Learning to Ask Good Questions: Ranking Clarification Questions using Neural Expected Value of Perfect Information

Sudha Rao
University of Maryland, College Park
raosudha@cs.umd.edu

Hal Daumé III
University of Maryland, College Park
Microsoft Research, New York City
hal@cs.umd.edu

Abstract

Inquiry is fundamental to communication, and machines cannot effectively collaborate with humans unless they can ask questions. In this work, we build a neural network model for the task of ranking clarification questions. Our model is inspired by the idea of expected value of perfect information: a good question is one whose expected answer will be useful. We study this problem using data from StackExchange, a plentiful online resource in which people routinely ask clarifying questions to posts so that they can better offer assistance to the original poster. We create a dataset of clarification questions consisting of ∼77K posts paired with a clarification question (and answer) from three domains of StackExchange: askubuntu, unix and superuser. We evaluate our model on 500 samples of this dataset against expert human judgments and demonstrate significant improvements over controlled baselines.

1 Introduction

A principle goal of asking questions is to fill information gaps, typically through clarification questions.[1] We take the perspective that a good question is the one whose *likely answer* will be useful. Consider the exchange in Figure 1, in which an initial poster (who we call "Terry") asks for help configuring environment variables. This post is underspecified and a responder ("Parker") asks a clarifying question (a) below, but could alternatively have asked (b) or (c):

(a) What version of Ubuntu do you have?

[1]We define 'clarification question' as a question that asks for some information that is currently missing from the given context.

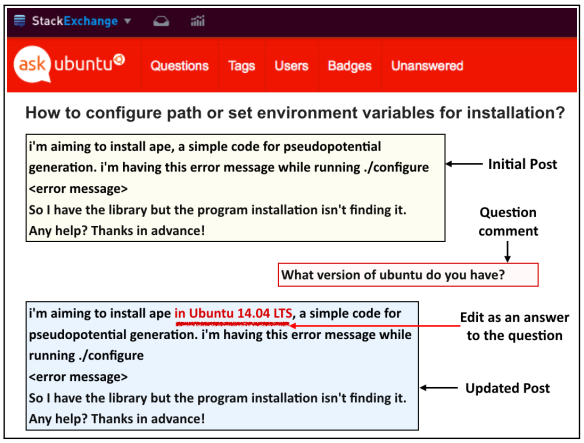

Figure 1: A post on an online Q & A forum "askubuntu.com" is updated to fill the missing information pointed out by the question comment.

(b) What is the make of your wifi card?

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

Parker should not ask (b) because an answer is unlikely to be useful; they should not ask (c) because it is too specific and an answer like "No" or "I do not know" gives little help. Parker's question (a) is much better: it is both likely to be useful, and is plausibly answerable by Terry.

In this work, we design a model to rank a candidate set of clarification questions by their usefulness to the given post. We imagine a use case (more discussion in § 7) in which, while Terry is writing their post, a system suggests a shortlist of questions asking for information that it thinks people like Parker might need to provide a solution, thus enabling Terry to immediately clarify their post, potentially leading to a much quicker resolution. Our model is based on the decision theoretic framework of the Expected 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 questions are most likely to elicit an answer that would make the post more informative.

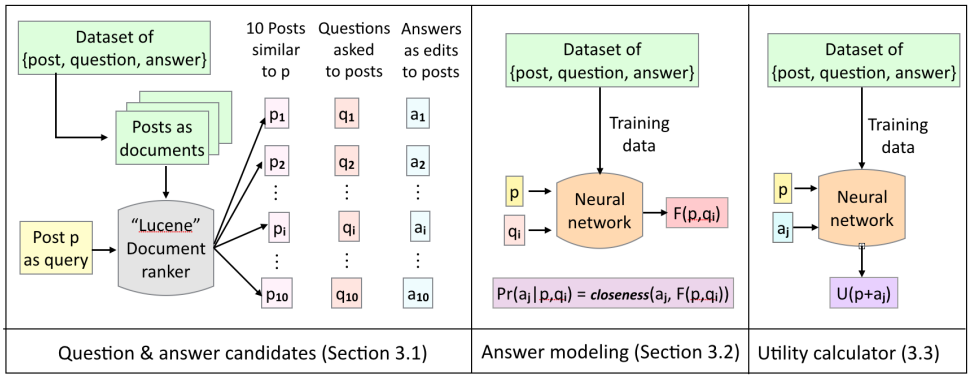

Figure 2: The behavior of our model during test time: Given a post p, we retrieve 10 posts similar to post p using Lucene. The questions asked to those 10 posts are our question candidates Q and the edits made to the posts in response to the questions are our answer candidates A. For each question candidate q_i, we generate an answer representation $F(p, q_i)$ and calculate how close is the answer candidate a_j to our answer representation $F(p, q_i)$. We then calculate the utility of the post p if it were updated with the answer a_j. Finally, we rank the candidate questions Q by their expected utility given the post p (Eq 1).

Our work has two main contributions:

1. A novel neural-network model for addressing the task of ranking clarification question built on the framework of expected value of perfect information (§2).
2. A novel dataset, derived from StackExchange[2], that enables us to learn a model to ask clarifying questions by looking at the types of questions people ask (§3).

We formulate this task as a ranking problem on a set of potential clarification questions. We evaluate models both on the task of returning the original clarification question and also on the task of picking any of the candidate clarification questions marked as good by experts (§4). We find that our EVPI model outperforms the baseline models when evaluated against expert human annotations. We include a few examples of human annotations along with our model performance on them in the supplementary material. We have released our dataset of ∼77K (p, q, a) triples and the expert annotations on 500 triples to help facilitate further research in this task.[3]

2 Model description

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 our setting, for any given question q_i that we can ask, there is a set A of possible answers that could be given. For each possible answer $a_j \in A$, there is some probability of getting that answer, and some utility if that were the answer we got. The value of this question q_i is the expected utility, over all possible answers:

$$\text{EVPI}(q_i|p) = \sum_{a_j \in A} \mathbb{P}[a_j|p, q_i]\mathbb{U}(p + a_j) \quad (1)$$

In Eq 1, p is the post, q_i is a potential question from a set of candidate questions Q and a_j is a potential answer from a set of candidate answers A. Here, $\mathbb{P}[a_j|p, q_i]$ measures the probability of getting an answer a_j given an initial post p and a clarifying question q_i, and $\mathbb{U}(p + a_j)$ is a utility function that measures how much more complete p would be if it were augmented with answer a_j. The modeling question then is how to model:

1. The probability distribution $\mathbb{P}[a_j|p, q_i]$ and
2. The utility function $\mathbb{U}(p + a_j)$.

In our work, we represent both using neural networks over the appropriate inputs. We train the parameters of the two models jointly to minimize a joint loss defined such that an answer that has a higher potential of increasing the utility of a post gets a higher probability.

Figure 2 describes the behavior of our model during test time. Given a post p, we generate a set of candidate questions and a set of candidate

[2]We use data from StackExchange; per license cc-by-sa 3.0, the data is "intended to be shared and remixed" (with attribution).

[3]https://github.com/raosudha89/ranking_clarification_questions

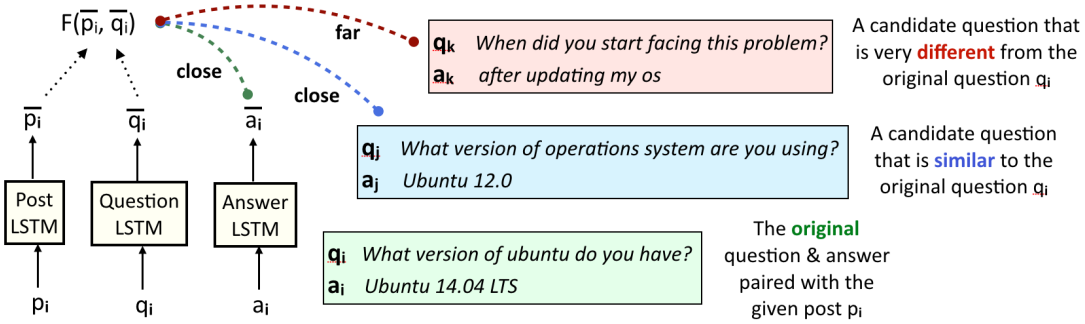

Figure 3: Training of our answer generator. Given a post p_i and its question q_i, we generate an answer representation that is not only close to its original answer a_i, but also close to one of its candidate answers a_j if the candidate question q_j is close to the original question q_i.

answers (§2.1). Given a post p and a question candidate q_i, we calculate how likely is this question to be answered using one of our answer candidates a_j (§2.2). Given a post p and an answer candidate a_j, we calculate the utility of the updated post i.e. $\mathbb{U}(p + a_j)$ (§2.3). We compose these modules into a joint neural network that we optimize end-to-end over our data (§2.4).

2.1 Question & answer candidate generator

Given a post p, our first step is to generate a set of question and answer candidates. One way that humans learn to ask questions is by looking at how others ask questions in a similar situation. Using this intuition we generate question candidates for a given post by identifying posts similar to the given post and then looking at the questions asked to those posts. For identifying similar posts, we use Lucene[4], a software extensively used in information retrieval for extracting documents relevant to a given query from a pool of documents. Lucene implements a variant of the term frequency-inverse document frequency (TF-IDF) model to score the extracted documents according to their relevance to the query. We use Lucene to find the top 10 posts most similar to a given post from our dataset (§3). We consider the questions asked to these 10 posts as our set of question candidates Q and the edits made to the posts in response to the questions as our set of answer candidates A. Since the top-most similar candidate extracted by Lucene is always the original post itself, the original question and answer paired with the post is always one of the candidates in Q and A. §3 describes in detail the process of extracting the (*post, question, answer*) triples from the StackExchange datadump.

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

2.2 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_j. We first generate an answer representation by combining the neural representations of the post and the question using a function $F_{ans}(\bar{p}, \bar{q}_i)$ (details in §2.4). Given such a representation, we measure the distance between this answer representation and one of the answer candidates a_j using the function below:

$$dist(F_{ans}(\bar{p},\ \bar{q}_i),\ \hat{a}_j) = 1 - cos\_sim(F_{ans}(\bar{p},\ \bar{q}_i),\ \hat{a}_j)$$

The likelihood of an answer candidate a_j being the answer to a question q_i on post p is finally calculated by combining this distance with the cosine similarity between the question q_i and the question q_j paired with the answer candidate a_j:

$$\mathbb{P}[a_j|p, q_i] = \exp^{-dist(F_{ans}(\bar{p},\ \bar{q}_i),\ \hat{a}_j)} * cos\_sim(\hat{q}_i,\ \hat{q}_j) \quad (2)$$

where $\hat{a}_j$, $\hat{q}_i$ and $\hat{q}_j$ are the average word vector of a_j, q_i and q_j respectively (details in §2.4) and cos_sim is the cosine similarity between the two input vectors.

We model 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, for a given post and a question, there can be

several different answers to that question. For instance, "Ubuntu 14.04 LTS", "Ubuntu 12.0", "Ubuntu 9.0", are all valid answers. To generate an answer representation capturing these generalizations, we train our answer generator on our triples dataset (§3) using the loss function below:

$$\text{loss}_{\text{ans}}(p_i, q_i, a_i, Q_i) = dist(F_{ans}(\bar{p}_i, \bar{q}_i),\ \hat{a}_i) + \sum_{j \in Q} \Big(dist(F_{ans}(\bar{p}_i, \bar{q}_i),\ \hat{a}_j) * cos\_sim(\hat{q}_i,\ \hat{q}_j) \Big) \quad (3)$$

where, $\hat{a}$ and $\hat{q}$ is the average word vectors of a and q respectively (details in §2.4), cos_sim is the cosine similarity between the two input vectors.

This loss function can be explained using the example in Figure 3. Question q_i is the question paired with the given post p_i. In Eq 3, the first term forces the function $F_{ans}(\bar{p}_i, \bar{q}_i)$ to generate an answer representation as close as possible to the correct answer a_i. Now, a question can be asked in several different ways. Let Q_i be the set of candidate questions for post p_i, retrieved from the dataset using Lucene (§ 2.1). Suppose a question candidate q_j is very similar to the correct question q_i (i.e. $cos\_sim(\hat{q}_i,\ \hat{q}_j)$ is near zero). Then the second term forces the answer representation $F_{ans}(\bar{p}_i, \bar{q}_i)$ to be close to the answer a_j corresponding to the question q_j as well. Thus in Figure 3, the answer representation will be close to a_j (since q_j is similar to q_i), but may not be necessarily close to a_k (since q_k is dissimilar to q_i).

2.3 Utility calculator

Given a post p and an answer candidate a_j, the third step is to calculate the utility of the updated post i.e. $\mathbb{U}(p + a_j)$. As expressed in Eq 1, this utility function measures how useful it would be if a given post p were augmented with an answer a_j paired with a different question q_j in the candidate set. Although theoretically, the utility of the updated post can be calculated only using the given post (p) and the candidate answer (a_j), empirically we find that our neural EVPI model performs better when the candidate question (q_j) paired with the candidate answer is a part of the utility function. We attribute this to the fact that much information about whether an answer increases the utility of a post is also contained in the question asked to the post. We train our utility calculator using our dataset of (p, q, a) triples (§3). We label all the (p_i, q_i, a_i) pairs from our triples dataset with label $y = 1$. To get negative samples, we make use of the answer candidates generated using Lucene as described in §2.1. For each $a_j \in A_i$, where A_i is the set of answer candidates for post p_i, we label the pair (p_i, q_j, a_j) with label $y = 0$, except for when $a_j = a_i$. Thus, for each post p_i in our triples dataset, we have one positive sample and nine negative samples. It should be noted that this is a noisy labelling scheme since a question not paired with the original question in our dataset can often times be a *good* question to ask to the post (§4). However, since we do not have annotations for such other good questions at train time, we assume such a labelling.

Given a post p_i and an answer a_j paired with the question q_j, we combine their neural representations using a function $F_{util}(\bar{p}_i, \bar{q}_j, \bar{a}_j)$ (details in § 2.4). The utility of the updated post is then defined as $\mathbb{U}(p_i + a_j) = \sigma(F_{util}(\bar{p}_i, \bar{q}_j, \bar{a}_j))$[5]. We want this utility to be close to 1 for all the positively labelled (p, q, a) triples and close to 0 for all the negatively labelled (p, q, a) triples. We therefore define our loss using the binary cross-entropy formulation below:

$$\text{loss}_{\text{util}}(y_i, \bar{p}_i, \bar{q}_j, \bar{a}_j) = y_i \log(\sigma(F_{util}(\bar{p}_i, \bar{q}_j, \bar{a}_j))) \quad (4)$$

2.4 Our joint neural network model

Our fundamental representation is based on recurrent neural networks over word embeddings. We obtain the word embeddings using the GloVe (Pennington et al., 2014) model trained on the entire datadump of StackExchange.[6]. In Eq 2 and Eq 3, the average word vector representations $\hat{q}$ and $\hat{a}$ are obtained by averaging the GloVe word embeddings for all words in the question and the answer respectively. Given an initial post p, we generate a post neural representation $\bar{p}$ using a post LSTM (long short-term memory architecture) (Hochreiter and Schmidhuber, 1997). The input layer consists of word embeddings of the words in the post which is fed into a single hidden layer. The output of each of the hidden states is averaged together to get our neural representation $\bar{p}$. Similarly, given a question q and an answer a, we generate the neural representations $\bar{q}$ and $\bar{a}$ using a question LSTM and an answer LSTM respectively. We define the function F_{ans} in our answer model as a feedforward neural network with five hidden layers on the inputs $\bar{p}$ and $\bar{q}$. Likewise, we

[5] σ is the sigmoid function.

[6] Details in the supplementary material.

define the function F_{util} in our utility calculator as a feedforward neural network with five hidden layers on the inputs $\bar{p}$, $\bar{q}$ and $\bar{a}$. We train the parameters of the three LSTMs corresponding to p, q and a, and the parameters of the two feedforward neural networks jointly to minimize the sum of the loss of our answer model (Eq 3) and our utility calculator (Eq 4) over our entire dataset:

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

Given such an estimate $\mathbb{P}[a_j|p, q_i]$ of an answer and a utility $\mathbb{U}(p + a_j)$ of the updated post, we rank the candidate questions by their value as calculated using Eq 1. The remaining question, then, is how to get data that enables us to train our answer model and our utility calculator. Given data, the training becomes a multitask learning problem, where we learn simultaneously to predict utility and to estimate the probability of answers.

3 Dataset creation

StackExchange is a network of online question answering websites about varied topics like academia, ubuntu operating system, latex, etc. The data dump of StackExchange contains timestamped information about the posts, comments on the post and the history of the revisions made to the post. We use this data dump to create our dataset of (*post, question, answer*) triples: where the *post* is the initial unedited post, the *question* is the comment containing a question and the *answer* is either the edit made to the post after the question or the author's response to the question in the comments section.

Extract posts: We use the post histories to identify posts that have been updated by its author. We use the timestamp information to retrieve the initial unedited version of the post.

Extract questions: For each such initial version of the post, we use the timestamp information of its comments to identify the first question comment made to the post. We truncate the comment till its question mark '?' to retrieve the question part of the comment. We find that about 7% of these are rhetoric questions that indirectly suggest a solution to the post. For e.g. *"have you considered installing X?"*. We do a manual analysis of these non-clarification questions and hand-crafted a few rules to remove them. [7]

	Train	Tune	Test
askubuntu	19,944	2493	2493
unix	10,882	1360	1360
superuser	30,852	3857	3856

Table 1: Table above shows the sizes of the train, tune and test split of our dataset for three domains.

Extract answers: We extract the answer to a clarification question in the following two ways:
(a) Edited post: Authors tend to respond to a clarification question by editing their original post and adding the missing information. In order to account for edits made for other reasons like stylistic updates and grammatical corrections, we consider only those edits that are longer than four words. Authors can make multiple edits to a post in response to multiple clarification questions.[8] To identify the edit made corresponding to the given question comment, we choose the edit closest in time following the question.
(b) Response to the question: Authors also respond to clarification questions as subsequent comments in the comment section. We extract the first comment by the author following the clarification question as the answer to the question.

In cases where both the methods above yield an answer, we pick the one that is the most semantically similar to the question, where the measure of similarity is the cosine distance between the average word embeddings of the question and the answer.

We extract a total of 77,097 (*post, question, answer*) triples across three domains in StackExchange (Table 1). We will release this dataset along with the the nine question and answer candidates per triple that we generate using lucene (§ 2.1). We include an analysis of our dataset in the supplementary material.

4 Evaluation design

We define our task as given a post p, and a set of candidate clarification questions Q, rank the questions according to their usefulness to the post.

[7] Details in the supplementary material.

[8] On analysis, we find that 35%-40% of the posts get asked multiple clarification questions. We include only the first clarification question to a post in our dataset since identifying if the following questions are clarifications or a part of a dialogue is non-trivial.

Since the candidate set includes the original question q that was asked to the post p, one possible approach to evaluation would be to look at how often the original question is ranked higher up in the ranking predicted by a model. However, there are two problems to this approach: 1) Our dataset creation process is noisy. The original question paired with the post may not be a useful question. For e.g. *"are you seriously asking this question?"*, *"do you mind making that an answer?"*[9]. 2) The nine other questions in the candidate set are obtained by looking at questions asked to posts that are similar to the given post.[10] This greatly increases the possibility of some other question(s) being more useful than the original question paired with the post. This motivates an evaluation design that does not rely solely on the original question but also uses human judgments. We randomly choose a total of 500 examples from the test sets of the three domains proportional to their train set sizes (`askubuntu:160`, `unix:90` and `superuser:250`) to construct our evaluation set.

4.1 Annotation scheme

Due to the technical nature of the posts in our dataset, identifying useful questions requires technical experts. We recruit 10 such experts on Upwork[11] who have prior experience in unix based operating system administration.[12] We provide the annotators with a post and a randomized list of the ten question candidates obtained using Lucene (§2.1) and ask them to select a *single* "best" (B) question to ask, and additionally mark as "valid" (V) other questions that they thought would be okay to ask in the context of the original post. We enforce that the "best" question be always marked as a "valid" question. We group the 10 annotators into 5 pairs and assign the same 100 examples to the two annotators in a pair.

4.2 Annotation analysis

We calculate the inter-annotator agreement on the "best" and the "valid" annotations using Cohen's Kappa measurement. When calculating the agreement on the "best" in the strict sense, we get a low agreement of 0.15. However, when we relax this to a case where the question marked as"best" by one annotator is marked as "valid" by another, we get an agreement of 0.87. The agreement on the "valid" annotations, on the other hand, was higher: 0.58. We calculate this agreement on the binary judgment of whether a question was marked as valid by the annotator.

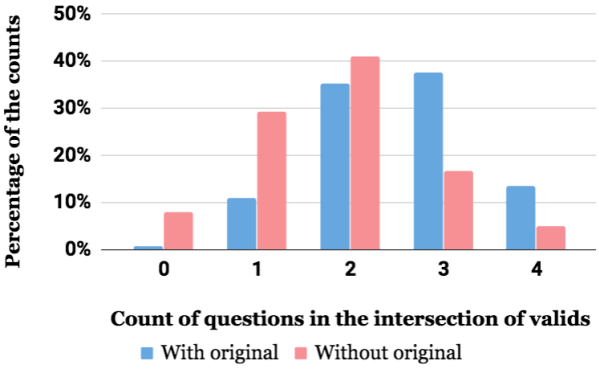

Figure 4: Distribution of the count of questions in the intersection of the "valid" annotations.

Given these annotations, we calculate how often is the original question marked as "best" or "valid" by the two annotators. We find that 72% of the time one of the annotators mark the original as the "best", whereas only 20% of the time both annotators mark it as the "best" suggesting against an evaluation solely based on the original question. On the other hand, 88% of the time one of the two annotators mark it as a "valid" question confirming the noise in our training data.[13]

Figure 4 shows the distribution of the counts of questions in the intersection of "valid" annotations (blue legend). We see that about 85% of the posts have more than 2 valid questions and 50% have more than 3 valid questions. The figure also shows the distribution of the counts when the original question is removed from the intersection (red legend). Even in this set, we find that about 60% of the posts have more than two valid questions. These numbers suggests that the candidate set of questions retrieved using Lucene (§2.1) very often contains useful clarification questions.

5 Experimental results

Our primary research questions that we evaluate experimentally are:

1. Does a neural network architecture improve upon non-neural baselines?

[9] Data analysis included in the supplementary material suggests 9% of the questions are not useful.

[10] Note that this setting is different from the distractor-based setting popularly used in dialogue (Lowe et al., 2015) where the distractor candidates are chosen randomly from the corpus.

[11] https://upwork.com

[12] Details in the supplementary material.

[13] 76% of the time both the annotators mark it as a "valid".

Model	$B1 \cup B2$ p@1	p@3	p@5	MAP	$V1 \cap V2$ p@1	p@3	p@5	MAP	Original p@1
Random	17.5	17.5	17.5	35.2	26.4	26.4	26.4	42.1	10.0
Bag-of-ngrams	19.4	19.4	18.7	34.4	25.6	27.6	27.5	42.7	10.7
Community QA	23.1	21.2	20.0	40.2	33.6	30.8	29.1	47.0	18.5
Neural (p, q)	21.9	20.9	19.5	39.2	31.6	30.0	28.9	45.5	15.4
Neural (p, a)	24.1	**23.5**	20.6	41.4	32.3	**31.5**	29.0	46.5	18.8
Neural (p, q, a)	25.2	**22.7**	**21.3**	42.5	34.4	**31.8**	**30.1**	47.7	20.5
EVPI	**27.7**	**23.4**	**21.5**	**43.6**	**36.1**	**32.2**	**30.5**	**49.2**	**21.4**

Table 2: Model performances on 500 samples when evaluated against the union of the "best" annotations ($B1 \cup B2$), intersection of the "valid" annotations ($V1 \cap V2$) and the original question paired with the post in the dataset. The difference between the bold and the non-bold numbers is statistically significant with $p < 0.05$ as calculated using bootstrap test. p@k is the precision of the k questions ranked highest by the model and MAP is the mean average precision of the ranking predicted by the model.

2. Does the EVPI formalism provide leverage over a similarly expressive feedforward network?
3. Are answers useful in identifying the right question?
4. How do the models perform when evaluated on the candidate questions excluding the original?

5.1 Baseline methods

We compare our model with following baselines:

Random: Given a post, we randomly permute its set of 10 candidate questions uniformly.[14]

Bag-of-ngrams: Given a post and a set of 10 question and answer candidates, we construct a bag-of-ngrams representation for the post, question and answer. We train the baseline on all the positive and negative candidate triples (same as in our utility calculator (§2.3)) to minimize hinge loss on misclassification error using cross-product features between each of (*p, q*), (*q, a*) and (*p, a*). We tune the ngram length and choose n=3 which performs best on the tune set. The question candidates are finally ranked according to their predictions for the positive label.

Community QA: The recent SemEval2017 Community Question-Answering (CQA) (Nakov et al., 2017) included a subtask for ranking a set of comments according to their relevance to a given post in the Qatar Living[15] forum. Nandi et al. (2017), winners of this subtask, developed a logistic regression model using features based on string similarity, word embeddings, etc. We train this model on all the positively and negatively labelled (p, q) pairs in our dataset (same as in our utility calculator (§2.3), but without a). We use a subset of their features relevant to our task.[16]

Neural baselines: We construct the following neural baselines based on the LSTM representation of their inputs (as described in §2.4):
1. **Neural(p, q):** Input is concatenation of $\bar{p}$ and $\bar{q}$.
2. **Neural(p, a):** Input is concatenation of $\bar{p}$ and $\bar{a}$.
3. **Neural(p, q, a):** Input is concatenation of $\bar{p}$, $\bar{q}$ and $\bar{a}$.

Given these inputs, we construct a fully connected feedforward neural network with 10 hidden layers and train it to minimize the binary cross entropy across all positive and negative candidate triples (same as in our utility calculator (§2.3)). The major difference between the neural baselines and our EVPI model is in the loss function: the EVPI model is trained to minimize the joint loss between the answer model (defined on $F_{ans}(p, q)$ in Eq 3) and the utility calculator (defined on $F_{util}(p, q, a)$ in Eq 4) whereas the neural baselines are trained to minimize the loss directly on $F(p, q)$, $F(p, a)$ or $F(p, q, a)$. We include the implementation details of all our neural models in the supplementary material.

5.2 Results

5.2.1 Evaluating against expert annotations

We first describe the results of the different models when evaluated against the expert annotations we collect on 500 samples (§4). Since the annotators

[14] We take the average over 1000 random permutations.
[15] http://www.qatarliving.com/forum
[16] Details in the supplementary material.

had a low agreement on a single best, we evaluate against the union of the "best" annotations ($B1 \cup B2$ in Table 2) and against the intersection of the "valid" annotations ($V1 \cap V2$ in Table 2).

Among non-neural baselines, we find that the bag-of-ngrams baseline performs slightly better than random but worse than all the other models. The Community QA baseline, on the other hand, performs better than the neural baseline (Neural (p, q)), both of which are trained without using the answers. The neural baselines with answers (Neural(p, q, a) and Neural(p, a)) outperform the neural baseline without answers (Neural(p, q)), showing that answer helps in selecting the right question.

More importantly, EVPI outperforms the Neural (p, q, a) baseline across most metrics. Both models use the same information regarding the true question and answer and are trained using the same number of model parameters.[17] However, the EVPI model, unlike the neural baseline, additionally makes use of alternate question and answer candidates to compute its loss function. This shows that when the candidate set consists of questions similar to the original question, summing over their utilities gives us a boost.

5.2.2 Evaluating against the original question

The last column in Table 2 shows the results when evaluated against the original question paired with the post. The bag-of-ngrams baseline performs similar to random, unlike when evaluated against human judgments. The Community QA baseline again outperforms Neural(p, q) model and comes very close to the Neural (p, a) model.

As before, the neural baselines that make use of the answer outperform the one that does not use the answer and the EVPI model performs significantly better than Neural(p, q, a).

5.2.3 Excluding the original question

In the preceding analysis, we considered a setting in which the "ground truth" original question was in the candidate set Q. While this is a common evaluation framework in dialog response selection (Lowe et al., 2015), it is overly optimistic. We, therefore, evaluate against the "best" and the "valid" annotations on the nine other question candidates. We find that the neural models beat the non-neural baselines. However, the differences between all the neural models are statistically insignificant.[18]

[17] We use 10 hidden layers in the feedforward network of the neural baseline and five hidden layers each in the two feedforward networks F_{ans} and F_{util} of the EVPI model.

6 Related work

Most prior work on question generation has focused 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; Olney et al., 2012). Comprehension questions, by definition, are answerable from the provided text. Clarification questions–our interest–are not.

Outside reading comprehension questions, Labutov et al. (2015) generate high-level question templates by crowdsourcing which leads to significantly less data than we collect using our method. Liu et al. (2010) use template question generation to help authors write better related work sections. Mostafazadeh et al. (2016) introduce a Visual Question Generation task where the goal is to generate natural questions that are not about what is present in the image rather about what can be inferred given the image, somewhat analogous to clarification questions. Penas and Hovy (2010) identify the notion of missing information similar to us, but they fill the knowledge gaps in a text with the help of external knowledge bases, whereas we instead ask clarification questions. Artzi and Zettlemoyer (2011) use human-generated clarification questions to drive a semantic parser where the clarification questions are aimed towards simplifying a user query; whereas we generate clarification questions aimed at identifying missing information in a text.

Among works that use community question answer forums, the keywords to questions (K2Q) system (Zheng et al., 2011) generates a list of candidate questions and refinement words, given a set of input keywords, to help a user ask a better question. Figueroa and Neumann (2013) rank different paraphrases of query for effective search on forums. (Romeo et al., 2016) develop a neural network based model for ranking questions on forums with the intent of retrieving similar other question. The recent SemEval-2017 Community Question-Answering (CQA) (Nakov et al., 2017) task included a subtask to rank the comments according to their relevance to the post. Our task primarily differs from this task in that we want to identify a

[18] Results included in the supplementary material.

question comment which is not only relevant to the post but will also elicit useful information missing from the post. Hoogeveen et al. (2015) created the CQADupStack dataset using StackExchange forums for the task of duplicate question retrieval. Our dataset, on the other hand, is designed for the task of ranking clarification questions asked as comments to a post.

7 Conclusion

We have constructed a new dataset for learning to rank clarification questions, and proposed a novel model for solving this task. Our model integrates well-known deep network architectures with the classic notion of expected value of perfect information, which effectively models a pragmatic choice on the part of the questioner: how do I *imagine* the other party would answer if I were to ask this question. Such pragmatic principles have recently been shown to be useful in other tasks as well (Golland et al., 2010; Smith et al., 2013; Orita et al., 2015; Andreas and Klein, 2016). One can naturally extend our EVPI approach to a full reinforcement learning approach to handle multi-turn conversations.

Our results shows that the EVPI model is a promising formalism for the question generation task. In order to move to a full system that can help users like Terry write better posts, there are three interesting lines of future work. First, we need it to be able to generalize: for instance by constructing templates of the form "What version of ___ are you running?" into which the system would need to fill a variable. Second, in order to move from question ranking to question generation, one could consider sequence-to-sequence based neural network models that have recently proven to be effective for several language generation tasks (Sutskever et al., 2014; Serban et al., 2016; Yin et al., 2016). Third is in evaluation: given that this task requires expert human annotations and also given that there are multiple possible good questions to ask, how can we automatically measure performance at this task?, a question faced in dialog and generation more broadly (Paek, 2001; Lowe et al., 2015; Liu et al., 2016).

Acknowledgments

The authors thank the three anonymous reviewers of this paper, and the anonymous reviewers of the previous versions for their helpful comments and suggestions. They also thank the members of the Computational Linguistics and Information Processing (CLIP) lab at University of Maryland for helpful discussions.

This work was supported by NSF grant IIS-1618193. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the view of the sponsors.

References

Jacob Andreas and Dan Klein. 2016. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. pages 1173–1182.

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, pages 421–432.

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

Alejandro Figueroa and Günter Neumann. 2013. Learning to rank effective paraphrases from query logs for community question answering. In *AAAI*. volume 13, pages 1099–1105.

Dave Golland, Percy Liang, and Dan Klein. 2010. A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 conference on empirical methods in natural language processing*. Association for Computational Linguistics, pages 410–419.

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* 9(8):1735–1780.

Doris Hoogeveen, Karin M Verspoor, and Timothy Baldwin. 2015. Cqadupstack: A benchmark data set for community question-answering research. In *Proceedings of the 20th Australasian Document Computing Symposium*. ACM, page 3.

Igor Labutov, Sumit Basu, and Lucy Vanderwende. 2015. Deep questions without deep understanding. In *ACL (1)*. pages 889–898.

Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike 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 *Proceedings of the*

2016 Conference on 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 *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. page 285.

Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende. 2016. Generating natural questions about an image. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 1802–1813.

Preslav Nakov, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin, and Karin Verspoor. 2017. Semeval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. pages 27–48.

Titas Nandi, Chris Biemann, Seid Muhie Yimam, Deepak Gupta, Sarah Kohail, Asif Ekbal, and Pushpak Bhattacharyya. 2017. Iit-uhh at semeval-2017 task 3: Exploring multiple features for community question answering and implicit dialogue identification. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. pages 90–97.

Andrew McGregor Olney, Arthur C Graesser, and Natalie K Person. 2012. Question generation from concept maps. *D&D* 3(2):75–99.

Naho Orita, Eliana Vornov, Naomi Feldman, and Hal Daumé III. 2015. Why discourse affects speakers' choice of referring expressions. In *ACL (1)*. pages 1639–1649.

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.

Anselmo Penas and Eduard Hovy. 2010. Filling knowledge gaps in text for machine reading. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics, pages 979–987.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. pages 1532–1543.

Salvatore Romeo, Giovanni Da San Martino, Alberto Barrón-Cedeno, Alessandro Moschitti, Yonatan Belinkov, Wei-Ning Hsu, Yu Zhang, Mitra Mohtarami, and James Glass. 2016. Neural attention for learning to rank questions in community question answering. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. pages 1734–1745.

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 Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*. pages 3776–3784.

Nathaniel J Smith, Noah Goodman, and Michael Frank. 2013. Learning and using language via recursive pragmatic reasoning about other agents. In *Advances in neural information processing systems*. pages 3039–3047.

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. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, pages 2972–2978.

Zhicheng Zheng, Xiance Si, Edward Chang, and Xiaoyan Zhu. 2011. K2q: Generating natural language questions from keywords with user refinements. In *Proceedings of 5th International Joint Conference on Natural Language Processing*. pages 947–955.