Are you asking the right questions? Teaching Machines to Ask Clarification Questions

Sudha Rao
Department of Computer Science
University Of Maryland, College Park
raosudha@cs.umd.edu

Abstract

Inquiry is fundamental to communication, and machines cannot effectively collaborate with humans unless they can ask questions. In this thesis work, we explore how can we teach machines to ask clarification questions when faced with uncertainty, a goal of increasing importance in today's automated society. We do a preliminary study using data from StackExchange, a plentiful online resource where people routinely ask clarifying questions to posts so that they can better offer assistance to the original poster. We build neural network models inspired by the idea of the expected value of perfect information: a good question is one whose expected answer is going to be most useful. To build generalizable systems, we propose two future research directions: a template-based model and a sequence-to-sequence based neural generative model.

1 Introduction

A main goal of asking questions is to fill information gaps, typically through clarification questions, which naturally occur in conversations (Purver, 2004; Ginzburg, 2012). A good question is one whose *likely answer* is going to be the most useful. Consider the exchange in Figure 1, in which an initial poster (who we'll call "Terry") asks for help configuring environment variables. This question is underspecified and a responder ("Parker") asks a clarifying question "(a) What version of Ubuntu do you have?" Parker could alternatively have asked one of:

(b) Is the moon waxing or waning?

(c) Are you running Ubuntu 14.10 kernel 4.4.0-59-generic on an x86_64 architecture?

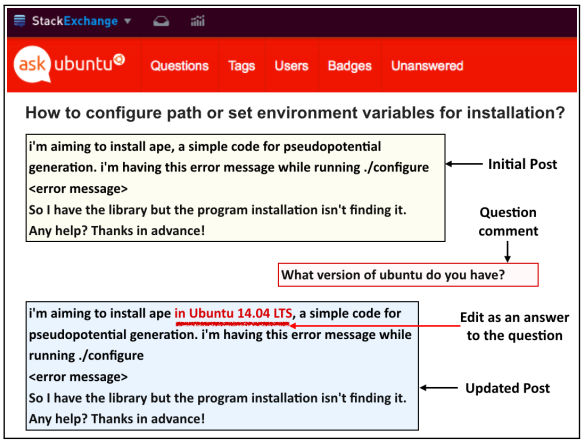

Figure 1: A post on an online Q & A forum "askubuntu.com" is updated to fill the missing information pointed out by the question comment

Parker should not ask (b) because it's not useful; they should not ask (c) because it's too specific and an answer of "No" gives little help. Parker's question (a) is optimal: it is both likely to be useful, and is plausibly answerable by Terry. Our goal in this work is to automate Parker. Specifically, after Terry writes their initial post, we aim to generate a clarification question so that Terry can immediately amend their post in hopes of getting faster and better replies.

Our work has two main contributions:

1. A novel neural-network model for addressing this task that integrates the notion of expected value of perfect information (§2).
2. A novel dataset, derived from StackExchange, that enables us to learn a model to ask clarifying questions by looking at the types of questions people ask (§4.1).[1]

To develop our model we take inspiration from the decision theoretic framework of the Expected

[1]We use data from StackExchange; per license cc-by-sa 3.0, the data is "intended to be shared and remixed" (with attribution). We will release all of the data we extract.

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics- Student Research Workshop, pages 30–35
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-3006

Value of Perfect Information (EVPI) (Avriel and Williams, 1970), a measure of the value of gathering additional information. In our setting, we use EVPI to calculate which question is most likely to elicit an answer that would make the post more informative. Formally, for an input post p, we want to choose a question q that maximizes $\mathbb{E}_{a\sim p,q}[\mathbb{U}(p+a)]$, where a is a hypothetical answer and U is a utility function measuring the *completeness* of post p if a were to be added to it. To achieve this, we construct two models: (1) an answer model, which estimates $\mathbb{P}[a \mid p, q]$, the likelihood of receiving answer a if one were to ask question q on post p; (2) a completeness model, $\mathbb{U}(p)$, which measures how complete a post is. Given these two models, at prediction time we search over a shortlist of possible questions for that which maximizes the EVPI.

We are able to train these models jointly based on (p, q, a) triples that we extract automatically from StackExchange. Figure 1 depicts how we do this using StackExchange's edit history. In the figure, the initial post fails to state what version of Ubuntu is being run. In response to Parker's question in the comments section, Terry, the author of the post, edits the post to answer Parker's clarification question. We extract the initial post as p, question posted in the comments section as q, and edit to the original post as answer a to form our (p, q, a) triples.

Our results show significant improvements from using the EVPI formalism over both standard feedforward network architectures and bag-of-ngrams baselines, even when our system builds on strong information retrieval scaffolding. In comparison, without this scaffolding, the bag-of-ngrams model outperforms the feedforward network. We additionally analyze the difficulty of this task for non-expert humans.

2 Related Work

The problem of question generation has received sparse attention from the natural language processing community. Most prior work focuses on generating reading comprehension questions: given text, write questions that one might find on a standardized test (Vanderwende, 2008; Heilman, 2011; Rus et al., 2011). Comprehension questions, by definition, are answerable from the provided text. Clarification questions are not. Outside reading comprehension questions, Labutov et al. (2015) studied the problem of generating question templates via crowdsourcing, Liu et al. (2010) use template-based question generation to help authors write better related work sections, Mostafazadeh et al. (2016) consider question generation from images, and Artzi and Zettlemoyer (2011) use human-generated clarification questions to drive a semantic parser.

3 Model Description

In order to choose what question to ask, we build a neural network model inspired by the theory of expected value of perfect information (EVPI). EVPI is a measurement of: if I were to acquire information X, how useful would that be to me? However, because we haven't acquired X yet, we have to take this quantity in expectation over all possible X, weighted by each X's likelihood. In the question generation setting, for any given question q that we can ask, there is set A of possible answers that could be given. For each possible answer $a \in A$, there is some probability of getting that answer, and some utility if that were the answer we got. The value of this question q is the expected utility, over all possible answers. The theory of EVPI then states that we want to choose the question q that maximizes:

$$\arg\max_{q \in Q} \sum_{a \in A} \mathbb{P}[a|p, q]\mathbb{U}(p + a) \qquad (1)$$

In Eq 1, p is the post, q is a potential question from a set of candidate questions Q (§3.1) and a is a potential answer from a set of candidate answers A (§3.1). $\mathbb{P}[a|p, q]$ (§3.2) measures the probability of getting an answer a given an initial post p and a clarifying question q. $\mathbb{U}(p + a)$ (§3.3) measures how useful it would be if p were augmented with answer a. Finally, using these pieces, we build a joint neural network that we can optimize end-to-end over our data (§3.4). Figure 2 describes the behavior of our model during test time.

3.1 Question & Answer Candidate Generator

Given a post, our first step is to generate a set of candidate questions and answers. Our model learns to ask questions by looking at questions asked in previous similar situations. We first identify 10 posts similar to the given post in our dataset using Lucene[2] (a software extensively used in information retrieval) and then consider the ques-

[2] https://lucene.apache.org/

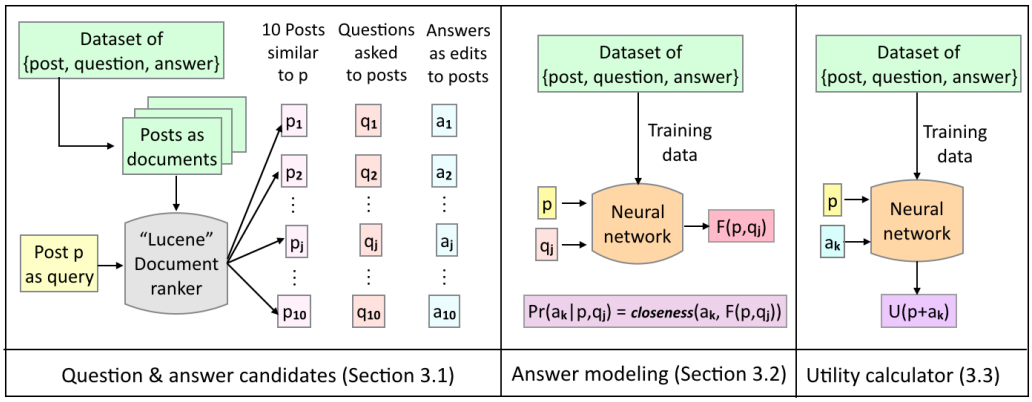

Figure 2: The behavior of our model during test time. Given a post p, we retrieve 10 posts similar to p using Lucene and consider the questions asked to those as question candidates and the edits made to the posts in response to the questions as answer candidates. Our answer model generates an answer representation $F_{ans}(p, q_j)$ for each question candidate q_j and calculates how close is an answer candidate a_k to $F_{ans}(p, q_j)$. Our utility calculator calculates the utility of the post if it were updated with the answer a_k. We select the question q_j that maximizes the expected utility of the post p (Equation 1).

tions asked to these posts as our set of question candidates and the edits made to the posts in response to the questions as our set of answer candidates.

3.2 Answer Modeling

Given a post p and a question candidate q_i, our second step is to calculate how likely is this question to be answered using one of our answer candidates a_k. To calculate this probability, we first generate an answer representation $F_{ans}(p, q_i)$ and then measure how close is the answer candidate a_k to our answer representation using the equation:

$$\mathbb{P}[a_k|p, q_i] = \frac{1}{Z} \exp\left[-\lambda||a_k - F_{ans}(p, q_i)||^2\right] \quad (2)$$

where λ is a tunable parameter that controls the variance of the distribution.

We train our answer generator using the following intuition: a question can be asked in several different ways. For e.g. in Figure 1, the question "What version of Ubuntu do you have?" can be asked in other ways like "What version of operating system are you using?", "Version of OS?", etc. Additionally, a question can generate several different answers. For instance, "Ubuntu 14.04 LTS", "Ubuntu 12.0", "Ubuntu 9.0", are all valid answers. To capture these generalizations, we define the following loss function:

$$\text{loss}_{\text{ans}}(\bar{p}, \bar{q}, \bar{a}, Q) = ||F_{ans}(\bar{p}, \bar{q}) - \bar{a}||^2 \quad (3)$$
$$+ \sum_{j \in Q} \left(||F_{ans}(\bar{p}, \bar{q}) - \bar{a_j}||^2 (1 - \tanh\left(||\bar{q} - \bar{q_j}||^2\right)) \right)$$

In equation 3, the first term forces the answer representation $F_{ans}(\bar{p_i}, \bar{q_i})$ to be as close as possible to the correct answer a_i and the second term forces it to be close to the answer a_j corresponding to a question q_j very similar to q_i (i.e. $||\bar{q_i} - \bar{q_j}||$ is near zero).

3.3 Utility Calculator

Given a post p and an answer candidate a_k, our third step is to calculate the utility of the updated post i.e. $\mathbb{U}(p + a_k)$ which measures how useful it would be if a given post p were augmented with an answer a_k. We use the intuition that a post p_i, when updated with the answer a_i that it is paired with in our dataset, would be more complete than if it is updated with some other answer a_j. Therefore we label all the (p_i, a_i) pairs from our dataset as positive ($y = 1$) and label p_i paired with other nine answer candidates generated using Lucene (§3.1) as negative ($y = 0$). The utility of the updated post is then defined as $\mathbb{U}(p + a) = \sigma(F_{utility}(\bar{p}, \bar{a}))$ where $F_{utility}$ is a feedforward neural network. We want this utility to be close to one for all the positively labelled (p, a) pairs and close to zero for all the negatively labelled (p, a) pairs. We therefore define our loss using the binary cross-entropy formulation below:

$$\text{loss}_{\text{util}}(y, \bar{p}, \bar{a}) = y \log(\sigma(F_{utility}(\bar{p}, \bar{a}))) \quad (4)$$

3.4 Our joint neural network model

Our fundamental representation is based on recurrent neural network, specifically long short-term memory architecture (LSTM) (Hochreiter and Schmidhuber, 1997) over word embeddings

Models	Lucene negative candidates				Random negative candidates			
	Acc	MRR	R@3	R@5	Acc	MRR	R@3	R@5
Random	10.0	29.3	30.0	50.0	10.0	29.3	30.0	50.0
Bag-of-ngrams	11.6	31.3	32.5	54.6	54.9	70.5	83.1	92.0
Feed-forward	17.4	37.8	43.2	63.9	49.0	66.8	81.3	92.8
EVPI	**23.3**	**43.4**	**51.0**	**70.3**	**61.1**	**75.5**	**87.9**	**95.8**

Table 1: Results of two setups 'Lucene negative candidates' and 'Random negative candidates' on askubuntu when trained on a combination of three domains: askubuntu, unix and superuser. We report four metrics: accuracy (percent of time the top ranked question was correct), mean reciprocal rank (the reciprocal of the ranked position of the correct question in the top 10 list), recall at 3 (percent of time the correct answer is in the top three) and recall at 5.

obtained using a GloVe (Pennington et al., 2014) model trained on the entire datadump of StackExchange. We define three LSTMs corresponding to p, q and a and two feedforward neural networks corresponding to our answer model $F_{ans}(\bar{p}, \bar{q})$ and our utility calculator $F_{utility}(\bar{p}, \bar{a})$. We jointly train the parameters of all our neural network models to minimize the sum of the loss of our answer model (Eq 3) and our utility calculator (Eq 4):

$$\sum_i \text{loss}_{\text{ans}}(\bar{p}_i, \bar{q}_i, \bar{a}_i, Q_i) + \text{loss}_{\text{util}}(y_i, \bar{p}_i, \bar{a}_i) \quad (5)$$

Given such an estimate $\mathbb{P}[a|p, q]$ of an answer and a utility $\mathbb{U}(p + a)$ of the updated post, predictions can be done by choosing that "q" that maximizes Eq 1.

4 Experiments and Results

4.1 Dataset

StackExchange is a network of online question answering websites containing timestamped information about the posts, comments on the post and the history of the revisions made to the post. Using this, we create our dataset of {*post, question, answer*} triples: where *post* is the initial unedited post, *question* is the comment containing a question and *answer* is the edit made to the post that matches the question comment [3]. We extract a total of 37K triples from the following three domains of StackExchange: askubuntu, unix and superuser.

4.2 Experimental Setups

We define our task as given a post and 10 question candidates, select the correct question candidate. For every post p in our dataset of (p, q, a) triples, the question q paired with p is our positive question candidate. We define two approaches to generate negative question candidates:

Lucene Negative Candidates: We retrieve nine question candidates using Lucene (§3.1) and

Random Negative Candidates: We randomly sample nine other questions from our dataset.

[3] We measure the cosine similarity between the averaged word embeddings of the question and the edit.

4.3 Primary Research Questions

Our primary research questions that we evaluate experimentally are:

a. Does a neural architecture improve upon a simple bag-of-ngrams baseline?

b. Does the EVPI formalism provide leverage over a similarly expressive feed-forward network?

c. How much harder is the task when the negative candidate questions come from Lucene rather than selected randomly?

4.4 Baseline Methods

Random: Randomly permute the set of 10 candidate questions uniformly.

Bag-of-ngrams: Construct a bag-of-ngrams representation for the post, the question and the answer and train a classifier to minimize hinge loss on misclassification loss.

Feed-forward neural: Concatenate the post LSTM representation, the question LSTM representation and the answer LSTM representation and feed it through a feed forward neural network of two fully-connected hidden layers.

4.5 Results

We describe results on a test split of askubuntu when our models are trained on the union of all data, summarized in Table 1. The left half of this table shows results when the candidate sets is from Lucene—the "hard" setting and the right half of this table shows the same results when the candidate set is chosen randomly—the "easy" setting. Here, we see that for all the evaluation metrics, EVPI outperforms all the baselines by at least a few percentage points. A final performance of 51% recall at 3 in the "hard" setting is encouraging, though clearly there is a long way to go for a perfect system.

5 How good are humans at this task?

In this section we address two natural questions: (a) How does the performance of our system compare to a human solving the same task? (b) Just because the system selects a question that is not the exact gold standard question, is it certainly wrong? To answer these questions, we had 14 computer science graduate students perform the task on 50 examples. Most of these graduate students are *not* experts in unix or ubuntu, but are knowledgable. Given a post and a randomized list of ten possible questions, they were instructed to select what they thought was the *single* best question to ask, and additionally mark as "valid" any additional questions that they thought would also be okay to ask. We also asked them to rate their confidence in $\{0, 1, 2, 3\}$. Most found this task quite challenging because many of the questions are about subtle nuances of operating system behavior.

These annotator's accuracy on the "hard" task of Lucene-selected questions, was only 36%, significantly better than our best system (23%), but still far from perfect. If we limited to those examples on which they were more confident (confidence of 2 or 3), their accuracy raised to 42%, but never surpassed that. A major problem for the human annotators is the amount of background knowledge required to solve this problem. On an easier domain, or with annotators who are truly experts, we might expect these numbers to be higher.

6 Proposed Research Directions

In our preliminary work, we focus on the question selection problem i.e. select the right clarification question from a set of prior questions. To enable our system to generalize well to new context, we propose two future research directions:

6.1 Template Based Question Generation

Consider a template like "What version of ___ are you running?". This template can generate thousands of specific variants found in the data like "What version of Ubuntu are you running?", "What version of apt-get are you running?", etc. We propose the following four step approach to our template-based question generation method:

1. Cluster questions based on their lexical and semantic similarity.
2. Generate a template for each cluster by removing topic specific words from questions.
3. Given a post, select a question template from a set of candidate question templates using a model similar to our preliminary work.
4. Finally, fill in the blanks in the template using topic specific words retrieved from the post.

6.2 Neural Network Generative Model

Sequence-to-sequence neural network models have proven to be effective for several language generation tasks like machine translation (Sutskever et al., 2014), dialog generation (Serban et al., 2016), etc. These models are based on an encoder-decoder framework where the encoder takes in a sequence of words and generates a vector representation which is then taken in by a decoder to generate the output sequence of words.

On similar lines, we propose a model for generating the clarification question one word at a time, given the words of a post. A recent neural generative question answering model (Yin et al., 2016) built an answer language model which decides, at each time step, whether to generate a common vocabulary word or an answer word retrieved from a knowledge base. Inspired from this work, we propose to build a question generation model which will decide, at each time step, whether to generate a common vocabulary word or a topic specific word retrieved from the current post, thus incorporating the template-based method into a more general neural network framework.

7 Conclusion

In our work, we introduce a novel dataset for clarification question generation, and build a model that integrates neural network structure with the classic notion of expected value of perfect information. Our preliminary model learns to select the right question from a set of candidate questions. We propose two future directions for automatically generating clarification questions.

One main avenue for improvement of this work is in evaluation: given that this task is so difficult for humans, but also given that there is no single right question to ask, how can we better measure performance at this task? This is exactly the same question faced in dialog and generation (Paek, 2001; Lowe et al., 2015; Liu et al., 2016; Kannan and Vinyals, 2017). Finally, asking question is a natural component of dialog, and building a collaborative dialog system that can naturally converse with a user is a broad long term goal.

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Empirical Methods in Natural Language Processing*. pages 421–432.

Mordecai Avriel and AC Williams. 1970. The value of information and stochastic programming. *Operations Research* 18(5):947–954.

Jonathan Ginzburg. 2012. *The interactive stance*. Oxford University Press.

Michael Heilman. 2011. *Automatic factual question generation from text*. Ph.D. thesis, Carnegie Mellon University.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* pages 1735–1780.

Anjuli Kannan and Oriol Vinyals. 2017. Adversarial evaluation of dialogue models. *arXiv preprint arXiv:1701.08198* .

Igor Labutov, Sumit Basu, and Lucy Vanderwende. 2015. Deep questions without deep understanding. In *Association for Computational Linguistics*. pages 889–898.

Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Empirical Methods in Natural Language Processing*. pages 2122–2132.

Ming Liu, Rafael A Calvo, and Vasile Rus. 2010. Automatic question generation for literature review writing support. In *International Conference on Intelligent Tutoring Systems*. Springer, pages 45–54.

Ryan Lowe, Nissan Pow, Iulian V Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Special Interest Group on Discourse and Dialogue*.

Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende. 2016. Generating natural questions about an image. In *Association for Computational Linguistics*. pages 1802–1813.

Tim Paek. 2001. Empirical methods for evaluating dialog systems. In *Proceedings of the workshop on Evaluation for Language and Dialogue Systems-Volume 9*. Association for Computational Linguistics, page 2.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods on Natural Language Processing*.

Matthew Richard John Purver. 2004. *The theory and use of clarification requests in dialogue*. Ph.D. thesis, Citeseer.

Vasile Rus, Paul Piwek, Svetlana Stoyanchev, Brendan Wyse, Mihai Lintean, and Cristian Moldovan. 2011. Question generation shared task and evaluation challenge: Status report. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 318–320.

Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Lucy Vanderwende. 2008. The importance of being important: Question generation. In *Proceedings of the 1st Workshop on the Question Generation Shared Task Evaluation Challenge, Arlington, VA*.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. 2016. Neural generative question answering. *International Joint Conference on Artificial Intelligence* .