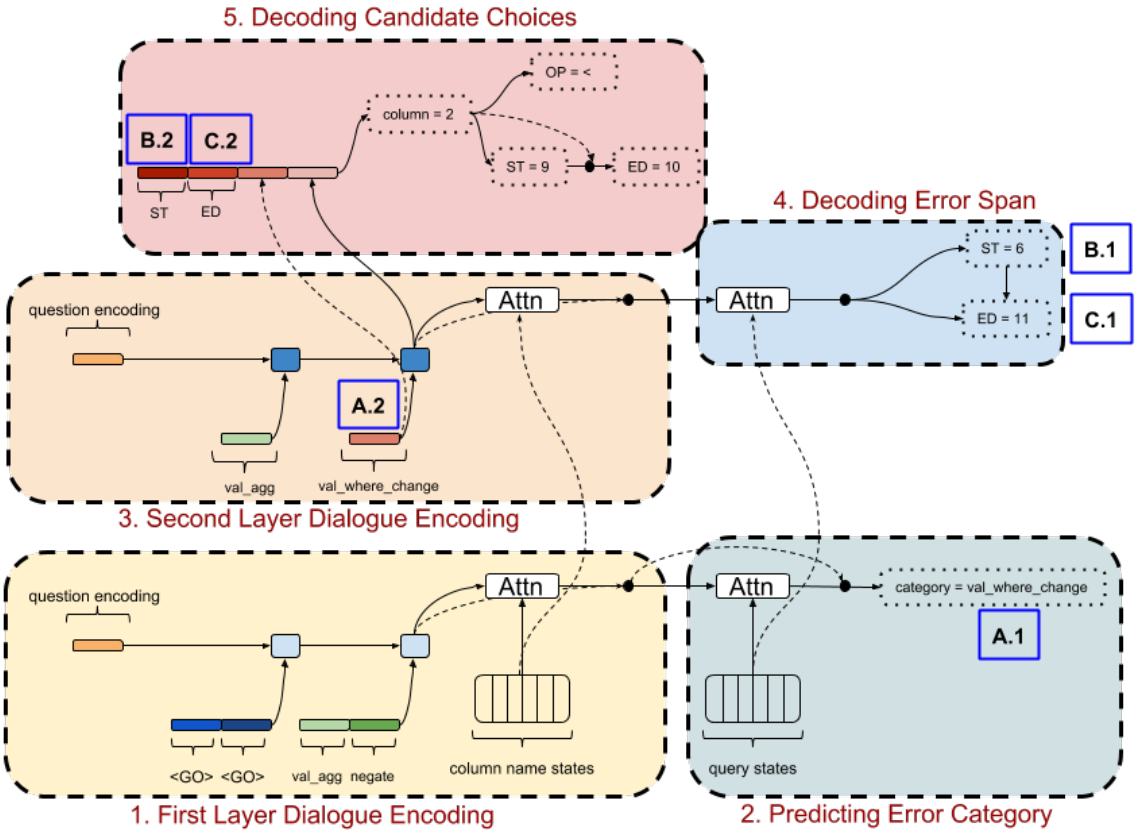


Figure 2.5: DialSQL model: Boxes are RNN cells, colors indicate parameter sharing. Dashed lines denote skip connections, dashed boxes denote classifications, and black circles denote vector concatenation. Blue boxes with capital letters and numbers (X.1, X.2) denote that the embeddings of predicted token at X.1 is passed as input to X.2. Each component in the pipeline is numbered according to execution order. `<GO>` is a special token to represent the start of a sequence and `ST` and `ED` denote the start and end indices of a span, respectively.

error category from this encoding. The second layer is conditioned on the error category and decodes the start and end positions of the error span by attending over the outputs of query encoder. Finally, the last layer is conditioned on both error category and error span and decodes a list of choices to offer to the user.

Preliminaries and Notation

Each token w is associated with a vector e_w from rows of an embeddings matrix E. We aim at obtaining vector representations for question, table headers, and query, then generating error category, error span, and candidate choices.

For our purposes, we use GRU units [17] in our RNN encoders which are defined as

$$h_t = f(x_t; h_{t-1})$$

where h_t is the hidden state at time t. f is a nonlinear function operating on input vector x_t and previous state h_{t-1}. We refer to the last hidden state of an RNN encoder as the encoding of a sequence.

Encoding

The core of our model is a hierarchical encoder-decoder neural network that encodes dialogue history and decodes errors and candidate choices at the end of each user turn. The input to the model is the previous system turn and the current user turn and the output is the next system question.

Encoding Question, Column Names, and Query. Using decoupled RNNs (Enc), we encode natural language question, column names, and query sequences in parallel and produce outputs and hidden states. o^Q, o^{T_i}, and o^U denote the sequence of hidden states at each step and h^Q, h^{T_i}, and h^U denote the last hidden states of question, column name, and query encoders, respectively. Parameters of the encoders are decoupled and only the word embedding matrix E is shared.

Encoding System and User Turns Since there is only a single candidate choice dur-

ing training, we ignore the index and encode user turn by doing an embedding lookup using the validation answer (affirmation or negation). Each element (error category, error span, and candidate choice) of the system response is encoded by doing an embedding lookup and different elements are used as input at different layers of our model.

Encoding Dialogue History At the end of each user turn, we first concatenate the previous error category and the current user turn encodings to generate the turn level input. We employ an RNN to encode dialogue history and current turn into a fixed length vector as

$$\begin{aligned} h_0^{D_1} &= h^Q \\ o_t^{D_1}, g_t^{D_1} &= Enc([E_c, E_a]) \\ h_t^{D_1} &= [Attn(g_t^{D_1}, H^T), o_t^D] \end{aligned}$$

where [.] is vector concatenation, E_c is the error category encoding, E_a is the user turn encoding, $h_0^{D_1}$ is the initial hidden state, and $h_t^{D_1}$ is the current hidden state. $Attn$ is an attention layer with a bilinear product defined as in [66]

$$Attn(h, O) = \sum softmax(tanh(hWO)) * O$$

where W is attention parameter.

Predicting Error Category

We predict the error category by attending over query states using the output of the dialogue encoder as

$$c_t = tanh(Lin([Attn(h_t^{D_1}, O^U), h_t^{D_1}]))$$
$$l_t = softmax(c_t \cdot E(C))$$

where Lin is a linear transformation, $E(C)$ is a matrix with error category embeddings, and l_t is the probability distribution over categories.

Decoding Error Span

Consider the case in which there are more than one different `WHERE` clauses in the query and each clause has an error. In this case, the model needs to monitor previous error spans to avoid decoding the same error. DialSQL runs another RNN to generate a new dialogue encoding to solve the aforementioned problem as

$$h_0^{D_2} = h^Q$$
$$o_t^{D_2}, g_t^{D_2} = Enc(E_c)$$
$$h_t^{D_2} = [Attn(g_t^{D_2}, H^T) o_t^{D_2}]$$

where $h_0^{D_2}$ is the initial hidden state, and $h_t^{D_2}$ is the current hidden state. Start position i of the error span is decoded using the following probability distribution over query tokens

$$p_i = softmax(tanh(h_t^{D_2} L_1 H^U))$$

where p_i is the probability of start position over the ith query token. End position j of the error span is predicted by conditioning on the start position

$$c_i = \sum p_i * H^U$$
$$\hat{p}_j = softmax(tanh([h_t^{D_2}, c_i]L_2 H^U))$$

where $\hat{p}_j$ is the probability of end position over the jth query token. Conditioning on the error category will localize the span prediction problem as each category is defined by only a small segment of the query.

Decoding Candidate Choices

Given error category c and error span (i, j) , DialSQL decodes a list of choices that will potentially replace the error span based on user feedback. Inspired by SQLNet [110], we describe our candidate choice decoding approach as follows.

Select column choice. We define the following scores over column names,

$$h = Attn(Lin([o_{i-1}^U, o_j^U, E_c]), H^T)$$
$$s_{sel} = u^T * tanh(Lin([H^T, h]))$$

where o_{i-1}^U is the output vector of the query encoder preceding the start position, and o_j^U is the output of query encoder at the end position.

Aggregation choice. Conditioned on the encoding e of the select column, we define the following scores over the set of aggregations (`MIN`, `MAX`, `COUNT`, `NO_AGGREGATION`)

$$s_{agg} = v^T * tanh(Lin(Attn(e, H^Q)))$$

Where condition choice. We first decode the condition column name similar to decoding select column. Given the encoding e of condition column, we define the following scores over the set of operators (=, <, >)

$$s_{op} = w^T * tanh(Lin(Attn(e, H^Q)))$$

Next, we define the following scores over question tokens for the start and end positions of the condition value

$$s_{st} = Attn(e, H^Q)$$
$$s_{ed} = Attn([e, h_{st}, H^Q])$$

where h_{st} is the context vector generated from the first attention. We denote the number of candidate choices to be decoded by k. We train DialSQL with $k = 1$. The list of $k > 1$ candidate choices is decoded similar to beam search during testing. As an example, we select k column names that have the highest scores as the candidate where column choices. For each column name, we first generate k different operators and from the set of $k * 2$ column name and operator pairs; select k operators that have the highest joint probability. Ideally, DialSQL should be able to learn the type of errors present in the generated query, extract precise error spans by pointing to query tokens, and using the location of the error spans, generate a set of related choices.

2.2.5 Experimental Results and Discussion

In this section, we evaluate DialSQL on WikiSQL using several evaluation metrics by comparing with previous literature.

Evaluation Setup and Metrics

We measure the query generation accuracy as well as the complexity of the questions and the length of the user interactions.

Query-match accuracy. We evaluate DialSQL on WikiSQL using query-match accuracy [126, 110]. Query-match accuracy is the proportion of testing examples for which the generated query is exactly the same as the ground truth, except the ordering of the `WHERE` clauses.

Dialogue length. We count the number of turns to analyze whether DialSQL generates any redundant validation questions.

Question complexity. We use the average number of tokens in the generated validation questions to evaluate if DialSQL can generate simple questions without overwhelming users.

Since SQLNet and Seq2SQL are single-step models, we can not analyze DialSQL's performance by comparing against these on the last two metrics. We overcome this issue by generating simulated dialogues using an oracle system that has access to the ground truth query. The system compares `SELECT` and `AGGREGATION` clauses of the predicted query and the ground truth; asks a validation question if they differ. For each `WHERE` clause pairs of generated query and the ground truth, the system counts the number of matching segments namely `COLUMN, OP`, and `VALUE`. The system takes all the pairs with the highest matching scores and asks a validation question until one of the queries has no remaining `WHERE` clause. If both queries have no remaining clauses, the dialogue terminates. Otherwise, the system asks a `validate_where_added` (`validate_where_removed`) question when the generated query (ground truth query) has more remaining clauses. We call this strategy *Oracle-Matching (OM)*. OM ensures that the generated dialogues have the minimum number of turns possible.

Model	QM-Dev	QM-Test
Seq2SQL [110]	53.5%	51.6%
SQLNet [110]	63.2%	61.3%
BiAttn [37]	64.1%	62.5%
Seq2SQL - DialSQL	62.2%	61%
SQLNet - DialSQL	70.9%	69.0%
Seq2SQL - DialSQL$^+$	68.9%	67.8%
SQLNet - DialSQL$^+$	74.8%	73.9%
Seq2SQL - DialSQL*	84.4%	84%
SQLNet - DialSQL*	82.9%	83.7%

Table 2.8: Query-match accuracy on the WikiSQL development and test sets. The first two scores of our model are generated using 5 candidate choices, ($^+$) denotes a variant where users can revisit their previous answers, and (*) denotes a variant with more informative user responses.

Training Details

We implement DialSQL in TensorFlow [1] using the Adam optimizer [51] for the training with a learning rate of 1e−4. We use an embedding size of 300, RNN state size of 50, and a batch size of 64. The embeddings are initialized from pretrained GloVe embeddings [75] and fine-tuned during training. We use bidirectional RNN encoders with two layers for questions, column names, and queries. Stanford CoreNLP tokenizer [68] is used to parse questions and column names. Parameters of each layer are decoupled from each other and only the embedding matrix is shared. The total number of turns is limited to 10 and 10 simulated dialogues are generated for each example in the WikiSQL training set. SQLNet and Seq2SQL models are trained on WikiSQL using the existing implemention provided by their authors. The code is available at `https://github.com/izzeddingur/DialSQL`.

Evaluation on the WikiSQL Dataset

Table 2.8 presents the results of query match accuracy. We observe that DialSQL model with a number of 5 choices improves the performance of both SQLNet and

Model	QC Dev	DL Dev	QC Test	DL Test
Seq2SQL - OM	3.47 (2.25)	0.84 (1.77)	3.51 (2.41)	0.88 (1.8)
SQLNet - OM	3.37 (2.63)	0.61 (1.45)	3.34 (2.51)	0.63 (1.49)
Seq2SQL - DialSQL	3.53 (1.79)	5.54 (2.32)	3.55 (1.81)	5.55 (2.34)
SQLNet - DialSQL	3.6 (1.86)	5.57 (2.34)	3.17 (1.55)	4.77 (1.57)

Table 2.9: Average query complexity and dialogue length on the WikiSQL datasets (values in paranthesis are standard deviations). Metrics for SQLNet and Seq2SQL models are generated by the OM strategy as described earlier.

Seq2SQL by 7.7% and 9.4%, respectively. The higher gain on Seq2SQL model can be attributed that the single-step Seq2SQL makes more errors: DialSQL has more room for improvement. We also show the results of DialSQL where users are allowed to revisit their previous answers and with more informative user responses; instead the model only validates the error span and the user directly gives the correct choice. In this scenario, the performance further improves on both development and test sets. It seems decoding candidate choices is a hard task and has room for improvement. For the rest of the evaluation, we present results with multi-choice questions.

Query Complexity and Dialogue Length

In Table 2.9, we compare DialSQL to the OM strategy on query complexity (QC) and dialogue length (DL) metrics. DialSQL and SQLNet-OM both have very similar query complexity scores showing that DialSQL produces simple questions. The number of questions DialSQL asks is around 3 for both query generation models. Even though SQLNet-OM dialogues have much smaller dialogue lengths, we attribute this to the fact that 61.3% of the dialogues have empty interactions since OM will match every segment in the generated query and the ground truth. The average number of turns in dialogues with non-empty interactions, on the other hand, is 3.10 which is close to DialSQL.

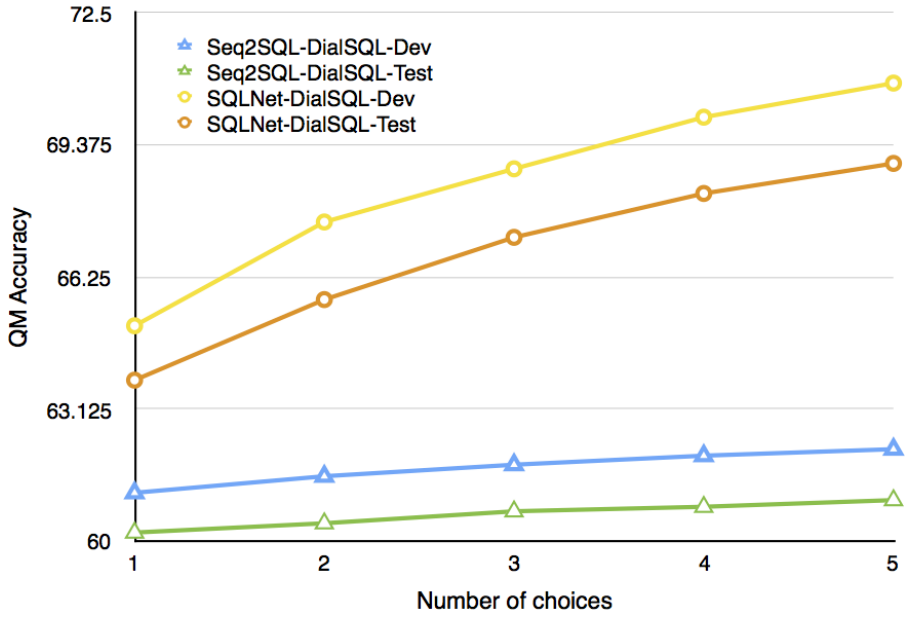


Figure 2.6: DialSQL performance on WikiSQL with a varying number of choices at each turn.

A Varying Number of Choices

In Figure 2.6, we plot the accuracy of DialSQL on WikiSQL with a varying number of choices at each turn. We train DialSQL once and generate a different number of choices at each turn by offering top-k candidates during testing. We observe that offering even a single candidate improves the performance of SQLNet remarkably, 1.9% and 2.5% for development and test sets, respectively. As the number of choices increases, the performance of DialSQL improves in all the cases. Particularly, for the SQLNet-DialSQL model we observe more accuray gain. We increased the number of choices to 10 and observed no notable further improvement in the development set which suggests that 5 is a good value for the number of choices.

Error Distribution

We examine the error distribution of DialSQL and SQLNet. In DialSQL, almost all the errors are caused by `validate_sel` and `validate_where_change`, while in SQLNet

`validate_where_change` is the major cause of error and other errors are distributed uniformly.

Human Evaluation

We extend our evaluation of DialSQL using human subject experiment so that real users interact with the system instead of our simulated user. We randomly pick 100 questions from WikiSQL development set and run SQLNet to generate initial candidate queries. Next, we run DialSQL using these candidate queries to generate 100 dialogues, each of which is evaluated by 3 different users. At each turn, we show users the headers of the corresponding table, original question, system response, and list of candidate choices for users to pick. For each error category, we generate 5 choices except for the `validate_where_added` category for which we only show 2 choices (YES or NO). Also, we add an additional choice of *None of the above* so that users can keep the previous prediction unchanged. At the end of each turn, we also ask users to give an *overall score* between 1 and 3 to evaluate whether they had a successful interaction with the DialSQL for the current turn. On average, the length of the generated dialogues is 5.6.

In Table 2.10, we compare the performance of SQLNet, DialSQL with user simulation, and DialSQL with real users using QM metric. We present the average performance across 3 different users with the standard deviation estimated over all dialogues. We observe that when real users interact with our system, the overall performance of the generated queries are better than SQLNet model showing that DialSQL can improve the performance of a strong NLIDB system in a real setting. However, there is still a large room for improvement between simulated dialogues and real users.

In Figure 2.7, we present the correlation between DialSQL ranking of the candidate choices and user preferences. We observe that, user answers and DialSQL rankings are positively correlated; most of the time users prefer the top-1 choice. Interestingly,

Model	Accuracy
SQLNet	58
DialSQL w/ User Simulation	75
DialSQL w/ Real Users	65 (1.4)

Table 2.10: QM accuracies of SQLNet, DialSQL with user simulation, and DialSQL with real users (value in paranthesis is standard deviation).

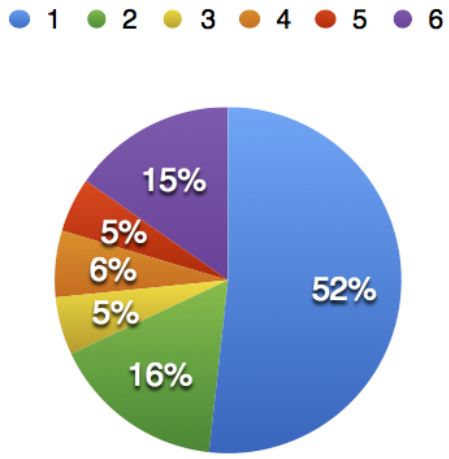


Figure 2.7: Distribution of user preference for DialSQL ranking (scaled to 1-6 with 6 is *None of the above.*).

15% of the user answers is *None of the above*. This commonly happens in the scenario where DialSQL response asks to replace a correct condition and users prefer to keep the original prediction unchanged. Another scenario where users commonly select *None of the above* is when table headers without the content remain insufficient for users to correctly disambiguate condition values from questions. We also compute the Mean Reciprocal Rank (MMR) for each user to measure the correlation between real users and DialSQL. Average MMR is 0.69 with standard deviation of 0.004 which also shows that users generally prefer the choices ranked higher by DialSQL. The overall score of each turn also suggests that users had a reasonable conversation with DialSQL. The average score is 2.86 with standard deviation of 0.14, showing users can understand DialSQL responses and can pick a choice confidently.

2.3 Modeling Users in Task-Oriented Dialogue Systems

In the previous section, we introduced a new type of structured query generation framework where humans are integrated into the generation process. We extracted precise errors from candidate queries which are then refined over multiple turns via having multi-turn dialogues with users. Our models are trained with rule-based user simulators as collecting large-scale labeled dataset is costly and time consuming; however, rule-based user simulators lack the necessary diversity of user responses and might not accurately reflect user behavior.

Besides training end-to-end dialogue systems, modeling user behavior has other use cases. Google Duplex demo[1] is a good example of machines mimicking humans for achieving certain tasks. Furthermore, these simulators can be used to evaluate the alternative approaches for a task oriented dialogue system. Most systems are evaluated on fixed testing corpora (collected via crowd sourcing), a very limited way in its ability to test if a system can accurately interact with various different users. A scalable and accurate evaluation criteria that can compare multiple systems in a similar set-up is critical to ameliorate the aforementioned bottlenecks. We propose several end-to-end user simulation models that aim to mimic realistic and diverse user behaviors. Our user simulators aim to enable side-by-side comparisons of dialogue systems or their components across a given set of user goals and dialogue scenarios. Another goal of our approach is to enable training of dialogue policies via supervised and reinforcement learning using a diverse set of interactions.

[1] https://www.youtube.com/watch?v=bd1mEm2Fy08

User Goal: *date*=Friday, *num_tickets*=2, *theatre_name*=DontCare, *movie*=Sully, *time*=DontCare

0 -	SYSTEM	Hi, how can I help you? `greeting()`

	USER	Hi, I'd like to buy tickets to see the Sully movie.
		`greeting() intent(buy_movie_tickets)`
		`inform(movie=Sully)`

1 -	SYSTEM	You wanna see Sully. How many tickets would you need? `confirm(movie=Sully) request(num_tickets)`
	USER	2 tickets please. `inform(num_tickets=2)` ...

Figure 2.8: An example dialogue from the movie ticket booking domain. Each dialogue has an initial user goal randomly assigned at the beginning.

2.3.1 Overview

Inspired by recent conversational models [88], we develop a hierarchical seq2seq user simulator (HUS) that implicitly tracks user goal over multiple turns. HUS first encodes a user goal and each system turn observed in the dialogue history into vector representations. Initialized from the user goal vector, a higher level encoder generates a dialogue history representation using system turn vectors as input at each time step. Finally, the simulated user turns are decoded from this dialogue history representation.

While HUS can generate successful dialogues, it will always generate the same simulated user turns given the same user goal and system turns. To induce human-like variations and generate a richer set of user turns, we propose a variational framework where an unobserved latent variable generates user turns. Furthermore, with HUS, user turns and user goals are related only through a user turn decoder, resulting in repetition of previously specified information throughout the interaction. We introduce a new goal regularization approach where a penalization term that aims to maximize the overlap of tokens between the user turns and the user goals is added to the final loss. The proposed user simulators work at the dialogue act level and generate responses in the form of acts and associated slot and value pairs (a sample dialogue is shown in Figure 2.8). Final out-

puts are then converted into natural language utterances using a template-based natural language generation approach.

We evaluate our models on a *movie ticket booking* domain by systematically interacting each user simulator with multiple system policies trained with different objectives and architectures. We show that a reinforcement learning based policy is more robust to random variations in user behavior, while a supervised model suffers from a considerable drop in accuracy. Our goal regularization approach generates significantly shorter dialogues (i.e. avoiding loops around user or system misunderstandings) across all system policies and shows higher task completion rates. Finally, human evaluations show high naturalness scores for all proposed approaches.

2.3.2 Related Work

The proposed model architectures in our work are inspired by hierarchical deep learning models studied in [91, 88, 57] which are also shown to perform better than plain seq2seq and language models on dialogue generation tasks. Variational approaches such as Variational Autoencoders are used for unsupervised or semi-supervised model training [50, 25] that improves the final performance metrics as well as the diversity of the generated outputs.

Agenda based user simulations have been investigated in task oriented dialogues for model training [86]. Supervised learning approaches using linear models [32, 33] as well as hidden markov models [21] have also been proposed for simulating users. Recently, seq2seq neural network models are proposed for user simulation [5, 74, 53, among others] that utilize additional state tracking signals and dialogue turns are encoded more coarsely. Kreyssig et al. [53] further showed that plain seq2seq models with state tracking signals at each turn can outperform agenda-based user simulators on several metrics

including success rate. Another work that also uses seq2seq models for user simulation is [20], where seq2seq models are compared with language modeling based approaches for generating user turns in natural language. They encode the context (previous user turn) and current system turn using two independent encoders and concatenate the output vectors. Simulated user turn is then decoded from this final vector. However, we propose an end-to-end hierarchical seq2seq approach that can encode the entire dialogue history without any feature extraction and does not require any external state tracking annotations. Furthermore, we introduce several novel variants using variational approaches and goal regularization that improves the dialogue metrics and allows a more thorough comparison with different system policies.

2.3.3 Problem Description

Following the recent progress on end-to-end supervised dialogue models, we consider the inverse problem of generating a user turn U_t given a user goal C and the history of system turns $S_1, S_2, \cdots, S_t$. A user goal is a set of slot-value pairs (ex. *time: 12pm*) with a predefined user personality (e.g., *aggressive, cooperative*) that defines the sampling distribution when generating user turns. Similarly, system and user turns are sets of actions where each action has a dialogue act (ex. *inform*) and a set of corresponding slot-value pairs, e.g., *inform(time=12pm, theatre="AMC Theatre").*

We first generate a more coarse level representation of each input by replacing the value of each slot with one of the following values: {`Requested, DontCare, ValueInGoal, ValueContradictsGoal, Other` }. If the value of a slot is requested by system, we replace those slot values with `Requested`. If the value of a slot appears in or contradicts the user goal, we replace those values with `ValueInGoal` or `ValueContradictsGoal`, respectively. If the value of a slot in the user goal is flexible, meaning the user is open to accept any offered value, we replace the value of the corresponding slot with `DontCare`

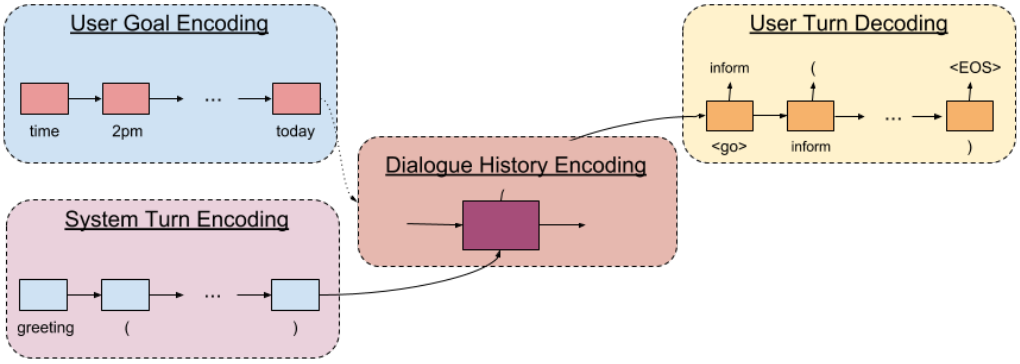


Figure 2.9: HUS model: Boxes are RNN cells, colors indicate parameter sharing.

in each input. Finally, we replace other values with `Other`. During testing, based on the coarse value, we sample an actual value either from the user goal, system turn, or from the knowledge base. Inspired by the success of recent approaches to structured data, we next linearize each input (user scenario, system and user turns) following [100] to generate token sequences. As an example, we replace the actual movie name *Sully* in system Turn-1 from Figure 2.8 with `ValueInGoal` and convert the sequence of actions into a sequence of tokens as

`"confirm", "(", "movie=ValueInGoal", ")",`

`"request", "(" "num_tickets", ")".`

We now describe our problem more formally. Given a user goal sequence C and the history of system turn sequences S_t at each turn, our task is to generate a new sequence of tokens that match the correct user turn sequence $U_t = \{u_{t1}, u_{t2}, \cdots, u_{tN_U}\}$ at turn t. Here, a successful user turn might require blending information from the user goal, and history of system and user turns. Furthermore, we do not assume that a supervised signal for dialogue state tracking is given.

2.3.4 Hierarchical Seq2Seq Models for User Simulation

We now present our supervised models, in Figure 2.9, and describe their operations in a fully end-to-end setting. Our baseline model is a hierarchical seq2seq neural network that operates on individual system and user turns as well as at the overall dialogue level. Given a (C, S_t, U_t) triplet, the model first encodes user goal C and the system turn dialogue act at turn t, S_t, into vector space representations in parallel with decoupled Recurrent Neural Networks (RNN). Another RNN is initialized from the goal representation and encodes the sequence of system turn representations into a hidden vector at turn t. Finally, simulated user response in dialogue act and arguments at turn t, U_t, is decoded from this hidden vector using an RNN sequence decoder. We next discuss the shortcomings of this model and propose several model improvements that are unsupervised in the sense that no additional supervised data is required.

Preliminaries and Notation

Each token w comes from a vocabulary V and is associated with a vector e_w. E^C, E^{S_t} and E^{U_t} denotes the sequence of token embedding vectors for user goal, system and user turns at turn t, respectively. Our models utilize the power of RNN encoders and decoders where we utilize GRU units [16]. We refer to the average of hidden states of a turn level RNN encoder as the encoding of a sequence.

Hierarchical Seq2Seq User Simulation (HUS)

The core of our model is a hierarchical seq2seq neural network that first encodes user goal and system turns and generates user turns from dialogue level representations.

Encoding User Scenario and System Turns Using a RNN (Enc), we encode user

goal as

$$h^C = Enc(e^C; \theta_C) \tag{2.13}$$

where h^C is the encoding and θ_C represents the parameters of goal encoder. We employ another RNN to encode each system turn as

$$h_i^S = Enc(e^{S_i}; \theta_S) \tag{2.14}$$

where h_i^S is encoding and θ_S represents the parameters of system turn encoder. θ_S is shared for every system turn encoder and is decoupled from θ_C.

Encoding Dialogue History Given the sequence of system turn encodings, $h_0^S, h_1^S, \cdots, h_t^S$, we employ another RNN conditioned on the user goal encoding to encode dialogue history as

$$h_0^D = h^C \tag{2.15}$$

$$h_t^D = Enc(\{h_i^S\}_{i=1,\cdots,t}; \theta_D) \tag{2.16}$$

where h_0^D is the initial hidden state, h_t^D is the history encoding, and θ_D represents the parameters of the dialogue level RNN. Ideally, the dialogue level RNN should keep track of the user goal and generate meaningful user turns.

Decoding User Turns Given the history encoding at turn t, an RNN decoder (Dec)

generates the user turn token sequence:

$$h_0^U = h_t^D \tag{2.17}$$

$$h_{t,i}^U = Dec(h_{t-1,i}^U; \omega_U) \tag{2.18}$$

$$P(U_{t,i}^* = w_j) \approx exp(e_j^T(W_U h_{t,i}^U + b_U)) \tag{2.19}$$

$$U_{t,i}^* = argmax_j(P(U_{t,i}^* = w_j)) \tag{2.20}$$

where h_0^U is the initial hidden state and $H_{t,i}^U$ is the hidden vector at turn t and timestep i. U_t^* is the sequence of user turn tokens generated. W_U, b_U and ω_U are the parameters of the user turn decoder. The training objective is to minimize the cross-entropy error $L_{crossent}$ between the candidate sequence U^* and the correct user turn U.

Incorporating Dialogue Length During training, we observe that sampling probabilities of different user personalities randomly generate shorter or longer dialogues which is not captured by HUS. To overcome this problem, we append the length of the corresponding dialogue to system turn encodings at each turn during training. During testing, we randomly sample a dialogue length from the following normal distribution $\mathcal{N}(5, 2)$. For the subsequent sections, we always incorporate dialogue length in our models.

An alternative model would also incorporate the previous user turn more explicitly, such as by encoding user turns and conditioning dialogue history encoder on user turn encodings; however we observe no significant benefit. One plausible reason is that, user turns are decoded from history encodings which are implicitly conditioned on previous user turns via previous history encodings.

Variational HUS

The HUS model will generate exactly the same user turns given the same user goal and dialogue history. However, to explore the robustness of a system policy and model

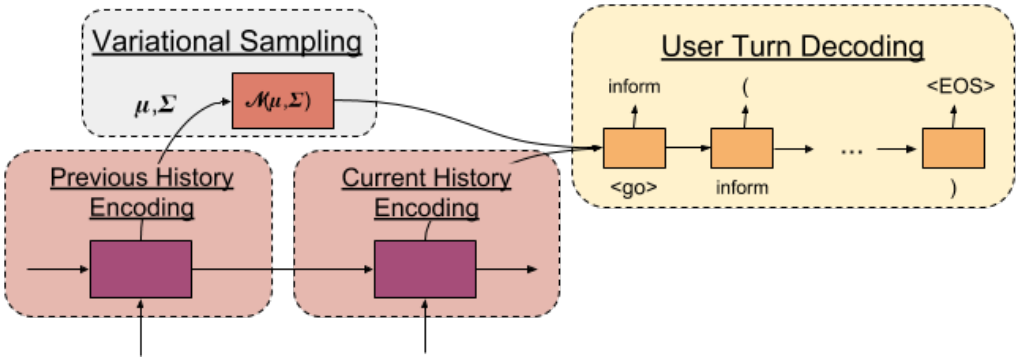


Figure 2.10: VHUS model: A variational sampling step before user turn decoder is proposed. Boxes are RNN cells, colors indicate parameter sharing.

different types of users, we need a model that can generate more diverse and at the same time meaningful user turns. Here, we propose a novel variational hierarchical seq2seq user simulator (VHUS) where an unobserved latent random variable generates the user turn sequence (Figure 2.10). The encoders are exactly the same as HUS, but the hidden state h_t^D is not directly passed to the decoder. Instead, it is first concatenated with a latent vector generated from a Gaussian distribution with a diagonal covariance matrix and then passed to the decoder.

More formally, given the sequence of dialogue representations $h_0^D, h_1^D, \cdots, h_t^D$, we learn a prior Gaussian distribution $\mathcal{N}(z|\mu_x, \Sigma_x)$ using the previous dialogue history: The mean and covariance is estimated as follows

$$\mu_x = W_\mu h_{t-1}^D + b_\mu \tag{2.21}$$

$$\Sigma_x = W_\Sigma h_{t-1}^D + b_\Sigma \tag{2.22}$$

where W_μ, W_Σ, b_μ, and b_Σ are new parameters for prior distribution. The decoder is then initialized with a single vector $\hat{h}_t^D = FC([h_t^D; z_x])$ where [.] is vector concatenation, FC is a single layer neural network, and z_x is sampled from the prior distribution, $z_x \sim \mathcal{N}(z|\mu_x, \Sigma_x)$. We also learn a posterior Gaussian distribution using the current

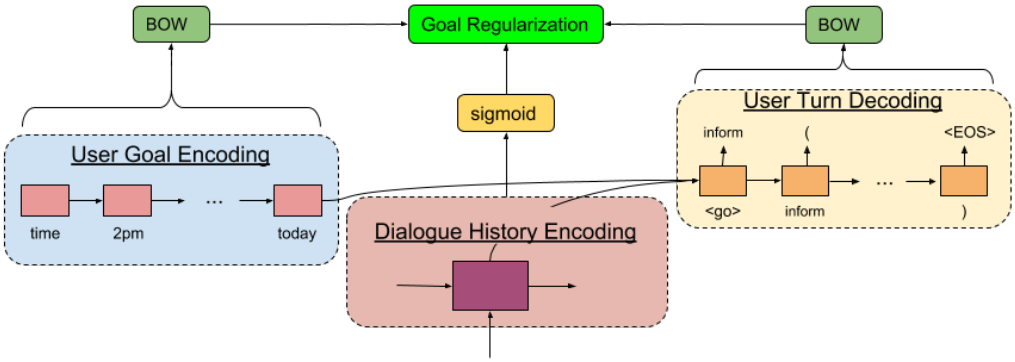


Figure 2.11: VHUSReg model: Divergence between user turns and user goal is regularized. Boxes are RNN cells, colors indicate parameter sharing.

dialogue history as

$$\mu_y = \tilde{W}_\mu h_t^D + \tilde{b}_\mu \tag{2.23}$$

$$\Sigma_y = \tilde{W}_\Sigma h_t^D + \tilde{b}_\Sigma \tag{2.24}$$

where $\tilde{W}_\mu$, $\tilde{W}_\Sigma$, $\tilde{b}_\mu$, and $\tilde{b}_\Sigma$ are new parameters for posterior distribution. We add the Kullback-Leibler divergence between trained prior and posterior distribution, i.e.,

$$L_{var} = \alpha KL(\mathcal{N}(z|\mu_x, \Sigma_x)|\mathcal{N}(z|\mu_y, \Sigma_y)),$$

to the cross-entropy loss, where α is a balancing parameter between the two losses.

Our main intuition is that, when the decoder is conditioned on a noisy history, it will generate a slightly different user turn. By penalizing the KL divergence between prior and posterior distributions, we control the level of noise by ensuring that the previous and current histories are consistent.

Goal Regularization

HUS and VHUS models learn the relationship between a user turn and a user goal only through the decoder loss which might generate considerably longer dialogues when user turns diverge from the initial user goal. Here, we introduce a new goal regularization approach called VHUSReg to eliminate the aforementioned problem (Figure 2.11). Our turn level encoders are exactly the same as in HUS, but instead of initializing the dialogue level RNN with user goal representation, we initialize it with zero vector and condition the decoder on user goal more directly. Furthermore, we penalize the divergence between dialogue level representations and user goal to enforce a more direct correspondence.

More formally, given the sequence of system turn encodings, we generate dialogue representation as

$$h_t^D = Enc(\{h_i^S\}_{i=1,\cdots,t}; \theta_D) \tag{2.25}$$

where the encoder is initialized with zeros. Next, we generate a new dialogue representation as

$$\hat{h}_t^D = FC([h_t^D; h^C]) \tag{2.26}$$

where new output is generated by blending user goal and old dialogue representation. Similarly, an RNN decoder is used to generate the user turn token sequence from new dialogue representation.

To regularize the divergence of user turn and user goal, we minimize the discrepancy between user turn and user goal tokens conditioned on the current system turn. We first generate a bag-of-words approximation of current user turn and current system turn:

$$b_t^D = FC(h_t^D) \tag{2.27}$$

$$b_t^S = FC(h_t^S) \tag{2.28}$$

where FC is a single layer neural network with sigmoid activation function. Following, we build a new bag-of-words representation for current user turn conditioned on the current system turn:

$$b_t^u = FC([b_t^D; b_t^S]) \tag{2.29}$$

Next, we introduce a new loss to minimize the divergence between initial user goal and user turn tokens by ensuring that each bag-of-words approximation is accurate :

$$L_{reg} = ||b_t^u - BOW(C)|| + ||b_t^D - BOW(U_t)|| + ||b_t^S - BOW(S_t)|| \tag{2.30}$$

where $BOW(x)$ is a function that outputs a bag-of-words vector for a set of tokens x, for example we use $BOW(C)$ to generate a bag-of-words vector for initial user goal. By minimizing the discrepancy between the bag-of-words representations in each term, we align the tokens in user goal and user turn.

Variational HUS with Goal Regularization

Finally, we combine both approaches in a hybrid framework and sum the three losses:

$$L = L_{crossent} + L_{var} + L_{reg} \tag{2.31}$$

2.3.5 Experimental Results and Discussion

We present our experimental results by systematically interacting each user simulator model with two different system policies, trained with supervised learning and reinforcement learning. We use a dataset from *movie ticket booking* domain and use several metrics to evaluate and compare our models, including task completion rates [77] and dialogue

length. We use template-based NLG to produce utterances for user and system action sequences, where we sample a template from a set of templates written by crowd workers for each dialogue act and the set of arguments that are possible for this application.

System Policy Models

For comparison, we train two state-of-the-art end-to-end system policy models: (i) supervised policy, and (ii) reinforcement learning policy.

Supervised policy is an end-to-end deep neural network that learns to map natural language user turns into system actions [63].

Reinforcement learning policy is first initialized with the supervised policy model and further fine-tuned by interacting with an agenda-based user simulator (different from our simulators) [63] to explore and generate more training data. Parameters of the neural network is updated using the REINFORCE algorithm [103]. At each turn a small negative reward (such as -1) is given and at the end of each dialogue a large positive reward is given (such as 20) if the dialogue is successfully terminated. As a result, RL policy tries to minimize the number of turns while achieving a high success rate.

These policies use natural language utterances as input and generate a sequence of system actions which are then converted into natural language using template-based generation. However, the user simulator only utilizes the dialogue acts and arguments of the system turns. In our user simulator, we also leverage a template-based NLG to generate utterances as input to system policies where templates were sampled from a template set (about 10000) collected from crowdworkers. Each policy is trained on a dataset different from ours and fixed. To maintain a fair comparison, we first randomly generate 1000 user goals. To evaluate policies and simulators, we generate a dialogue for all user goals by interacting each user simulator and policy pair.

Dataset

We use a task oriented dialogue corpus following [89]. At the beginning of each dialogue, a user goal is randomly generated and the dialogue is populated using a dialogue agent with finite state machine and an agenda based user simulator [86]. Each dialogue also incorporates a user personality with different cooperativeness and randomness settings. The training and testing datasets have 10,000 dialogues each. Maximum number of dialogue turns is set to 20 and at each turn there are at most 3 sequences of dialogue actions with at most 5 slot-value pairs.

Metrics

We use three metrics to assess the successful interactions between user simulators and system policies: exact goal match, partial goal match, and dialogue length.

- A simulated dialogue has an *exact goal match (EM)* score of 1 if the final slot-value pairs that system confirmed at the end of dialogue fully matches user's goal.
- *Partial goal match (PM)* score of a simulated dialogue is the number of correct slot-value pairs over the set of all possible slots in user goals.
- We also report numbers on *dialogue length*, the average number of turns (both user and system) per dialogue, to assess the effect of different setups such as inducing noise.

Two response diversity metrics are also utilized to measure the diversity of user responses and the robustness of system policies: entropy and perplexity of the dialogue acts of the user responses per system response.

	Exact Match (%)	Partial Match(%)	Dialogue Length	
HUS	75.67	94.3	12.03	**SL**
	94.69	98.27	7.45	**RL**
+ dialogue length	86.1	96.51	9.615	**SL**
	94.33	98.2	7.076	**RL**
VHUS	82.52	95.69	11.8005	**SL**
	95.53	98.43	7.803	**RL**
HUSReg	88.8	97.08	7.92	**SL**
	96.19	**98.56**	**6.878**	**RL**
VHUSReg	91.90	97.67	8.0555	**SL**
	95.98	98.52	6.905	**RL**

Table 2.11: Comparison of user simulation models and system policies using task completion and dialogue length metrics. Rows in gray show supervised learning based (SL) policy and rows in white show reinforcement learning based (RL) policy.

Training Details

We use randomly initialized word embeddings of size 150. We set state size of turn level and dialogue level RNN as 200. We train our models using Adam optimization method [51] with initial learning rate of 1e-3. We apply dropout with probability 0.5 and use mini-batches of size 32. We train our models for 10 epochs and choose the best model via validation.

Task Completion and Dialogue Length Results

In Table 2.11, we present our results related to task-completion and dialogue length metrics. We used 10 different user simulation and system policy pairs. In our earlier experiments, we also implemented a plain seq2seq user simulation model without explicit dialogue state tracking but we omitted from the paper due to having poor results.

Evaluating Dialogue Policies When compared to SL policy, RL policy outperforms SL policy and generates remarkably higher task completion rates. We observe that in the EM metric, the gap is larger than the PM metric. One plausible explanation is that SL policy gets stuck in a local minima and can not recover some of the slot-value pairs.

	Entropy	Perplexity	
HUS	0.075	1.053	**SL**
	0.119	1.086	**RL**
+ dialogue length	0.091	1.065	**SL**
	0.102	1.073	**RL**
VHUS	**0.284**	**1.218**	**SL**
	0.201	1.149	**RL**
HUSReg	0.018	1.012	**SL**
	0.0358	1.025	**RL**
VHUSReg	0.211	1.158	**SL**
	0.204	1.152	**RL**

Table 2.12: Comparison of user simulation models and system policies using response diversity metrics at the level of dialogue acts.

RL policy, on the other hand, is able to successfully produce the correct set of pairs. RL policy also yields considerably fewer turns per dialogue across all user simulators. When we decrease the complexity and effectiveness of user simulators (from bottom to top), the gap between RL and SL increases. The performance of RL policy on EM and PM metrics is not affected by different user simulators; however SL policy shows a drop when a weaker user is present. These observations indicate that RL is more robust to different types of users even when the responses are more stochastic (as in VHUS). When we exclude dialogue length from HUS, SL performance drops due to its lack of robustness to a weaker user. Note that these evaluation results obtained using fully automatic methods with a user simulator are in line with human evaluation results presented in [63].

Evaluating User Simulators We also compare user simulators and their components using the same set of policies and metrics. When we incorporate dialogue length, HUS is able to untangle the effect of user types on dialogue length and other factors which lead to more successful and shorter conversations. VHUS produces more diverse and more stochastic user responses which decreases the performance of SL policy and increases the average number of turns. Although, RL policy can recover from these more unpredictable situations, the average length of dialogues increase; more than 0.7 turns

Model	Average Score (Standard Deviation)
Agenda-based	4.56 (0.859)
HUS+dialogue length	4.86 (0.545)
VHUS	4.88 (0.472)
HUSReg	4.88 (0.452)
VHUSReg	4.83 (0.594)

Table 2.13: Comparison of user simulator models using real user evaluation.

per dialogue. Penalizing the user responses at each turn based on their divergence from the initial user goal generates more successful dialogues that are also shorter and more goal oriented. One reason for the increase in EM and PM is that goal regularization tends to generate the shortest conversations possible and effectively alleviates the shortcomings of policies such as infinite loops. When we augment HUSReg with the variational step, the performance of RL policy is not affected but the task completion rate of SL increases. We also observe that average dialogue length does not increase which is attributed to the inverse effect of goal regularization to produce shorter conversations.

Response Diversity Results

Using the same set of simulated dialogues, in Table 2.12, we present our results related to response diversity metrics. Note that entropy and perplexity measures are computed at the level of dialogue acts to measure the ability of each approach in producing diverse responses, and the measures are expected to increase significantly when acts are converted to natural language utterances.

Evaluating Dialogue Policies RL policy produces slightly higher entropy and perplexity scores and generates richer dialogues over all user simulations except variational simulations. We reason that inducing diversity into user responses is positively correlated with average dialogue length; hence RL policy will try to reduce very high uncertainty in user responses by balancing the number of turns and the total reward. We also ob-

serve that the diversity of user responses drops when SL policy interacts with VHUSReg instead of VHUS while it is very similar in RL policy. This similarity in RL policy performance can be associated to our reasoning above; RL policy tries to balance dialogue length and total reward which bounds the maximum diversity of user responses.

Evaluating User Simulators VHUS and VHUSReg produces higher diversity scores over all metrics and system policies which confirms our hypothesis that introducing a variational step increases user response diversity. The diversity scores on VHUSReg is smaller than VHUS in SL policy which is the result of the proposed goal regularization loss that controls the divergence of user responses from the initial goal. HUSReg also generates dialogues that are more succinct with the smallest diversity scores and the smallest dialogue lengths compared to other user simulators. When we incorporate dialogue length into HUS, we observe no change in diversity scores while improving task completion and dialogue length performances.

Evaluating with Real Users

We presented 100 dialogue subset of each dataset to crowdworkers for human evaluation. All turns of these dialogues were transformed into natural language by template based generation and annotated by 3 crowdworkers in a scale from 1 to 5 in terms of the clarity and appropriateness of that turn in the context of the conversation. Table 2.13 shows the turn-based average scores. We observe that all user simulation models generate highly successful turns. We also compare our user simulation models to a hand-crafted agenda-based user simulation model [86]. Our models have higher average user scores with less standard deviation compared to agenda-based user simulation. One particular explanation for this difference is that agenda-based simulation can generate out of context turns deviating from the dialogue history.

Chapter 3

Learning from Unstructured Data

In the previous chapter, we described our non-interactive and interactive deep neural models that can map natural language utterances to structured query representations while having multi-turn dialogues with users. Final queries are executed against the corresponding database and the answer is presented to users. In general, these structured databases are updated via collaborative human efforts where new entries are harvested from many sources including online text documents and user submitted contributions, and validated by individual users. Compared to structured databases, unstructured databases such as Wikipedia documents offer a richer set of information which is more dynamically updated.

In this chapter, we propose supervised and semi-supervised models for reading comprehension on long Wikipedia documents. Typically, RNNs suffer from modeling long-term dependencies when the documents are substantially longer. We alleviate this problem by chunking a long document into multiple overlapping windows and soft-selecting the window that gives the desired answer. When the size of the labeled dataset is small, the models are prone to overfitting and suffer from generalization to new questions and documents. We propose the first unsupervised Wikipedia encoding approach where chunked windows are collectively encoded and reconstructed using Recurrent Variational Auto-Encoders (RVAE). These unsupervised models are effectively utilized in

downstream supervised tasks such as reading comprehension using a novel progressive reviewer model which learn task-specific architectures without altering the pretrained unsupervised models.

3.1 Reading Comprehension on Wikipedia

Reading comprehension and question answering tasks over documents [102, 43, 79] require extracting precise information from documents conditioned on a given query. While a basic sequence to sequence (seq2seq) model [96] can perform these tasks by encoding a question and document sequence and decoding an answer sequence [45], it has some disadvantages. The answer may be encountered early in the text and need to be stored across all the further recurrent steps, leading to forgetting or corruption; Attention can be added to the decoder to solve this problem [43]. Even with attention, approaches based on Recurrent Neural Networks (RNNs) require a number of sequential steps proportional to the document length to encode each document position. Hierarchical reading models address this problem by breaking the document into sentences [19]. To mitigate these problems, we introduce a simpler hierarchical model that achieves state-of-the-art performance on our benchmark task without this linguistic structure, and use it as framework to explore semi-supervised learning for reading comprehension.

In the following, we describe our supervised and semi-supervised models for reading comprehension on Wikipedia. We first develop a hierarchical reader called Sliding-Window Encoder Attentive Reader (SWEAR) that circumvents the aforementioned bottlenecks of existing readers. SWEAR, illustrated in Figure 3.1, first encodes each question into a vector space representation. It then chunks each document into overlapping, fixed-length windows and, conditioned on the question representation, encodes each window in parallel. Inspired by recent attention mechanisms such as Hermann et al. [43], SWEAR

attends over the window representations and reduces them into a single vector for each document. Finally, the answer is decoded from this document vector. Our results show that SWEAR outperforms the previous state-of-the-art on the supervised WikiReading task [45], improving Mean F1 to 76.8 from the previous 75.6 [19].

While WikiReading is a large dataset with millions of labeled examples, many applications of machine reading have a much smaller number of labeled examples among a large set of unlabeled documents. To model this situation, we constructed a semi-supervised version of WikiReading by downsampling the labeled corpus into a variety of smaller subsets, while preserving the full unlabeled corpus (i.e., Wikipedia). To take advantage of the unlabeled data, we evaluated multiple methods of reusing unsupervised recurrent autoencoders in semi-supervised versions of SWEAR. Importantly, in these models we are able to reuse all the autoencoder parameters *without fine-tuning*, meaning the supervised phase only has to learn to condition the answer on the document and query. This allows for more efficient training and online operation: Documents can be encoded in a single pass offline and these encodings reused by all models, both during training and when answering queries. Our semi-supervised learning models achieve significantly better performance than supervised SWEAR on several subsets with different characteristics. The best-performing model reaches 66.5 with 1% of the WikiReading dataset, compared to the 2016 state of the art of 71.8 (with 100% of the dataset).

3.2 Problem Description

Following the recent progress on end-to-end supervised question answering [43, 79], we consider the general problem of predicting an answer A given a query-document pair (Q, D). We do not make the assumption that the answer should be present verbatim in the document.

3.2.1 Supervised Version

Given a document $D = \{d_1, d_2, \cdots, d_{N_D}\}$ and a query $Q = \{q_1, q_2, \cdots, q_{N_Q}\}$ as sequences of words, our task is to generate a new sequence of words that matches the correct answer $A = \{a_1, a_2, \cdots, a_{N_A}\}$. Because we do not assume that A is a subsequence of D, the answer may require blending information from multiple parts of the document, or may be precisely copied from a single location. Our proposed architecture supports both of these use cases.

The WikiReading dataset [45], which includes a mix of categorization and extraction tasks, is the largest dataset matching this problem description. In WikiReading, documents are Wikipedia articles, while queries and answers are Wikidata properties and values, respectively. Example Wikidata property-value pairs are `(place of birth, Paris)`, `(genre, Science Fiction)`. The dataset contains 18.58M instances divided into training, validation, and test with an 85/10/5 split. The answer is present verbatim in the document only 47.1% of the time, severely limiting models that label document spans, such as those developed for the popular SQUAD dataset [79].

3.2.2 Semi-Supervised Version

We also consider a semi-supervised version of the task, where an additional corpus of documents without labeled (Q, A) pairs is available. Taking advantage of the large size of the WikiReading dataset, we created a series of increasingly challenging semi-supervised problems with the following structure:

- **Unsupervised:** The entire document corpus (about 4M Wikipedia articles), with queries and answers removed.
- **Supervised:** Five smaller training sets created by sampling a random (1%, 0.5%, 0.1%) of the WikiReading training set, and taking (200, 100) random samples from

each property in the original training set.

3.3 Supervised Model Architecture

We now present our model, called Sliding-Window Encoder Attentive Reader (SWEAR), shown in Figure 3.1, and describe its operation in a fully supervised setting. Given a (Q, D) pair, the model encodes Q into a vector space representation with a Recurrent Neural Network (RNN). The first layer of the model chunks the document D into overlapping, fixed-length windows and encodes all windows in parallel with an RNN conditioned on the question representation. The second layer attends over the window representations, reducing them into a single vector representing the latent answer. Finally, the answer sequence A is decoded from this vector using an RNN sequence decoder.

3.3.1 Preliminaries and Notation

Each word w comes from a vocabulary V and is associated with a vector e_w which constitutes the rows of an embedding matrix E. We denote by e^D, e^Q, and e^A the vector sequences corresponding to the document, question, and answer sequences, respectively. More specifically, we aim at obtaining vector representations for documents and questions, then generating the words of the answer sequence.

Our model makes extensive use of RNN encoders to transform sequences into fixed length vectors. For our purposes, an RNN encoder consists of GRU units [16] defined as

$$h_t = f(x_t; h_{t-1}; \theta) \tag{3.1}$$

where h_t is hidden state at time t. f is a nonlinear function operating on input vector x_t and previous state, h_{t-1} with θ being its parameter vector. Given an input sequence,

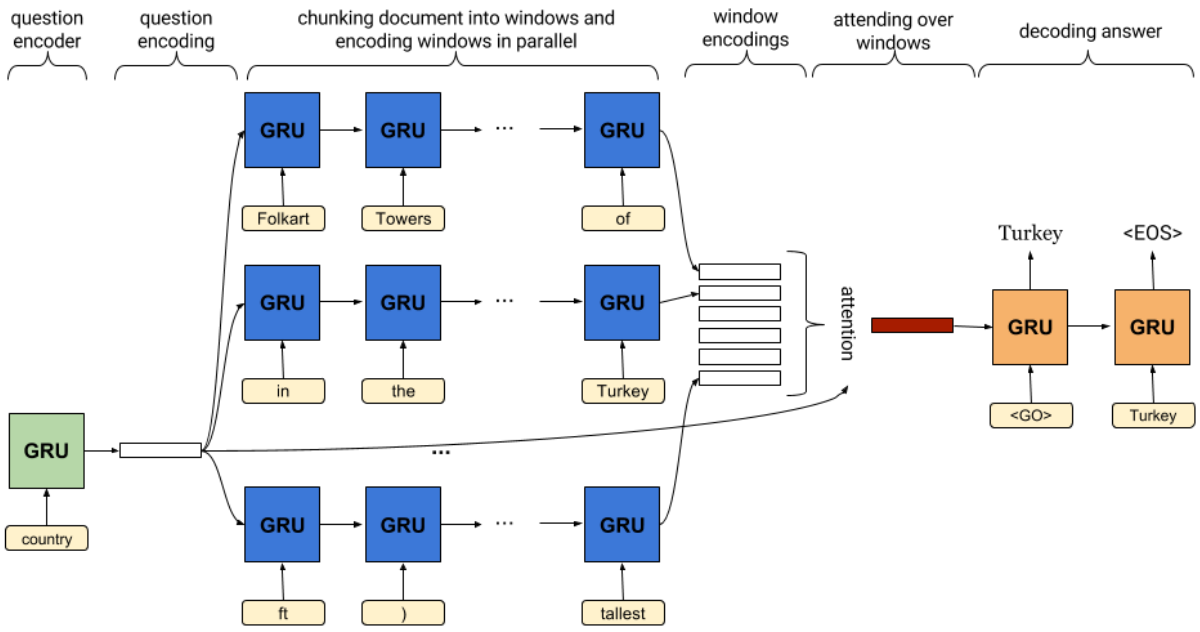


Figure 3.1: SWEAR model: Boxes are RNN cells, colors indicate parameter sharing.

the encoder runs over the sequence of words producing the hidden vectors at each step. We refer to the last hidden state of an RNN encoder as the *encoding* of a sequence.

3.3.2 Sliding Window Recurrent Encoder

The core of the model is a sequence encoder that operates over sliding windows in a manner analogous to a traditional convolution. Before encoding the document, we slide a window of length l with a step size s over the document and produce $n = \lfloor \frac{N_D - l}{s} \rfloor$ document windows. This yields a sequence of sub-documents $(D_1, D_2, \cdots, D_n)$, where each D_i contains a subsequence of l words from the original document D. Intuitively, a precise answer may be present verbatim in one or more windows, or many windows may contain evidence suggestive of a more categorical answer.

Next, the model encodes each window conditioned on a question encoding. We first

encode the question sequence once using a RNN (Enc) as

$$h^q = Enc(e^Q; \theta_Q) \tag{3.2}$$

where h^q is the last hidden state and θ_Q represents the parameters of the question encoder. Initialized with this question encoding, we employ another RNN to encode each document window as

$$\begin{aligned} h^w_{i,0} &= h^q \\ h^w_i &= Enc(e^{D_i}; \theta_W) \end{aligned} \tag{3.3}$$

where $h^w_{i,0}$ is the initial hidden state, h^w_i is the last hidden state, and θ_W represents the parameters of the window encoder. θ_W is shared for every window and is decoupled from θ_Q. As the windows are significantly smaller than the documents, encodings of windows will reflect the effect of question encodings better, mitigating any long-distance dependency problems.

3.3.3 Combining Window Encodings

SWEAR attends over the window encoder states using the question encoding to produce a single vector h^d for the document, given by

$$p_i \propto \exp(u_R^T \, tanh(W_R[h^w_i, h^q])) \tag{3.4}$$

$$h^d = \sum_i p_i h^w_i \tag{3.5}$$

where [.] is vector concatenation, and p_i is the probability window i is relevant to answering the question. W_R and u_R are parameters of the attention model.

Model	Mean F1
Placeholder seq2seq (HE16)	71.8
SoftAttend (CH17)	71.6
Reinforce (CH17)	74.5
Placeholder seq2seq (CH17)	75.6
SWEAR (w/ zeros)	76.4
SWEAR	**76.8**

Table 3.1: Results for SWEAR compared to top published results on the WikiReading test set.

3.3.4 Answer Decoding

Given the document encoding h^d, an RNN decoder (Dec) generates the answer word sequence:

$$h_0^a = h^d$$

$$h_t^a = Dec(h_{t-1}^a; \omega_A) \tag{3.6}$$

$$P(a_t^* = w_j) \propto \exp(e_j^T (W_A h_t^a + b_A)) \tag{3.7}$$

$$a_t^* = argmax_j(P(a_t^* = w_j)) \tag{3.8}$$

where h_0^a is the initial hidden state and h_t^a is the hidden vector at time t. $A^* = \{a_1^*, a_2^*, \cdots, a_{N_A}^*\}$ is the sequence of answer words generated. W_A, b_A, and ω_A are the parameters of the answer decoder. The training objective is to minimize the average cross-entropy error between the candidate sequence A^* and the correct answer sequence A.

3.3.5 Supervised Results

Before exploring unsupervised pre-training, we present summary results for SWEAR in a fully supervised setting, for comparison to previous work on the WikiReading task,

	HE16 Best	SWEAR
Categorical	**88.6**	**88.6**
Relational	56.5	**63.4**
Date	73.8	**82.5**

Table 3.2: Mean F1 for SWEAR on each type of property compared with the best results for each type reported in Hewlett et al. (2016), which come from different models. Other publications did not report these sub-scores.

Doc length	pct	seq2seq	SWEAR	imp
$[0, 200)$	44.6	79.7	80.7	1.2
$[200, 400)$	19.5	76.7	77.8	1.5
$[400, 600)$	11.0	74.5	76.3	2.3
$[600, 800)$	6.6	72.8	74.3	2.1
$[800, 1000)$	4.3	71.5	72.8	1.8
$[1000, max)$	14.0	64.8	65.9	1.7

Table 3.3: Comparison of Mean F1 for SWEAR and a baseline seq2seq model on the WikiReading test set across different document length ranges. *pct* indicates the percentage of the dataset falling in the given document length range. *imp* is the percentage improvement of SWEAR over baseline.

namely that of Hewlett et al. [45] and Choi et al. [19], which we refer to as HE16 and CH17 in tables. For further experiments, results, and discussion see Section 3.5.2. Table 3.1 shows that SWEAR outperforms the best results for various models reported in both publications, including the hierarchical models SoftAttend and Reinforce presented by Choi et al. Choi et al. [19].[1] Interestingly, SoftAttend computes an attention over sentence encodings, analogous to SWEAR's attention over overlapping window encodings, but it does so on the basis of less powerful encoders (BoW or convolution vs RNN), suggesting that the extra computation spent by the RNN provides a meaningful boost to performance.

To quantify the effect of initializing the window encoder with the question state, we report results for two variants of SWEAR: In *SWEAR* the window encoder is initialized with the question encoding, while in *SWEAR w/ zeros*, the window encoder is initialized

[1]Document lengths differ between publications: We truncate documents to the first 600 words, while Choi et al. truncate to 1000 words or 35 sentences and Hewlett et al. truncate to 300 words.

with zeros. In both cases the question encoding is used for attention over the window encodings. For SWEAR w/ zeros it is additionally concatenated with the document encoding and passed through a 2-layer fully connected neural network before the decoding step. Conditioning on the question increases Mean F1 by 0.4.

Hewlett et al. [45] grouped properties by answer distribution: *Categorical* properties have a small list of possible answers, such as countries, *Relational* properties have an open set of answers, such as spouses or places of birth, and *Date* properties (a subset of relational properties) have date answers, such as date of birth. We reproduce this grouping in Table 3.2 to show that SWEAR improves performance for Relational and Date properties, demonstrating that it is better able to extract precise information from documents.

Finally, we observe that SWEAR outperforms a baseline seq2seq model on longer documents, as shown in Table 3.3. The baseline model is roughly equivalent to the best previously-published result, Placeholder seq2seq (CH17) in Table 3.1, reaching a Mean F1 of 75.5 on the WikiReading testset. SWEAR improves over the baseline in every length category, but the differences are larger for longer documents.

3.4 Semi-Supervised Model Architecture

We now describe semi-supervised versions of the SWEAR model, to address the semi-supervised problem setting described in Section 3.2.2. A wide variety of approaches have been developed for semi-supervised learning with Neural Networks, with a typical scheme consisting of training an unsupervised model first, and then reusing the weights of that network as part of a supervised model. We consider each of these problems in turn, describing two types of unsupervised autoencoder models for sequences in Section 3.4.1 before turning to a series of strategies for incorporating the autoencoder weights into a

final supervised model in Section 3.4.3. All of these models reuse the autoencoder weights without modification, meaning a document can be encoded once by an offline process, and the resulting encodings can be used both during training and to answer multiple queries online in a more efficient manner.

3.4.1 Recurrent Autoencoders for Unsupervised Pre-training

Autoencoders are models that reconstruct their input, typically by encoding it into a latent space and then decoding it back again. Autoencoders have recently proved useful for semi-supervised learning [22, 25]. We now describe two autoencoder models from the recent literature that we use for unsupervised learning. The Recurrent Autoencoder (RAE) is the natural application of the seq2seq framework [96] to autoencoding documents [22]: In seq2seq, an encoder RNN already produces a latent representation h_N, which is used to initialize a decoder RNN. In RAE, the output sequence is replaced with the input sequence, so learning minimizes the cross-entropy between the reconstructed input sequence and the original input sequence. Encoder and decoder cells share parameters θ_U.

Variational Recurrent Autoencoder

The Variational Recurrent Autoencoder (VRAE), introduced by Fabius et al. Fabius and van Amersfoort [25], is a RAE with a variational Bayesian inference step where an unobserved latent random variable generates the sequential data. The encoder and decoder are exactly the same as RAE, but the latent state h_N is not directly passed to the decoder. Instead, it is used to estimate the parameters of a Gaussian distribution with a diagonal covariance matrix: The mean is given by $\mu_x = W_\mu h_N + b_\mu$ and the covariance by $\Sigma_x = W_\Sigma h_N + b_\Sigma$, where W_μ, W_Σ, b_μ, and b_Σ are new variational step

parameters. The decoder is initialized with a single vector sampled from this distribution, $z_x \sim \mathcal{N}(z|\mu_x, \Sigma_x)$. For VRAE, the Kullback-Leibler divergence between trained Normal distribution and standard normal distribution, i.e., $KL(\mathcal{N}(\mu_x, \Sigma_x)|\mathcal{N}(0, I))$, is added to the loss.

Window Autoencoders

We take advantage of the SWEAR architecture by training autoencoders for text windows, as opposed to the standard document autoencoders. These autoencoders operate on the same sliding window subsequences as the supervised SWEAR model, autoencoding all subsequences independently and in parallel. This makes them easier to train as they only have to compress short sequences of text into a fixed-length representation. As the task of autoencoding is independent from our supervised problem, we refer to the generated encodings as *global encodings*.

3.4.2 Baseline: Initialization with Autoencoder Embeddings

Our baseline approach to reusing an unsupervised autoencoder in SWEAR is to initialize all embeddings with the pre-trained parameters and fix them. We call this model SWEAR-SS (for semi-supervised). The embedding matrix is fixed to the autoencoder embeddings. All other parameters are initialized randomly and trained as in the fully supervised version. We found that initializing the encoders and decoder with autoencoder weights hurts performance.

3.4.3 Reviewer Models

Unfortunately, this baseline approach to semi-supervised learning has significant disadvantages in our problem setting. Pre-trained RNN parameters are not fully exploited

since we observed catastrophic forgetting when initializing and fine-tuning SWEAR with pre-trained weights. This includes fixing window encoder parameters with autoencoders and only fine-tuning question encoders. Second, conditioning the window encoders on the question eliminates the possibility to train window representations offline and utilize them later which causes a significant overhead during testing.

Inspired by recent trends in deep learning models such as Progressive Neural Networks [84] and Reviewer Models [114], we propose multiple solutions to these problems. All of the proposed models process text input first through a fixed *autoencoder layer*: fixed pre-trained embeddings and fixed RNN encoder parameters, both initialized from the autoencoder weights. Above this autoencoder layer, we build layers of abstraction that learn to adapt the pre-trained models to the QA task.

Multi-Layer Reviewer (SWEAR-MLR)

The most straightforward extension to the baseline model is to fix the pretrained autoencoder RNN as the first layer and introduce a second, trainable *reviewer layer*. To make this approach more suitable for question answering, reviewer layers utilize corresponding global encodings as well as hidden states of the pre-trained autoencoders as input (Figure 3.2). The aim is to review both pre-trained question and window encodings to compose a single vector representing the window conditioned on the question.

Encoding questions: The question is first encoded by the autoencoder layer, $\tilde{h}^q = Enc(e^Q; \theta_U)$ where both word embeddings (E and e^Q) and encoder (θ_U) are fixed and initialized with pretrained parameters. A second, learnable RNN layer then takes the output of the autoencoder layer and corresponding input embeddings as input and produces the final question encoding, $h^q = Enc(FC([e^Q, \tilde{h}^q, \tilde{h}_t]); \theta_Q)$ where FC is a fully connected layer with ReLU activation function, and $\tilde{h}_t$ is the output of the autoencoder layer at time step t. Figure 3.3 illustrates a single time-step of the question encoder.

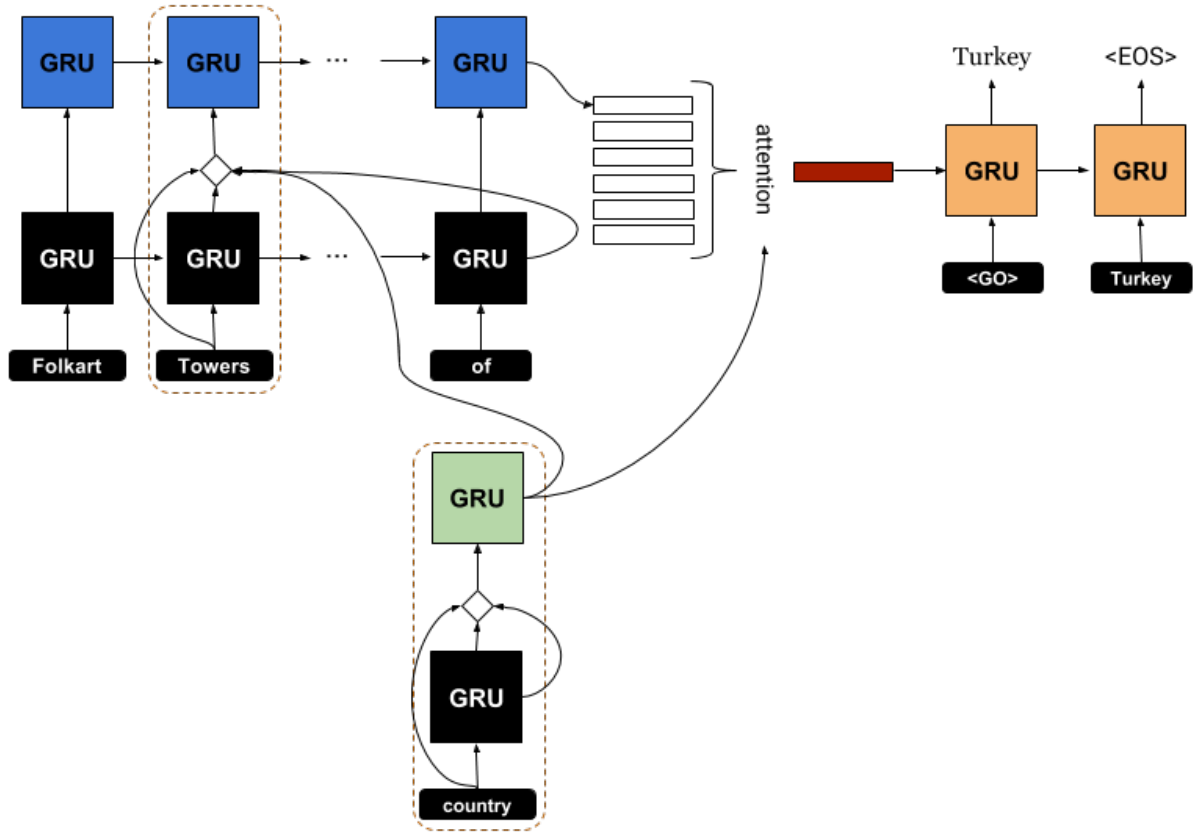


Figure 3.2: Multi-layer reviewer model (SWEAR-MLR), shown operating over a single window: Black boxes are RNN cells with fixed weights copied from the autoencoder, diamonds indicate vector concatenation with adapter and FC layers. For simplicity, only the cells in dashed boxes are fully illustrated (detailed in Figure 3.3), but the same structure is repeated for each cell.

Encoding windows: Similarly, windows are encoded first by the fixed autoencoder layer and then by a reviewer layer, $\tilde{h}_i^w = Enc(e^{D_i}; \theta_U)$ and $h_i^w = Enc(FC([e^{D_i}, \tilde{h}_i^w, \tilde{h}_t^w, h^q]); \theta_W)$ where $\tilde{h}_t^w$ is the output of the autoencoder layer at time step t. Unlike supervised SWEAR, in SWEAR-MLR the window encoder is not initialized with the question encoder state. Instead, the question encoder state is an additional input to each unit in the reviewer layer (illustrated as the dashed line in Figure 3.3). Intuitively, the reviewer layer should reuse the global window and question information and encode only information relevant to the current question.

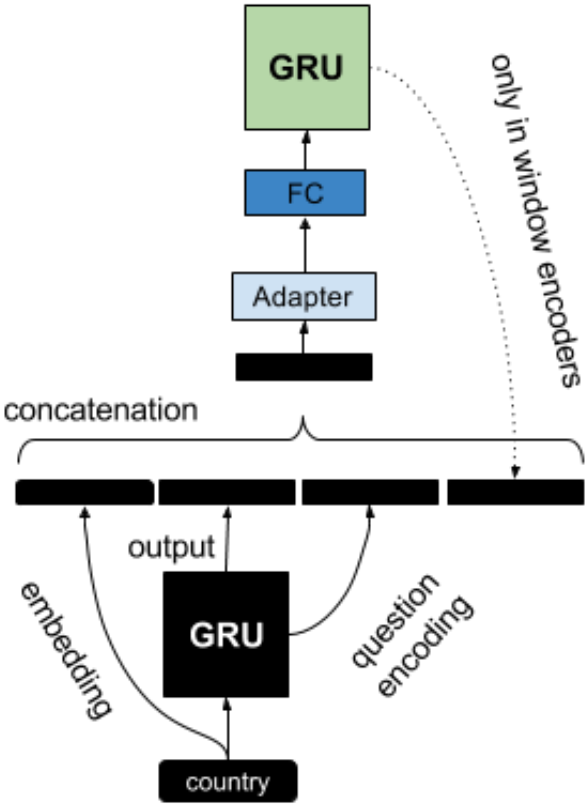


Figure 3.3: Detailed illustration of the dashed box in SWEAR-MLR question encoder. Black boxes are fixed parameters and encodings. The window encoder is similar, except that the output of the question reviewer layer is also added to the concatenated input (dashed line).

Progressive Reviewer (SWEAR-PR)

Although the reviewer layer in SWEAR-MLR has global window and question encodings as input, it requires a number of sequential steps equal to the window size, plus any additional reviewer steps. The reviewer layer also has to re-encode windows for each question, which is not ideal for online use. To address these issues, we now present a *Progressive Reviewer* model (SWEAR-PR) that reviews the outputs of the encoders using a separate RNN that is decoupled from the window size (Figure 3.4).

Encoding questions and windows: Similar to SWEAR-MLR, SWEAR-PR first encodes the questions and windows independently using autoencoder layers, $\tilde{h}^q = Enc(e^Q; \theta_U)$

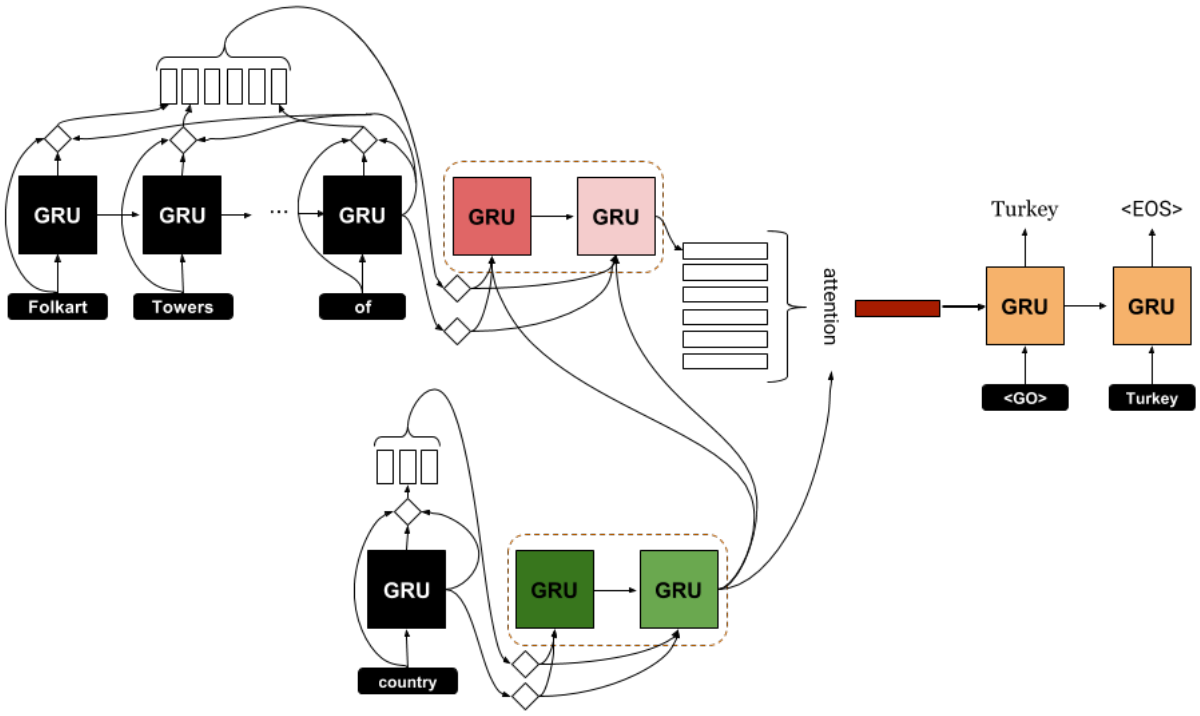


Figure 3.4: Progressive reviewer model (SWEAR-PR), shown operating over a single window: Black boxes are RNN cells with fixed weights copied from the autoencoder, diamonds indicate vector concatenation with adapter and FC layers. Dashed boxes contain reviewer layers. Cells within reviewer layers are decoupled as indicated by different colors.

and $\tilde{h}_i^w = Enc(e^{D_i}; \theta_U)$. To decouple the question and window encoders, however, SWEAR-PR does not have a second layer as a reviewer.

Reviewing questions and windows: SWEAR-PR employs two other RNNs to review the question and window encodings and to compose a single window representation conditioned on the question. Question reviewer takes the same pre-trained question encoding at each time step and attends over the hidden states and input embeddings of the pre-trained question encoder, $h^q = AttnEnc(FC(\tilde{h}^q); FC([\tilde{h}_t^q, e^Q]); \theta_Q)$ where $AttnEnc$ is an RNN with an attention cell which is illustrated in Figure 3.5. Outputs of the fixed autoencoder layer and fixed word embeddings, $[\tilde{h}_t^q, e^Q]$, are the attendable states. Window reviewer on the other hand takes the pre-trained window encoding and reviewed

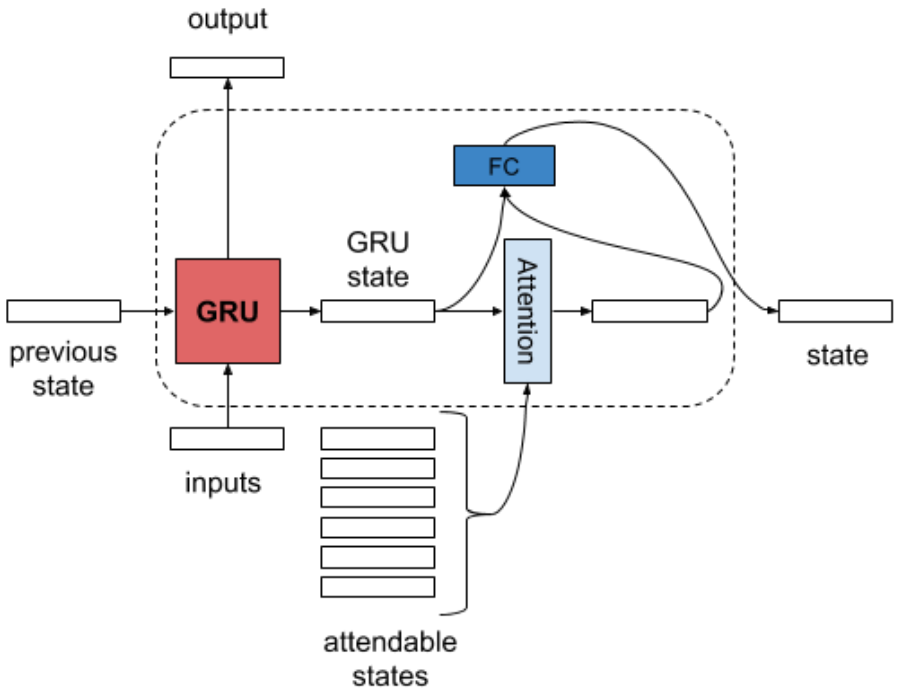


Figure 3.5: Illustration of attention cell: GRU state is used to attend over attendable states, then final state is computed by concatenating GRU state with context vector and passing through a fully connected neural network.

question encoding at each time step and attends over the hidden states of pre-trained window encoder, $h_i^w = AttnEnc(FC([\tilde{h}_i^w, h^q]); FC([\tilde{h}_t^w, e^{D_i}]); \theta_W)$ where outputs of the fixed autoencoder layer and fixed word embeddings, $[\tilde{h}_t^w, e^{D_i}]$, are the attendable states. As length of the windows is smaller than length of the reviewers, SWEAR-PR has significantly smaller overhead compared to other supervised and semi-supervised SWEAR variants.

Shared Components

Reducing window encodings and decoding: As in the supervised case described in 3.4, both reviewer models attend over the window encodings using the question encoding and reduce them into a single document encoding. Identical to answer decoding described in 3.6, the answer is decoded using another RNN taking the document state

as the initial state. The parameters of this answer decoder are initialized randomly.

Adapter layer: As the distribution and scale of parameters may differ significantly between the autoencoder layer and the reviewer layer, we use an *adapter layer* similar to the adapters in Progressive Neural Networks [84] to normalize the pre-trained parameters:

$$W_{out} = a * tanh(b * W_{in}) \tag{3.9}$$

where a and b are scalar variables to be learnt and W_{in} is a pre-trained input parameter. We put adapter layers after every pre-trained parameter connecting to a finetuned parameter such as on the connections from pre-trained embeddings to reviewer layer. We use dropout [92] regularization on both inputs and outputs of the reviewer cells.

3.5 Experimental Evaluation

As described in Section 3.2, we evaluate our models on the WikiReading task. In Section 3.3.5 we presented results for the supervised SWEAR on the full WikiReading dataset, establishing it as the highest-scoring method so far developed for WikiReading. We now compare our semi-supervised models SWEAR-MLR and SWEAR-PR over various subsets of the WikiReading dataset, using SWEAR as a baseline.

3.5.1 Experimental Setup

Following Hewlett et al. [45], we use the *Mean F1* metric for WikiReading, which assigns partial credit when there are multiple valid answers. We ran hyperparameter tuning for all models and report the result for the configuration with the highest Mean F1 on the validation set.

The supervised SWEAR model was trained on both the full training (results reported

Model	1%	0.5%	0.1%
SWEAR	63.5	57.6	39.5
SWEAR-SS (RAE)	64.7	62.8	55.3
SWEAR-SS (VRAE)	**65.7**	**64.0**	**60.7**

Table 3.4: Mean F1 results for SWEAR (fully supervised) and SWEAR-SS (semi-supervised) trained on 1%, 0.5%, and 0.1% subsets, respectively. Variants of SWEAR-SS indicate different sources of fixed encoder weights. [2]

in Section 3.3.5) and on each subset of training data (results reported below). Unsupervised autoencoders were trained on all documents in the WikiReading training set. We selected the autoencoder with the lowest reconstruction error for use in semi-supervised experiments. After initialization with weights from the best autoencoder, learnable parameters in the semi-supervised models were trained exactly as in the supervised model.

Training Details

We implemented all models in a shared framework in TensorFlow [1]. We used the Adam optimizer [51] for all training, periodically halving the learning rate according to a hyperparameter. Models were trained for a maximum of 4 epochs.

Table 3.7 shows which hyperparameters were tuned for each type of model, and the range of values for each hyperparameter. The parameters in the second group of the table are tuned for supervised SWEAR and the best setting (shown in bold) was used for other models where applicable. We fixed the batch size to 8 for autoencoders and 64 for semi-supervised models. We used a truncated normal distribution with a standard deviation of 0.01 for VRAE.

3.5.2 Results and Discussion

Table 3.4 and 3.5 show the results of SWEAR and semi-supervised models with pretrained and fixed embeddings. Results show that SWEAR-SS always improves over

[2]Initialization with Word2Vec [70] embeddings on 1% subset gives 64.0 Mean F1 score.

Model	**100**	**200**
SWEAR	25.0	33.0
SWEAR-SS (VRAE)	**39.0**	**45.0**

Table 3.5: Results for SWEAR and the best SWEAR-SS initialization (VRAE) trained on 100- and 200- per-property subsets, respectively.

Model	**Mean F1**
SWEAR-PR	**66.5**
dropout on input only	65.4
no dropout	64.6
shared reviewer cells	63.8
SWEAR-MLR	63.0
w/o skip connections	60.0

Table 3.6: Results for semi-supervised reviewer models trained on the 1% subset of WikiReading.

SWEAR at small data sizes, with the difference become dramatic as the dataset becomes very small. VRAE pretraining yields the best performance. As training and testing datasets have different distributions in per-property subsets, Mean F1 for supervised and semi-supervised models drops compared to uniform sampling. However, initialization with pretrained VRAE model leads to a substantial improvement on both subsamples. We further experimented by initializing the decoder (vs. only the encoder) with pretrained autoencoder weights but this resulted in a lower Mean F1.

Table 3.6 shows the results of semi-supervised reviewer models. When trained on 1% of the training data, SWEAR-MLR and the supervised SWEAR model perform similarly. Without using skip connections between embedding and hidden layers, the performance drops. The SWEAR-PR model further improves Mean F1 and outperforms the strongest SWEAR-SS model, even without fine-tuning the weights initialized from the autoencoder.

The success of SWEAR-PR rests on multiple design elements working together, as shown by the reduced performance caused by altering or disabling them. Using dropout only on the inputs, or not using any dropout on reviewer cells, causes a substantial

Parameter	Space
All Models	
Learning Rate	[0.0001; 0.005]
Learning Rate Decay Steps	{25k, 50k}
Gradient Clip	[0.01; 1.0]
Supervised & Semi-Supervised	
Embedding Size	{128, **256**, 512}
RNN State Size	{256, **512**}
Window Size	{10, 20, **30**}
Batch Size	{64, **128**}
Dropout	{0.3, **0.5**, 0.8}
Autoencoders Only	
Embedding Sharing	{Input, **All**}
KL-weight	[0.0001;0.01]
Semi-Supervised Only	
Finetune Pretrained Parameters	{YES, NO}
Dropout	{0.7, 0.8, **0.9**}
Reviewer State Size	{256, **512**}
Question Reviewer Steps	{0, 2, 3}
Window Reviewer Steps	{2, 3, 5, 8}

Table 3.7: Hyperparameter search spaces for each model type. We use {...} to denote a set of discrete values and [...] to denote a continuous range. Following Hewlett et al. [45], we ran a random search over the possible configurations.

decrease in Mean F1 score (by 1.1 and 1.9, respectively). Configuring the model with many more review steps (15) but with a smaller hidden vector size (128) reduced Mean F1 to 62.5. Increasing the number of review steps for the question to 5 caused a decrease in Mean F1 of 2.1.

Chapter 4

Learning from Web Interfaces

In the previous chapters, we detailed our models for learning from structured and unstructured databases using hierarchical neural networks that also work well in low labeled data regimes. These databases are accessible by our models and arbitrary number of queries can be executed to collect rich set of examples to train models. While this setup is important for developing accurate models for specific domains, their generalizability is limited to well manufactured and scarcely available set of Web APIs and the accessibility of the underlying database. On the other hand, the Web offers a rich source of tasks with human-readable web interfaces that can expand the capabilities of current systems.

In this chapter, we introduce a range of guided deep reinforcement learning (RL) approaches that can navigate designated web pages by following natural language instructions. We face several challenges when training RL agents through trial-and-error — large state and action spaces with large number of vocabulary tokens, and sparse and delayed rewards which are only non-zero at the end of an episode. In Section 4.3, we first explain reinforcement learning formulation of our problem with state and action representations. We detail our baseline deep Q network and curriculum learning approaches to alleviate the aforementioned problems in Section 4.4. In Section 4.5, we develop a new meta-learning framework where a generative model is trained to generate new web navigation tasks and this model is used to guide a navigation agent towards more valuable

states. The proposed meta-learning framework generates new user instructions from goal states that are discovered by following arbitrary navigation policies.

4.1 Learning to Navigate Web Pages

We study the problem of training reinforcement learning agents to navigate the Web (*navigator agent*) by following certain instructions, such as *book a flight ticket* or *interact with a social media web site*, that require learning through large state and action spaces with sparse and delayed rewards. In a typical web environment, an agent might need to carefully navigate through a large number of web elements to follow highly dynamic instructions formulated from large vocabularies. For example, in the case of an instruction "Book a flight from WTK to LON on 21-Oct-2016", the agent needs to fill out the origin and destination drop downs with the correct airport codes, select a date, hit submit button, and select the cheapest flight among all the options. Note the difficulty of the task: The agent can fill-out the first three fields in any order. The options for selection are numerous, among all possible airport / date combination only one is correct. The form can only be submitted once all the three fields are filled in. At that point the web environment / web page changes, and flight selection becomes possible. Then the agent can select and book a flight. Reaching the true objective in these tasks through trial-and-error is cumbersome, and reinforcement learning with the sparse reward results in the majority of the episodes generating no signal at all. The problem is exacerbated when learning from large set of instructions where visiting each option could be infeasible. As an example, in the flight-booking environment the number of possible instructions / tasks can grow to more than 14 millions, with more than 1700 vocabulary words and approximately 100 web elements at each episode.

A common remedy for these problems is guiding the exploration towards more valu-

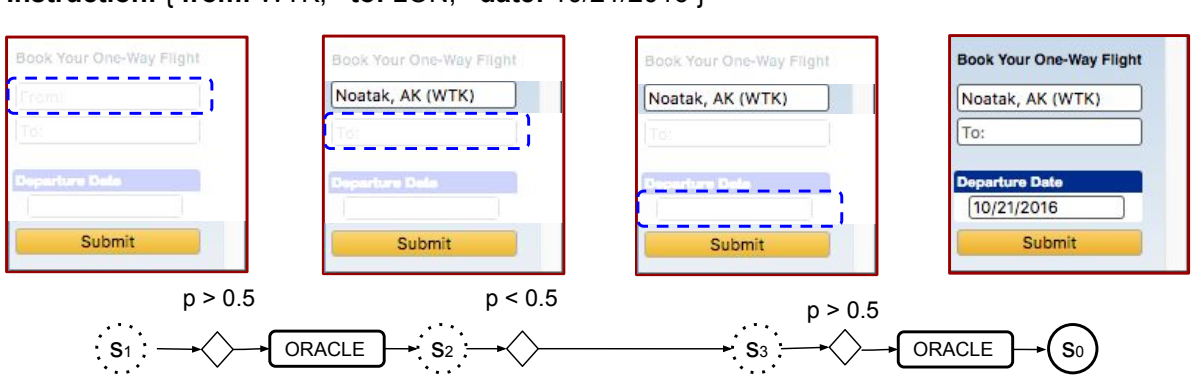


Figure 4.1: Curriculum learning. Final state is used as the initial state for training the navigator agent. The navigator is assigned only with the easier sub-instruction {to: "LON"}. We detail this process in Section 4.4.3.

able states by learning from human demonstrations and using pretrained word embeddings. Previous work ([64, 90]) has shown that the success rate of an agent on Web navigation tasks (Miniwob ([90])) can be improved via human demonstrations and pretrained word embeddings; however, they indeed use separate demonstrations for each environment and as the complexity of an environment increases, these methods fail to generate any successful episode (such as flight booking and social media interaction environments). But in environments with large state and action spaces, gathering the human demonstrations does not scale, as the training needs large number of human demonstrations for each environment.

In this section, we present two methods for reinforcement learning in large state and action spaces with sparse rewards for the web navigation. First, when expert demonstrations or an instruction-following policy (ORACLE) are available, we develop *curriculum-DQN*, a curriculum learning that guides the exploration by starting with an easier instruction following task and gradually increasing the difficulty over a number of training steps. Curriculum-DQN decomposes an instruction into multiple sub-instructions and assigns the web navigation agent (navigator) with an easier task of solving only a subset

of these sub-instructions (Figure 4.1). An expert instruction-following policy (ORACLE) places the agent and goal closer to each other.

Second, when demonstrations and ORACLE policies are not available, we present a novel *meta-learning* framework that trains a generative model for expert instruction-following demonstrations using an arbitrary web navigation policy *without instructions*. The key insight here is that we can treat an arbitrary navigation policy (e. g. random policy) as if it was an expert instruction-following policy for some hidden instruction. If we recover the underlying instruction, we can autonomously generate new expert demonstrations, and use them to improve the training of the navigator. Intuitively, generating an instruction from a policy is easier than following an instruction, as the navigator does not need to interact with a dynamic web page and take complicated actions. Motivated by these observations, we develop an instructor agent, a *meta-trainer*, that trains the navigator by generating new expert demonstrations.

In addition to the two trainers, *curriculum-DQN* and *instructor meta-trainer*, the paper introduces two novel neural network architectures for encoding web navigation Q-value functions, QWeb and INET, combining self-attention, LSTMs, and shallow encoding. QWeb serves as Q-value function for the learned instruction-following policy, trained with either *curriculum-DQN* or *instructor* agent. The INET is Q-value function for the instructor agent. We test the performance of our approaches on a set of Miniwob and Miniwob++ tasks ([64]). We show that both approaches improve upon a strong baseline and outperform previous state-of-the-art.

While we focus on the Web navigation, the methods presented here, automated curriculum generation with attention-equipped DQN, might be of interest to the larger task planning community working to solve goal-oriented tasks in large discrete state and action Markov Decision Processes.

4.2 Related Work

Our work is closely related to previous works on training reinforcement learning policies for navigation tasks. [90] and [64] both work on web navigation tasks and aim to leverage human demonstrations to guide the exploration towards more valuable states where the focus of our work is investigating the potential of curriculum learning approaches and building a novel framework for meta-training of our deep Q networks. Compared to our biLSTM encoder for encoding the Document Object Model (DOM) trees, a hierarchy of web elements in a web page, [64] encodes them by extracting spatial and hierarchical features. Diagonal to using pretrained word embeddings as in ([64]), we used shallow encodings to enhance learning semantic relationships between DOM elements and instructions. [89] also utilizes an attention-based DQN for navigating in home environments with visual inputs. They jointly encode visual representation of the environment and natural language instruction using attention mechanisms and generate Q values for a set of atomic actions for navigating in a 3D environment.

Modifying the reward function ([73, 2]) is one practice that encourages the agent to get more dense reward signals using potentials which also motivated our augmented rewards for Web navigation tasks. Curriculum learning methods ([8, 121, 35]) are studied to divide a complex task into multiple small sub-tasks that are easier to solve. Our curriculum learning is closely related to ([27]) where a robot is placed closer to a goal state in a 3D environment and the start state is updated based on robot's performance.

Meta-learning is used to exploit past experiences to continuously learn on new tasks ([4, 24, 101]). [28] introduced a meta-learning approach where task-specific policies are trained in a multi-task setup and a set of primitives are shared between tasks. Our meta-learning framework differs from previous works where we generate instruction and goal pairs to set the environment and provide dense rewards for a low level navigator agent

to effectively train.

4.3 Setup

We train a RL agent using DQN ([72]) that learns a value function $Q(s, a)$ which maps a state s to values over the possible set of actions a. At each time step, the agent observes a state s_t, takes an action a_t, and observes a new state s_{t+1} and a reward $r_t = r(s_{t+1}, a_t)$. DQN aims to maximize the sum of discounted rewards $\sum_t \gamma^t r_t$ by rolling out episodes as suggested by $Q(s, a)$ and accumulating the reward. We particularly focus on the case where the reward is sparse and only available at the end of an episode. More specifically, for only a small fraction of episodes that are successful, the reward is $+1$; in other cases it is -1. This setup combined with large state and action spaces make it difficult to train a Q learning model that can successfully navigate in a Web environment.

In this section, we further make the following assumption where we are given an instruction, $I = [F = (K, V)]$, as a list of fields F where each field is represented as a key-value pair (K, V) (ex. {*from*: "San Francisco", *to*: "LA", *date*: "12/04/2018"}). At each time step, the state of the environment s_t consists of the instruction I and a representation of the web page as a tree D_t of DOM elements (DOM tree). Each DOM element is represented as a list of named attributes such as *tag*, *value*, *name*, *text*, *id*, *class*. We also assume that the reward of the environment is computed by comparing the final state of an episode (D_N)

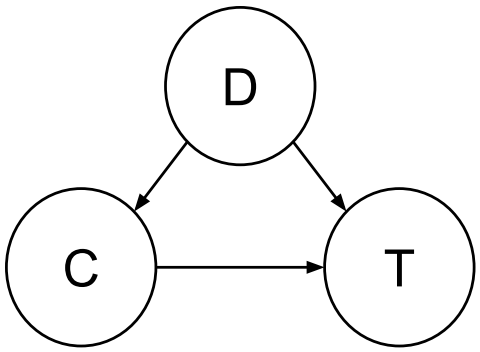


Figure 4.2: Action dependency graph for hierarchical Q learning. D denotes picking a DOM element, C denotes a click or type action, and T denotes generating a type sequence.

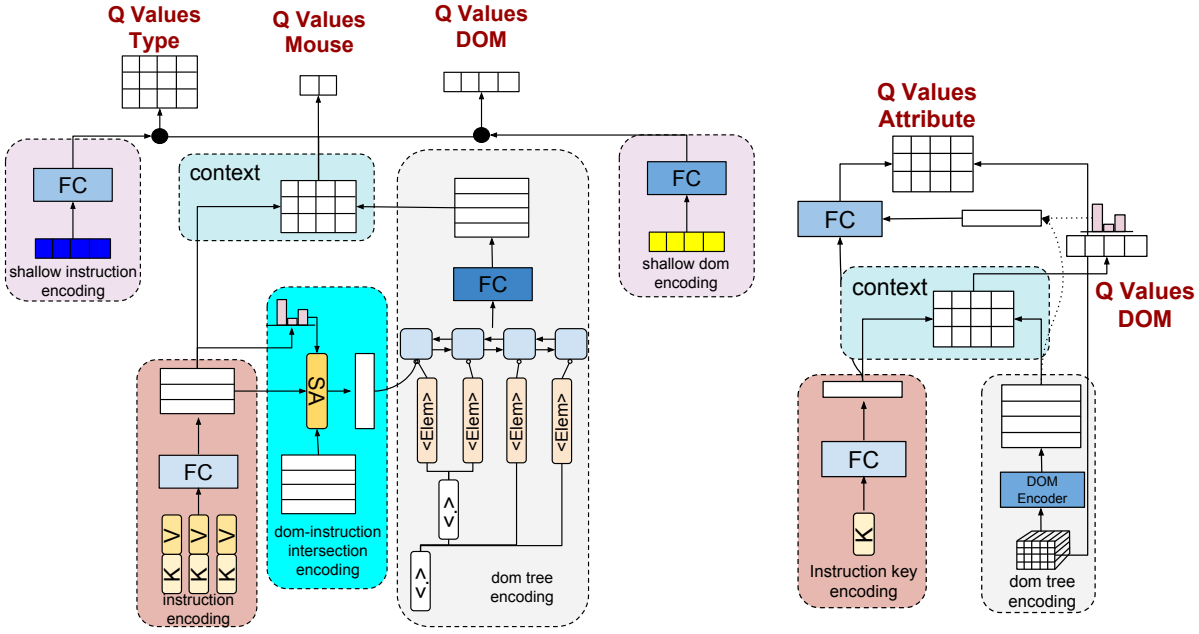
with the final goal state $G(I)$. Following [64], we constrain the action space to `Click(e)` and `Type(e, y)` actions where e is a leaf DOM element in the DOM tree and y is a value of a field from the instruction. Both of these composite actions are mostly identified by the DOM element (e), e.g., a *text box* is typed with a sequence whereas a *date picker* is clicked. This motivates us to represent composite actions using a hierarchy of atomic actions defined by the dependency graph in Figure 4.2. Following this layout, we define our composite Q value function by modeling each node in this graph considering its dependencies :

$$Q(s,a) = Q(s,a_D) + Q(s,a_C|a_D) + Q(s,a_T|a_D,[a_C == \text{"}type\text{"}]) \tag{4.1}$$

where $a = (a_D, a_C, a_T)$ is the composite action, a_D denotes selecting a DOM element, $a_C|a_D$ denotes a "click" or "type" action on the given DOM element, and $a_T|a_D, [a_C == \text{"}type\text{"}]$ denotes "typing a sequence from instruction" on the given DOM element. When executing the policy (during exploration or during testing), the agent first picks a DOM element with the highest $Q(s, a_D)$; then decides to `Type` or `Click` on the chosen DOM element based on $Q(s, a_C|a_D)$; and for a type action, the agent selects a value from the instruction using $Q(s, a_T|a_D, [a_C == \text{"}type\text{"}])$.

4.4 Guided Q Learning for Web Navigation

We now describe our proposed models for handling large state and action spaces with sparse rewards. We first describe our deep Q network, called QWeb, for generating Q values for a given observation ($s_t = (I, D_t)$) and for each atomic action a_D, a_C, a_T. Next, we explain how we extend this network with shallow DOM and instruction encoding layers to mitigate the problem of learning a large number of input vocabulary. Finally,



(a) QWeb network (b) INET network

Figure 4.3: Network architectures a) QWeb and b) INET with attribute pointing. Boxes indicate fully connected layers (FC) with ReLU (for instruction encoding) or tanh (for shallow encoding) activation. (K, V) indicates embeddings of key and value pairs on the instruction; <*Elem*> shows the leaf DOM element embeddings. SA denotes a self-attention mechanism that generates a distribution over the instruction fields. Black circles indicate the gating mechanism to join Q values generated by shallow and deep encodings.

we delve into our reward augmentation and curriculum learning approaches to solve the aforementioned problems.

4.4.1 Deep Q Network for Web Navigation (QWeb)

QWeb is composed of three different layers linked in a hierarchical structure where each layer encodes a different component of a given state (Figure 4.3a): (i) Encoding user instruction, (ii) Encoding the overlapping words between attributes of the DOM elements and instruction, and (iii) Encoding DOM tree.

Given an instruction $I = [F = (K, V)]$, QWeb first encodes each field F into a fixed

length vector by learning an embedding for each K and V. The sequence of overlapping words between DOM element attributes and instruction fields is encoded into a single vector to condition each element on contextually similar fields. Finally, the DOM tree is encoded by linearizing the tree structure ([100]) and running a bidirectional LSTM (biLSTM) network on top of the DOM elements sequence. Output of the LSTM network and encoding of the instruction fields are used to generate Q values for each atomic action.

Encoding User Instructions We represent an instruction with a list of vectors where each vector corresponds to a different instruction field. A field is encoded by encoding its corresponding key and value and transforming the combined encodings via a fully connected layer (FC) with $ReLU$ activation. Let $E^f_K(i,j)$ ($E^f_V(i,j)$) denote the embedding of the j-th word in the key (value) of i-th field. We represent a key or value as the average of these embeddings over the corresponding words, i.e., $E^f_K(i) = \frac{1}{|F(i)|}\sum_j E^f_K(i,j)$ represents the encoding of a key. Encoding of a field is then computed as follows : $E^f(i) = FC([E_K(i), E_V(i)])$ where [.] denotes vector concatenation.

Algorithm 1 Curriculum-DQN.

Input: ORACLE policy

Input: Sampling probability p and decay rate d

Web Environment, ENV_W

for number of training steps **do**

$(I, D, G) \leftarrow$ Sample instruction, initial, and goal states from environment.

//Warm-starting the episode.

for each DOM element e in D **do**

$\hat{D} \leftarrow$ Get new state from ORACLE(I, e) with probability p.

end for

// Collect experience and train QWeb starting from the final state $\hat{D}$*.*

Run Algorithm 2 (QWeb, I, $\hat{D}$, ENV_W)

Decrease p by a factor of d

end for

Encoding DOM-Instruction Intersection For each field in the instruction and each attribute of a DOM element, we generate the sequence of overlapping words. By encoding these sequences in parallel, we generate instruction-aware DOM element encodings. We average the word embeddings over each sequence and each attribute to compute the embedding of a DOM element conditioned on each instruction field. Using a self-attention mechanism, we compute a probability distribution over instruction fields and reduce this instruction-aware embedding into a single DOM element encoding. Let $E(f, D_t(i))$ denote the embedding of a DOM element conditioned on a field f where $D_t(i)$ is the i-th DOM element. Conditional embedding of $D_t(i)$ is the weighted average of these embeddings, i.e., $E_C = \sum_f p_f * E(f, D_t(i))$ where self-attention probabilities are computed as $p_f = softmax_i(u * E^f)$ with u being a trainable vector.

Encoding DOM Trees We represent each attribute by averaging its word embeddings. Each DOM element is encoded as the average of its attribute embeddings. Given conditional DOM element encodings, we concatenate these with DOM element embeddings to generate a single vector for each DOM element. We run a bidirectional LSTM (biLSTM) network on top of the list of DOM element embeddings to encode the DOM tree. Each output vector of the biLSTM is then transformed through a FC layer with tanh to generate DOM element representations.

Algorithm 2 One-step DQN training.

Input: DQN

Input: Goal state, G

Input: Set of initial states, $\hat{S}$

Input: Environment, ENV

//Run DQN policy to collect experience.

for $S \in \hat{S}$ **do**

$S_i \leftarrow S$

while environment is not terminated **do**

$o \leftarrow$DQN(G, S_i) to get logits over actions.

$a \leftarrow$Sample from a categorical distribution $Cat(o)$.

$(S_{i+1}, R_i) \leftarrow$ Run action a in the environment and collect new state and reward.

Add (S_i, a, s_{i+1}, R_i) to replay buffer.

$S_i \leftarrow S_{i+1}$

end while

end for

// Update DQN parameters.

for number of updates per episode **do**

Sample experience batch from replay buffer.

Update DQN parameters by running a single step of gradient descent.

end for

Generating Q Values Given encodings for each field in the instruction and each DOM element in the DOM tree, we compute the pairwise similarities between each field and each DOM element to generate a context matrix M. Rows and columns of M show the posterior values for each field and each DOM element in the current state, respectively.

By transforming through a FC layer and summing over the rows of M, we generate Q values for each DOM element, i.e., $Q(s_t, a_t^D)$. We use the rows of M as the Q values for typing a field from the instruction to a DOM element, i.e., $Q(s_t, a_t^T) = M$. Finally, Q values for *click* or *type* actions on a DOM element are generated by transforming the rows of M into 2 dimensional vectors using another FC layer, i.e., $Q(s_t, a_t^C)$. Final Q value for a composite action a_t is then computed by summing these Q values : $Q(s_t, a_t) = Q(s_t, a_t^D) + Q(s_t, a_t^T) + Q(s_t, a_t^C)$.

Incorporating Shallow Encodings In a scenario where the reward is sparse and input vocabulary is large, such as in flight-booking environments with hundreds of airports, it is difficult to learn a good semantic similarity using only word embeddings. We augment our deep Q network with shallow instruction and DOM tree encodings to alleviate this problem. A joint shallow encoding matrix of fields and elements is generated by computing word-based similarities (such as jaccard similarity, binary indicators such as subset or superset) between each instruction field and each DOM element attribute. We also append shallow encodings of siblings of each DOM element to explicitly incorporate DOM hieararchy.

Algorithm 3 Meta-learning for training QWeb.

instruction generation environment, ENV_I

// Training instructor agent INET.

for number of training steps **do**

(G, K_0, I) ←Sample goal (a DOM tree), initial state, and instruction from ENV_I.

Run Algorithm 2 ($INET$, I, (K_0, G), ENV_I)

end for

Fix the parameters of $INET$.

// QWeb meta-training via task generation.

for number of training steps **do**

// Generate new instruction following task.

(G, P) ← Run $RRND()$ and get goal state and sequence of state, G, action pairs, P.

Using P get a dense reward $\hat{R}$.

I ← Run $INET(G)$ and get instruction I.

Set up the ENV_{DR} with new task (I, G) and dense rewards.

D_0 ← Sample an initial state from the web environment.

// Collect experience and train QWeb in environment with dense rewards ENV_{DR}.

Run Algorithm 2 (QWeb, I, D_0, ENV_{DR})

end for

We sum over columns and rows of the shallow encoding matrix to generate a shallow input vector for DOM elements and instruction fields, respectively. These vectors are transformed using a FC layer with tanh and scaled via a trainable variable to generate a single value for a DOM element and a single value for an instruction field. Using a gating mechanism between deep Q values and shallow Q values, we compute final Q values as

follows:

$$\hat{Q}(s,a) = Q_{deep}(s_t, a_t^D)(1-\sigma(u)) + Q_{shallow}(s, a_t^D)(\sigma(u)) \tag{4.2}$$

$$\hat{Q}(s,a) = Q_{deep}(s_t, a_t^T)(1-\sigma(v)) + Q_{shallow}(s, a_t^T)(\sigma(v)) \tag{4.3}$$

where u and v are scalar variables learned during training.

4.4.2 Reward Augmentation

Following [73], we use potential based rewards for augmenting the environment reward function. Since the environment reward is computed by evaluating if the final state is exactly equal to the goal state, we define a potential function ($Potential(s, g)$) that counts the number of matching DOM elements between a given state (s) and the goal state (g); normalized by the number of DOM elements in the goal state. Potential based reward is then computed as the scaled difference between two potentials for the next state and current state:

$$R_{potential} = \gamma(Potential(s_{t+1}, g) - Potential(s_t, g)) \tag{4.4}$$

where g is the goal state. For example, in the flight-booking environment, there are 3 DOM elements that the reward is computed from. Let's assume that at current time step the agent correctly enters the date. In this case, the potential for the current state will increase by 1/3 compared to the potential of the previous state and the agent will receive a positive reward. Keep in mind that not all actions generate a non-zero potential difference. In our example, if the action of the agent is to click on the date picker but not to choose a specific date, the potential will remain unchanged.

4.4.3 Curriculum Learning

We perform curriculum learning by decomposing an instruction into multiple sub-instructions and assigning the agent with an easier task of solving only a subset of these sub-instructions. We practiced two different curriculum learning strategies to train QWeb: (i) Warm-starting an episode and (ii) Simulating sub-goals.

Warm-Start. We warm-start an episode by placing the agent closer to the goal state where the agent only needs to learn to perform a small number of sub-instructions to successfully finish the episode (illustrated in Algorithm 1). We independently visit each DOM element with a certain probability p and probe an ORACLE policy to perform a correct action on the selected element. The environment for the QWeb is initialized with the final state of the Warm-Start process and the original goal of the environment is kept the same. This process is also illustrated in Figure 4.1 for the flight-booking environment. In this example scenario, QWeb starts from the partially filled web form (origin and departure dates are already entered) and only tasked with learning to correctly enter the destination airport. At the beginning of the training, we start p with a large number (such as 0.85) and gradually decay towards 0.0 over a predefined number of steps. After this limit, the initial state of the environment will revert to the original state of plain DOM tree with full instruction.

Goal Simulation. We simulate simpler but related sub-goals for the agent by constraining an episode to a subset of the elements in the environment where only the corresponding sub-instructions are needed to successfully finish an episode. We randomly pick a subset of elements of size K and probe the ORACLE to perform correct set of actions on this subset to generate a sub-goal. The goal of the environment for QWeb will be assigned with the final state of this process and the initial state of the environment will remain unchanged. QWeb will receive a positive reward if it can successfully reach to

this sub-goal. At the beginning of the training, we start K with 1 and gradually increase it towards the maximum number of elements in the DOM tree over a number of steps. After this limit, the environment will revert to the original environment as in warm-start approach.

4.5 Meta-Trainer for Training QWeb

To address situations when the human demonstrations and ORACLE policy are not available, we combine the curriculum learning and reward augmentation under a more general unified framework, and present a novel meta-training approach that works in two phases. First, we learn to generate new instructions with a DQN agent, *instructor*. *Instructor* learns to recover instructions implied by a non-expert policy (e.g. rule-based or random). We detail the instructor agent, its training environment and neural network architecture in Section 4.5.1. Second, once the *instructor* is trained, we fix it and use it to provide demonstrations for the QWeb agent from a simple, non-expert, rule-based policy. This process, including the rule-based policy is described in Section 4.5.2. The whole training process is outlined in Algorithm 3.

4.5.1 Learning Instructions from Goal States

Instruction Generation Environment We define an *instruction state* by the sampled goal and a single *key (K)* sampled without replacement from the set of possible keys predefined for each environment. *Instruction actions* are composite actions: *select a DOM element as in web navigation environments* ($\hat{a}_t^D$) and *generate a value that corresponds to the current key (K),* ($\hat{a}_t^K$). For example, in flight-booking environment, the list of possible keys are defined by the set {*from, to, date*}. Let's assume that the current key is *from* and the agent selected the *departure* DOM element. A value can be generated

by copying one of the attributes of the departure element, e.g., *text : "LA"*. After each action, agent receives a positive *reward* (+1) if the generated value of the corresponding key is correct; otherwise a negative reward (-1). To train the *instructor*, we use the same sampling procedure used in the curriculum learning (Algorithm 2 to sample new pairs, but states, actions, and rewards in instruction generation environments are set up differently).

Deep Q Network for Instruction Generation (INET) We design a deep Q network, called INET and depicted in Figure 4.3b, to learn a Q value function approximation for the instruction generation environment. Borrowing the DOM tree encoder from QWeb, INET generates a vector representation for each DOM element in the DOM tree using a biLSTM encoder. Key in the environment state is encoded similarly to instruction encoding in QWeb where only the key is the input to the encoding layer. Q value for selecting a DOM element is then computed by learning a similarity between the key and DOM elements, i.e., $Q^I(s_t, \hat{a}_t^D)$ where Q^I denotes the Q values for the instructor agent. Next, we generate a probability distribution over DOM elements by using the same similarity between the key and DOM elements; and reduce their encodings into a single DOM tree encoding. Q values for each possible attribute is generated by transforming a context vector, concatenation of DOM tree encoding and key encoding, into scores over the possible set of DOM attributes, i.e., $Q^I(s_t, \hat{a}_t^K)$. Final Q values are computed by combining the two Q values : $Q^I(s_t, a_t) = Q^I(s_t, \hat{a}_t^D) + Q^I(s_t, \hat{a}_t^K)$.

4.5.2 Meta-Training of the QWeb (MetaQWeb)

We design a simple rule-based randomized policy (RRND) that iteratively visits each DOM element in the current state and takes an action. If the action is `Click(e)`, the agent on click on the element, and the process continues. If the DOM element is

part of a group, that their values depend on the state of other elements in the group (such as radio buttons), RRND clicks on one of them randomly and ignores the others. However, if the action is `Type(e, t)`, the type sequence is randomly selected from a given knowledge source (KS). Consider the flight-booking environment as an example, if the visited element is a text box, RRND randomly picks an airport from a list of available airports and types it into the text box. RRND stops after each DOM element is visited and final DOM tree (D) is generated. Using pretrained INET model, we generate an instruction I from D and set up the web navigation environment using (I, D) pair. After QWeb takes an action and observes a new state in the web navigation environment, new state is sent to the meta-trainer to collect a meta-reward ($R1$). Final reward is computed by adding $R1$ to the environment reward, i.e., $R = R1 + R2$.

Discussion. Even though we use a simple rule-based policy to collect episodes, depending on the nature of an environment, a different kind of policy could be designed to collect desired final states. Also, note that the generated goal states (D) need not be a valid goal state. MetaQWeb can still train QWeb by leveraging those incomplete episodes as well as the instruction and goal pairs that the web navigation environment assigns. There are numerous other possibilities to utilize MetaQWeb to provide various signals such as generating supervised episodes and performing behavioral cloning, scheduling a curriculum from the episodes generated by meta-trainer, using MetaQWeb as a behavior policy for off-policy learning, etc. We practiced using potential-based dense rewards ($R1$) and leave the other cases as future work.

4.6 Experimental Results

We evaluate our approaches on a number of environments from Miniwob ([90]) and Miniwob++ ([64]) benchmark tasks. We use a set of tasks that require various com-

binations of clicking and typing to set a baseline for the QWeb including a difficult environment, *social-media-all*, where previous approaches fail to generate any successful episodes. We then conduct extensive experiments on a more difficult environment, *book-flight-form* (a clone of the original book-flight environment with only the initial web form is used), that require learning through a large number of states and actions. Each task consists of a structured instruction (also presented in natural language) and a 160px x 210px environment represented as a DOM tree. We cap the number of DOM elements at 100 and the number of fields is 3 for *book-flight-form* environment. Hence, the number of possible actions and number of variables in the state can reach up to 300 and 600, respectively. However, these numbers only reflect a sketch and do not reflect a realization of DOM elements or instruction fields. With more than 700 airports, these numbers greatly increase and generate even larger spaces. *Social-media-all* environment has more than 7000 different possible values in instructions and DOM elements and the task length is 12 which are both considerably higher than other environments. All the environments return a sparse reward at the end of an episode with (+1) for successful and (-1) for failure episodes, respectively. We also use a small step penalty (-0.1) to encourage QWeb to find successful episodes using as small number of actions as possible.

As evaluation metric, we follow previous work and report *success rate* which is the percentage of successful episodes that end with a +1 reward. For instruction generation tasks, *success rate* is +1 if all the values in the instruction is correctly generated.

Previous Approaches: We compare the performance of QWeb to previous state-of-the-art approaches. First, SHI17 ([90]) pre-trains with behavioral cloning using approximately 200 demonstrations for each environment and fine-tunes using RL.They mainly use raw pixels with several features to represent states. Second, LIU18 ([64]) uses an alternating training approach where a program-policy and a neural-policy are trained iteratively. Program-policy is trained to generate a high level workflow from human

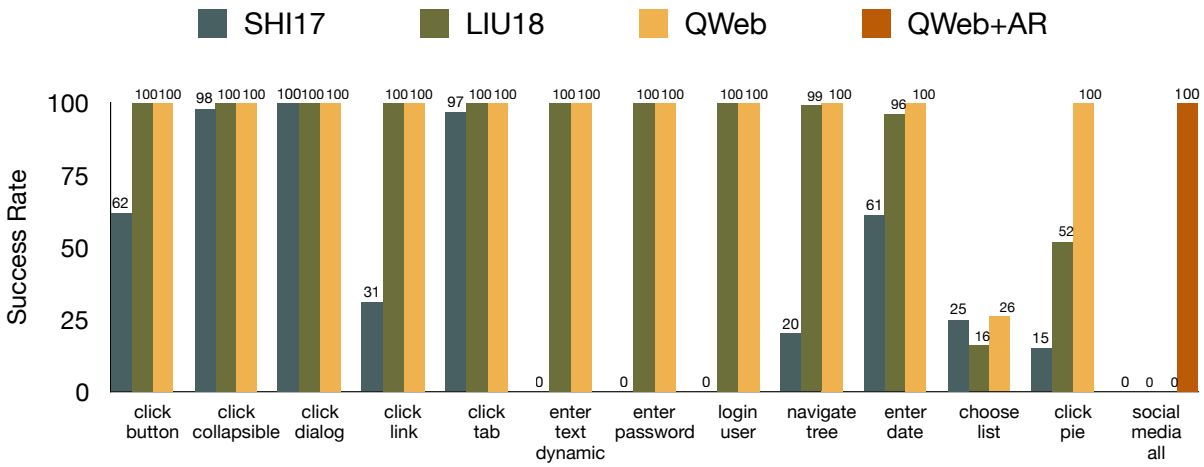


Figure 4.4: Performance of QWeb on a subset of Miniwob and Miniwob++ environments compared to previous state-of-the-art approaches. AR denotes augmented reward.

demonstrations. Neural-policy is trained using an actor-critic network by exploring states suggested by program-policy.

4.6.1 Performance on Miniwob Environments

We evaluate the performance of QWeb on a set of simple and difficult Miniwob environments based on the size of state and action spaces and the input vocabulary. On the simple Miniwob environments (first 11 environments in Figure 4.4), we show the performance of QWeb without any shallow encoding, reward augmentation or curriculum learning. Figure 4.4 presents the performance of QWeb compared to previous approaches. We observe that, QWeb can match the performance of previous state-of-the-art on every simple environment; setting a strong baseline for evaluating our improvements on more difficult environments. We can also loosely confirm the effectiveness of using a biLSTM encoder instead of extracting features for encoding DOM hierarchy where biLSTM encoding gives consistently competitive results on each environment.

Using shallow encoding and augmented rewards, we also evaluate on more difficult environments (*click-pie* and *social-media-all*). On *click-pie* environment, the episode

lengths are small (1) but the agent needs to learn a relatively large vocabulary (all the letters and digits in English language) to correctly deduce the semantic similarities between instruction and DOM elements. The main reasons that QWeb is not able to perform well on *social-media-all* environment is that the size of the vocabulary is more than 7000 and task length is 12 which are both considerably larger compared to other environments. Another reason is that the QWeb can not learn the correct action by focusing on a single node; it needs to incorporate siblings of a node in the DOM tree to generate the correct action. In both of these environments, QWeb+SE successfully solves the task and outperforms the previous approaches without using any human demonstration. Our empirical results suggest that it is critical to use both shallow encoding and augmented rewards in *social-media-all* environment where without these improvements, we weren't able to train a successful agent purely from trial-and-error. Without augmented reward, the QWeb overfits very quickly and gets trapped in a bad minima; in majority of these cases the policy converges to terminating the episode as early as possible to get the least step penalty.

We analyze the number of training steps needed to reach the top performance on several easy (*login-user*, *click-dialog*, *enter-password*) and difficult (*click-pie*) environments to understand the sample efficiency of learning from demonstrations (as in LIU18 ([64])). We observe that using demonstrations can lead to faster learning where on average it takes less than 1000 steps for LIU18 and more than 3000 steps for QWeb ($96k$ steps for *login-user* environment) however with a drop in performance on the more difficult *click-pie* environment: LIU18 reaches 52% success rate at $13k$ steps while QWeb reaches the same performance at $31k$ steps and 100% success at $175k$ steps.

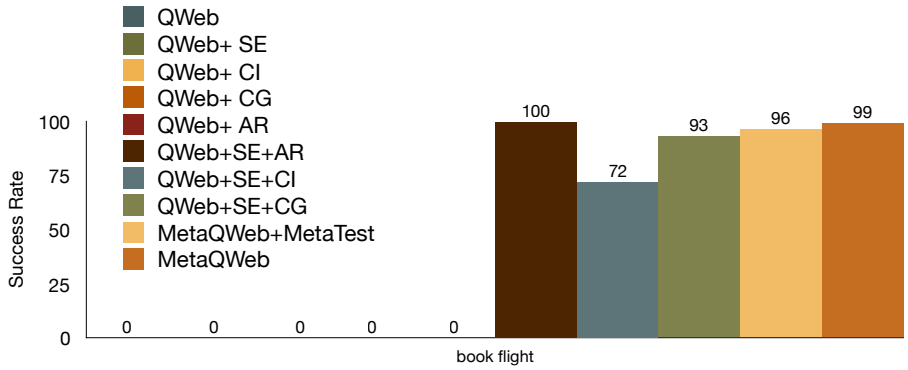


Figure 4.5: Performance of variants of QWeb and MetaQWeb on the book-flight-form environment. SE, CI, CG, and AR denote shallow encoding, curriculum with warm-start, curriculum with simulated sub-goals, and augmented reward, respectively.

4.6.2 Performance on Book Flight Environment

In Figure 4.5, we present the effectiveness of each of our improvements on *book-flight-form* environment. Without using any of the improvements or using only a single improvement, QWeb is not able to generate any successful episodes. The main reason is that without shallow encoding, QWeb is not able to learn a good semantic matching even with a dense reward; with only shallow encoding vast majority of episodes still produce -1 reward that prevents QWeb from learning. When we analyze the cause of the second case, we observe that as the training continues QWeb starts clicking submit button at first time step to get the least negative reward. When we remove the step penalty or give no reward for an unsuccessful episode, QWeb converges to one of the modes: clicking submit button at first time step or generating random sequence of actions to reach step limit where no reward is given. Using shallow encoding, both curriculum approaches offer very large improvements reaching above 90% success rate. When the reward can easily be augmented with a potential-based dense reward, we get the most benefit and completely solve the task.

Before we examine the performance of MetaQWeb, we first evaluate the performance of INET on generating successful instructions to understand the effect of the noise that

MetaQWeb introduces into training QWeb. We observe that INET gets 96% success rate on fully generating an instruction. When we analyze the errors, we see that the majority of them comes from incorrect DOM element prediction (75%). Furthermore, most of the errors are on date field (75%) where the value is mostly copied from an incorrect DOM element. After we train QWeb using meta-training, we evaluate its performance in two different cases : (i) We meta-test QWeb using the instruction and goal pairs generated by MetaQWeb to analyze the robustness of QWeb to noisy instructions and (ii) We test using the original environment. Figure 4.5 suggests that in both of the cases, MetaQWeb generates very strong results reaching very close to solving the task. When we examine the error cases for meta-test use case, we observe that 75% of the errors come from incorrect instruction generations where more than 75% of those errors are from incorrect date field. When evaluated using the original setup, the performance reaches up to 99% success rate showing the effectiveness of our meta-training framework for training a successful QWeb agent.

The main reason of the performance difference between the QWeb+SE+AR and the MetaQWeb can be explained by the difference between the generated experience that these models learn from. In training QWeb+SE+AR, we use the original and clean instructions that the environment sets at the beginning of an episode. MetaQWeb, however, is trained with the instructions that instructor agent generates. These instructions are sometimes incorrect (as indicated by the error rate of INET : 4%) and might not reflect the goal accurately. These noisy experiences hurt the performance of MetaQWeb slightly and causes the 1% drop in performance.

Chapter 5

Understanding Neural Network Predictions

In the previous chapters, we proposed neural approaches that can learn from various natural language interfaces with structured and unstructured databases. We designed deep neural architectures that captures hierarchical features within the data and can generalize well to unseen tasks. Other recent advances of neural model have also delivered promising results for a variety of NLP tasks such as question answering [119], machine comprehension [107], and machine translation [105]. However, the success of these models has been largely driven by their ability to perform well on a held-out data using models of increasingly higher complexity [42], at the cost of transparency and interpretability. Take, as example, our reading comprehension model in Chapter 3. There are three components that are connected in a hierarchy: (i) Encoding user questions, (ii) Encoding document windows in parallel, and (iii) Decoding answers from soft-selected windows. In this setting, it is unclear how question and document windows are related, which word and phrase level features are important when selecting a window, and how categorical and span-level answers are generated from the window representations. This hurdles progressive model development and limits the application of these models in areas like medicine and health care, where model understanding is critical.

Question					
What	**movie**	did	James	Frey	**write**?
0.11	**0.37**	0.05	0.06	0.03	**0.38**

Correct Knowledge-base Path
`film.writer.film`

Figure 5.1: An example of the identified neural indicators in a question and its correct knowledge base path. When perturbing the word embedding of "*movie*" and "*write*", the prediction of the neural network will change significantly, indicating that the model's prediction heavily depends on these words.

In this chapter, we propose several models that can extract salient input features, called neural indicators, that shows which word and phrase level components are critical while learning and making predictions. In Section 5.3, we formalize the definition of neural indicators and give baseline deep neural models for question answering and sentiment analysis tasks. We describe how these indicators can be identified using bayesian uncertainty and forward and backward propagation in Section 5.4. By regularizing models based on the confidence of indicator selection scores, we propose new objective formulations in Section 5.5.

5.1 Understanding and Regularizing Neural Models via Prediction Uncertainty

In this section, we propose a general framework to understand neural predictions by perturbing inputs of a model and analyzing the caused variation on the output. Through this analysis, we can identify important features, called *neural indicators*, for which a small perturbation will generate a significant variation on the output distribution. Take knowledge base question answering [11] as an example, where the main task is to predict the correct knowledge-base path that can answer a natural language question (Figure

5.1). In this case, our framework will assign a probability distribution over input words, which indicates the sensitivity (or *uncertainty*) of the neural network predictions with respect to each input word.

We define two perspectives of uncertainty to identify neural indicators: forward and backward view. Forward view uncertainty estimates the variation on the prediction distribution by applying a small perturbation to one input feature such as a word vector while keeping other word vectors fixed. Backward view uncertainty, on the other hand, examines the variation in the gradients backpropagated to each input word vector when prediction distribution perturbed. In Figure 5.1, the words "*movie*" and "*write*" cause the highest uncertainty on the final prediction, and we call them neural indicators.

In addition to identifying neural indicators, predictive uncertainty provides a natural way to measure the confidence of neural network predictions. A highly peaked prediction distribution on an input example shows that a neural network is very confident on its prediction, which may imply a high risk of overfitting. Based on this observation, we develop regularization techniques for neural networks to discourage predictions that are too confident. Our method does not have any assumption on the type of applicable neural networks and does not need additional annotated data.

We present quantitative and qualitative results on two example tasks: factoid question answering (QA) and sentiment classification (SC). Qualitatively, we show that the neural indicators identified by our method are semantically related to the corresponding knowledge-base path of a question. Quantitatively, we show that the proposed regularization methods can improve the accuracy of a baseline model on the QA task and the SC task by more than 2% and 1.5%, respectively.

5.2 Related Work

Recent studies for understanding neural networks include visualizing state activations [44, 48, 58, 60], generating short and descriptive texts [55], and learning sparse word vectors with interpretability [26]. Lei et al. [55] learn a model to generate short texts using a multi-encoder approach. Salient features are extracted using first-order derivatives assuming decision score is a linear combination of input features [58]. Li et al. [60] searches a minimal set of input words that causes the prediction to change. Our work differs from these studies in both the definition of neural indicators and how they are applied. We define indicators using FVU as the input words that lead to an uncertainty on the output prediction. In BVU, we examine the gradients using backpropagation when the prediction distribution varies.

Attention models provide another meaning of understanding by focusing on parts of neural networks [6, 15, 14, 69, 108, 115, 66, 94, 71]. Recently, they have been applied to different NLP tasks such as textual entailment [82], abstractive summarization [83], and machine comprehension [43, 45]. Our neural indicator model can be applied to attention-based models too, and can be used to generate indicators where attention is not well suited.

Our work is also related to regularization techniques such as dropout [92, 52]. Szegedy et al. [97] and Xie et al. [106] penalize confident predictions using a label smoothing method on prediction distributions. Kingma et al. [52] introduces variational dropout where dropout weights from gaussian distribution are learnt during training. Recently, dropouts are shown in a Bayesian setting [31] and uncertainty of an example is computed using variation ratios [29, 30]. In our work, we described the uncertainty of a prediction distributions w.r.t. a single input feature rather than all the features in an instance. We also presented an empirical connection between uncertainty of prediction distribution

and overfitting. We showed two different ways of incorporating this feedback mechanism to learn better models.

5.3 Neural Indicators

Gaining insights into how neural networks make predictions and mining the possible reasons that cause an error is critical for understanding neural networks. Consider a typical deep learning task of building a classifier using a vector representation of words in a corpus. Quantifying the importance of words that are essential for model's prediction can provide significant insights on neural networks. To this end, we introduce and formalize the task of *identifying neural indicators* as pinpointing word vectors that lead into high uncertainty in the prediction distribution. We hypothesize that if a small perturbation of an input vector causes high uncertainty on the output, then there is a strong dependency between model's prediction and the input vector. Dropout [92] is one way of perturbing word vectors, but various ways of perturbation can be used to estimate uncertainty.

As a motivating example, we define the single path factoid question answering task, where a question x and a set of candidate paths $Y = \{y_1, y_2, \cdots, y_n\}$ from a knowledge base are given, and the task is to find the correct path y^* that can answer the question. An example question with the corresponding one-hop path is shown in Figure 5.1. Each question and path is represented as a sequence of word vectors, i.e., $x = (x_1, x_2, \cdots, x_m)$ and $y_i = (y_{i1}, y_{i2}, \cdots, y_{il})$. The goal of a neural network is to map the sequences into a vector space such that x is closer to the correct path y^* than the others in this space. We call this mapping an encoder (Enc) and the corresponding vector an encoding of the sequence, $h = Enc(x)$. Recurrent Neural Networks (RNN) provide a natural way to

implement this mapping. For our purpose, we define an RNN encoder as

$$h_t = f(x_t, h_{t-1}; \theta) \tag{5.1}$$

where h_t is the output, and x_t is the input vector at time t. f is a nonlinear function operating on x_t and h_t with θ being its parameter vector. The input vectors are obtained from an embedding matrix E where each row corresponds to a word in the vocabulary V. Given an input sequence, a RNN runs over the word sequence and produces the output vectors at each step. We get the encoding h of a sequence using average pooling over the output vectors of RNN.

We define a multi-encoder QA model that employs two different RNNs to encode questions and knowledge base paths seperately. Let h_x and h_i denote the encoding of the question x and the i-th path by the multi-encoder model. Given a question x, we define the *prediction distribution* as the candidate path probabilities: $s(x, y_i) \propto exp(h_x^T h_i)$. Our objective is to minimize the cross-entropy error between the prediction distribution $s(x, y) = [s(x, y_1), \cdots, s(x, y_n)]$ and the correct path distribution.

Although RNNs are very effective at encoding sequences, their complex structure makes it difficult to assess the underlying reasons for their predictions. Considering our QA task, one natural question among many others is, *which words in the question that make the RNN favor a path than another?* We formulate the concept of *uncertainty* as a measure of the importance of each input word to the final prediction. Next we introduce two ways to estimate uncertainty.

5.4 Identifying Neural Indicators

We argue that identifying neural indicators is an unsupervised task and can be extended to other neural networks. In order to identify neural indicators, we introduce two different approaches to estimate uncertainty: forward and backward view. Forward view uncertainty (FVU) first applies a small perturbation to a word vector, then runs a forward propagation in the network, and calculates variation on the prediction distribution. Backward view uncertainty (BVU), on the other hand, first applies a small perturbation to the prediction distribution, then runs a backward propagation in the network, and calculates variation on gradients w.r.t. each word vector in the network.

5.4.1 Forward View Uncertainty

Let's assume that ψ is a vector sampled from a distribution such as Bernoulli distribution and having the same dimensionality as x_t. We denote $\hat{x} = [x_1, \cdots, x_t * \psi, \cdots, x_m]$ as a sequence of word vectors where x_t is randomly perturbed with ψ using elementwise multiplication. We first estimate the prediction distribution by running a forward propagation in the network with $\hat{x}$ as input. Using an appropriate uncertainty function, such as entropy, we estimate the predictive uncertainty. We repeat the same process for T iterations by sampling a new ψ each time we run forward propagation. The forward view uncertainty of x_t is then estimated as a sample statistics over these uncertainty values.

Take the running example in Figure 5.1. In order to estimate the FVU of "movie" (x_2), we first pick a sample vector ψ', e.g., $\psi' = [1, 0, \cdots, 0]$ where the first dimension is one and others are zero. We perturb the embedding by multiplying with ψ' which will keep the first dimension of x_2 and zero the rest. We run the RNN model described in Section 5.3 and estimate the probabilities over the candidate paths. Predictive uncertainty is then estimated over these probabilities.

Let's assume that $s_k(\hat{x}, y)$ is the prediction distribution at iteration k. We now describe how to estimate uncertainty on $s_k(\hat{x}, y)$ and combine these values to estimate FVU w.r.t. x_t.

Variation Ratios (VR): Variation ratios (VR) [29] uses mode of the prediction distribution to estimate uncertainty in bayesian neural networks. Traditional definition of VR incorporates all input features by perturbing every feature randomly. Here we apply VR to RNNs and focus on only a single feature where we perturb a single word vector and keep others fixed:

$$VR_t = 1 - \frac{1}{T}\sum_{k=1}^{T} max_i(s_k(\hat{x}, y_i)) \tag{5.2}$$

VR gives a measure of the variation of prediction distribution around its mode when only a single word embedding of the network is perturbed.

Entropy (Ent): Mode statistics is a relatively local measure of uncertainty which does not capture some distributional characteristics such as uniformity. To mitigate this effect, we also define an uncertainty measure using entropy. Similar to VR, we focus on only a single word:

$$Ent_t = \frac{1}{T}\sum_{k=1}^{T} H(s_k(\hat{x}, y)) \tag{5.3}$$

where $H(z) = -z^T log(z)$. Here, we measure a weighted variation incorporating all probabilities in the prediction distribution.

The above uncertainty estimators aggregate the effect of each individual dimension. Consider a case where only a small set of dimensions causes high uncertainty on the neural prediction. In this case, mean estimators (Eq. 5.2 and Eq. 5.3) are mostly determined by the less important dimensions and this will result in small uncertainty on the output.

To mitigate this problem, we propose to use standard-deviation instead of mean:

$$VRStd_t = STD(\{1 - max_i(s_k(\hat{x}, y_i))\}_k) \tag{5.4}$$

and

$$EntStd_t = STD(\{H(s_k(\hat{x}, y_i))\}_k) \tag{5.5}$$

where $\{.\}_k$ represents a set of values indexed by k. When the mean uncertainty of a word is small, the std-estimators ($VRStd$ and $EntStd$) can determine whether a subset of dimensions are important for the prediction.

5.4.2 Backward View Uncertainty

FVU looks at how the inference of a neural network changes when the input features such as word vectors are perturbed. Alternatively, changes in the training of a neural network can also give an estimation on uncertainty. We give a new definition of uncertainty by measuring the changes in the gradients w.r.t. input word vectors when the prediction distribution is perturbed. Let's denote $\tilde{x} = [x_1 * \psi_1, \cdots, x_t * \psi_t, \cdots, x_m * \psi_m]$ where we perturb each word vector simultaneously using different random samples from the same distribution. We first estimate the prediction distribution by running a forward propagation in the network with $\tilde{x}$ as input. Next, we estimate $H(s(\tilde{x}, y))$ and run a backward propagation to compute gradients w.r.t. x_t. We repeat the same process T iterations where at each iteration we perturb prediction distribution by sampling a new ψ_i for each word vector. The backward view uncertainty of x_t is then estimated as a sample statistics over these gradients.

Let's assume that $s_k(\tilde{x}, y)$ is the prediction distribution at iteration k. We define $g_{t,k} = \nabla(H(s_k(\tilde{x}, y)), x_t)$ as the gradient of entropy w.r.t. x_t at iteration k. We estimate

	VR	Ent	BVU	VR-Std	Ent-Std	BVU-Std
VR	1.0	0.76	0.49	0.77	0.74	0.46
Ent		1.0	0.49	0.76	0.79	0.47
BVU			1.0	0.69	0.7	0.93
VR-Std				1.0	0.95	0.68
Ent-Std					1.0	0.68
BVU-Std						1.0

Table 5.1: Pearson-correlation statistics between different uncertainty estimators on the WebQSP dataset. Darker color values correspond to higher correlations.

BVU of x_t using expected gradient length:

$$BVU_t = \frac{1}{T}\sum_k \|g_{t,k}\| \tag{5.6}$$

Similar to FVU, we also define a std-estimator for BVU:

$$BVUStd_t = \frac{1}{T}\sum_k STD(\{g_{t,k}\}_k) \tag{5.7}$$

Here, we estimate the standard-deviation over each gradient estimation rather than over the length of gradients to incorporate the effect of individual dimensions on the uncertainty.

5.4.3 Identifying Neural Indicators

We use FVU and BVU to quantify the importance of each word and to identify neural indicators probabilistically. Given an uncertainty measure unc_t for x_t, we define the following *uncertainty distribution* as the probability of a word being a neural indicator:

$$\tilde{unc}(x_t) = U^{-1} unc(x_t) \tag{5.8}$$

where U is the normalization factor over uncertainty estimations. As we are interested in indicators that cause a large variation, we also define a word as a *discriminative neural indicator* if:

$$u\tilde{n}c(x_t) > \mu_t + \sigma_t \tag{5.9}$$

where μ_t and σ_t are the mean and standard deviation of the uncertainty distribution for input x_t.

5.4.4 Experimental Results for Neural Indicators

Before exploring predictive uncertainty to regularize neural networks, we present results for identifying neural indicators in our QA task. For the experimental setup and more experimental results, see Section 5.6.

In Table 5.1, we report the average Pearson correlation coefficient over uncertainty distributions between each pair of estimators. We observe that FVU and BVU models are more similar within their respective groups but more different across groups. VR-Std and Ent-Std give the highest correlation suggesting that they provide a very similar uncertainty distribution. Also, BVU has more similarity to FVU std-estimators, than VR and Ent.

Table 5.2 illustrates several example question and path pairs where our neural network model makes incorrect path prediction. Both FVU and BVU models generate indicator words that describe the corresponding paths well, even though the predictions are incorrect. This suggests that neural predictions might depend on words that are descriptive of the true path but they suffer from poor generalization during training which motivates us to utilize predictive uncertainty to learn better models (Section 5.5). BVUs, frequently identify question words such as "where" that describe a constraint

(in this case `location`) on the path prediction. Consider the question "*What did Alvin Smith died from?*" the VR-Std model produces a single indicator *died* with a very high probability. In another example question "*Where did Jerome David Salinger died?*" the VR-Std model produces *died* and *where* as neural indicators. In comparison with the first example where the correct path is `people.deceases_person.`
`cause_of_death`, the model also identifies "where" as an additional indicator to discriminate from the first example.

5.5 Regularizing Neural Predictions

In the previous section, we showed that FVU and BVU might identify words that are descriptive of the correct path even though the neural network model makes incorrect predictions. Motivated by this observation, we study several methods to learn better representations by regularizing neural networks using predictive uncertainty. To motivate our new objective further, we first give an intuitive explanation for the relationship between predictive uncertainty and overfitting, then we explain how FVU and BVU can be used to learn better representations.

5.5.1 Predictive Uncertainty and Overfitting

Consider the following scenario where we make several simplifying assumptions to illustrate our intuition. We assume two independent regions in the encoding space for each question x. A *sampling region* S is defined such that each forward propagation using random perturbations (such as dropout) of input word vectors generates encodings within S. C denotes a *confidence region* where only encodings within C produces high confidence predictions. Figure 5.2 illustrates a sample setting with associated prediction distributions and regions. x' has more probability of overfitting, as samples from S' are

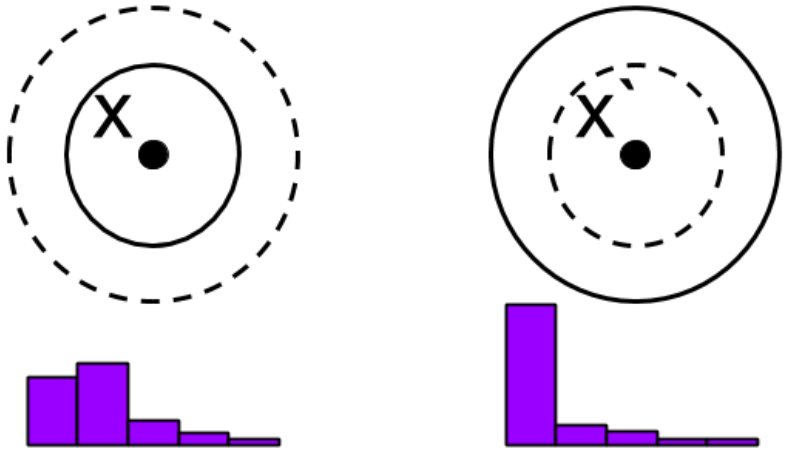


Figure 5.2: Two examples with their confidence (solid) and sampling (dotted) regions, and their prediction distributions. The example on the left has more uncertainty (less confidence) than the example on the right.

less likely to be outside of C' which produces more confident predictions. On the other hand, x incurs more error on its prediction, as samples from S are more likely to generate different prediction probabilities which produces more uncertainty on its output.

Let's assume that confidence regions are fixed for each example. Generating a larger sampling region for confident predictions will reduce the probability of overfitting. On the other hand, a smaller sampling region for examples with uncertainty will improve the prediction accuracy. An ideal objective would be searching for the best of both worlds using the uncertainty of an example to generate better regions.

5.5.2 Regularization via Uncertainty

We propose to regularize neural networks using predictive uncertainty to alleviate the overfitting problem motivated in Section 4.1.

Regularization using FVU

Traditional dropout notion [92] is widely used to regularize neural network parameters. We note that increasing the dropout weight in the case of small uncertainty can

yield to a better regularization of confident predictions. Inspired by this, we propose to introduce a new penalty using a dropout guided by FVU.

Given FVU of the model using VR, we generate a new dropout weight as follows:

$$\eta_x = \begin{cases} (1+\beta)\eta & \text{VR(x)} \geq \theta \\ (1+\beta)^{-1}\eta & \text{VR(x)} < \theta \end{cases}$$

where η is the global dropout weight and $0 \leq \beta, \theta \leq 1$ are hyperparameters. Here, we define $1-\eta$ as the probability of randomly removing a dimension. Early in the training, the uncertainty of the predictions will be large and new dropout weight will be small; later in the training, the uncertainty of the predictions will be small hence the model will be penalized more. It is important to note that, we perturb each word vector simultaneously which corresponds to $\tilde{x}$ rather than $\hat{x}$.

To incur a penalty using the new dropout, we modify the loss function by

$$L^*(z) = L(z) + \alpha L_{reg}(z) \tag{5.10}$$

where L_{reg} is the loss of the model with new dropout η_x while L is the original loss of the model using global dropout η. α is a hyperparameter balancing the effect regularization. Similar to L, we estimate L_{reg} by applying a forward propagation with the new dropout η_x. To estimate the new loss, we first apply a forward propagation and estimate new dropout weight η_x. Next, we estimate the new dropout weight for the question using VR. We then apply another forward propagation with the new dropout weight and estimate L_{reg}.

Regularization using BVU

Entropy of a prediction distribution yields a natural uncertainty measure. Hence, adding a penalty term using the entropy can inherently regularize confident prediction distributions. Here, we establish a connection between such a regularizer and our proposed BVU: as the gradients of an entropy term w.r.t. input word embeddings reveal important words, updating the word embeddings using these gradients will enable the network to focus more on important words. Formally, we define a new loss

$$L^*(z) = L(z) - \alpha H(s(x, y)) \tag{5.11}$$

that integrates this entropy based regularizer term. α is again a positive hyperparameter.

5.6 Experimental Evaluation

We evaluate the proposed model on two tasks, single path factoid question answering and sentiment classification. Our aim is to show that by gaining insights into neural predictions using predictive uncertainty, better objectives can be designed without altering the underlying model architectures.

5.6.1 Single Path Factoid Question Answering

Dataset We use the WebQSP factoid question answering dataset [120]. The dataset contains 4,737 questions generated through Google Suggest API. Each question is annotated with a set of logical forms from Freebase. Logical forms have two components: (i) a core inferential path and (ii) a set of constraints. Core inferential path is a Freebase path capturing the core relation in the question. Constraints such as *type* and *gender* capture the auxiliary information in the question.

We assume that the topic entity in a question is already linked to Freebase. Our QA task is to predict the 1-hop core inferential path of a given question. We generate 1-hop candidate paths for each topic entity from the Freebase graph. As the WebQSP dataset may have multiple correct core inferential paths for some questions, we cast our problem as multi-label classification. We report the mean accuracy over all the questions.

We reserve 10% of the training dataset as the validation set and train the multi-encoder QA model from Section 5.3 on the remaining. We use LSTM [46] units in the RNN encoders. We set the size of the input and hidden vectors as 150 and 100, respectively, and search the dropout parameter weight from a range of values, $\{0.5, 0.6\}$, using the validation set. We utilize Adam [51] optimizer with a batch size of 16 training examples, a learning rate of 0.001, and 0.1 for the uniform distribution scale (i.e., [-0.1, 0.1]). We use $T = 100$ for identifying neural indicators. We use $\alpha = 0.5$ and $\alpha = 1.0$ for regularization using FVU and BVU, respectively. We take $\beta = 0.25$ and $\theta = 0.5$ by hand tuning. We implement our models in Tensorflow [1].

Attentive-Encoder LSTM In addition to the multi-encoder approach that we described in Section 5.3, we also implemented a variant of the Attentive Reader [43]. Instead of taking the average of output vectors of the question encoder, we take a weighted average of outputs using the path encoder. We estimate a weight for each output vector of question encoder as follows:

$$w_t = A^{-1} exp(v^T tanh(Kh_t + Lh_i)) \tag{5.12}$$

where v, K, and L are attention parameters to learn, and A is the normalization factor over weights w_t. We then estimate the new output vector of the question encoder as

follows:

$$h = \sum_t w_t h_t \tag{5.13}$$

We use dropout regularization on inputs and ouputs of the LSTM cells for both multi-encoder and attentive-encoder models.

Classification Results In Table 5.3, we give a comparison between the performance of multi-encoder LSTM models and their corresponding variants using the proposed regularization methods. We show that all of our uncertainty regularization approaches outperform both LSTM and LSTM-Att models that use fixed dropout weights. This demonstrates that uncertainty values are not only useful for identifying neural indicators but can also be used to improve the performance of neural networks via regularization.

5.6.2 Sentiment Analysis

Dataset We use Stanford Large Movie Dataset for the sentiment classification task [67]. The dataset contains 25,000 training and 25,000 testing reviews collected from the IMDB database. Each review contains multiple sentences that describe the overall impression of a movie. Additionally, each review is associated with a rating (on a scale from 0 to 9) that indicates the overall sentiment of the reviewer. We use the ratings as the labels for our classification task.

We reserve 5% of the training dataset as the validation dataset and the remaining as the training dataset. We clip each review to the first 200 words. We use a single RNN encoder on the reviews and build a logistic regression classifier on the output vector h to minimize the cross-entropy error between sentiment predictions and correct labels. We set the size of the input and hidden vectors as 128. We search the dropout weight from $\{0.6, 0.7, 0.8\}$ using the validation set. We utilize Adam [51] optimizer with a batch size of 32 examples, a learning rate of 0.001, and 0.1 for the uniform distribution scale

(i.e., [-0.1, 0.1]). We use $\alpha = 0.5$ for both regularization using FVU and BVU. We take $\beta = 0.25$ and $\theta = 0.5$ by hand tuning. We report both fine-grained accuracy over all 10 labels and a coarse-grained binary accuracy by mapping fine-grained sentiment predictions from $[0, 4]$ and $[5, 9]$ to 0 and 1, respectively.

Classification Results In Table 5.4, we present a comparison between the accuracy of the standard (LSTM) and regularization (LSTM-Reg) models. We observe that LSTM-Reg outperforms the standard LSTM model on both fine-grained and coarse-grained accuracy measures. LSTM-Reg (FVU) and LST-Reg (BVU) give comparable results with LSTM-Reg (FVU) slightly better.

	Questions					
VR	What	did	Alvin	Smith	died	from
	Where	was		Ted	Kennedy	buried
	Where	did		Pavlova		originate
	What	city	did	Esther	live	in
	What	kind	of	government	is	China
	Where	did	Jerome	David	Salinger	died
VR-Std	What	did	Alvin	Smith	died	from
	Where	was		Ted	Kennedy	buried
	Where	did		Pavlova		originate
	What	city	did	Esther	live	in
	What	kind	of	government	is	China
	Where	did	Jerome	David	Salinger	died
BVU	What	did	Alvin	Smith	died	from
	Where	was		Ted	Kennedy	buried
	Where	did		Pavlova		originate
	What	city	did	Esther	live	in
	What	kind	of	government	is	China
	Where	did	Jerome	David	Salinger	died
BVU-Std	What	did	Alvin	Smith	died	from
	Where	was		Ted	Kennedy	buried
	Where	did		Pavlova		originate
	What	city	did	Esther	live	in
	What	kind	of	government	is	China
	Where	did	Jerome	David	Salinger	died

	Corresponding True Paths
1	`people.deceased_person.cause_of_death`
2	`people.deceased_person.place_of_burial`
3	`food.dish.cuisine`
4	`people.person.place_of_birth`
5	`location.country.form_of_government`
6	`people.person.place_of_death`

Table 5.2: Here, we show several example question and path pairs where the multi-encoder RNN model makes incorrect predictions. Neural indicators are identified using VR and BVU. Higher color values correspond to higher uncertainty probabilities.

	Model	Accuracy
Dropout	LSTM	75.6
	LSTM-Att	75.6
Uncertainty Reg.	LSTM-Reg (FVU)	76.6
	LSTM-Att-Reg (FVU)	76.5
	LSTM-Reg (BVU)	**77.6**
	LSTM-Att-Reg (BVU)	77.3

Table 5.3: Comparison of the LSTM encoders with different regularization approaches on the WebQSP dataset. Here, LSTM and LSTM-Att denote multi-encoder and attentive-encoder models. The suffix on each model denotes the type of regularization or using an attentive-encoder model.

Model	Fine-grained Accuracy	Coarse-grained Accuracy
LSTM	32.2	80.5
LSTM-Reg (FVU)	**33.8**	**81.5**
LSTM-Reg (BVU)	**33.8**	81.3

Table 5.4: Comparison of the LSTM and the proposed regularization models on the IMDB Sentiment Analysis dataset.

Chapter 6

Conclusions

In this dissertation, we have introduced neural models for learning natural language interfaces over different data sources. We focused on structured and unstructured data with interactive and non-interactive learning methods. We also studied learning from constrained interfaces where the underlying database is only accessible via a user interface, such as web pages. We concluded with understanding neural predictions by investigating correlations between neural predictions and inputs.

In Chapter 2, we described our neural models for learning from structured data such as knowledge bases and relational databases. In Section 2.1, we introduced a ranking model which is shown to be effective at end-to-end ranking of logical forms. By encoding structured logical forms using a hierarchical neural network, similarity between logical forms and user utterances are measured. In Section 2.2, we proposed a novel approach for learning natural language interfaces on structured tables where the labeled data is insufficient to generalize well. The proposed system improves its own performance by collecting more data via having multi-turn conversations with users. In Section 2.3, we discussed the shortcomings of rule-based users simulators and developed end-to-end models that can generate diverse and goal-oriented user responses by injecting hierarchical latent variables.

In Chapter 3, we introduced supervised and semi-supervised models for learning from

long unstructured data such as Wikipedia documents. We developed a supervised hierarchical encoder model which chunks a long document into pieces and composes a document representation via attention. The corresponding answer is decoded from this final representation using sequence decoder models. Next, we study reading comprehension in a low labeled data regime where all Wikipedia is available as an unlabeled data source. Using variational recurrent encoder-decoder architectures, we built unsupervised document representations over Wikipedia. We proposed neural progressive reviewer models that augments fixed document representations with trainable architectures which are tailed to the supervised task. We show that our semi-supervised models with only a handful of labeled data reach the performance of previous state-of-the-art models that use all the available labeled dataset.

In Chapter 4, we studied the problem of learning from constrained interfaces, such as web pages. We first investigated the challenges of learning from web interfaces such as large action spaces and sparse rewards. Next, we proposed curriculum learning methods that solve an easier problem by placing the agent and the goal state closer to each other and iteratively converging to the original problem setup. We introduced two curriculum learning methods – warm-start, where agent is placed closer to the goal by partially navigating the page at the start of the episode, and goal-simulation, where the goal is placed closer to the agent by randomly filtering out elements in the web page. When gold experience is not available in curriculum learning, we developed a novel meta-training approach. In meta-training, we first trained a model to predict the instruction given the final state; next, an arbitrary policy randomly navigates the web page and generates a final state; instruction is predicted from this final state and trajectory, instruction, and final state triples are used to pretrain a policy.

Finally, in Chapter 5, we develop models for understanding neural predictions. We use statistical uncertainty to measure the sensitivity of outputs of a neural network to its

inputs. By using variational dropout, we perturb inputs of a neural model and measure the change in the output predictions. We compute mode, standard deviation, and entropy uncertainty measures for extracting neural indicators which are input tokens that affect the neural prediction the most. On a question answering over knowledge base task, we show that these neural indicators are surface forms for entities and relations in a question. We also incorporate these measures to regularize confident neural predictions which are shown to cause overfitting of neural networks.

The research in learning neural interfaces has grown immensely in the past few years and we are excited to make contributions to the field. Nonetheless, there is still a large body of problems that we need to solve to achieve high performing interfaces. One key problem that remains to be solved is learning transferable models. As an example, web interfaces are greatly diverse where the visible user interface as well as the underlying structure of the elements vary between web pages. In the future, we need to jointly understand the semantics of the web pages and user utterances to generalize better over web interfaces.

With this thesis, we aim to inspire researchers towards studying and improving different types of natural language interfaces. We believe it will lead us towards better user experiences and hope to see these ideas to be adopted in industry and academia.

Bibliography

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

[2] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015430. URL `http://doi.acm.org/10.1145/1015330.1015430`.

[3] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 3981–3989. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6461-learning-to-learn-by-gradient-descent-by-gradient-descent.pdf`.

[5] Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. *CoRR*, abs/1607.00070, 2016. URL `http://arxiv.org/abs/1607.00070`.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine trans-

lation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

[7] Jun-Wei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. Knowledge-based question answering as machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL http://doi.acm.org/10.1145/1553374.1553380.

[9] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[10] Jonathan Berant and Percy Liang. Imitation learning of agenda-based semantic parsers. In *Transactions of the Association for Computational Linguistics*, 2015.

[11] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2013.

[12] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014.

[13] Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[14] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. ABC-CNN: an attention based convolutional neural network for visual question answering. *CoRR*, abs/1511.05960, 2015. URL http://arxiv.org/abs/1511.05960.

[15] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016. URL http://arxiv.org/abs/1601.06733.

[16] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259.

[17] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

[18] Eunsol Choi, Tom Kwiatkowski, and Luke Zettlemoyer. Scalable semantic parsing with partial ontologies. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[19] Eunsol Choi, Daniel Hewlett, Alexandre Lacoste, Illia Polosukhin, Jakob Uszkoreit, and Jonathan Berant. Coarse-to-fine question answering for long document. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[20] Paul Crook and Alex Marin. Sequence to sequence modeling for user simulation in dialog systems. In *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH 2017)*, pages 1706–1710, 2017.

[21] H. Cuayahuitl, S. Renals, O. Lemon, and H. Shimodaira. Human-computer dialogue simulation using hidden markov models. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 290–295, Nov 2005. doi: 10.1109/ASRU.2005.1566485.

[22] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3079–3087. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5949-semi-supervised-sequence-learning.pdf`.

[23] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[24] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL `http://arxiv.org/abs/1611.02779`.

[25] Otto Fabius and Joost R van Amersfoort. Variational Recurrent Auto-Encoders. *ArXiv*, December 2014.

[26] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. Sparse overcomplete word vector representations. *CoRR*, abs/1506.02004, 2015. URL `http://arxiv.org/abs/1506.02004`.

[27] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495. PMLR, 13–15 Nov 2017.

[28] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. META LEARNING SHARED HIERARCHIES. In *Proceedings of the International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyX0IeWAW`.

[29] Linton Clarke Freeman. *Elementary Applied Statistics*. 1965.

[30] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[31] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 2015.

[32] Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *INTERSPEECH*, 2005.

[33] Kallirroi Georgila, James Henderson, and Oliver Lemon. User simulation for spoken dialogue systems: learning and evaluation. In *INTERSPEECH*, 2006.

[34] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2009.

[35] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 1311–1320, 2017. URL `http://proceedings.mlr.press/v70/graves17a.html`.

[36] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *arxiv*, 2015.

[37] T. Guo and H. Gao. Bidirectional Attention for SQL Generation. *ArXiv e-prints*, December 2018.

[38] Izzeddin Gur, Daniel Hewlett, Llion Jones, and Alexandre Lacoste. Accurate supervised and semi-supervised machine reading for long documents. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2017.

[39] Izzeddin Gur, Dilek Hakkani-Tür, Gökhan Tür, and Pararth Shah. User modeling for task oriented dialogues. *IEEE Workshop on Spoken Language Technology*, 2018.

[40] Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. DialSQL: Dialogue based structured query generation. In *acl*, Melbourne, Australia, July 2018.

[41] Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJemQ209FQ`.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[43] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL `http://arxiv.org/abs/1506.03340`.

[44] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf`.

[45] Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, and David Berthelot. Wikireading: A novel large-scale language understanding task over wikipedia. *CoRR*, abs/1608.03542, 2016. URL `http://arxiv.org/abs/1608.03542`.

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

[47] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *CoRR*, abs/1704.08760, 2017. URL `http://arxiv.org/abs/1704.08760`.

[48] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015. URL `http://arxiv.org/abs/1506.02078`.

[49] Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. Learning to transform natural to formal languages. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2005.

[50] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

[51] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[52] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *CoRR*, abs/1506.02557, 2015. URL `http://arxiv.org/abs/1506.02557`.

[53] Florian Kreyssig, Iñigo Casanueva, Pawel Budzianowski, and Milica Gasic. Neural user simulation for corpus-based policy optimisation for spoken dialogue systems. *CoRR*, abs/1805.06966, 2018. URL `http://arxiv.org/abs/1805.06966`.

[54] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2013.

[55] Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. Rationalizing neural predictions. *CoRR*, abs/1606.04155, 2016. URL `http://arxiv.org/abs/1606.04155`.

[56] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, 2014.

[57] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. *CoRR*, abs/1506.01057, 2015. URL `http://arxiv.org/abs/1506.01057`.

[58] Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. Visualizing and understanding neural models in NLP. *CoRR*, abs/1506.01066, 2016. URL `http://arxiv.org/abs/1506.01066`.

[59] Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Dialogue learning with human-in-the-loop. *CoRR*, abs/1611.09823, 2016. URL `http://arxiv.org/abs/1611.09823`.

[60] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *CoRR*, abs/1612.08220, 2016. URL `http://arxiv.org/abs/1612.08220`.

[61] Ming Zhou Li Dong, Furu Wei and Ke Xu. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[62] Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 2013.

[63] Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. In *NIPS 2017 Workshop on Conversational AI*, 2017.

[64] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *Proceedings of the International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ryTp3f-0-`.

[65] Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[66] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

[67] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

[68] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014.

[69] André F. T. Martins and Ramón Fernández Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068, 2016. URL `http://arxiv.org/abs/1602.02068`.

[70] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[71] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016. URL `http://arxiv.org/abs/1606.03126`.

[72] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):

529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

[73] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613.

[74] Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue system via hierarchical deep reinforcement learning. *arxiv:1704.03084v2*, 2017.

[75] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014.

[76] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2014.

[77] Olivier Pietquin and Helen Hastie. A survey on metrics for the evaluation of user simulations. *The knowledge engineering review*, 28(1):59–73, 2013.

[78] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

[79] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[80] Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2014.

[81] Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 2016.

[82] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, and Phil Blunsom. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015. URL http://arxiv.org/abs/1509.06664.

[83] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015. URL `http://arxiv.org/abs/1509.00685`.

[84] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks. *arxiv*, 2016.

[85] A. Saha, V. Pahuja, M. M. Khapra, K. Sankaranarayanan, and S. Chandar. Complex Sequential Question Answering: Towards Learning to Converse Over Linked Question Answer Pairs with a Knowledge Graph. *ArXiv e-prints*, January 2018.

[86] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York, April 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N/N07/N07-2038`.

[87] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *CoRR*, abs/1507.04808, 2015. URL `http://arxiv.org/abs/1507.04808`.

[88] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069, 2016. URL `http://arxiv.org/abs/1605.06069`.

[89] Pararth Shah, Dilek Hakkani-Tur, Bing Liu, and Gokhan Tur. Bootstrapping a neural conversational agent with dialogue self-play, crowdsourcing and on-line reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 41–51. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/N18-3006`.

[90] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[91] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. *CoRR*, abs/1507.02221, 2015. URL `http://arxiv.org/abs/1507.02221`.

[92] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[93] Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, 2016.

[94] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015. URL `http://arxiv.org/abs/1503.08895`.

[95] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. Table cell search for question answering. In *Int. Conf. on World Wide Web (WWW)*, 2016.

[96] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

[97] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL `http://arxiv.org/abs/1512.00567`.

[98] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[99] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arxiv*, 2016.

[100] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 2773–2781, 2015.

[101] Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. Learning to reinforcement learn. *CoRR*, abs/1611.05763, 2016. URL `http://arxiv.org/abs/1611.05763`.

[102] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL `http://arxiv.org/abs/1410.3916`.

[103] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[104] William A Woods. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the American Federation of Information Processing Societies Conference*, 1973.

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

[106] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing CNN on the loss layer. *CoRR*, abs/1605.00055, 2016. URL http://arxiv.org/abs/1605.00055.

[107] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016. URL http://arxiv.org/abs/1611.01604.

[108] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015. URL http://arxiv.org/abs/1502.03044.

[109] Kun Xu, Yansong Feng, Siva Reddy, Songfang Huang, and Dongyan Zhao. Enhancing freebase question answering using textual evidence. In *arxiv*, 2016.

[110] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017. URL http://arxiv.org/abs/1711.04436.

[111] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):63:1–63:26, October 2017. ISSN 2475-1421. doi: 10.1145/3133887. URL http://doi.acm.org/10.1145/3133887.

[112] Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. Joint relational embeddings for knowledge-based question answering. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014.

[113] Yi Yang and Ming-Wei Chang. S-MART: novel tree-based structured learning algorithms applied to tweet entity linking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[114] Zhilin Yang, Ye Yuan, Yuexin Wu, Ruslan Salakhutdinov, and William W. Cohen. Encode, review, and decode: Reviewer module for caption generation. *arxiv*, 2016.

[115] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alexander J. Smola. Stacked attention networks for image question answering. *CoRR*, abs/1511.02274, 2015. URL `http://arxiv.org/abs/1511.02274`.

[116] Xuchen Yao. Lean question answering over freebase from scratch. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

[117] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[118] Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. Improving semantic parsing via answer type inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016.

[119] WenTau Yih, MingWei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[120] Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016. URL `http://aclweb.org/anthology/P/P16/P16-2033.pdf`.

[121] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014. URL `http://arxiv.org/abs/1410.4615`.

[122] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *arxiv*, 2012.

[123] John M Zelle and Mooney Ray. Learning to parse database queries using inductive logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1996.

[124] Luke Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[125] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 658–666, 2005.

[126] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL `http://arxiv.org/abs/1709.00103`.