Toward Data-Driven Tutorial Question Answering with Deep Learning Conversational Models

Mayank Kulkarni and **Kristy Elizabeth Boyer**
Department of Computer & Information Science & Engineering
University of Florida
Gainesville, Florida, USA
mayankk91@ufl.edu, keboyer@ufl.edu

Abstract

There has been an increase in popularity of data-driven question answering systems given their recent success. This paper explores the possibility of building a tutorial question answering system for Java programming from data sampled from a community-based question answering forum. This paper reports on the creation of a dataset that could support building such a tutorial question answering system and discusses the methodology to create the 106,386 question strong dataset. We investigate how retrieval-based and generative models perform on the given dataset. The work also investigates the usefulness of using hybrid approaches such as combining retrieval-based and generative models. The results indicate that building data-driven tutorial systems using community-based question answering forums holds significant promise.

1 Introduction

Question answering in dialogue is a central concern for designing the next generation of dialogue systems. Recent work has made great strides in generating dialogue, for example, with neural conversation models (Vinyals and Le, 2015), persona-based conversation models (Li et al., 2014) and adversarial models (Li et al., 2017). Specifically, for responding to questions, information-retrieval techniques have long been explored (Jeon et al., 2005; Ramos, 2003; Lowe et al., 2015). A critical open question is how to build data-driven systems for specific domains. A challenge that is faced by the community for such systems is the availability of data for those domains. Given that transfer learning has not yet been shown to yield good results (Mou et al., 2016), there has been investigation in the area of partially data-driven and hand-crafted systems (Williams et al., 2017). However, handcrafted systems face tremendous limitations in authoring. *Data-driven* dialogue systems, which derive their functionality from corpora, have the potential to eliminate this bottleneck.

This work explores the possibility of building a data-driven question-answering system for Java programming. We leverage a promising source of data by drawing from community-based question answering forums of Stack Exchange. Forums typically also have sub-forums, such as Stack Overflow for programming questions and Ask Ubuntu for Ubuntu operating system related questions. Such community-based forums serve as excellent datasets for specific domains, such as programming or IT support, that are otherwise not easily available to the general public. The promise of this data is further demonstrated by other work done using the Stack Exchange data: Campbell and Treude (2017) explore how to use semantic parsing to convert an English sentence or query into a code snippet, while Campos et al. (2016) investigate returning relevant question answer pairs for Swing, Boost and LINQ by using indexing techniques and building feature-based classifiers.

With technology becoming ubiquitous, having programming skills are highly sought after. In a University or MOOC setting, 'Introduction to Programming' courses typically have a large class size, and with a limited number of Teaching Assistants, providing individual help becomes a difficult task. The work in this paper focuses on attempting to assist in helping students learn Java programming with a data-driven tutorial question answering system.

This work attempts to build the tutorial question-answering system as both a retrieval-based question answering system (Ji et al., 2014) via the

Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 273–283
New Orleans, Louisiana, June 5, 2018.

Dual Encoder architecture (Medsker and Jain, 2001; Bromley et al., 1994) and as a generative question answering system (Ritter et al., 2011) via the Sequence-to-Sequence architecture (Sutsveker et al., 2014; Cho et al., 2014). The retrieval-based model answers the user's question by predicting the most relevant answer from a set of predefined answers. In contrast to the retrieval-based model, the generative model answers the user's question by generating new answers based on the data on which the model was trained. Both of these approaches rely on building good semantic representations of the input in the vector space using word embeddings (Mikolov et al., 2013; Mikolov et al., 2013).

This work also explores the usefulness of a hybrid approach involving the combination of the retrieval-based and generative models. This paper thus represents the first work to explore deep learning techniques for data-driven tutorial dialogue for Java programming.

2 Related Work

Recently, there has been work using natural language processing and machine learning techniques within tools for programming support and computer science education. Zhang et al. (2016) explored using Deep Belief Networks to grade short-answer texts and showed that this approach outperformed conventional machine learning models. They also explored using student modeling and clustering based on engineered features to predict the grades with reasonable success. Wang et al. (2017) used a recurrent neural network to attempt to represent a student's knowledge states for programming exercises and found that the model was able to successfully identify students with knowledge gaps and provide indications that assistance may be necessary.

Work is also being done to build models from data that can generate their own answers to questions. Bengio et al. (2003) and Mikolov (2012) (Mikolov et al., 2010) were able to successfully construct a neural language model using recurrent neural networks, further reinforcing the prevailing conclusion that recurrent neural networks are the architecture of choice for this task. Sordoni et al. (2015) and Shang et al. (2015) were also able to model short conversations using a recurrent neural network.

A critical turning point for generative models was when Sutskever et al. (2014) & Cho et al. (2014) introduced the sequence-to-sequence framework in the domain of machine translation. The authors proposed an architecture to convert one sequence to another sequence using recurrent neural networks as encoders and decoders. Inspired by the previous success of recurrent neural networks and the sequence-to-sequence framework, Vinyals and Le (2015) proposed applying this framework to conversational modeling, framing question answering as a machine translation problem. While Vinyals and Le (2015) showed that the model was able to give short, coherent answers for queries in a variety of settings, they also mentioned limitations of the system: it is restricted to short answers and lacks a personality.

In addition to generative systems, retrieval-based systems have also shown success in the recent past. Kannan et al. (2016) used semi-supervised learning with an LSTM RNN along with semantic intent clustering to generate high-quality responses for the Google Smart Reply system. Lu et al. (2017) explored how to generate responses from a large answer space by using a dual encoder LSTM network and employing clustering to generate templates from their large answer set, reducing the answer set space for a customer support question answering system. Jeon et al. (2005) investigated how to find question similarity using word translation probabilities. Lowe et al. (2015) constructed a corpus of one million multi-turn dialogues from the Ask Ubuntu forum, then performed experiments with retrieval-based models that demonstrated that a useful question answering system could be built using a dataset sampled from a community-based question answering forum. These techniques helped us gain insight on how to identify the most appropriate responses from a knowledge base.

The work in this paper attempts to employ deep learning techniques to support computer science education by developing a programming support tool for Java Programming that provides automated tutorial question answering. The work builds upon recent work in retrieval-based and generative models to construct answers that combine the English language with the Java programming language.

3 Dataset

Stack Exchange is a set of community-based question answering websites, with each website covering a specific topic. Stack Overflow deals

Figure 1: Sample Stack Overflow question[1]

with programming questions and relies on self-moderation through peer upvoting mechanisms. The user who posts the question can select the answer that they deem most appropriate. In some cases, the original poster does not select an answer, and in these cases the highest upvoted answer could be considered the best answer.

A typical Stack Overflow question can be seen in Figure 1. We see a title for the question at the top "Java String Declaration", followed by a description, "What is the difference between ... performance variation." An important piece of information is the meta tags seen underneath the description. We see the meta tags of "java" and "string", which describe on a high level to what the post is related. We see an upvote count to the left of the answer, a measure of how many other users agree with this answer. For the question in Figure 1, we see that there is an answer that has received the user's accepted answer status as well as 29 upvotes by the community.

Stack Exchange provides an anonymized data dump of all the user-contributed content, with the most recent version published on Dec 1, 2017. The data dump is in the format of a SQL database consisting of various components of the website represented in the form of SQL tables such as the Posts table, Users table, and Comments table. For the purposes of this work we consider only the Stack Overflow data and the Posts table.

[1] https://stackoverflow.com/questions/3652369/

3.1 Working with the Stack Exchange Database

The Posts table contained about 38 million posts, i.e. all the post data on Stack Overflow as of the data dump publication date. Every question and answer posted on the website is part of the Posts table, with different identifiers to signify the type of Post and relationships between the Posts. The question-answer relationship was defined as follows: the original question had a post ID, and answers corresponding to this question had the same post ID in their parentID column.

3.2 Filtering Posts

This work focuses on Java programming questions, which required us to narrow our search to Java-related questions from the Posts table. We first filtered to ignore questions containing the '<code>' tag in the 'Body' column, as our present goal is to answer general questions within a future tutorial system.

In order to obtain posts related to Java, we used the Post table's 'Tags' column, which contained meta tags related to the post, as seen in Figure 1. In order to ignore technology-specific questions such as a question about 'Spring' or 'Hibernate', we created a list of tags to ignore based on frequency counts and prefixes (such as 'google-api-xx' or 'facebook-api-xx'). Once these filters were in place, we filtered to ignore all unanswered questions based on the 'AnswerCount' column in the Posts table. Another filtering step was to take all the answers that contained code snippets defined by the <code> token and replace the tokens with 'CODE_START' and 'CODE_END' as labels to mark the beginning and end of the code snippet.

3.3 Dataset Statistics

We collected all corresponding answers from our set of filtered questions to create an initial corpus. This corpus contained 107,961 question-description-answer triplets, of which 47,220 questions did not have a 'user accepted best answer'. A statistical analysis based on a naive word split showed that there were outliers in the corpus, with very large maximum lengths of up to 10,000 words in an answer. We identified and removed the outliers in the corpus by removing the current largest sample and monitoring the average length of the corpus. We continued to remove the largest sample till we obtained a rela-

Average Question Length	8.68013
Average Description Length	71.45428
Average Answer Length	87.54342
Vocabulary Size	284,827

Table 1: Final Dataset Statistics

tively stable average value. This outlier determination was performed for each sample type of question, description and answer separately. Ultimately, we removed questions longer than 19 words, or whose descriptions were longer than 125 words, or with answers longer than 175 words.

The questions, descriptions and answers in the dataset were then converted into a sequence of numbers using word indexing techniques, in order to be usable by a machine learning model. The word indexing techniques involved first tokenizing the sentences into word tokens by using an open-source tokenizer (Python NLTK).[2] Each word was labelled with a unique index and stored as a key-value pair in a data structure. Secondly, the words in each sentence were replaced by the corresponding indexes using the data structure created above to obtain a sequence of numbers which corresponded to the original sentence. A total of 284, 827 words were obtained through tokenization and subsequently indexed in the data structure.

To maintain the uniformity of sentence length, we 'pre-pad' the sequence with 0 before the original sequence. Adding zeros at the start of the original sequence (if required) allows the network to accept a fixed sequence length and the nature of the number zero also allows us to denote that the element in the sequence is an empty space. We 'pre-pad' and thus structure the sequence with actual content towards the end of the sequence because a time-based neural network is more likely to 'remember' time steps towards the end of the sequence, as those would be stored in the more recent memory which is captured by the network.

The filtering of the sentences with length thresholds is important, as it is difficult to capture semantic representations for lengthy text using word embeddings. Setting these thresholds resulted in a reduced dataset of 106,386 questions. The statistics for the final dataset are shown in Table 1. We also make this dataset available for public use as a contribution of this paper.[3]

4 Methods and Techniques

With the future objective of building a data-driven tutorial question-answering system, we first explore three overarching approaches of retrieval-based models, generative models, and hybrid models for Java programming-based tutorial question answering.

The challenges associated with this dataset are that unlike traditional question answering datasets, this dataset has three streams of inputs. Each stream has its own unique descriptors such as vocabulary and length. The answers in the dataset contain interspersed English and Java, which could make building meaningful word vector representations difficult. Long sentences are typically more difficult to represent in a vector space and this dataset contains longer typical sentences for the description and answer than those seen in previous work of Lowe et al. (2015) and Lu et al. (2017). As a part of this work, we investigate which combinations of inputs from the dataset yield the most optimal results.

4.1 Dual Encoder LSTM (Siamese network)

The Siamese Network or Dual Encoder architecture (Medsker and Jain, 2001; Bromley et al., 1994) has shown success in the recent past to build a retrieval-based question answering system (Lowe et al., 2015; Lu et al., 2017).

To use the dataset with the Dual Encoder architecture, we needed to perform some additional pre-processing. We first built a dataset containing the question along with its description and the corresponding correct answer, and we assigned a label of 1 to these samples. We then created a sample containing the incorrect answer for a given question and description pair. This was done by randomly choosing another answer from the rest of the answer set and assigning a label of 0 to these samples.

Description & Answer Dual Encoder (DADE): This architecture consisted of a Dual Encoder Bidirectional LSTM network, where the first encoder encoded the description of the question and the second encoder encoded the answer

[2] NLTK implementation: https://www.nltk.org/

[3] https://cise.ufl.edu/research/learndialogue/data/java-stackoverflow-QA-dec2017.zip

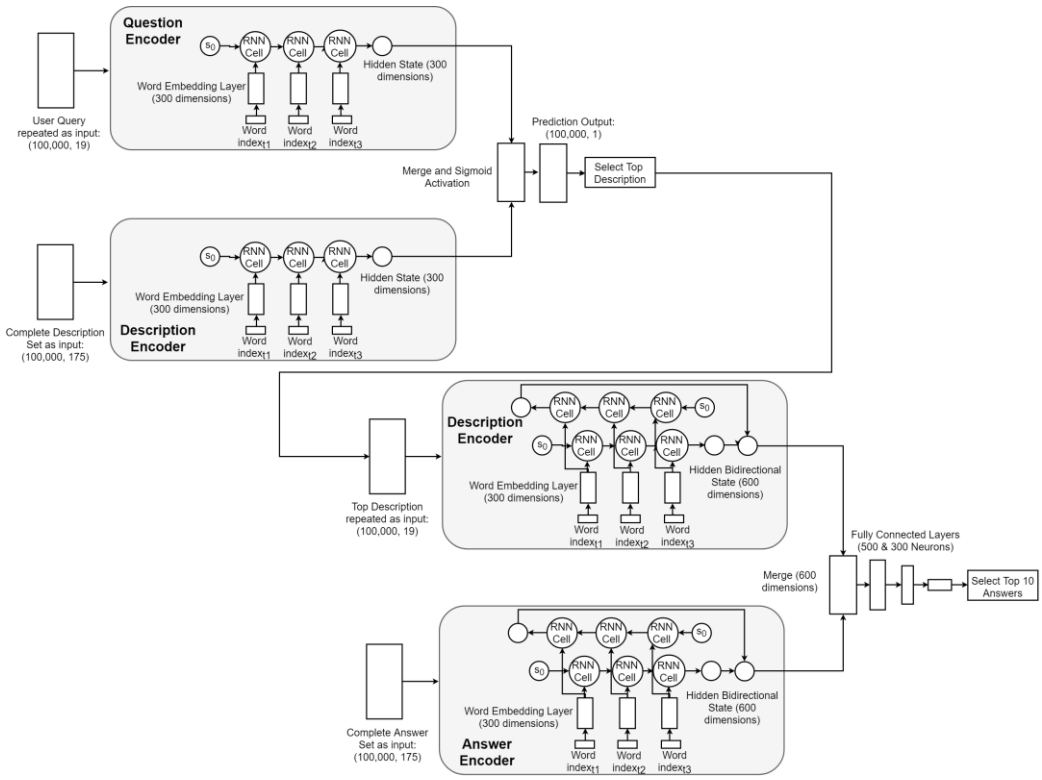

Figure 2. Question Description Matching followed by Description Answer Matching

statement. The large maximum sequence lengths influenced us to choose to use a Bidirectional LSTM (Schuster and Paliwal, 1997) as it allows the network to understand the context of a word with respect to both previous and next words and thus build better vector representations of the words. Each bidirectional encoder's output was merged together to obtain a single 600-dimensional output, and then this output was fed to a fully connected network of two layers, with the first layer containing 500 neurons and the second layer containing 300 neurons. This was then run through a sigmoid activation function in order to obtain the result.

This architecture used pre-trained GloVe word embeddings which were updated during the training phase. The LSTM cells contained 300 hidden units and 2 layers and optimized the binary cross-entropy loss function.

Question & Description Dual Encoder (QDDE): This architecture was similar to the Description and Answer Dual Encoder in that it consisted of a Dual Encoder LSTM network, where the first encoder encoded the question statement and the second encoder encoded the description statement. Each encoder's outputs were merged together to obtain a single 300-dimensional output and then this output was run through a sigmoid activation function in order to obtain the result.

Again, this architecture also used pre-trained GloVe word embeddings which were updated during the training phase. The LSTM cells contained 300 hidden units and a single layer and optimized the binary cross-entropy loss function.

The rationale for this architecture was to build a dual encoder that would be able to predict a description given a question. We wanted to investigate whether the dual encoder could learn relationships between smaller questions and longer descriptions. If we could successfully predict the description for a given question, it would allow us to leverage the similar lengths of the description and answer to obtain better results.

4.2 Techniques to answer queries

The aforementioned architectures were able to determine answers for the given training, validation and testing sets, where the correct answers are predetermined. To extend our model's use to the real world, we needed to define a different set of strategies to answer questions for which we do not know the predetermined answer. We explore our proposed strategies in the following section.

Question Description Matching followed by Description Answer Matching: This approach attempted to find a similarity measure between a given user question and a description of the given

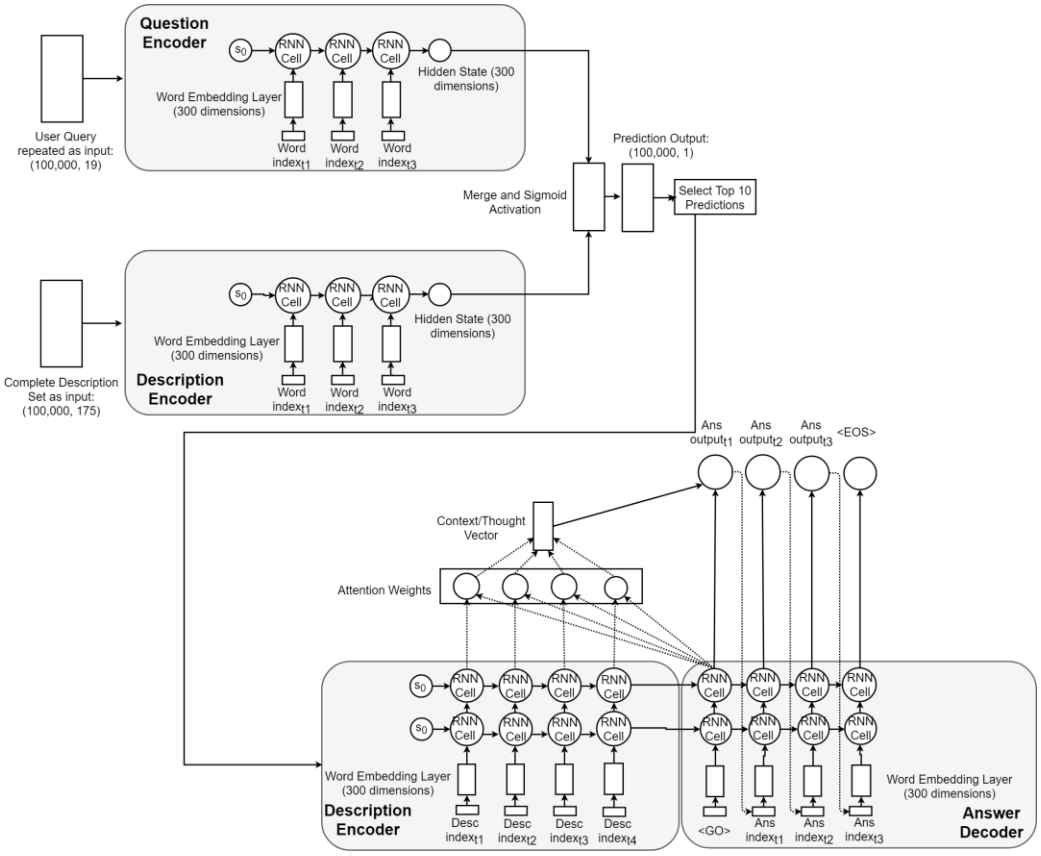

Figure 3. Hybrid Architecture Combining Dual Encoder with Sequence-to-Sequence Model

question via QDDE. The best matching description was then run against all the answers to determine the top 10 best possible answers, as seen in Figure 2. The intuition behind using this approach was that the description-answer dual encoder should provide better results as it used a bi-directional LSTM network and both sequence lengths were approximately the same.

4.3 Sequence-to-Sequence Models

Sequence-to-Sequence models are generative models that, unlike their retrieval-based counterparts, do not rely on choosing from an existing set of answers but rather generate answers on their own.

The preprocessing steps for the sequence-to-sequence model were identical to the preprocessing steps specified for the dual encoder model.

Description to Answer Encoder - Decoder: This architecture used the description of a question as the input to the encoder and attempted to match the actual answer to the question using the decoder. The intuition behind matching a description to the answer was that as the sequences are of almost equal length, this could then be framed as a machine translation problem, which has seen significant success with the sequence-to-sequence model (Sutskever et al., 2014; Cho et al., 2014; Vinyals and Le, 2015).

The encoder was a bidirectional recurrent neural network using LSTM cells. We chose bidirectionality for better sentence vector representation, and LSTM cells for their ability to capture long-term dependencies. The decoder is a standard recurrent neural network with LSTM cells.

The LSTM cells contained 512 hidden units and 2 layers. We used a dropout (Srivastava et al, 2014) probability of 0.2 and gradient normalization (Pascanu et al., 2013) of 3.6. We used 15 buckets, as the length of 175 (maximum answer length) would then be equally split into smaller chunks of size 12 increments. The Luong attention mechanism (Luong et al., 2015) was implemented in order to boost accuracy, as was beam search (Wiseman and Rush, 2016) of beam width 10 in order to obtain a better output for a given input. The vocabulary had to be reduced to 60,000 due to memory constraints.

All the hyperparameters stated for the networks discussed above were determined by performing a

Model Name		recall@1	recall@2	recall@5
DA	TF-IDF	0.7837	0.8437	0.9137
	DADE	0.7052	**0.8672**	**0.9798**
QA	TF-IDF	0.9483	0.9685	0.9785
	QDDE	0.8542	0.951	**0.9928**

Table 2. Testing recall@k for group size 10

grid search and cross-validating with the validation dataset.

4.4 Hybrid architecture (Dual Encoder + Sequence-to-Sequence)

In this work, we built a hybrid structure that combined both the retrieval-based model of the dual encoder with the generative model of the sequence-to-sequence model. The intuition behind building this model was that a user typically asks questions with a length of fewer than 20 words and may not necessarily have enough of a description to fit the 125-word limit sufficiently. The proposed architecture combats this issue by obtaining the user question and trying to find the most appropriate description from a set of prefixed descriptions, this is done by the question and description dual encoder mentioned earlier. The entire workflow can be seen in Figure 3.

We take the top 10 predicted descriptions and feed these descriptions as input to the description to answer sequence-to-sequence model.

The Description to Answer model would result in 10 different generated answers and the answers were ranked based on the input descriptions ranking. This architecture also lets us leverage the nature of the dataset, in that it contains a question, a description and an answer as opposed to a traditional question-answer pair dataset.

5 Experiments

All the experiments performed as a part of this work were done on a desktop with the following specification i7 8-core CPU, 32GB RAM, and NVIDIA GTX 1070 8GB VRAM.

The dataset of 106,386 was split into separate training, testing and validation sets. Given the large network sizes used in the experiments, there were a correspondingly large number of parameters to be trained for each network, which in turn required a sufficiently large dataset to train on. Taking this into consideration we chose to not follow the traditional 80-20 train-test split but rather maintain a large enough training set and use the rest of the data for testing and validation. The training set thus contained 100,000 questions and corresponding description and answers triplets, the test dataset contained 5,000 triplets and the validation set contained 1,386 triplets.

For the dual encoder experiments, the training set size was 200,000 as we had to use both positive and negative samples while training. Whereas for the sequence-to-sequence model training we used only 100,000 description and answer pairs.

5.1 Quantitative Analysis

Dual Encoder: The recall@k metric works in conjunction with the group size. Given that we have a group size of 5, recall@1 tells us that if we had the option to choose 1 out of the 5 options, what is the probability that it would be correct. We take a look at Table 2, where the group size is 10 and we compare another popular retrieval-based method, TF-IDF (Ramos, 2003), to our obtained results. TF-IDF has been outperformed by dual encoders for conversational models in the past (Lowe et al., 2015), but we see some interesting results for our dataset.

We see that for DADE, TF-IDF is able to slightly outperform the dual encoder at the recall@1 scores, but the dual encoder outperforms TF-IDF for recall@2 and recall@5. We further see that QDDE is outperformed by TF-IDF in both recall@1 and recall@2, only for QDDE to do better in recall@5.

We believe that we see this behavior because TF-IDF works based on word similarity and rates rare words between two documents as highly related (Ramos, 2003). Questions and descriptions containing common phrases are better perceived by TF-IDF than by the dual encoders. In addition, previous results like Lowe et al. (2015), worked on a corpus with an average word count of 10 words where they showed that the dual encoder architecture significantly outperformed TF-IDF, whereas our work deals with much longer utterances. In spite of long utterances, we see that the dual encoders do a comparable or better job than TF-IDF.

Sequence-to-Sequence: We followed Google's Neural Machine Translation tutorial (Luong et al., 2017) to build our sequence-to-sequence models.

Model:	Best Output:
QDDE + DADE	you can use an arraylist or a list of integers instead so that you can add items to the list as and when required also the list would then have only as many elements as the number of inputs syntax CODE_START List<Integer> = new ArrayList<Integer>() CODE_END to add elements to the list use CODE_START elements.add(new Item()) CODE_END to access members of the list use CODE_START elements.get(index) CODE_END

Hybrid Model (Response #1)	CODE_START
	public static void main (string args)
	list<integer> list = new arraylist<integer>
	for (int i = 0; i < list.size(); i++){
	system.out.println(list.get(i));
	}
	CODE_END[4]
	Actual output: code start list lt integer gt list new arraylist lt integer gt for int i 0 i lt list size i system out println list get i code end

Hybrid Model (Response #2)	CODE_START
	public static void main(string args) {
	list<integer> list = new arraylist<integer>
	for (int i = 0; i < 10; i++){
	list.add(i);
	system.out.println(list.get(i));
	}
	CODE_END[3]
	Actual output: code start public static void main string args list lt integer gt list new arraylist lt integer gt for int i 0 i lt 10 i list add i system out println list get i code end

Table 3. Top Responses for "how can we create an integer array in java"

An important point to note is that while traditional machine translations are judged based on BLEU score (Papineni et al., 2002) and perplexity, a conversational model cannot be judged on BLEU and hence we used perplexity as the primary measure of judgment (Shao et al., 2017).

Description to Answer Sequence-to-Sequence Model: The perplexity for the dev set continued to decrease, thus we could assume no overfitting had occurred over the epochs of training this network. The dev and test perplexity scores were better than the previous model, with scores of 68.91 and 70.15 respectively, and this is also reflected in the coherent responses made by the model.

5.2 Qualitative Analysis

We chose a question which was neither part of the training, development nor test corpus to analyze the qualitative results. The reason these results are presented as qualitative is that since its part of neither of the corpus we do not have the actual expected response. The question that we chose is a fairly simple and straightforward question:

"How can we create an integer array in java"

We take a look at the responses given by the models in Table 3. As the models provide multiple answers, we have handpicked the answer that we thought was most relevant from the top 10. We have also cleaned the answer by referring the original post on Stack Overflow for readability.

The QDDE+DADE model produces a response that suggests using Java collections to achieve the same purpose of the array. The drawback here is that it diverges from the actual answer but is still relevant nonetheless. Another one of the top ten answers suggested looking at some of the Java documentation related to arrays.

We now take a look at the responses generated by sequence-to-sequence models and the hybrid model. While the hybrid model suggests using an 'ArrayList' instead of an array, it was able to form different codes for the condition of the 'for' loop in both the answers, suggesting that it may understand a relationship between functions such as 'get' and 'add' and the 'for' loop condition.

It is also interesting to see the response generated by the Description to Answer sequence-to-sequence model. We can analyze some of the aforementioned testing responses generated via the Description to Answer Sequence-to-Sequence model as can be seen in Table 4.

By analyzing the generated responses for the samples above, we can see that the model has learned how to create new objects and has also learned what kind of commands are related to a given object, such as the date in Java needs simpledateformat class or that the file could need a file path. Perhaps the most notable was the crea-

[4] The sequence-to-sequence model does not include non-word tokens such as '=' or '{'. These have been added for readability.

1	Question:	how to format a date in java
	Description:	how can change this date format 2011 09 07...[5]
	Generated Response:	CODE_START simpledateformat sdf new simpledateformat yyyy mm dd hh mm ss …...
2	Question:	java does not recognize a file when it begins with file
	Description:	java says a file does not exist when it is a valid file path ...[6]
	Generated Response:	CODE_START file file new file path to file CODE_END
3	Question:	how to track of other application's memory and cpu usage by java coding
	Description:	i want to show cpu and memory utilization of any application...[7]
	Generated Response:	you can take a look at the CODE_START java util concurrent CODE_END package http docs oracle com javase tutorial essential environment sysprop html

Table 4: Sample Sequence-to-Sequence Generated Responses from Test Set

tion of a coherent 'for' loop using the previously created object and referencing the appropriate method.

We also see that the models are able to successfully combine the English language along with java code, starting answers with phrases such as "you can use...", "i dont think there is a way...", "i am not sure but try…" and so on. The models are also able to draw a clear line between code snippets and English language and code start labels are mostly correctly completed with code end labels. There have also been instances where English phrases such "you can also try" are used between two code snippets.

While these examples have been sampled from a much larger set in which not all the responses are as appropriate, this still shows promise in using this architecture to build models that can appropriately respond to a query by generating their own response.

6 Conclusion

This work has examined how we can leverage community-based question answering forums as a source of data to build a dataset specific to general Java-based programming questions. We have seen that retrieval-based models obtain high recall rates on the testing set but are restricted only to the answer set available. On the other hand, generative models are able to successfully combine the English language along with Java code to make coherent responses at times, but the responses are small and do not completely answer the question. We found reasonable success with the hybrid model by combining the retrieval-based approach with the generative approach. The proposed approaches show promise in building a useful tutorial system based on the sampled dataset. These are the first steps made in that direction.

This work could be furthered by investigating jointly training the hybrid model to improve description selection and answer generation. One could also frame this task as a machine comprehension task, where the entire answer set could be used as the context. Doing so would allow us to leverage the memory network architecture, which performs better at tasks involving storing long-term memory. Finally, we could explore using adversarial training, as it has seen success on conversational models in the recent past (Li et al., 2017).

7 Acknowledgments

The authors wish to thank the members of the LearnDialogue group at the University of Florida for their helpful input. This work is supported in part by the National Science Foundation through grant CNS-1622438. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the authors, and do not necessarily represent the official views, opinions, or policy of the National Science Foundation.

[5] https://stackoverflow.com/questions/7631470
[6] https://stackoverflow.com/questions/40983790
[7] https://stackoverflow.com/questions/6390581

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research, 3(Feb), pp. 1137-1155.*

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger and Roopak Shah. 1994. Signature verification using a "siamese" time delay neural network. *In Advances in Neural Information Processing Systems, pp. 737-744.*

Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code snippet content assist via natural language tasks. *In Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on, pp. 628-632. IEEE.*

Eduardo C. Campos, Lucas BL Souza and Marcelo de A. Maia. 2016. Searching crowd knowledge to recommend solutions for API usage tasks. *Journal of Software: Evolution and Process 28, no. 10: 863-892*

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734.*

Jiwoon Jeon, W. Bruce Croft and Joon Ho Lee. 2005. Finding similar questions in large question and answer archives. *In Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 84-90. ACM.*

Zongcheng Ji, Zhengdong Lu and Hang Li. 2014. An information retrieval approach to short text conversation. *arXiv preprint arXiv:1408.6988.*

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado et al. 2016. Smart reply: Automated response suggestion for email. *In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 955-964. ACM.*

Jiwei Li, Michel Galley, Chris Brockett, Georgios P. Spithourakis, Jianfeng Gao and Bill Dolan. 2016. A persona-based neural conversation model. *In Proceedings of 54th Annual Meeting of Association for Computational Linguistics, pp. 994–1003.*

Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547.*

Ryan Lowe, Nissan Pow, Iulian Serban and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *In Proceedings of the SIGDIAL 2015 Conference, pp. 285–294.*

Yichao Lu, Phillip Keung, Shaonan Zhang, Jason Sun and Vikas Bhardwaj. 2017. A practical approach to dialogue response generation in closed domains. *arXiv preprint arXiv:1703.09439.*

Minh-Thang Luong and Eugene Brevdo and Rui Zhao. 2017. Neural Machine Translation (seq2seq) Tutorial, *https://github.com/tensorflow/nmt.*

Minh-Thang Luong, Hieu Pham and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1412–1421.*

L. R. Medsker and L. C. Jain. 2001. Recurrent neural networks. *Design and Applications 5.*

Tomáš Mikolov. 2012. Statistical language models based on neural networks. *PhD thesis, PhD Thesis, Brno University of Technology, 2012.*

Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *In Advances in Neural Information Processing Systems (pp. 3111-3119).*

Tomáš Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. *In Eleventh Annual Conference of the International Speech Communication Association, pp. 1045-1048.*

Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang and Zhi Jin. 2016. How Transferable are Neural Networks in NLP Applications*?. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 479–489.*

Kishore Papineni, Salim Roukos, Todd Ward and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. *In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pp. 311–318.*

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *In International Conference on Machine Learning (pp. 1310-1318)*

Juan Ramos. 2003. Using tf-idf to determine word relevance in document queries. *In Proceedings of*

the First Instructional Conference on Machine Learning, vol. 242, pp. 133-142.

Alan Ritter, Colin Cherry and William B. Dolan. 2011. Data-driven response generation in social media. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 583-593.*

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing 45, no. 11: 2673-2681.*

Lifeng Shang, Zhengdong Lu and Hang Li. 2015. Neural responding machine for short-text conversation. *In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 1577–1586.*

Yuanlong Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope and Ray Kurzweil. 2017. Generating high-quality and informative conversation responses with sequence-to-sequence models. *In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 2210-2219.*

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. *In Proceedings of Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pp. 196–205.*

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15(1), 1929-1958.*

Ilya Sutskever, Oriol Vinyals and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *In Advances in Neural Information Processing Systems, pp. 3104-3112.*

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *Proceedings of the 31st International Conference on Machine Learning, JMLR: W&CP volume 37.*

Lisa Wang, Angela Sy, Larry Liu, Chris Piech. 2017. Learning to represent student knowledge on programming exercises using deep learning. *In Proceedings of the 10th International Conference on Educational Data Mining; Wuhan, China (pp. 324-329).*

Jason D. Williams, Kavosh Asadi and Geoffrey Zweig. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *In Proceedings of 55th Annual Meeting of Association for Computational Linguistics, pp. 665-677.*

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 1296–1306.*

Yuan Zhang, Rajat Shah and Min Chi. 2016. Deep Learning+ Student Modeling+ Clustering: a Recipe for Effective Automatic Short Answer Grading. *In Proceedings of the 9th International Conference on Educational Data Mining (pp. 562-567).*