# Machine Comprehension by Text-to-Text Neural Question Generation

**Xingdi Yuan**[1*] **Tong Wang**[1,*] **Caglar Gulcehre**[2,*†] **Alessandro Sordoni**[1,*]
**Philip Bachman**[1] **Sandeep Subramanian**[2,†] **Saizheng Zhang**[2,†] **Adam Trischler**[1]

[1]Microsoft Maluuba, [2]Montreal Institute for Learning Algorithms, Université de Montréal
{eric.yuan,tong.wang,alsordon,phbachma,adam.trischler}@microsoft.com
gulcehrc@iro.umontreal.ca, sandeep.subramanian@gmail.com, saizheng.zhang@umontreal.ca

## Abstract

We propose a recurrent neural model that generates natural-language questions from documents, conditioned on answers. We show how to train the model using a combination of supervised and reinforcement learning. After teacher forcing for standard maximum likelihood training, we fine-tune the model using policy gradient techniques to maximize several rewards that measure question quality. Most notably, one of these rewards is the performance of a question-answering system. We motivate question generation as a means to improve the performance of question answering systems. Our model is trained and evaluated on the recent question-answering dataset *SQuAD*.

## 1 Introduction

People ask questions to improve their knowledge and understanding of the world. Questions can be used to access the knowledge of others or to direct one's own information-seeking behavior. Here we study the generation of natural-language questions by machines, based on text passages. This task is synergistic with *machine comprehension* (MC), which pursues the understanding of written language by machines at a near-human level. Because most human knowledge is recorded in text, this would enable transformative applications.

Many machine comprehension datasets have been released recently. These generally comprise (document, question, answer) triples (Hermann et al., 2015; Hill et al., 2015; Rajpurkar et al., 2016; Trischler et al., 2016a; Nguyen et al., 2016), where the goal is to predict an answer, conditioned on a document and question. The availability of large labeled datasets has spurred development of increasingly advanced models for question answering (QA) from text (Kadlec et al., 2016; Trischler et al., 2016b; Seo et al., 2016; Wang et al., 2016; Shen et al., 2016).

| **Text Passage** |
|---|
| **in 1066**[1,2] **, duke william ii**[3] of normandy conquered england killing king harold ii at the battle of hastings. **the invading normans and their descendants**[4] replaced the anglo-saxons as the ruling class of england. |
| **Questions Generated by our System** |
| 1) when did the battle of hastings take place? |
| 2) in what year was the battle of hastings fought? |
| 3) who conquered king harold ii at the battle of hastings? |
| 4) who became the ruling class of england? |

Table 1: Examples of conditional question generation given a context and an answer from the *SQuAD* dataset, using the scheme referred to as $R_{\text{PPL + QA}}$ below. Bold text in the passage indicates the answers used to generate the numbered questions.

In this paper we reframe the standard MC task: rather than *answering* questions about a document, we teach machines to *ask* questions. Our work has several motivations. First, we believe that posing appropriate questions is an important aspect of information acquisition in intelligent systems. Second, learning to ask questions may improve the ability to answer them. Singer and Donlan (1982) demonstrated that having students devise questions before reading can increase scores on subsequent comprehension tests. Third, answering the questions in most existing QA datasets is an *extractive* task – it requires selecting some span of text within the document – while question asking is comparatively *abstractive* – it requires generation of text that may not appear in the document. Fourth, asking good questions involves skills beyond those used to answer them. For instance, in existing QA

[*]Equal contribution.
[†]Supported by funding from Maluuba.



*Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 15–25,
Vancouver, Canada, August 3, 2017. 

datasets, a typical (document, question) pair specifies a unique answer. Conversely, a typical (document, answer) pair may be associated with multiple questions, since a valid question can be formed from any information or relations which uniquely specify the given answer. Finally, a mechanism to ask informative questions about documents (and eventually answer them) has many practical applications, e.g.: generating training data for question answering (Serban et al., 2016; Yang et al., 2017), synthesising frequently asked question (FAQ) documentation, and automatic tutoring systems (Lindberg et al., 2013).

We adapt the sequence-to-sequence approach of Cho et al. (2014) for generating questions, conditioned on a document and answer: first we encode the document and answer, then output question words sequentially with a decoder that conditions on the document and answer encodings. We augment the standard encoder-decoder approach with several modifications geared towards the question generation task. During training, in addition to maximum likelihood for predicting questions from (document, answer) tuples, we use policy gradient optimization to maximize several auxiliary rewards. These include a language-model-based score for fluency and the performance of a pretrained question-answering model on generated questions. We show quantitatively that policy gradient increases the rewards earned by generated questions at test time, and provide examples to illustrate the qualitative effects of different training schemes. To our knowledge, we present the first end-to-end, text-to-text model for question generation.

## 2 Related Work

Recently, automatic question generation has received increased attention from the research community. It has been harnessed, for example, as a means to build automatic tutoring systems (Heilman and Smith, 2010; Ali et al., 2010; Lindberg et al., 2013; Labutov et al., 2015; Mazidi and Nielsen, 2015), to reroute queries to community question-answering systems (Zhao et al., 2011), and to enrich training data for question-answering systems (Serban et al., 2016; Yang et al., 2017).

Several earlier works process documents as individual sentences using syntactic (Heilman and Smith, 2010; Ali et al., 2010; Kumar et al., 2015) or semantic-based parsing (Mannem et al., 2010; Lindberg et al., 2013), then reformulate questions using hand-crafted rules acting on parse trees. These traditional approaches generate questions with a high word overlap with the original text that pertain specifically to the given sentence by re-arranging the sentence parse tree. An alternative approach is to use generic question templates whose slots can be filled with entities from the document (Lindberg et al., 2013; Chali and Golestanirad, 2016). Labutov et al. (2015), for example, use ontology-derived templates to generate high-level questions related to larger portions of the document. These approaches comprise pipelines of independent components that are difficult to tune for final performance measures.

More recently, neural networks have enabled end-to-end training of question generation systems. Serban et al. (2016) train a neural system to convert knowledge base (KB) triples into natural-language questions. The head and the relation form a context for the question and the tail serves as the answer. Similarly, we assume that the answer is known *a priori*, but we extend the context to encompass a span of unstructured text. Mostafazadeh et al. (2016) use a neural architecture to generate questions from images rather than text. Contemporaneously with this work, Yang et al. (2017) developed generative domain-adaptive networks, which perform question generation as an auxiliary task in training a QA system. The main goal of their question generation is data augmentation, thus questions themselves are not evaluated. In contrast, our work focuses primarily on developing a neural model for question generation that could be applied to a variety of downstream tasks that includes question answering.

Our model shares similarities with recent end-to-end neural QA systems, e.g. Seo et al. (2016); Wang et al. (2016). I.e., we use an encoder-decoder structure, where the encoder processes answer and document (instead of question and document) and our decoder generates a question (instead of an answer). While existing question answering systems typically extract the answer from the document, our decoder is a fully generative model.

Finally, we relate the recent body of works that apply reinforcement learning to natural language generation, such as Li et al. (2016); Ranzato et al. (2016); Kandasamy and Bachrach (2017); Zhang and Lapata (2017). We similarly apply a REINFORCE-style (Williams, 1992) algorithm

to maximize various rewards earned by generated questions.

## 3 Encoder-Decoder Model for Question Generation

We adapt the simple encoder-decoder architecture first outlined by Cho et al. (2014) to the question generation problem. Particularly, we base our model on the attention mechanism of Bahdanau et al. (2015) and the *pointer-softmax* copying mechanism of Gulcehre et al. (2016). In question generation, we can condition our encoder on two different sources of information (compared to the single source in neural machine translation (NMT)): a document that the question should be about and an answer that should fit the generated question. Next, we describe how we adapt the encoder and decoder architectures in detail.

### 3.1 Encoder

Our encoder is a neural model acting on two input sequences: the document, $D = (d_1, \ldots, d_n)$ and the answer, $A = (a_1, \ldots, a_m)$. Sequence elements $d_i,\ a_j \in \mathbb{R}^{D_e}$ are given by embedding vectors (Bengio et al., 2001).

In the first stage of encoding, similar to current question answering systems, e.g. (Seo et al., 2016), we augment each document word embedding with a binary feature that indicates if the document word belongs to the answer. Then, we run a bidirectional long short-term memory (Hochreiter and Schmidhuber, 1997) (LSTM) network on the augmented document sequence, producing *annotation* vectors $\mathbf{h}^d = (\mathbf{h}^d_1, \ldots, \mathbf{h}^d_n)$. Here, $\mathbf{h}^d_i \in \mathbb{R}^{D_h}$ is the concatenation of the network's forward ($\overrightarrow{\mathbf{h}}^d_i$) and backward hidden states ($\overleftarrow{\mathbf{h}}^d_i$) for input token $i$, i.e., $\mathbf{h}^d_i = [\overrightarrow{\mathbf{h}}^d_i; \overleftarrow{\mathbf{h}}^d_i]$.[1]

Our model operates on QA datasets where the answer is extractive; thus, we encode the answer $A$ using the annotation vectors corresponding to the answer word positions in the document. We assume that, without loss of generality, $A$ consists of the sequence of words $(d_s, \ldots, d_e)$ in the document, s.t. $1 \leq s \leq e \leq n$. We concatenate the annotation sequence $(\mathbf{h}^d_s, \ldots, \mathbf{h}^d_e)$ with the corresponding answer word embeddings $(a_s, \ldots, a_e)$, i.e., $[\mathbf{h}^d_j; a_j], s \leq j \leq e$, then apply a second bidirectional LSTM (biLSTM) over the resulting sequence of vectors to obtain the extractive condition encoding $\mathbf{h}^a \in \mathbb{R}^{D_h}$. We form $\mathbf{h}^a$ by concatenating the final hidden states from each direction of the biLSTM.

We also compute an initial state $\mathbf{s}_0 \in \mathbb{R}^{D_s}$ for the decoder using the annotation vectors and the extractive condition encoding:

$$\mathbf{r} = \mathbf{L}\mathbf{h}^a + \frac{1}{n}\sum_i^{|D|} \mathbf{h}^d_i, \quad \mathbf{s}_0 = \tanh\left(\mathbf{W}_0\mathbf{r} + \mathbf{b}_0\right),$$

where $\mathbf{L} \in \mathbb{R}^{D_h \times D_h}$, $\mathbf{W}_0 \in \mathbb{R}^{D_s \times D_h}$, and $\mathbf{b}_0 \in \mathbb{R}^{D_s}$ are model parameters.[2]

### 3.2 Decoder

Our decoder is a neural model that generates outputs $y_t$ sequentially to yield the question sequence $Q = \{y_t\}$. At each time-step $t$, the decoder models a conditional distribution parametrized by $\theta$,

$$p_\theta(y_t | y_{<t}, D, A), \tag{1}$$

where $y_{<t}$ represents the outputs at earlier time-steps. In question generation, output $y_t$ is a word sampled according to (1).

When formulating questions based on documents, it is common to refer to phrases and entities that appear directly in the text. We therefore incorporate into our decoder a mechanism for copying relevant words from $D$. We use the pointer-softmax formulation (Gulcehre et al., 2016), which has two output layers: the *shortlist* softmax and the *location* softmax. The shortlist softmax places a distribution over words in a predefined output vocabulary. The location softmax is a pointer network (Vinyals et al., 2015) that places a distribution over document tokens to be copied. A source switching network enables the model to interpolate between these distributions.

In more detail, the decoder is a recurrent neural network. Its internal state, $\mathbf{s}_t \in \mathbb{R}^{D_s}$, evolves according to the long short-term memory update (Hochreiter and Schmidhuber, 1997), i.e.,

$$\mathbf{s}_t = \mathrm{LSTM}(\mathbf{s}_{t-1}, y_{t-1}, \mathbf{v}_t), \tag{2}$$

where $\mathbf{v}_t$ is a the context vector computed from the document and answer encodings.

At every time-step $t$, the model computes a soft-alignment score over the document to decide which words are more relevant to the question being generated. As in a traditional NMT architecture, the decoder computes a relevance weight $\alpha_{tj}$ for the $j$th

[1] We use the notation $[\cdot;\cdot]$ to denote concatenation of two vectors throughout the paper.

[2] Let $|X|$ denote the length of sequence $X$.

word in the document when generating the $t$th word in the question. Alignment score vector $\boldsymbol{\alpha}_t \in \mathbb{R}^{|D|}$ is computed with a single layer feedforward neural network $f(\cdot)$ using the $\tanh(\cdot)$ activation function. The scores $\boldsymbol{\alpha}_t$ are also used as the location softmax distribution. The network defined by $f(\cdot)$ computes energies according to (3) for the alignments, and the normalized alignments $\alpha_{tj}$ are computed as in (4):

$$e_{tj} = \exp(f(\mathbf{h}_j^d,\ \mathbf{h}^a,\ y_t,\ \mathbf{s}_{t-1})), \tag{3}$$

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{i=1}^{T} \exp(e_{ij})}. \tag{4}$$

To compute the context vector $\mathbf{v}_t$ used in (2), we first construct context vector $\mathbf{c}_t$ for the document and then concatenate it with $\mathbf{h}^a$:

$$\mathbf{c}_t = \sum_{i=1}^{|D|} \alpha_{ti} \mathbf{h}_i^d, \tag{5}$$

$$\mathbf{v}_t = [\mathbf{c}_t; \mathbf{h}^a]. \tag{6}$$

We use a deep output layer (Pascanu et al., 2013) at each time-step for the shortlist softmax vector $\mathbf{o}_t$. This layer fuses the information coming from $\mathbf{s}_t$, $\mathbf{v}_t$ and $y_{t-1}$ through a simple MLP to predict the word logits for the softmax as in (7). Parameters of the softmax layer are denoted as $\mathbf{W}_o \in \mathbb{R}^{|V| \times D_h}$ and $\mathbf{b}_o \in \mathbb{R}^{|V|}$, where $|V|$ is the size of the shortlist vocabulary (we used 2000 words).

$$\begin{aligned} \mathbf{e}_t &= g(\mathbf{s}_t, \mathbf{v}_t, y_{t-1}) \\ \mathbf{o}_t &= \text{softmax}(\mathbf{W}_o \mathbf{e}_t + \mathbf{b}_o) \end{aligned} \tag{7}$$

A source switching variable $z_t$ enables the model to interpolate between document copying and generation from shortlist. It is computed by an MLP with two hidden layers using $\tanh$ units (Gulcehre et al., 2016). Similarly to the computation of the shortlist softmax, the switching network takes $\mathbf{s}_t$, $\mathbf{v}_t$ and $y_{t-1}$ as inputs. Its output layer generates the scalar $z_t$ through the logistic sigmoid activation function.

Finally, $p_\theta(y_t | y_{<t}, D, A)$ is approximated by the full pointer-softmax $\mathbf{p}_t \in \mathbb{R}^{|V|+|D|}$ by concatenating $\mathbf{o}_t$ and $\boldsymbol{\alpha}_t$ after both are weighted by $z_t$:

$$\mathbf{p}_t = [z_t \mathbf{o}_t;\ (1 - z_t)\boldsymbol{\alpha}_t]. \tag{8}$$

As is standard in NMT, during decoding we use a beam search (Graves, 2012) to maximize (approximately) the conditional probability of an output sequence. We discuss this in more detail in the following section.

### 3.3 Training

The model is trained initially to minimize the negative log-likelihood of the training data under the model distribution,

$$\mathcal{L} = -\sum_t \log p_\theta(y_t | y_{<t}, D, A), \tag{9}$$

where, in the decoder as defined in (2), the previous token $y_{t-1}$ comes from the source sequence rather than the model output (this is called *teacher forcing*).

Based on our knowledge of the task, we introduce additional training signals to aid the model's learning. First, we encourage the model not to generate answer words in the question. We use the soft answer-suppression constraint given in (10) with the penalty hyperparameter $\lambda_s$; $\bar{\mathcal{A}}$ denotes the set of words that appear in the answer but not in the ground-truth question:

$$\mathcal{L}_s = \lambda_s \sum_t \sum_{\bar{a} \in \bar{\mathcal{A}}} p_\theta(y_t = \bar{a} | y_{<t}, D, A). \tag{10}$$

We also encourage variety in the output words to counteract the degeneracy often observed in NLG systems towards common outputs (Sordoni et al., 2015). This is achieved with a loss term that maximizes entropy in the output softmax (8), i.e.,

$$\mathcal{L}_e = \lambda_e \sum_t \mathbf{p}_t^T \log \mathbf{p}_t. \tag{11}$$

## 4 Policy Gradient Optimization

As described above, we use teacher forcing to train our model to generate text by maximizing ground-truth likelihood. Teacher forcing introduces critical differences between the training phase (in which the model is driven by ground-truth sequences) and the testing phase (in which the model is driven by its own outputs) (Bahdanau et al., 2016). Significantly, teacher forcing prevents the model from making and learning from mistakes during training. This is related to the observation that maximizing ground-truth likelihood does not teach the model how to distribute probability mass among examples other than the ground-truth, some of which may be valid questions and some of which may be completely incoherent. This is especially problematic in language, where there are often many ways to say the same thing. A reinforcement learning (RL) approach, by which a model is rewarded or penalized for its own actions, could mitigate these

issues – though likely at the expense of reduced stability during training. A properly designed reward, maximized via RL, could provide a model with more information about how to distribute probability mass among sequences that do not occur in the training set (Norouzi et al., 2016).

We investigate the use of RL to fine-tune our question generation model. Specifically, we perform policy gradient optimization following a period of "pretraining" on maximum likelihood, using a combination of scalar rewards correlated to question quality. We detail this process below. To make clear that the model is acting freely without teacher forcing, we indicate model-generated tokens with $\hat{y}_t$ and sequences with $\hat{Y}$.

### 4.1 Rewards

**Question answering (QA)** One obvious measure of a question's quality is whether it can be answered correctly given the context document $D$. We therefore feed model-generated questions into a pretrained question-answering system and use that system's accuracy as a reward. We use the recently proposed Multi-Perspective Context Matching (MPCM) (Wang et al., 2016) model as our reference QA system, *sans* character-level encoding. Broadly, that model takes in a generated question $\hat{Y}$ and a document $D$, processes them through bidirectional recurrent neural networks, applies an attention mechanism, and points to the start and end tokens of the answer in $D$. After training a MPCM model on the *SQuAD* dataset, the reward $R_{\text{QA}}(\hat{Y})$ is given by MPCM's answer accuracy on $\hat{Y}$ in terms of the F1 score, a token-based measure proposed by Rajpurkar et al. (2016) that accounts for partial word matches:

$$R_{\text{QA}}(\hat{Y}) = \text{F1}(\hat{A}, A), \tag{12}$$

where $\hat{A} = \text{MPCM}(\hat{Y})$ is the answer to the generated question by the MPCM model. Optimizing the QA reward could lead to 'friendly' questions that are either overly simplistic or that somehow cheat by exploiting quirks in the MPCM model. One obvious way to cheat would be to inject answer words into the question. We prevented this by masking these out in the location softmax, a hard version of the answer suppression loss (10).

**Fluency (PPL)** Another measure of quality is a question's fluency – i.e., is it stated in proper, grammatical English? As simultaneously proposed in Zhang and Lapata (2017), we use a language model to measure and reward the fluency of generated questions. In particular, we use the perplexity assigned to $\hat{Y}$ by an LSTM language model:

$$R_{\text{PPL}}(\hat{Y}) = -2^{-\frac{1}{T}\sum_{t=1}^{T}\log_2 p_{\text{LM}}(\hat{y}_t|\hat{y}_{<t})}, \tag{13}$$

where the negation is to reward the model for minimizing perplexity. The language model is trained through maximum likelihood estimation on over $80,000$ human-generated questions from *SQuAD* (the training set).

**Combination** For the total scalar reward earned by the word sequence $\hat{Y}$, we also test a weighted combination of the individual rewards:

$$R_{\text{PPL + QA}}(\hat{Y}) = \lambda_{\text{QA}} R_{\text{QA}}(\hat{Y}) + \lambda_{\text{PPL}} R_{\text{PPL}}(\hat{Y}),$$

where $\lambda_{\text{QA}}$ and $\lambda_{\text{PPL}}$ are hyperparameters. The individual reward functions use neural models to tune the neural question generator. This is reminiscent of recent work on GANs (Goodfellow et al., 2014) and actor-critic methods (Bahdanau et al., 2016). We treat the reward models as black boxes, rather than attempting to optimize them jointly or back-propagate error signals through them. We leave these directions for future work.

We also experimented with several other rewards, most notably the BLEU score (Papineni et al., 2002) between $\hat{Y}$ and the ground-truth question for the given document and answer, and a softer measure of similarity between output and ground-truth based on skip-thought vectors (Kiros et al., 2015). Empirically, we were unable to obtain consistent improvements on these rewards through training, though this may be an issue with hyperparameter settings.

### 4.2 REINFORCE

We use the REINFORCE algorithm (Williams, 1992) to maximize the model's expected reward. For each generated question $\hat{Y}$, we define the loss

$$\mathcal{L}_{\text{RL}} = -\mathbb{E}_{\hat{Y}\sim\pi(\hat{Y}|D,A)}[R(\hat{Y})], \tag{14}$$

where $\pi$ is the policy to be trained. The policy is a distribution over discrete actions, i.e. words $\hat{y}_t$ that make up the sequence $\hat{Y}$. It is the distribution induced at the output layer of the encoder-decoder model (8), initialized with the parameters determined through likelihood optimization.[3]

[3]The policy also depends on the switch values but we omit these for brevity.

REINFORCE approximates the expectation in (14) with independent samples from the policy distribution, yielding the policy gradient

$$\nabla \mathcal{L}_{\text{RL}} \approx \sum_{t=1} \nabla \log \pi(\hat{y}_t | \hat{y}_{<t}, D, A) \frac{R(\hat{Y}) - \mu_R}{\sigma_R}. \tag{15}$$

The optional $\mu_R$ and $\sigma_R$ are the running mean and standard deviation of the reward, which push $R(\hat{Y})$ toward zero mean and unit variance. This "whitening" of rewards is a simple version of PopArt (van Hasselt et al., 2016), and we found empirically that it stabilized learning.

It is straightforward to combine policy gradient with maximum likelihood, as both gradients can be computed by backpropagating through a properly reweighted sequence-level log-likelihood. The sequences for policy gradient are sampled from the model and weighted by a whitened reward, and the likelihood sequences are sampled from the training set and weighted by 1.

### 4.3 Training Scheme

Instead of sampling from the model's output distribution, we use beam-search to generate questions from the model and approximate the expectation in Eq. 14. Empirically we found that rewards could not be improved through training without this approach. Randomly sampling from the model's distribution may not be as effective for estimating the modes of the generation policy and it may introduce more variance into the policy gradient.

Beam search keeps a running set of candidates that expands and contracts adaptively. At each time-step $t$, $k$ output words that maximize the probabilities of their respective paths are selected and added to the candidate sequences, where $k$ is the beam size. The probabilities of these candidates are given by their accumulated log-likelihood up to $t$.[4]

Given a complete sample from the beam search and its accumulated log-likelihood, the gradient in (15) can be estimated as follows. After calculating the reward with a sequence generated by beam search, we use the sample to teacher-force the decoder so as to recreate exactly the model states from which the sequence was generated. The model can then be accurately updated by coupling the parameter-independent reward with the log-likelihood of the generated sequence. This approach adds a computational overhead but it significantly increases the initial reward values earned by the model and stabilizes policy gradient training.

We also further tune the likelihood during policy gradient optimization to prevent the model from overwriting its earlier training. We combine the policy gradient update to the model parameters, $\nabla \mathcal{L}_{\text{RL}}$, with an update from $\nabla \mathcal{L}$ based on teacher forcing on the ground-truth signal.

## 5 Experiments

### 5.1 Dataset

We conducted our experiments on the *SQuAD* dataset for machine comprehension (Rajpurkar et al., 2016), a large-scale, human-generated corpus of (document, question, answer) triples. Documents are paragraphs from 536 high-PageRank Wikipedia articles covering a variety of subjects. Questions are posed by crowdworkers in natural language and answers are spans of text in the related paragraph highlighted by the same crowdworkers. There are 107,785 question-answer pairs in total, including 87,599 training instances and 10,570 development instances.

### 5.2 Baseline Seq2Seq System

Our baseline system, denoted "Seq2Seq," is based on the encoder-decoder architecture with attention and pointer-softmax outlined in Bahdanau et al. (2015) and Gulcehre et al. (2016). This is essentially the model outlined in Section 3, with a few key differences: (i) since the baseline was originally designed for translation, its encoder and decoder vocabularies are separate; (ii) the baseline conditions question generation on the answer simply by setting $\mathbf{h}^a$ as the average of the document encodings corresponding to the answer positions in $D$; (iii) the baseline has no constraint on generating answer words in the question (Equation (10)); and (iv) the baseline does not include the entropy-based loss defined in (11).

### 5.3 Quantitative Evaluation

We use several automatic evaluation metrics to judge the quality of generated questions with respect to the ground-truth questions from the dataset. We are undertaking a large-scale human evaluation to determine how these metrics align with human judgments. The first metric is BLEU (Papineni

[4] We also experimented with a stochastic version of beam search by randomly sampling $k$ words from top-$2k$ predictions sorted by candidate sequence probability at each time step. No performance improvement was observed.

| | NLL | BLEU | F1 | QA | PPL |
|---|---|---|---|---|---|
| Seq2Seq | 45.8 | 4.9 | 31.2 | 45.6 | 153.2 |
| Our System | **35.3** | 10.2 | 39.5 | 65.3 | 175.7 |
| + PG ($R_{\text{PPL}}$) | 35.7 | 9.2 | 38.2 | 61.1 | **155.6** |
| + PG ($R_{\text{QA}}$) | 39.8 | **10.5** | **40.1** | **74.2** | 300.9 |
| + PG ($R_{\text{PPL+QA}}$) | 39.0 | 9.2 | 37.8 | 70.2 | 183.1 |
| Question LM | - | - | - | - | 87.7 |
| MPCM | - | - | - | 70.5 | - |

Table 2: Automatic metrics on *SQuAD*'s dev set. NLL is the negative log-likelihood. BLEU and F1 are computed with respect to the ground-truth questions. QA is the F1 obtained by the MPCM model answers to generated questions and PPL is the perplexity computed with the question language model (LM) (lower is better). PG denotes policy gradient training. The bottom two lines report performance on ground-truth questions.

| **Text Passage** |
|---|
| ...the court of justice accepted that a requirement to speak **gaelic** to teach in a dublin design college could be justified as part of the public policy of promoting the irish language. |
| **Generated Questions** |
| 1) what did the court of justice not claim to do? |
| 2) what language did the court of justice say should be justified as part of the public language? |
| 3) what language did the court of justice decide to speak? |
| 4) what language did the court of justice adopt a requirement to speak? |
| 5) what language did the court of justice say should be justified as part of? |

Table 3: Examples of generated questions given a context and an answer. Questions are generated by the five systems in Table 2, in order.

et al., 2002), a standard in machine translation, which computes {1,2,3,4}-gram matches between generated and ground-truth questions. Next we use F1, which focuses on unigram matches (Rajpurkar et al., 2016). We also report fluency and QA performance metrics used in our reward computation. Fluency is measured by the perplexity (PPL) of the generated question computed by the pretrained question language model. The PPL score is proportional to the marginal probability $p(\hat{Y})$ estimated from the corpus. The QA performance is measured by running the pretrained MPCM model on the generated questions and measuring F1 between the predicted answer and the conditioning answer.

### 5.4 Results and qualitative analysis

Our results for automatic evaluation on *SQuAD*'s development set are presented in Table 2. Implementation details for all models are given in the supplementary material. One striking feature is that BLEU scores are quite low for all systems tested, which relates to our earlier argument that a typical (document, answer) pair may be associated with multiple semantically-distinct questions. This seems to be born out by the result since most generated samples look reasonable despite low BLEU scores (see Tables 1, 3).

**Our system vs. Seq2Seq** Comparing our model to the Seq2Seq baseline, we see that all metrics improve notably with the exception of PPL. Interestingly, our system performs worse in terms of PPL despite achieving lower negative log-likelihood. This, along with the improvements in BLEU, F1 and QA, suggests that our system learns a more powerful conditional model at the expense of accurately modelling the marginal distribution over questions. It is likely challenging for the model to allocate probability mass to rarer keywords that are helpful to recover the desired answer while also minimizing perplexity. We illustrate with samples from both models, specifically the first two samples in Table 3. The Seq2Seq baseline generated a well-formed English question, which is also quite vague – it is only weakly conditioned on the answer. On the other hand, our system's generated question is more specific, but still not correct given the context and perhaps less fluent given the repetition of the word *language*. We found that our proposed entropy regularization helped to avoid over-fitting and worked nicely in tandem with dropout: the training loss for our regularized model was 26.6 compared to 22.0 for the Seq2Seq baseline that used only dropout regularization.

**Policy gradient ($R_{\text{PPL}}$: $\lambda_{\text{PPL}} = 0.1$)** Policy gradient training with the negative perplexity of the pretrained language model improves the generator's PPL score as desired, which approaches that of the baseline Seq2Seq model. However, QA, F1, and BLEU scores decrease. This aligns with the above observation that fluency and answerability (as measured by the automatic scores) may be in competition. As an example, the third sample in Table 3 is more fluent than the previous examples but does not refer to the desired answer.

| Training | Generated Questions | QA | PPL |
|---|---|---|---|
| $R_{\text{PPL}}$ | what was the name of the library that was listed on the grainger market? | 0 | 73.2 |
| $R_{\text{QA}}$ | the grainger market architecture was listed in 1954 by what? | 100 | 775 |
| $R_{\text{QA+PPL}}$ | what language did the grainger market architecture belong to? | 0 | 257 |
| $R_{\text{PPL}}$ | what are the main areas of southern california? | 0 | 114 |
| $R_{\text{QA}}$ | southern california is famous for what? | 16.6 | 269 |
| $R_{\text{QA+PPL}}$ | what is southern california known for? | 16.6 | 179 |
| $R_{\text{PPL}}$ | what was the goal of the imperial academy of medicine? | 19.1 | 44.3 |
| $R_{\text{QA}}$ | why were confucian scholars attracted to the medical profession? | 73.7 | 405 |
| $R_{\text{QA+PPL}}$ | what did the confucian scholars believe were attracted to the medical schools? | 90.9 | 135 |
| $R_{\text{PPL}}$ | what is an example of a theory that can be solved in theory? | 0 | 38 |
| $R_{\text{QA}}$ | in complexity theory, it is known as what? | 100 | 194 |
| $R_{\text{QA+PPL}}$ | what is an example of a theory that can cause polynomial-time solutions to be useful? | 100 | 37 |

Table 4: Comparison of questions from different reward combinations on the same text and answer.

**Policy gradient ($R_{\text{QA}}$: $\lambda_{\text{QA}} = 1.0$)** Policy gradient is very effective at maximizing the QA reward, gaining 8.9% in accuracy over the improved Seq2Seq model and improving most other metrics as well. The fact that QA score is 3.7% higher than that obtained on the ground-truth questions suggests that the question generator may have learned to exploit MPCM's answering mechanism, and the higher reported perplexity suggests questions under this scheme may be less fluent. We explore this in more detail below. The fourth sample in Table 3, in contrast to the others, is clearly answered by the context word *gaelic* as desired.

**Policy gradient ($R_{\text{PPL + QA}}$: $\lambda_{\text{PPL}} = 0.25, \lambda_{\text{QA}} = 0.5$)** We attempted to improve fluency and answerability in tandem by combining QA and PPL rewards. The PPL reward adds a prior towards questions that look natural. According to Table 2, this optimization scheme yields a good balance of performance, improving over the maximum-likelihood model by a large margin in terms of QA performance and gaining back some PPL. In the sample shown in Table 3, however, the question is specific to the answer but ends prematurely.

In Table 4 we provide additional generated samples from the different PG rewards. This table reveals one of the 'tricks' encouraged by the QA reward for improving MPCM performance: questions are often phrased with the interrogative 'wh' word at the end. This gives the language high perplexity, since such questions are rarer in the training data, but brings the question form closer to the form of the source text for answer matching.

### 5.5 Discussion

Looking through examples revealed certain difficulties in the task and some pathologies in the model that should be rectified through future work.

**Entities and Verbs** Similar entities and related verbs are often swapped, e.g., *miami* for *jacksonville* in a question about population. This issue could be mitigated by biasing the pointer softmax towards the document for certain word types.

**Abstraction** We desire a system that generates *interesting* questions, which are not limited to reordering words from the context but exhibit some abstraction. Rewards from existing QA systems do not seem beneficial for this purpose. Questions generated through NLL training show more abstraction at the expense of decreased specificity.

**Commonsense and Reasoning** Commonsense understanding appears critical for generating questions that are well-posed and show abstraction from the original text. Likewise, the ability to reason about and compose relations between entities could lead to more abstract and interesting questions. The existing model has no such capacities.

**Evaluation** Due to the large number of possible questions given a predefined answer, it is challenging to evaluate the outputs using standard overlap-based metrics such as BLEU. In this sense, question generation from text is similar to other tasks with large output spaces (Galley et al., 2015) and may benefit from corpora with multiple ground-truth questions associated to a quality rating (Mostafazadeh et al., 2016).

## 6 Conclusion and Future Work

We proposed a recurrent neural model that generates natural-language questions conditioned on text passages and predefined answers. We showed how to train this model using a combination of maximum likelihood and policy gradient optimization, and demonstrated both quantitatively and qualitatively how several reward combinations affect the generated outputs. We are now undertaking a human evaluation to determine the correlation between rewards and human judgments, improving our model, and testing on additional datasets.

One of our interests is to build models that seek information autonomously through question asking, as people do. This would entail, among other things, the direct sampling of interesting, informative questions from documents, i.e., modelling distribution $p(Q|D)$ rather than the distribution conditioned on the answer, $p(Q|D, A)$, as in this work. The present work may serve as a useful first step toward this goal, since the larger problem can be tackled by factorizing $p(Q|D) = \sum_A p(Q|D, A)p(A|D)$ and first sampling a document's likely answers according to modelled distribution $p(A|D)$.

## References


Husam Ali, Yllias Chali, and Sadid A Hasan. 2010. Automation of question generation from sentences. *Proc. of QG2010: The Third Workshop on Question Generation* .

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086* .

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)* .

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2001. A neural probabilistic language model. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *NIPS'2000*. MIT Press, pages 932–938.

Yllias Chali and Sina Golestanirad. 2016. Ranking automatically generated questions using common human queries. *Proc. of INLG* .

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. *arXiv preprint arXiv:1406.1078* .

François Chollet. 2015. keras. https://github.com/fchollet/keras.

Michel Galley, Chris Brockett, Alessandro Sordoni, Yangfeng Ji, Michael Auli, Chris Quirk, Margaret Mitchell, Jianfeng Gao, and Bill Dolan. 2015. deltableu: A discriminative metric for generation tasks with intrinsically diverse targets. *arXiv preprint arXiv:1506.06863* .

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. pages 2672–2680.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711* .

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148* .

Michael Heilman and Noah A Smith. 2010. Good question! statistical ranking for question generation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 609–617.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*. pages 1693–1701.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9.8:1735–1780.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547* .

Kirthevasan Kandasamy and Yoram Bachrach. 2017. Batch policy gradient methods for improving neural conversation models. *Proc. of ICLR* .

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*. pages 3294–3302.

Girish Kumar, Rafael E Banchs, and Luis Fernando D'Haro Enriquez. 2015. Revup: Automatic gap-fill question generation from educational texts. *Proc. of ACL* .

Igor Labutov, Sumit Basu, and Lucy Vanderwende. 2015. Deep questions without deep understanding. *Proc. of ACL* .

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *Proc. of EMNLP* .

David Lindberg, Fred Popowich, and John Nesbit Phil Winne. 2013. Generating natural language questions to support learning on-line. *Proc. of ENLG* .

Prashanth Mannem, Rashmi Prasad, and Aravind Joshi. 2010. Question generation from paragraphs at upenn: Qgstec system description. In *Proceedings of QG2010: The Third Workshop on Question Generation*. pages 84–91.

Karen Mazidi and Rodney D Nielsen. 2015. Leveraging multiple views of text for automatic question generation. In *International Conference on Artificial Intelligence in Education*. Springer, pages 257–266.

Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende. 2016. Generating natural questions about an image. *arXiv preprint arXiv:1603.06059* .

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* .

Mohammad Norouzi, Samy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*.

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*. Association for Computational Linguistics, pages 311–318.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026* .

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* .

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. *Proc. of ICLR* .

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603* .

Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. *Proc. of ACL* .

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284* .

Harry Singer and Dan Donlan. 1982. Active comprehension: Problem-solving schema with question generation for comprehension of complex short stories. *Reading Research Quarterly* pages 166–186.

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. *arXiv preprint arXiv:1506.06714* .

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688. http://arxiv.org/abs/1605.02688.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2016a. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830* .

Adam Trischler, Zheng Ye, Xingdi Yuan, Phil Bachman, Alessandro Sordoni, and Kaheer Suleman. 2016b. Natural language comprehension with the epireader. In *Empirical Methods on Natural Language Processing (EMNLP)*.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. 2016. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*. pages 4287–4295.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. pages 2692–2700.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211* .

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William W Cohen. 2017. Semi-supervised qa with generative domain-adaptive nets. *arXiv preprint arXiv:1702.02206* .

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. *arXiv preprint arXiv:1703.10931* .

Shiqi Zhao, Haifeng Wang, Chao Li, Ting Liu, and Yi Guan. 2011. Automatically generating questions from queries for community-based question answering. *Proc. of IJCNLP* .

# Supplementary Material

## A Implementation details

All models are implemented using Keras (Chollet, 2015) with Theano (Theano Development Team, 2016) backend. We used Adam (Kingma and Ba, 2014) with an initial learning rate 2e-4 for both maximum likelihood and policy gradient updates. Word embeddings were initialized with the GloVe vectors (Pennington et al., 2014) and updated during training. The hidden size for all RNNs is 768.

Dropout (Srivastava et al., 2014) is applied with a rate of 0.3 to the embedding layers as well as all the RNNs (between both input-hidden and hidden-hidden connections).

Both $\lambda_s$ for answer-suppression and $\lambda_e$ for entropy maximization are set to $0.01$. We used beam search with a beam size of 32 in all experiments. The reward weights used in policy gradient training are listed in Table 5. These parameters are selected using grid search based on validation QA reward.

| | $\text{QA}_{\text{MPCM}}$ | $\text{PPL}_{\text{Quest. LM}}$ |
|---|---|---|
| $\lambda_{\text{QA}}$ | 1.0 | - |
| $\lambda_{\text{PPL}}$ | - | 0.1 |
| $\lambda_{\text{PPL+QA}}$ | 0.5 | 0.25 |

Table 5: Hyperparameter settings for policy gradient training.