TRANSFORMER-BASED MULTI-HOP QUESTION GENERATION

JOHN EMERSON
Bachelor of Science, University of Lethbridge, 2018

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

TRANSFORMER-BASED MULTI-HOP QUESTION GENERATION

JOHN EMERSON

Date of Defence: August 22, 2022

Dr. Y. Chali Thesis Supervisor	Professor	Ph.D.
Dr. W. Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. J. Zhang Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. J. Sheriff Chair, Thesis Examination Committee	Assistant Professor	Ph.D.

Dedication

For the underdogs.

Abstract

Question generation is the parallel task of question answering, where given an input context and, optionally, an answer, the goal is to generate a relevant and fluent natural language question. Although recent works on question generation have experienced success by utilizing sequence-to-sequence models, there is a need for question generation models to handle increasingly complex input contexts to produce increasingly detailed questions. Multi-hop question generation is a more challenging task that aims to generate questions by connecting multiple facts from multiple input contexts. In this work, we apply a transformer model to the task of multi-hop question generation without utilizing any sentence-level supporting fact information. We utilize concepts that have proven effective in single-hop question generation, including a copy mechanism and placeholder tokens. We evaluate our model's performance on the HotpotQA dataset using automated evaluation metrics, including BLEU, ROUGE and METEOR and show an improvement over the previous work.

Acknowledgments

First, I'd like to thank my thesis supervisor, Dr. Yllias Chali, for his guidance, support, and all the knowledge I've gained from him throughout my master's program.

I'd also like to thank my supervisory committee members, Dr. Wendy Osborn and Dr. John Zhang, for their time and effort, the feedback they provided on my work, and the many courses I took with them during my time at the University of Lethbridge.

I'd also like to thank my office mate, Deen Mohammad Abdullah, who provided valuable suggestions for my research and was always willing to answer any questions I had.

I also have to thank my family for supporting me and always taking an interest in my work. I couldn't have done it without you.

Lastly, I'd like to thank Two Guys & A Pizza Place for giving me the strength to power through many late nights at the university over the past two years. If it wasn't for you, I'd have a few thousand less in student loans to repay. I regret nothing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In this chapter, we discuss the motivation for the work presented in this thesis, list the research contributions of this thesis, and describe the organization of its various components.

1.2 Motivation

Question generation is a research area within natural language processing that has seen considerable interest in recent years. The increased research interest has primarily been driven by improvements in the effectiveness of deep learning and the advent of sequence-to-sequence models. There are numerous variations of the question generation task, all of which share the same basic structure: given a selection of text as input, a question generation system should produce a relevant, thought-provoking question that is grammatically correct. Figure 1.1 demonstrates the process of generating a question that requires a single reasoning hop over an input context.

Many practical applications can benefit from the improvement of question generation

Input Context: Oxygen is used in cellular respiration and released by photosynthesis, which uses the energy of sunlight to produce oxygen from water.

Generated Question: What life process produces oxygen in the presence of light?

Figure 1.1: An example of single-hop question generation.

systems. Most notably, question generation systems can generate query suggestions for search engines (Zamani et al., 2020). If a user searches the internet for information about a specific topic, the search engine can create query suggestions based on the articles returned by the user's original search. Similarly, e-commerce websites can utilize the technology to generate questions from user-supplied product reviews automatically (Yu et al., 2020). The generated questions can then be used to filter out irrelevant reviews, only showing the user reviews that are relevant to their interests. Question generation can also be applied in educational settings for reading comprehension evaluation (Heilman and Smith, 2010). Such questions can serve as a self-evaluation tool or spur the reader to investigate a particular topic further. Question generation research can also contribute to the advancement of conversational systems, such as those utilized by digital personal assistants (Zhao et al., 2018).

In addition to the practical applications, question generation systems can also be utilized to advance research in other areas of natural language processing, such as the parallel task of question answering. Given a query, a question answering system determines the correct answer by examining a selection of input text. Since question answering systems often rely on deep learning models trained in a supervised fashion, they require large, high-quality datasets which map queries and input contexts to answers. Manually generating these datasets is time-consuming and expensive. Question generation systems can be utilized to automate this process (Du et al., 2017).

Although improvements have been made in recent years, early research on question generation has focused on input contexts of restricted sizes, usually no more than a sentence or small paragraph. Additionally, the depth of reasoning required to answer automatically generated questions has also been limited. The multi-hop question generation task addresses these deficiencies. Instead of generating a question that requires simple reasoning over the input context, multi-hop question generation attempts to automatically create questions that require the reader to reason over multiple pieces of information from vari-

Input Context 1: The Androscoggin Bank Colisée (formerly Central Maine Civic Center and Lewiston Colisee) is a 4,000 capacity (3,677 seated) multi-purpose arena, in Lewiston, Maine, that opened in 1958. In 1965 it was the location of the World Heavyweight Title fight during which one of the most famous sports photographs of the century was taken of Muhammed Ali standing over Sonny Liston.

Input Context 2: The Lewiston Maineiacs were a junior ice hockey team of the Quebec Major Junior Hockey League based in Lewiston, Maine. The team played its home games at the Androscoggin Bank Colisée. They were the second QMJHL team in the United States, and the only one to play a full season. They won the President's Cup in 2007.

Generated Question: The arena where the Lewiston Maineiacs played their home games can seat how many people?

Figure 1.2: An example of multi-hop question generation.

ous input contexts. Figure 1.2 demonstrates the process of generating a multi-hop question from multiple input contexts. To answer the generated question, two separate reasoning hops must be performed, which requires answering two smaller questions in turn. The first reasoning hop is "where did the Lewiston Maineiacs play their home games?" The answer, the Androscoggin Bank Colisée, can be found within Input Context 2. The second reasoning hop is "How many people can the Androscoggin Bank Colisée seat?" The answer, 3,677, can be found within Input Context 1 and serves as the final answer to the generated question.

The work presented in this thesis seeks to evaluate the effectiveness of using a transformer-based model to perform multi-hop question generation.

1.3 Contributions

The work presented in this thesis contributes to the research on multi-hop question generation in the following ways:

- We establish that non-pre-trained transformer-based models are an effective way to generate high-quality questions that require multiple reasoning hops to be answered.
- We show that enhancements such as placeholder tokens and a copy mechanism,

which have proven effective for single-hop question generation, can be successfully leveraged for the multi-hop question generation task.

- We demonstrate that randomly initialized embeddings can outperform GloVe embeddings for the multi-hop question generation task.
- We compare the performance of our model to the previous work on multi-hop question generation and demonstrate a significant improvement in multiple evaluation metrics and the amount of training time required.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides the background information required to understand the various technologies used in our proposed model. Chapter 3 discusses the previously completed work on question generation. Chapter 4 describes the problem of multi-hop question generation and introduces our proposed model. Chapter 5 discusses the dataset utilized for this project, the automated evaluation metrics employed, the model's implementation details, and the results of our experiments. Chapter 6 summarizes the work that has been completed in this thesis and provides directions for future research on the multi-hop question generation task.

Chapter 2

Background

2.1 Introduction

This chapter provides the background information necessary to understand the previous work on question generation and the approach presented in this thesis. We start by introducing artificial neural networks, including deep learning concepts such as recurrent neural networks, sequence-to-sequence learning and attention. We then discuss some essential ideas for natural language processing, including tokenization, word embeddings, and automated evaluation metrics. We then provide an overview of The Transformer, a critical part of our proposed model. Lastly, we discuss concepts related to deep neural networks' training and inference process.

2.2 Artificial Neural Networks

2.2.1 The Perceptron

The perceptron is the simplest example of an artificial neural network (ANN). It comprises an input layer consisting of n input nodes, followed by a single output node. Each node within the input layer receives a numeric value, commonly referred to as a feature, as input. Each feature x_i is then multiplied with its corresponding weight w_i. The products are summed to form a linear combination z. Equation 2.1 formalizes this calculation.

$$z = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

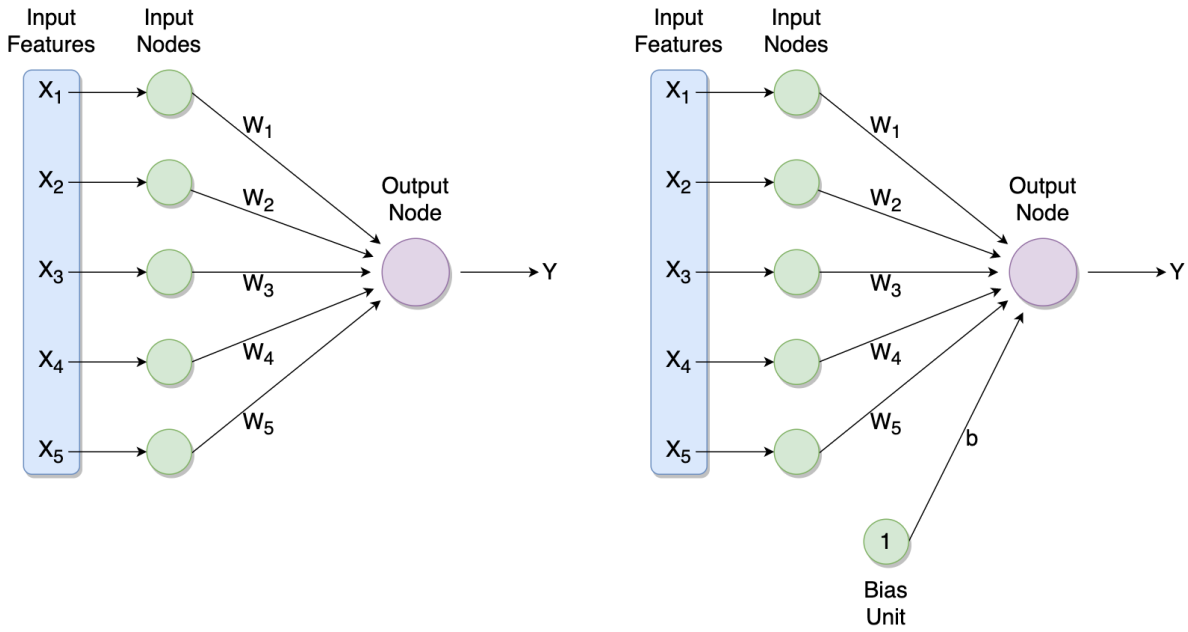


Figure 2.1: A perceptron with and without a bias unit.

$$y = \sigma(z) = \frac{1}{(1+e^{-z})} \tag{2.2}$$

The final prediction y produced by the perceptron's output node is determined by applying an activation function to the linear combination z. In practice, many different activation functions can be utilized. One of the most common activation functions is the sigmoid function (equation 2.2) which maps the linear combination to a value in the interval $(0,1)$. The weights inside the perceptron are randomly initialized and then gradually adjusted (learned) to reduce the incorrectness of the perceptron's predictions.

The perceptron commonly includes an additional node in the input layer known as a bias unit, which constantly emits the value 1. Similar to the other nodes in the input layer, the output of the bias unit is multiplied by a weight, and the result is included in the perceptron's linear combination. Including a bias unit in the perceptron increases the flexibility of the model by determining the extent to which each individual node is activated. Equation 2.3 formalizes the calculation of the linear combination z when the perceptron includes a bias unit.

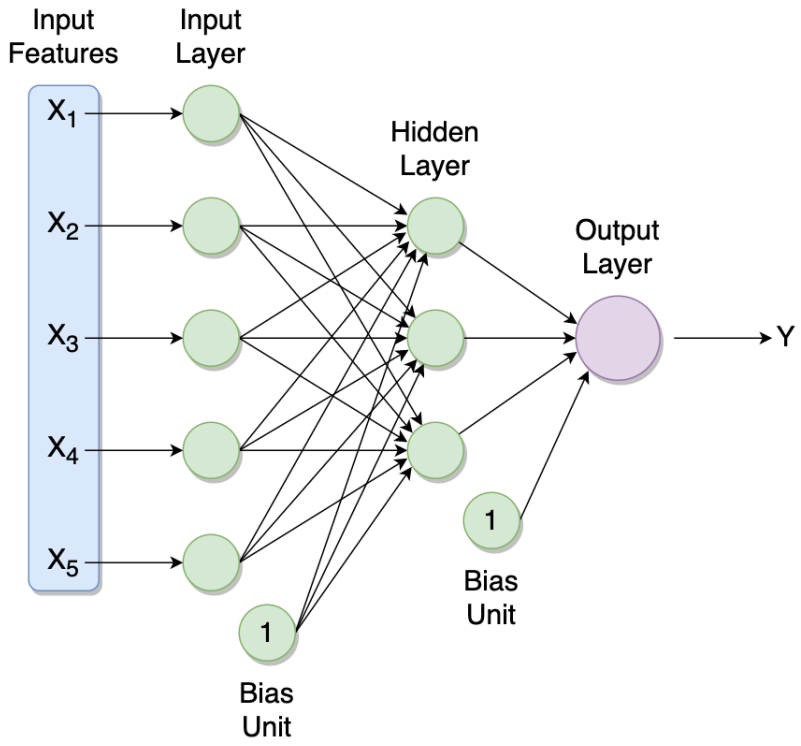


Figure 2.2: A feed-forward network with bias units.

$$z = (\sum_{i=0}^{n} w_i x_i) + b \tag{2.3}$$

2.2.2 Feed-Forward Neural Networks

Feed-forward networks are one of the most common types of neural networks. Their name is derived from the fact that the information within the network only flows forward, without any recurrence. Feed-forward networks consist of an input layer, zero or more hidden layers, and an output layer.

While the number of nodes within the input layer is equal to the number of input features present in the data being processed, the number of hidden layers, and the number of nodes within each of those layers, are largely hyperparameters that are dependent on the problem at hand. The number of nodes within the output layer depends on the network's desired output. If the network is applied to a binary classification problem, the output layer will likely consist of a single node that utilizes the sigmoid activation function. Since the output

of the sigmoid function, z, falls in the range $(0,1)$, $z < 0.5$ will indicate a prediction for class 1, while $z >= 0.5$ will indicate a prediction for class 2. If the network is applied to a multi-class classification problem consisting of n possible classes, the output layer will likely consist of n nodes. In such a case, the outputs, or logits, of each node in the output layer can be fed into the softmax function, which normalizes the logits so that they may be interpreted as probabilities for each possible class.

$$softmax(\vec{Z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.4}$$

Equation 2.4 demonstrates how the softmax calculation is performed. The function accepts a vector $\vec{Z}$ consisting of K logits as input. The standard exponential function is applied to each logit in $\vec{Z}$. Next, each result is divided by the sum of all exponentials. The resulting values sum to 1, allowing them to be interpreted as probabilities.

Similar to the perceptron, the hidden and output layers within a feed-forward network may also include a bias unit. Figure 2.2 illustrates the structure of a feed-forward neural network.

2.2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are another commonly used neural architecture. They were explicitly designed to process sequential data, making them an excellent choice for many problems within natural language processing, where the input to the model often consists of sequences of text of varying lengths. In contrast to feed-forward networks, the architecture of RNNs contains a loop which allows processed information to be fed back into the model. This allows the model to remember what has been processed in the sequence and utilize it while processing the current data, resulting in a rich encoding of the input sequence.

Figure 2.3 illustrates the unrolling process that takes place when an RNN processes an input sequence. At each time step t in a sequence of length n, the input x_t is fed into the

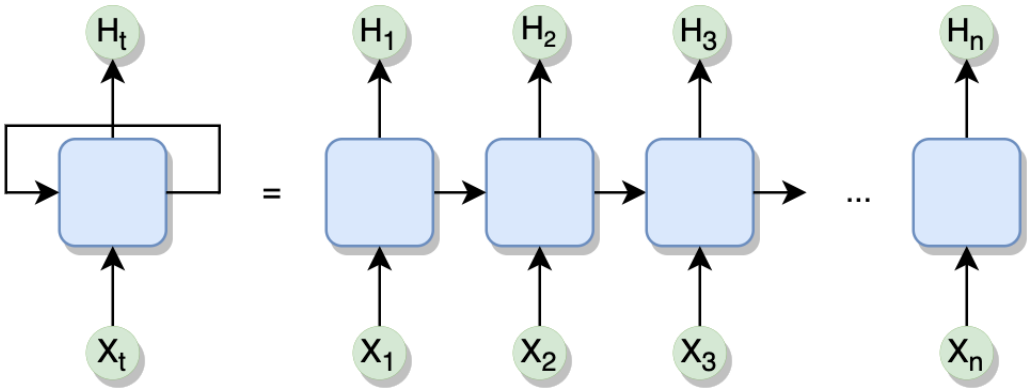


Figure 2.3: An unrolled recurrent neural network.

RNN along with the hidden state from the previous time step, h_{t-1}. The RNN utilizes these inputs to generate a new hidden state h_t. The output of an RNN at a given time step t is formalized in equation 2.5. Here, x_t represents the input to the RNN, h_{t-1} represents the hidden state from the previous time step, and f represents a non-linear function. The exact implementation of f is dependent on the type of RNN being used.

$$h_t = f(x_t, h_{t-1}) \tag{2.5}$$

Long Short-Term Memory

Long Short-Term Memory (LSTM) is a form of RNN that attempts to solve one of the most significant problems with RNNs: vanishing and exploding gradients (Hochreiter and Schmidhuber, 1997). The term 'gradient' refers to a derivative of a function with more than one input variable. Neural networks can be formalized as a function and their weights can be considered input variables. This allows gradients to be computed for each weight within a neural network, which are then utilized to adjust those weights. This process is referred to as training and it is discussed further in section 2.6.

Processing an extended sequence with an RNN means that the parameter matrices of the model will be repeatedly multiplied with each part of the input sequence. These repeated multiplications can make the gradients too small to learn long-term dependencies. It can also cause the gradients to become too large, making the training process unstable. LSTMs

incorporate a cell into their architecture which contains an input gate, an output gate, and a forget gate. These gates allow the cell to retain information from the previous cell state, allowing the model to handle long-range dependencies more effectively.

Gated Recurrent Unit (GRU)

Gated Recurrent Units are a more recent form of RNN that also aim to address the problem of vanishing/exploding gradients and model long-range dependencies (Cho et al., 2014). Instead of using a cell as a long-term memory, GRUs utilize a reset gate and an update gate that operate directly on the hidden state to control precisely how much information from the previous state should be retained or forgotten.

2.2.4 Sequence-to-Sequence Learning

Sutskever et al. (2014) combined two LSTMs into a single, unified model and trained it in an end-to-end fashion to perform machine translation between English and French. This model architecture proved so effective that it now forms the basis of a huge portion of research on natural language processing. This model architecture is often referred to as sequence-to-sequence learning since it accepts a variable-length sequence as input and produces an output sequence of a different length. The flexibility provided by this architecture means that it can be applied to many different problems, including question generation.

Sequence-to-sequence models consist of two major components, the first known as the encoder. The encoder is responsible for taking the input sequence and converting it into an encoded representation. The second part, the decoder, uses the encoded representation of the input sequence to generate an output sequence. In the case of question generation, a sequence-to-sequence model will typically accept a sequence of text, known as the context, as input and generate a relevant, high-quality question as output. The exact implementation of the encoder and decoder is dependent on the problem at hand. In the case of Sutskever et al. (2014), the encoder and decoder are simply multi-layer LSTMs, but for other tasks, the exemplary architecture can be much more intricate.

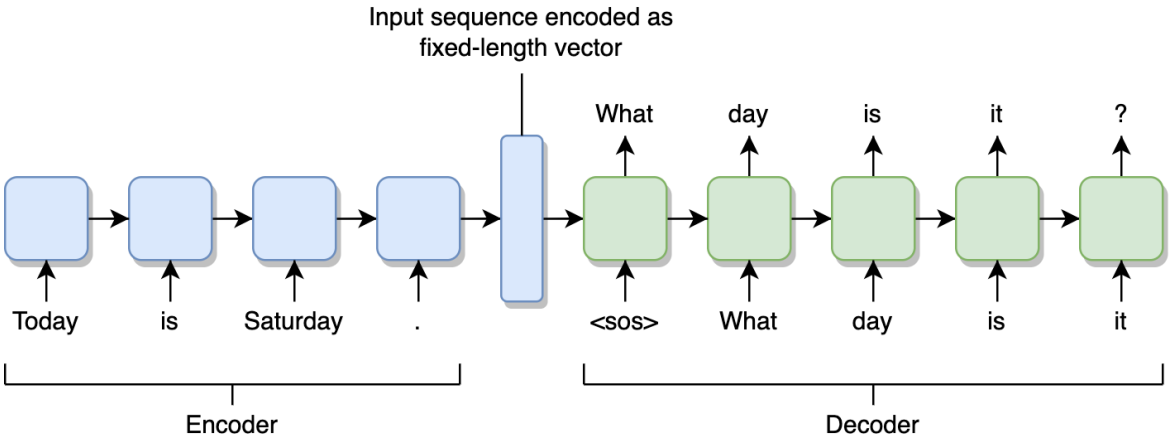


Figure 2.4: A recurrent sequence-to-sequence model.

Figure 2.4 illustrates how a trained RNN-based sequence-to-sequence model performs question generation. The input sequence is fed into the model and encoded into a fixed-length vector. This encoded representation of the input sequence is then fed into the decoder with a special token, the <sos> token. The <sos> (start-of-sequence) token signals the decoder to begin generating the output sequence. At each time step during the decoding process, the decoder greedily chooses the most likely token by conditioning on the input sequence and the previously generated tokens. The decoding process continues until a stopping condition is reached. In the case of question generation, the stopping conditions are usually the production of a question mark or the output sequence reaching a pre-determined maximum length. The overall goal of a sequence-to-sequence model is to generate the most probable output sequence y, given an input sequence x. Equation 2.6 formalizes this concept.

$$\arg\max_{y} p(y|x) \tag{2.6}$$

2.2.5 Attention

Although RNN-based sequence-to-sequence models are a powerful construct, they suffer from one critical deficiency: the fixed-length vector used to represent the encoded input sequence results in a bottleneck, limiting the amount of information stored. As the length

of the input sequence increases, the model's overall performance tends to decrease.

Bahdanau et al. (2014) introduced the concept of attention as a way of addressing this deficiency while researching neural machine translation. Their proposed approach extends the RNN-based sequence-to-sequence architecture, allowing the model to look back at the input sequence during the decoding process, determining which words from the input sequence are most relevant to the current time step while generating the output sequence. In addition to generating a fixed-length vector representing the entire input sequence, the hidden states from the encoding process are saved. At each time step during the decoding process, the model utilizes the saved encoder hidden states in conjunction with the current decoder hidden state to generate a context vector. The model conditions on this context vector to predict the next token in the output sequence.

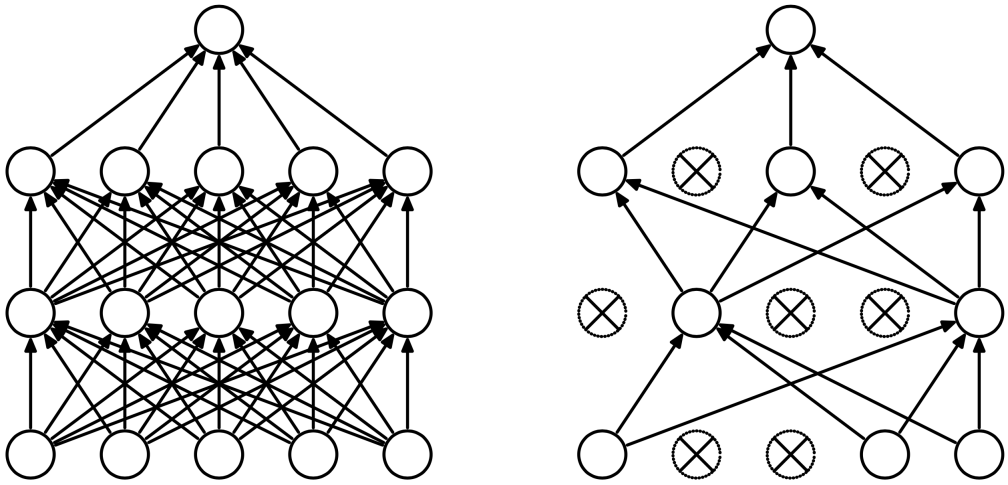
To form the context vector, the model must first calculate an alignment between the current decoder hidden state and each encoder hidden state. The alignment specifies how important each encoder state is to the current decoder state. The alignment, a_{ij}, is calculated as shown in equation 2.7, where h_j refers to the hidden state of the encoder at position j in the input sequence, s_i refers to the hidden state of the decoder at position i in the output sequence, and f is a feed-forward network.

$$a_{ij} = f(s_i, h_j) \tag{2.7}$$

Once the alignments between a decoder state and the encoder states have been calculated, the model computes the weights, w_{ij}, as follows:

$$w_{ij} = \frac{e^{a_{ij}}}{\sum_{k=1}^{T_x} e^{a_{ik}}} \tag{2.8}$$

Finally, the model computes the context vector as a sum of the encoder hidden states



(a) Standard Neural Net (b) After applying dropout.

Figure 2.5: Dropout applied to a feed-forward neural network (Srivastava et al., 2014).

weighted by the alignments, as shown below:

$$c_i = \sum_{j=1}^{T_x} w_{ij} h_j \tag{2.9}$$

2.2.6 Dropout

Ensemble methods are sometimes used to increase the performance of deep neural networks. These methods create and combine multiple models, resulting in improved performance compared to a single model (Aggarwal, 2018).

An ensemble method that is commonly used in deep neural networks is dropout. Dropout introduces a hyperparameter p, representing the probability that connections entering and exiting the input and hidden layers of a network will be randomly zeroed out. Figure 2.5 demonstrates this concept on a feed-forward neural network.

Randomly dropping out nodes during the training process has the effect of training a slightly different neural network configuration during each forward pass, which is why dropout can be considered an ensemble method (Aggarwal, 2018). This has the effect of reducing co-adaptation among the network's nodes. Co-adaptation refers to the situation where a node only produces helpful output when supplied with a specific input context

from several other nodes within the network (Hinton et al., 2012). By applying dropout to a network, each node is given a chance to learn to detect some feature, given a wide variety of internal contexts within the network. This is desirable since it reduces the phenomenon known as overfitting, where the model memorizes the training data but fails to generalize accurately when tested on unseen data. It should be noted that dropout is only applied during the training process and not during inference.

2.3 Natural Language Processing Concepts

2.3.1 Tokenization

Before a machine learning model can be applied to textual data, the data must be pre-processed into a cleaner, more predictable form. One of the most crucial steps in the pre-processing stage is tokenization, which refers to splitting a sequence of text into discrete elements capable of being represented numerically.

The simplest form of tokenization splits a sequence of text on its whitespace. For example, the sentence "The man rode his bicycle." would be tokenized as ['The', 'man', 'rode', 'his', 'bicycle.']. Notice how splitting the sentence on whitespace results in the period being included in the token representing the word bicycle, which is not ideal. Tokenizing on white space is also inadequate when multiple words should be regarded as a single token. For example, a city name such as San Francisco would be tokenized as ['San', 'Francisco'] when we want ['San Francisco'].

To alleviate these issues, rule-based tokenizers are provided by open-source libraries such as NLTK (Natural Language Toolkit)[1] and spaCy[2], and are often utilized for natural language processing and question generation research.

[1] https://www.nltk.org/
[2] https://spacy.io/

2.3.2 Part-of-Speech Tagging

Part-of-Speech (POS) tagging refers to annotating a sequence of text with labels that specify the grammatical role served by each token. The predicted tags for a sequence are represented numerically and fed into a machine learning model as an additional input feature. Part-of-speech taggers can be rule-based or utilize a machine learning model to generate tag predictions. Appendix B contains a list of the part-of-speech tags that belong to the spaCy library, which we use in our proposed model. Figure 2.6 demonstrates how a sentence is annotated with part-of-speech tags.

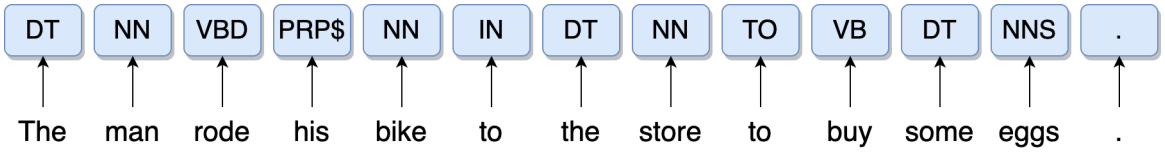


Figure 2.6: A sentence annotated with part-of-speech tags.

2.3.3 Named-Entity Recognition

Named-entity recognition (NER) refers to annotating a sequence of text with labels that indicate whether or not a token represents a named entity and, if so, the specific type of entity. Like part-of-speech tags, named-entity tags can also be an input feature to machine learning models. Figure 2.7 illustrates the result of tagging a sequence of text with a NER model belonging to the spaCy library.[3] Appendix C provides a complete list of the NER tags that belong to this library, which we use in our proposed model.

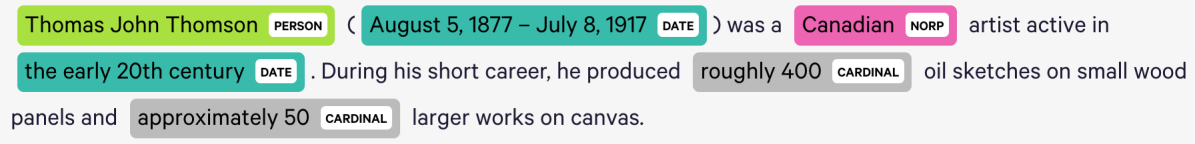


Figure 2.7: Sequence of text annotated with named entity tags.

[3] https://explosion.ai/demos/displacy-ent

2.3.4 Word Embeddings

Before a sequence of text can be fed into a neural model, the textual data must first be represented numerically. Early research on natural language processing utilized one-hot vectors for encoding tokens numerically. With one-hot encoding, each token in the system's vocabulary is represented by a vector consisting entirely of zeros except for a unique position in the vector which contains the value one. There are a couple of notable deficiencies with one-hot encoding. Firstly, the vectors are very sparse. As the size of the system's vocabulary grows, so must the dimension of the one-hot encodings. Secondly, the vectors do not contain helpful information about the tokens they represent.

Bengio et al. (2003) had the idea to represent each token in a system's vocabulary as a dense vector of floating-point values. These dense vectors, known as word embeddings, allow tokens to be represented by significantly fewer dimensions than one-hot encoding. Typically, dense word embeddings will consist of 200-300 dimensions (Mikolov et al., 2013; Pennington et al., 2014; Bojanowski et al., 2016), but they can also be larger or smaller. Word embeddings can also contain much more information about the tokens they represent. The values inside the word embeddings are randomly initialized and learned with the rest of a model's parameters. As a result of this process, tokens commonly seen near each other within the training data will be represented by embeddings that exhibit a high degree of similarity. In contrast, tokens rarely seen near each other will be represented by considerably different embeddings. The similarity between embeddings can be ascertained by computing the cosine similarity between them. Equation 2.10 formalizes how cosine similarity is calculated, where A and B are embeddings, $A \cdot B$ is the dot product between A and B, $||A||$ represents the magnitude of A, and $||B||$ represents the magnitude of B.

$$similarity(A,B) = \frac{A \cdot B}{||A||||B||} \tag{2.10}$$

As an example, table 2.1 contrasts one-hot encoding with dense word embeddings for a small vocabulary consisting of five tokens. It is safe to assume that the words 'coffee' and

'tea' will appear in similar contexts within a corpus. This will result in their embeddings displaying a high cosine similarity. The same is true for 'vehicle', 'truck' and 'motorcycle'. However, it is unlikely the words 'coffee' and 'motorcycle' will be seen close to each other very often, resulting in a much lower cosine similarity.

Table 2.1: One-hot encodings (left) contrasted with dense word embeddings (right).

$coffee$	$= [1,0,0,0,0]$	$coffee$	$= [-0.540, -0.299, -0.675, 0.157...]$
tea	$= [0,1,0,0,0]$	tea	$= [0.449, -0.002, -1.044, 0.426...]$
$vehicle$	$= [0,0,1,0,0]$	$vehicle$	$= [0.766, -0.464, 1.539, -0.072...]$
$truck$	$= [0,0,0,1,0]$	$truck$	$= [0.350, -0.361, 1.505, -0.070...]$
$motorcycle$	$= [0,0,0,0,1]$	$motorcycle$	$= [-0.404, -0.707, 0.034, 0.434...]$

Word embeddings are also highly effective at modelling relational concepts present in the training data from which they were derived. For example, the embeddings for different nouns will exhibit roughly equal offsets between their masculine and feminine variations. The same is true for other grammatical concepts, including verb tenses and singular/plural variations of nouns. Semantic relations such as the association between a country and its capital city also demonstrate this concept.

Pre-trained word embeddings are a form of transfer learning that have become very popular within question generation research in recent years (Du et al., 2017; Zhao et al., 2018; Su et al., 2020). Transfer learning refers to the process of training a model for a particular task and then reusing it as the starting point when training another model to perform a different task. With pre-trained word embeddings, rather than learning each embedding's representation from scratch, it is possible to use previously learned embeddings inside a new model. The pre-trained embeddings can be used as a starting point for the word embeddings within the new model, in which case their parameters are unfrozen, meaning that they continue to be trained with the rest of the model. This is commonly referred to as fine-tuning. The embeddings can also be frozen, in which case they remain unaltered while

the new model is trained. It is important to note that the effectiveness of using pre-trained embeddings in a given model depends on the problem at hand, the dataset from which the embeddings were derived, and the algorithm used to create the embeddings. As discussed below, pre-trained word embeddings can be classified as either context-free or context-dependent.

Context-Free Word Embeddings

Context-free word embeddings were the first type of pre-trained embedding to appear when the Word2Vec model was proposed (Mikolov et al., 2013). Word2Vec learns tokens' embeddings by training a small feed-forward network on a simple word prediction task. The resulting weights in the network are directly used to form the embedding for each token.

Pennington et al. (2014) proposed GloVe (Global Vectors for Word Representation), an alternative approach that generates word embeddings through a count-based process rather than deep learning. First, their approach builds a co-occurrence matrix by counting how often tokens co-occur in a corpus. Next, dimensionality reduction is applied to the co-occurrence matrix to produce the final embedding for each token.

In 2016, the fastText model was proposed, which was trained similarly to Word2Vec, but utilized subword tokenization to allow embeddings to be generated for tokens not seen during the training process (Bojanowski et al., 2016).

Pre-trained embeddings classified as context-free do not distinguish between the different meanings that a single token can have. Consider the following sentences:

1) The man went to the river bank.

2) The airplane began to bank.

3) The man withdrew some cash from the bank.

Even though the token 'bank' has a unique meaning in each sentence, context-free word embeddings only possess a single embedding for each token, which is shared by each pos-

sible meaning a token has.

Context-Dependant Word Embeddings

Pre-trained models that produce context-dependent word embeddings are not limited to a single embedding for each token in their vocabulary. Instead, they accept a sequence of tokens as input and create a contextual embedding for each token in the sequence by applying an encoding process. The embedding generated for each token depends on the other tokens present in the sequence and their relative positions. This allows the model to generate a unique embedding for each meaning that a single token can have.

BERT (Bi-Directional Encoder Representation from Transformers) (Devlin et al., 2018) generates contextual embeddings by using a transformer trained as a masked language model. ELMo (Embeddings from Language Model) utilizes a multi-layer bi-directional LSTM in a similar fashion (Peters et al., 2018).

2.4 Automated Evaluation Metrics

Natural language processing models can be evaluated using manual techniques or automated evaluation metrics. Manual evaluation techniques rely on humans to assess the performance of a model on some predetermined criteria. Automated evaluation metrics rank a model's performance by comparing its output to the reference output provided by the dataset. Manually evaluating the performance of a model is often seen as less desirable since it is time-consuming and tends to produce inconsistent results across evaluators (Clark et al., 2012). As a result, most of the research in natural language processing is evaluated using automated metrics. These metrics are deterministic and ensure that the results produced by different models can be compared consistently without requiring a significant turnaround time. A strong correlation between the automated metric and human evaluators should be identified for an automated metric to become widely used in natural language processing research. This allows researchers to conclude that an improvement in

the automated metric will also result in a real-world improvement (Clark et al., 2012).

2.4.1 BLEU

BLEU (Bilingual Evaluation Understudy) (Papineni et al., 2002) is a precision-based metric initially designed for evaluating machine translation systems that has been shown to correlate highly with human evaluations. BLEU generates a score by comparing a candidate sentence to one or more reference sentences. It calculates modified n-gram precisions by determining what percentage of n-grams within the candidate sentence are present in the reference sentence. A significant problem with precision-based metrics is their tendency to assign a high score to candidate sentences with extensive repetition. BLEU addresses this inadequacy by clipping the counts of the n-grams in the candidate sentence to the number of times they occur within the reference sentence. This technique naturally penalizes candidate sentences that are too long compared to the reference sentences. While some evaluation metrics utilize recall to penalize candidate sentences shorter than the reference, BLEU uses a brevity penalty.

By default, BLEU generates modified n-gram precision scores for unigrams, bigrams, trigrams and four-grams. The resulting scores can be combined into a single aggregate score by calculating a weighted mean, or they can be viewed individually. The resulting scores fall in $[0, 1]$, where 1 represents a candidate sentence that perfectly matches the reference. BLEU scores for n-grams of a lower cardinal value demonstrate the adequacy of the candidate sentence. In contrast, scores for n-grams of higher cardinal values represent the fluency of the candidate sentence.

2.4.2 ROUGE-L

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004) is an evaluation metric initially designed to evaluate text summarization and machine translation systems. There are multiple ROUGE variations, but the most commonly employed variation to evaluate question generation systems is ROUGE-L, which considers the order of tokens in

the candidate sentence when generating a score. It does this by first determining the longest common subsequence (LCS) between the candidate and reference sentences. Once the LCS has been found, the precision between the unigrams in the LCS and the candidate sentence is calculated. Next, the recall between the LCS and the reference sentence is calculated. The precision and recall calculations are then used to generate an F-score for the model's prediction.

2.4.3 METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) (Lavie and Agarwal, 2007) is another evaluation metric initially designed to evaluate machine translation systems. METEOR first calculates an alignment between the candidate and reference sentences' unigrams. It then calculates precision and recall between the aligned sentences and uses these two values to calculate an F-score, where recall is weighted more heavily than precision by default. In addition to performing this calculation using the surface forms of words, METEOR also allows this calculation to be performed using stemmed words and synonyms.

2.5 The Transformer Model

Noting the success that recurrent sequence-to-sequence models utilizing attention have experienced in natural language processing tasks, Vaswani et al. (2017) designed a new sequence transduction model known as The Transformer. The Transformer disposes of recurrence, relying almost entirely on the concept of attention, which allows it to be trained with a high degree of parallelization. Like other sequence-to-sequence models, The Transformer consists of an encoder and a decoder. The encoder comprises an embedding layer, a positional encoding, and a stack of encoder blocks. The decoder contains an embedding layer, a positional encoding, and a stack of decoder blocks followed by a linear layer and a softmax function. This section describes how The Transformer processes training data in

parallel, and then details each model component.

2.5.1 Input Feeding

While recurrence-based sequence transduction models process input sequentially, The Transformer processes the entire input sequence in parallel. Given an input sequence consisting of *n* tokens, The Transformer's encoder produces *n* hidden states corresponding to each token in the input sequence. The hidden state produced for each token is a function of all of the tokens in the sequence and their relative positions.

During the training process, The Transformer's decoder also operates in a parallel fashion. As seen in section 2.2.4, recurrence-based sequence-to-sequence models condition on the previously generated tokens of the output sequence when producing a new token to add to the output sequence. This approach is impossible when training a transformer since each token in the generated output sequence is predicted in parallel. Instead, a technique known as teacher-forcing is used when training a transformer-based model. This involves feeding a sample's reference output sequence into the decoder rather than previously predicted tokens. Before the reference output sequence is fed into the decoder, the <sos> token is prepended to the beginning of the sequence, which shifts each token in the reference sequence to the right by one position. This results in the correct token from the reference sequence being fed into the decoder at a given position rather than the decoder's prediction for the previous position.

During the inference process, the decoder of a trained transformer model must sequentially generate an output sentence since no reference output is available to the model.

2.5.2 Positional Encoding

Recurrent neural networks encode textual data sequentially, gradually building up an encoded representation of the sequence. This approach results in information about the relative positions of tokens within the sequence being implicitly incorporated into the encoded representation. Since The Transformer processes each sequence token in parallel,

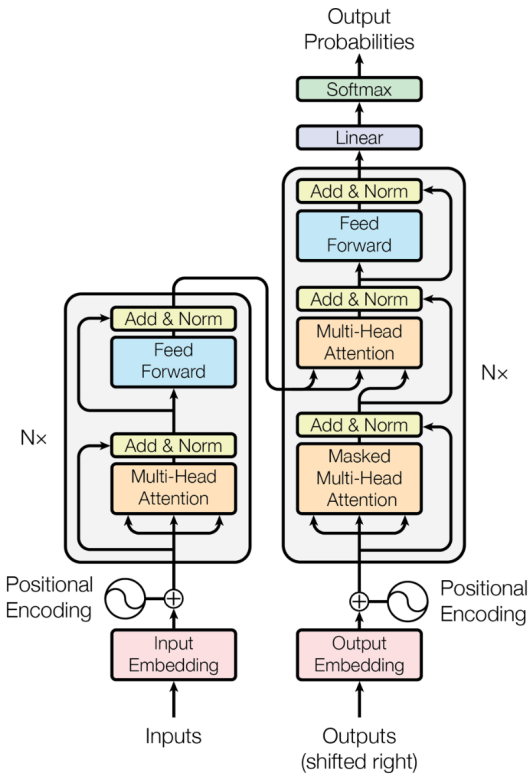


Figure 2.8: Transformer model architecture (Vaswani et al., 2017).

positional information cannot be automatically built into the encoded representation. A positional encoding must be manually applied to each token in the input and output sequences before The Transformer can process them.

Static Positional Encoding

Vaswani et al. (2017) utilize sine and cosine functions of different frequencies to generate static positional encodings for their model. Equations 2.11 and 2.12 formalize how these positional encodings are calculated. In the given equations, *pos* represents a position within a sequence, *i* refers to a particular dimension of the positional encoding, and d_{model} denotes the hidden dimension of the model. Note that the dimension of the positional encoding must match the dimension of the word embeddings so they may be added together.

Although frequency-based and learned positional encodings each perform well, Vaswani et al. (2017) hypothesized that frequency-based encodings allow the model to generalize

more effectively when processing sequences longer than those seen during the training process.

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.11}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.12}$$

Learned Positional Encoding

Recent transformer implementations such as BERT (Bi-direction Encoder Representations from Transformers) (Devlin et al., 2018) have had success with learned positional encodings. Rather than generating a fixed, frequency-based encoding and applying it to each token in a sequence, a vector corresponding to each position can be randomly initialized and learned with the rest of a model's parameters.

2.5.3 Encoder and Decoder Stacks

Encoder Stack

The Transformer's encoder is composed of a stack of identical encoder blocks. In figure 2.8, the number of encoder blocks within the encoder stack is denoted by Nx. Each encoder block is divided into two sublayers: the multi-head attention layer, detailed in section 2.5.4, and the position-wise feed-forward layer, which is described in section 2.5.5. There is a residual connection (He et al., 2015) around each of these sublayers, and layer normalization (Ba et al., 2016) is applied after each of them. To facilitate the application of the residual connections, the input and output dimensions of each of the sublayers must match the hidden dimension of the model.

Decoder Stack

The decoder is also composed of a stack of identical decoder blocks, where each block consists of three sublayers. In figure 2.8, the number of decoder blocks within the decoder stack is denoted by Nx. The first sublayer is a multi-head attention layer, whose implementation differs slightly from the implementation seen in the encoder blocks. Since The Transformer utilizes teacher-forcing during the training process to predict each token of the output sequence in parallel, a masking scheme is needed to ensure that each position within the output sequence does not attend to subsequent positions. Without such a masking scheme, the model would not learn to make predictions based on the previous tokens supplied by the reference output. The second sublayer is another multi-head attention layer. Unlike the first attention layer, which calculates self-attention on the output sequence, the second attention layer calculates attention between the output sequence and the encoded input sequence. The third sublayer is a position-wise feed-forward layer, identical to the one used in the encoder blocks. Each of the three sublayers in the decoder stack utilizes a residual connection (He et al., 2015), followed by layer normalization (Ba et al., 2016).

2.5.4 Multi-Head Attention

The multi-head attention layer is the most vital part of The Transformer. It allows the model to process a sequence by applying several attention mechanisms, known as heads, in a parallel fashion. The weights corresponding to each head are randomly initialized, and each head learns to attend to different features within a sequence.

Within the multi-head attention layer, each head utilizes scaled dot-product attention. Equation 2.13 demonstrates how scaled dot-product attention is calculated. The equation has three inputs: Q, which represents queries of dimension d_k, K, which represents keys of dimension d_k, and V, which represents values of dimension d_v. Q, K and V are matrices generated by applying linear transformations to the model's hidden states. To calculate attention, we first calculate the dot-product between the queries and the keys and then scale

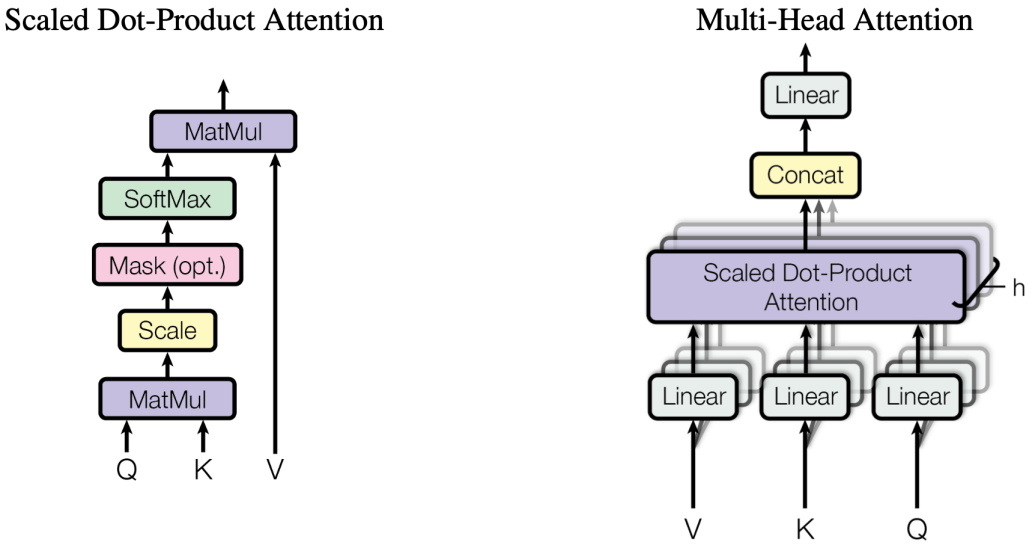


Figure 2.9: (Left) Scaled dot-product attention. (Right) Multi-head attention consists of several attention layers running in parallel (Vaswani et al., 2017).

the result by $\sqrt{d_k}$. We then take the softmax of this result and utilize it to weight the values. Essentially, the output of equation 2.13 is a weighted sum of the values, where the weight applied to each value is calculated by applying a compatibility function to the queries and the keys.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.13}$$

Each head within the multi-head attention layer has three linear layers that are applied to the model's hidden states to generate the matrices represented by Q, K and V. Equation 2.14 demonstrates this concept. Scaled dot-product attention is calculated in parallel for each head. The results, which are of dimension d_v, are concatenated together to form vectors of dimension d_{model} and are then passed through a final linear layer. Equation 2.15 demonstrates this concept.

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.14}$$

$$MultiHead(Q,K,V) = Concat(head_1,...,head_h)W^O \tag{2.15}$$

The multi-head attention layers within the encoder blocks function as self-attention layers. The query, key and value are all taken from the output of the previous encoder block. Each position within the input sequence is permitted to attend to every other position.

The first multi-head attention layer within the decoder blocks also functions as a self-attention layer. The inputs to this layer are taken from the outputs of the previous decoder block. In contrast to the encoder, the decoder's self-attention layer is only permitted to attend to earlier positions within the output sequence rather than every position within the sequence. As mentioned in section 2.5.3, masking is utilized to enforce this constraint when calculating dot product attention.

The second multi-head attention layer within the decoder blocks calculates attention between the encoded input and output sequences. The previous decoder block supplies queries while keys and values are taken from the output of the encoder.

2.5.5 Position-Wise Feed-Forward Layer

The position-wise feed-forward (PWFF) layer consists of a linear layer followed by a ReLU activation function and a second linear layer. The PWFF layer can be formalized by equation 2.16, where W represents a trainable weight matrix, and b denotes bias. Each encoder and decoder block has a separate PWFF layer with unique parameters. The PWFF layer for a given block is applied to each position within the sequence in parallel.

$$PWFF(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.16}$$

2.6 Training and Inference

2.6.1 Learning Paradigms

Machine learning algorithms can be divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning. The two main differences between these categories are how their training data is structured and how feedback is given to the system for learning.

In supervised learning approaches, the dataset that the model is trained on contains samples consisting of an input and an expected output. The goal of the model under supervised learning is to determine how to map the input data to the expected result. Problems within natural language processing that perform language generation usually fall under this category, including question generation.

With unsupervised learning approaches, the system is not provided with training data that dictates the expected output. Instead, the system is only supplied with input data, and the learning algorithm is expected to identify patterns present within the data. Unsupervised learning is commonly used in natural language processing to construct language models (Devlin et al., 2018) and pre-trained word embeddings (Mikolov et al., 2013).

Reinforcement learning approaches are often applied to problems consisting of dynamic situations where the number of possible states is too large to explicitly model and the effectiveness of the system's actions is easy to evaluate. Initially, the system does not know the actions it should perform in a specific situation, and it gradually learns which steps to take through a reward-based reinforcement process. Rewards are generated based on the effectiveness of the system's actions, and the learning process seeks to maximize the reward (Aggarwal, 2018).

2.6.2 Gradient-Based Optimization

Gradient-based optimizers are the backbone of deep learning models. Their goal is to reduce the incorrectness of the model's output as much as possible by gradually adjusting

its learnable parameters.

During the training process, there are two related phases: the forward pass and the backward pass. The forward pass, also called forward propagation, involves passing a sample from the training dataset through the model to predict the ideal output. During the backward pass, also known as backpropagation, the optimizer determines how much each trainable parameter in the model should be adjusted and performs the update accordingly.

Before the backward pass can be completed, the result produced by the forward pass must be compared to the reference output provided by the dataset to derive a numeric value that represents the incorrectness of the model's prediction. The function utilized to calculate this measure of incorrectness is known as the loss function. The choice of loss function for a given model depends on the problem being solved and the format of the model's output. For example, binary cross-entropy loss and multi-class cross-entropy loss can be considered the default loss functions for binary and multi-class classification problems, respectively (Brownlee, 2019).

The derivative of the loss function with respect to each trainable parameter in the model is calculated by applying the chain rule. This process starts at the output layer and continues toward the input layer, which is why it is known as the backward pass. Naive implementations of this process can be very costly due to repeated calculations. Dynamic programming ensures that each calculation is only performed once, which significantly improves efficiency (Aggarwal, 2018).

Once the gradient of the loss function has been calculated with respect to each of the model's trainable parameters, the parameters are updated simultaneously. Although the exact way that the parameters are updated is dependent on the optimization algorithm in use, the parameters are adjusted by stepping them in the opposite direction of the gradient (Aggarwal, 2018). The size of the adjustments made to the trainable parameters is determined by a hyperparameter α, known as the learning rate.

Batching

Batching is a concept that determines how much data is used to calculate the gradient of the loss function each time the model's trainable parameters are updated. Stochastic gradient descent (SGD) updates the trainable parameters in the model after a single sample is processed. Batch gradient descent updates the model's parameters after processing all of the samples in the training data. Mini-batch gradient descent is a compromise of the previous two approaches; it updates the trainable parameters in the model based on a mini-batch consisting of n samples (Ruder, 2016).

Researchers will often specify how many epochs were required to train their model. A single epoch refers to a complete pass through the training dataset. For example, if a dataset consists of 1024 samples with a mini-batch size of 16, then the trainable parameters in the model will be updated 64 times in a single epoch. Each of these updates can also be referred to as a training step.

Learning Rate Scheduling

The learning rate, α, does not have to remain constant throughout the training process. A more significant learning rate can be used near the beginning of the training process to allow the model to make large jumps toward a local minima. As training progresses, gradually lowering the learning rate results in the optimizer making increasingly smaller adjustments to the model's parameters. The combined effect is a training phase that is both quick and stable.

Gradient Clipping

As mentioned in section 2.2.3, recurrent neural networks are particularly susceptible to the exploding gradients problem. Gradient clipping is a technique that can be used to reduce this problem. It refers to modifying the derivatives calculated during backpropagation (Aggarwal, 2018).

Value-based gradient clipping defines maximum and minimum thresholds for the com-

puted gradients. Gradients that fall above or below these thresholds will be clipped to the specified maximum or minimum. Norm-based clipping scales the gradient vector so that the norm equals a specified value (Aggarwal, 2018).

2.6.3 Greedy Decoding

At inference time, trained models are used to generate an output sequence from a given input sequence. The simplest way to do this is through greedy decoding. Once the input sequence has been encoded, we initialize the output sequence with the <sos> token and feed it into the decoder. The model will select the most probable token from the vocabulary and append it to the generated sequence. The updated output sequence is then fed back into the decoder. Again, the most probable token is appended to the generated sequence. This process repeats until either a pre-determined maximum output sequence length has been reached or a token signalling the end of the output sequence has been generated.

2.6.4 Beam Decoding

Although greedy decoding is guaranteed to choose the token with the highest probability at each decoding step, the resulting sequence is not guaranteed to have the highest overall probability. To find the output sequence with the highest overall likelihood, we would have to perform an exhaustive search, which requires us to generate every possible sequence and choose the most likely one. The computational requirements for this approach are astronomical. Beam decoding improves upon greedy decoding while maintaining reasonable computational requirements. It generates a fixed number of candidate sequences and allows the most probable one to be selected as the final output sequence.

To utilize beam decoding, we start by choosing the beam width, k, which determines how many candidate output sequences will be tracked during the decoding process. At the first time step in the decoding process, we feed k candidate sequences initialized with the <sos> token into the decoder to begin the generation process. We then append one of the k most probable tokens to each of the k candidate sequences. The k candidate sequences

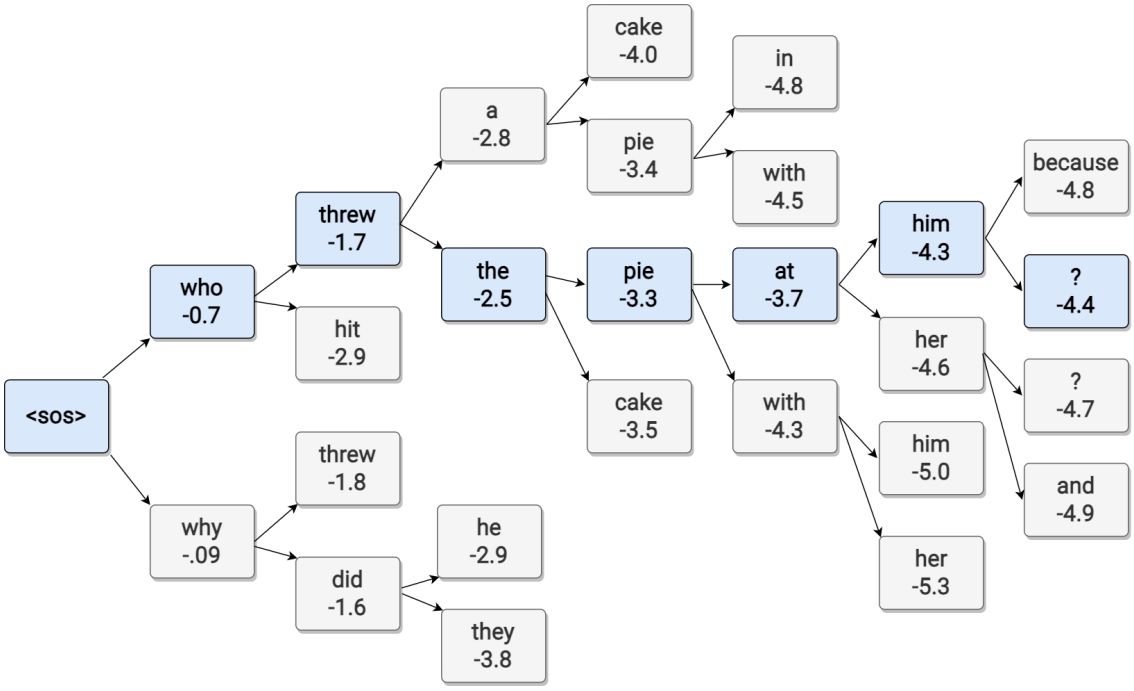


Figure 2.10: An example of beam decoding, where $k = 2$.

are fed back into the decoder at the subsequent step, and the top k tokens are selected for each candidate. Note that while at this point there are k^2 candidates in total, only the top k candidates are kept while the rest are discarded. This process repeats until k candidates have reached a stopping condition. Once k candidates have reached a stopping condition, the candidate with the highest overall probability will be chosen. Like greedy decoding, the stopping conditions include reaching a pre-determined maximum output sequence length or the production of a token that signals the end of the generated sequence.

The candidates that will be kept or pruned are computed by multiplying the log probabilities of a candidate's tokens and dividing the result by the total length of the sequence. Calculating the probabilities in log space ensures that arithmetic underflow does not occur. Dividing the result by the sequence length allows sequences of different sizes to be compared.

2.7 Summary

This chapter provided the background information necessary to understand the previous work on question generation and the approach presented in this thesis. We introduced artificial neural networks, including deep learning concepts such as recurrent neural networks, sequence-to-sequence learning and attention. We then discussed some essential ideas for natural language processing, including tokenization, word embeddings, and automated evaluation metrics. We then provided an overview of The Transformer, a critical part of our proposed model. Lastly, we discussed concepts related to deep neural networks' training and inference process. In the next chapter, we discuss the previous work on question generation.

Chapter 3

Related Work

3.1 Introduction

This chapter discusses the previous work completed on question generation. We start by briefly describing the techniques that early question generation works relied on. We then detail how neural models following the sequence-to-sequence architecture have been applied to solve different variations of the question generation task. Lastly, we cover the previous work on multi-hop question generation.

3.2 Early Approaches

Research into question generation began with The First Question Generation Shared Task Evaluation Challenge (Rus et al., 2010), which challenged researchers to automatically generate questions from single sentences and paragraphs. Early research on question generation largely utilized approaches that split the task into two distinct phases: content selection and question construction (Pan et al., 2019). The content selection phase determines which parts of an input context are worth asking about and which question type is the most suitable. The question construction phase is responsible for assembling a fluent and grammatically correct question directly answerable by the input context. To perform content selection, early works mainly relied on syntactic (Liu et al., 2010; Smith and Heilman, 2011) and semantic parsing (Yao et al., 2012; Lindberg et al., 2013; Mazidi and Nielsen, 2014; Chali and Hasan, 2015). The question construction phase typically applied handcrafted rules or templates to the input context to produce questions.

Rule-based approaches operate by transforming the input context from its declarative form into a question (Ali et al., 2010; Pal, 2010; Smith and Heilman, 2011). In contrast, template-based strategies generate questions by plugging the input context into hand-crafted question templates (Chen and Aist, 2009; Liu et al., 2010; Liu et al., 2012; Labutov et al., 2015).

3.3 Seq-to-Seq Approaches

3.3.1 Introduction

The hand-crafted rules and templates limited the variety and quality of questions that early question generation systems could produce. Neural question generation systems based on the sequence-to-sequence architecture overcome these limits by utilizing a data-driven approach, eliminating the requirement to spend time designing rules and templates. Instead, they leverage the information provided by large, high-quality datasets to jointly learn to select content from the input context and generate a fluent and relevant question. This approach results in greater flexibility in the type of questions produced.

3.3.2 Paragraph-Based Question Generation

Du et al. (2017) were the first to perform question generation using a sequence-to-sequence model when they attempted to generate questions for reading comprehension. They experimented with input contexts consisting of a single sentence and an entire paragraph. They successfully generated questions from single sentences, while paragraph-based question generation proved much more difficult due to the increased amount of information that must be disregarded.

More recent works on neural question generation have attempted to improve performance when working with paragraph-sized inputs and have experienced greater success. Du and Cardie (2018) experimented with generating paragraph-level question-answer pairs from Wikipedia articles. They utilized a coreference resolution system that operates on the

context paragraph and answer span to filter out irrelevant information. Zhao et al. (2018) noted the importance of paragraph-level question generation and pointed out that approximately twenty percent of the questions in the SQuAD dataset (Rajpurkar et al., 2016) require paragraph-level information to be asked. They utilize a gated self-attention encoder to focus on essential and related parts of the input context. They also apply a maxout pointer mechanism to allow out-of-vocabulary tokens to be copied from the input context to the generated question while limiting repetition. Kumar et al. (2020) experiment further with paragraph-level question generation by utilizing copy mechanisms and paragraph-specific dictionaries.

3.3.3 Answer-Aware vs. Answer-Agnostic Question Generation

Typically, the datasets used to train neural question generation systems explicitly label the answer span within the input context. This information can be leveraged to improve the performance of question generation models by helping them learn which parts of the input context are most closely related to the reference question. Zhou et al. (2017) were the first to explicitly use answer position information as an input feature to their model by applying the BIO tagging scheme. This approach has been adopted by many subsequent works on question generation (Zhao et al., 2018; Li et al., 2019; Kumar et al., 2020). Guo et al. (2018) take a slightly different approach and encode the input context and answer using separate RNNs. Kim et al. (2018) successfully applied this technique as well while researching ways to limit answer tokens from appearing in generated questions. Sun et al. (2018) posited that context tokens near that answer span are crucial to generating a high-quality question. They designed a model that focuses more on these tokens by utilizing learnable positional embeddings that reflect the distance between a given token and the labelled answer span.

Answer-agnostic question generation, also called answer-unaware question generation, is a variation where the training data does not explicitly label the answer span. This task is

more challenging than answer-aware question generation since the system must determine which part of the input context to focus on before generating a question. It is also viewed as more desirable since fewer constraints are placed upon the training data, and the question generation process is not overly influenced by being trained on specific answers. Du and Cardie (2017) perform answer-agnostic question generation on paragraphs by learning to select important sentences from the input context. Subramanian et al. (2018) created a model that learns to predict phrases from the input context that are capable of serving as the answer to a question. Their model then conditions on these phrases to generate questions. Scialom et al. (2019) generate questions in an answer-agnostic setting using transformer-based models, named-entity recognition and contextualized word embeddings. Nakanishi et al. (2019) and Wu et al. (2020) utilize question type prediction to guide the question generation process in the absence of an explicitly labelled answer span.

3.3.4 Question Type Prediction

Question type prediction has also been applied in the answer-aware setting since automatically generated questions often contain interrogative words that do not match the content of the question. Duan et al. (2017) created a system that consists of two sequence-to-sequence models. The first model generates a question template beginning with an interrogative word by conditioning on the input context. The second model generates a question by filling the template with tokens from the input context. Sun et al. (2018) explicitly model the probability of generating different interrogative words by conditioning on the answer span within the input context. Zhou et al. (2019) further experiment with question type prediction by creating a single, unified model that performs question type prediction and generation. Wang et al. (2020b) use question type prediction to generate multiple questions of varying types, rather than simply treating question generation as a one-to-one task.

3.3.5 Inquisitive Question Generation

Curiosity-driven question generation, also referred to as inquisitive question generation, refers to the task of automatically generating questions that are not directly answerable by the input context (Scialom and Staiano, 2020; Ko et al., 2020). The produced questions should be closely related to the input context and aim to improve the reader's understanding of the discussed topic. They are meant to provoke the reader to investigate the topic at hand further to gain a deeper understanding.

3.3.6 Sequential Question Generation

Sequential question generation attempts to produce a series of related and interconnected questions from the input context. The tendency of errors to cascade through recurrently generated sequential questions complicates this task. Questions generated sequentially also tend to become shorter with each generation. Chai and Wan (2020) experimented with generating sequential questions in a semi-autoregressive fashion. Their approach grouped target questions based on their type and generated questions for each group in parallel to limit error cascades and dependencies between questions. Nakanishi et al. (2019) tackle sequential question generation within the answer-agnostic setting by generating questions that are conversational and facilitate a back and forth between the information provided by the input context and the reader.

3.3.7 Other Types of Question Generation

Explicitly modelling facts and relations between entities within the input context has been applied to decrease the likelihood of producing irrelevant questions. Wang et al. (2020a) built a knowledge graph to represent facts about the relationships between entities from the input context and then used it to generate questions. Li et al. (2019) utilized off-the-shelf information extraction tools to extract relations from the input context relevant to the given answer to assist the question generation process and ensure it focuses on critical information.

Nema et al. (2019) implement a neural question generation model that uses two separate passes to generate questions. The refinement pass improves the first (rough) pass, which increases the likelihood that the generated question is syntactically correct and semantically close to the input context.

Yu et al. (2020) generate questions from online product reviews. The authors overcome the absence of a dataset linking questions and product reviews by utilizing reinforcement learning to combine relevant reviews with crowdsourced question-answer pairs. Zhang and Bansal (2019) also experiment with reinforcement learning to prevent the topic covered by a generated question from drifting too far away from the input context and desired answer.

Varanasi et al. (2020) perform question generation by leveraging the powerful self-attentions provided by a pre-trained BERT-based model and experience significantly faster training times than previous works.

3.4 Multi-hop Question Generation

Multi-hop question generation is a relatively new area of research, and little work has been completed on it thus far. Gupta et al. (2020a) assert that a common problem with sequence-to-sequence question generation systems is their limited capacity to focus on more than one supporting fact. They take inspiration from recent works that perform multi-hop question answering and create a multi-hop question generation system that utilizes reinforcement learning to address this problem. Their proposed model uses an answer-aware bi-directional LSTM to encode the input context. The encoded input context is then fed into a supporting fact prediction layer, which utilizes a feed-forward network to predict which sentences in the input context are supporting facts that can be used to generate a question that requires multi-hop reasoning. The output of the supporting fact prediction layer is concatenated with the encoded input context and fed into another bi-directional LSTM to obtain a supporting fact-aware encoding of the input context. The decoder consists of a pointer-generator similar to the design proposed by See et al. (2017). During training, the

model's reward is generated by conditioning on the generated question and the input context to predict which sentences in the input context are supporting facts. These predictions are compared to the ground-truth supporting facts to compute an F1 score, which is regularized by a ROUGE-L score between the generated question and the ground truth question. The downfall of the approach introduced by Gupta et al. (2020a) is that it requires supporting facts to be explicitly labelled within the training data.

Su et al. (2020) create a sequence-to-sequence multi-hop question generation system that does not require supporting facts to be explicitly labelled within the training data. Their model's encoder uses two bidirectional LSTMs to encode the input context and the answer separately. A third bidirectional LSTM combines the encoded representations into an answer-aware context encoding. They build a graph representing named entities in the input context, where edges between entities indicate that they are seen in the same paragraph, masking out entities that are not closely related to the answer. They use a graph convolutional network to propagate multi-hop information on the answer-aware subgraph and apply a bi-attention mechanism Seo et al. (2016) to produce an updated multi-hop answer encoding. The multi-hop answer encoding is combined with the original answer-aware context encoding via bidirectional LSTM, resulting in an updated answer-aware context encoding. Lastly, a reasoning gate is applied to the original and updated answer-aware context encodings. The decoder consists of a uni-directional LSTM and a maxout pointer generator. Although the work published by Su et al. (2020) improves upon the previous research by not requiring supporting sentences to be explicitly labelled within the training data, it still relies on an LSTM-based model, which processes input contexts sequentially. This is less than ideal since the input contexts for the multi-hop question generation task tend to be longer than those seen in traditional question generation research, which leads to increased training times.

Pan et al. (2021) attempted to use multi-hop question generation to build a dataset suitable for training a multi-hop question answering system in a setting where no manually

created training data is available. Their system consists of operators, reasoning graphs, and question filtration. Operators are operations that retrieve, generate and fuse information from the input context. They are implemented with rules and pre-trained models. Reasoning graphs accept operators as input, combining them in the correct order to generate question/answer pairs. Question filtration uses a pretrained GPT-2 (Radford et al., 2019) model to remove unnatural and irrelevant questions. The downfall of the approach introduced by Pan et al. (2021) is that it relies on hand-crafted operators and reasoning graphs, which limit the diversity of the questions it can generate. In contrast, question generation models based entirely on neural models are not limited to generating certain types of questions.

3.5 Summary

This chapter briefly discussed the techniques that early question generation systems relied on. We then detailed how neural models following the sequence-to-sequence architecture have been applied to solve different variations of the question generation task. Lastly, we covered the previous work on multi-hop question generation. The next chapter provides an in-depth description of our proposed multi-hop question generation system.

Chapter 4

Multi-Hop Question Generation

4.1 Introduction

We begin this chapter by describing the motivation behind the multi-hop question generation task. Next, we formally define our proposed model's task and training objective. Lastly, we introduce our proposed model, which is based on the Transformer architecture, and detail its various components.

4.2 Motivation

There is a need for research that improves the ability of question generation systems to handle increasingly complex input contexts. Previous research on question generation has mainly focused on generating questions from a single sentence or paragraph. The complexity of the questions generated by these systems and the depth of reasoning required to answer them are limited. Multi-hop question generation is a relatively new area of research that attempts to create more complex questions from larger input contexts consisting of multiple related paragraphs. They require the reader to understand the content in each context paragraph and then make multiple reasoning hops over that information to answer the question.

Two main challenges arise from the added complexity of multi-hop question generation. First, the proposed model must find a way to identify connected pieces of information scattered throughout the context paragraphs. Second, the model must determine how to combine the collected information to generate a syntactically correct question that the input

context can directly answer. The larger input contexts that are utilized in multi-hop question generation are what complicate these two challenges. On the one hand, much more information is available to the model that can be used to generate a specific, high-quality question. On the other hand, there is an increase in the amount of irrelevant information that must be disregarded so that the generated question does not stray too far from the desired topic.

A transformer forms the base of our proposed multi-hop question generation model. We enhance the model by applying effective mechanisms from research on single-hop question generation. The rest of this chapter details our proposed multi-hop question generation model.

4.3 Task Definition

The multi-hop question generation task can be formalized as the conditional probability shown in equation 4.1. Given an input context $C = (c_1, c_2, c_3, ..., c_n)$, we wish to generate the question $Q = (q_1, q_2, q_3, ..., q_m)$ that has the highest probability. c_i and q_i represent the tokens at position i within the input context and generated question. n and m represent the number of tokens within the input context and generated question. The input context C is created by concatenating the tokens of two related context paragraphs. The generated question should be directly answerable by performing multiple reasoning hops over the information provided by the input context C.

$$p(Q|C) = \prod_{1}^{m} p(q_i|q_{[1:i-1]}, C) \tag{4.1}$$

$$Loss = \sum_{i=1}^{N} -logp(Q_i|C_i;\theta) \tag{4.2}$$

All the trainable parameters in the proposed model are learned in unison in an end-to-end fashion. This is accomplished by minimizing the negative log-likelihood loss of

the model over the training dataset. Equation 4.2 formalizes this concept. The training dataset consists of N samples containing a reference question Q_i and an input context C_i. θ represents all of the model's parameters, learnable and fixed.

4.4 Proposed Model

4.4.1 Model Overview

All of the previous works on multi-hop question generation have relied on sequence-to-sequence models where both the encoder and decoder are comprised of LSTMs (Gupta et al., 2020a; Su et al., 2020). Although these models have proven very effective, models based on recurrence offer little opportunity for parallelization during the training process. Vaswani et al. (2017) proposed The Transformer as an efficient alternative to recurrence-based sequence transduction models. The Transformer relies heavily on the concept of self-attention and positional encoding to process sequential data in a parallel fashion. Our model utilizes the transformer architecture as the basis of both the encoder and the decoder. We modify the decoder with a pointer-generator, allowing it to copy out-of-vocabulary tokens from the input context and select tokens from a fixed vocabulary. Our model also employs a pre-processing technique that utilizes placeholder tokens to allow out-of-vocabulary named-entity tokens to be copied from the input context to the generated question. We now discuss each of the model's components in detail.

4.4.2 Placeholder Tokens

Scialom et al. (2019) studied the effectiveness of transformer-based models for the single-hop question generation task. They found that the transformer model outlined by Vaswani et al. (2017) was ineffective at producing satisfactory results. For their research, they utilized SQuAD (Rajpurkar et al., 2016), which is one of the most popular datasets for single-hop question generation. Within this dataset, approximately 52% of samples contain answers with named entities. Scialom et al. (2019) attributed the poor performance of the

base transformer model to this fact. Since neural question generation models typically utilize vocabularies of a fixed size, many tokens within the input context will fall outside of this vocabulary. In such a case, it is common practice to represent each out-of-vocabulary token with a single, common token that signifies that the original token is unknown to the model. For example, consider the sentence, "Taylor Swift was born on December 13, 1989." Suppose this sentence is given to a question generation model as an input context, and the tokens 'Taylor' and 'Swift' are not included in the model's fixed vocabulary. In that case, the resulting input to the model will be, "[unk] [unk] was born on December 13, 1989," where [unk] represents an unknown token. Given the original sentence as input, the model should generate a question like, "When was Taylor Swift born?" The replacement of the tokens 'Taylor' and 'Swift' with the [unk] token makes this impossible.

Scialom et al. (2019) dealt with this issue by performing named-entity recognition on each sample's input context and reference question. For each sample in their training data, they replaced each named-entity token with a unique placeholder token corresponding to the detected type of named entity and its order of appearance. These unique placeholders were included in their model's fixed vocabulary. Using this pre-processing technique, our example sentence would be pre-processed into, "[person_1] [person_2] was born on December 13, 1989." This allows the model to generate a question such as, "When was [person_1] [person_2] born?". As a post-processing step, any placeholder tokens within the generated question are replaced with the original token they represent, resulting in the question, "When was Taylor Swift born?" This mechanism alone allowed Scialom et al. (2019) to increase the performance of their model to a level on par with the previous works based on recurrent models.

We utilized the spaCy library (Honnibal and Montani, 2017)[4] to perform named-entity recognition on the HotpotQA dataset (Yang et al., 2018) to determine the prevalence of named entities. We found that $\sim$75% of answers contained at least one named-entity

[4]https://spacy.io/

token while ∼60% of answers contained only named-entity tokens. Noting the success that Scialom et al. (2019) achieved using this technique in a single-hop setting where only ∼52% of answers contained named entities, we utilize the same technique in our multi-hop question generation model. Details of the HotpotQA dataset are provided in section 5.2.

4.4.3 Encoder

Figure 4.1 illustrates the design of our encoder, which is based on the standard transformer encoder detailed in section 2.5.

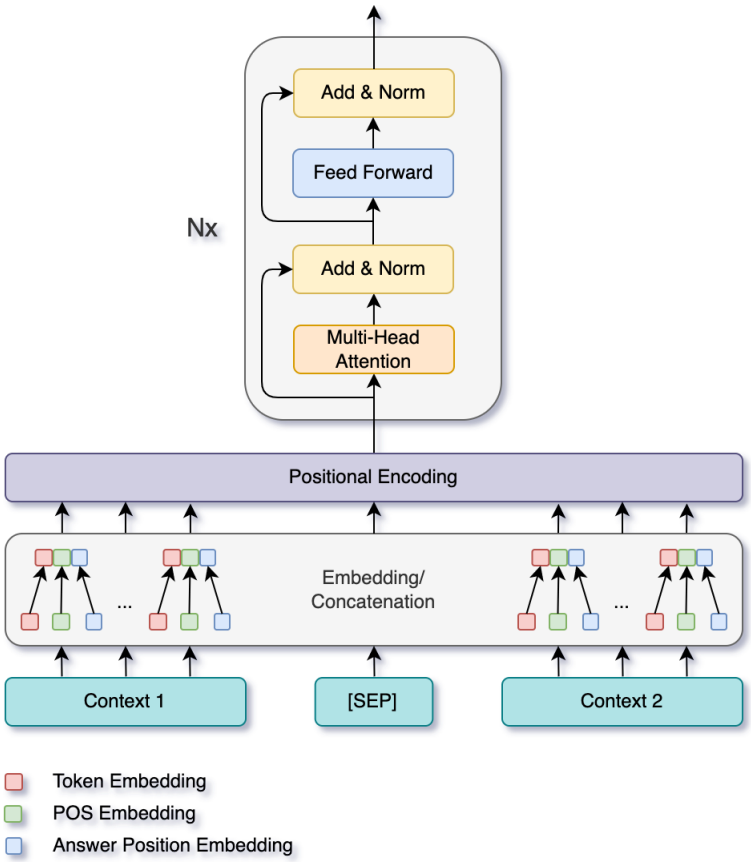


Figure 4.1: The encoder of our proposed model.

In section 4.3, we stated that the input context is created by concatenating the tokens of two separate, related context paragraphs. When joining these paragraphs, we separate them with a [sep] token. The motivation behind explicitly marking this boundary is to aid the en-

coder in building a connection between relevant entities across the two context paragraphs, resulting in a question that requires multiple reasoning hops to be answered.

For each token in the input context, the vector fed into the encoder is created by concatenating three separate embeddings. These embeddings correspond to the token being encoded, the part-of-speech tag for the token, and an answer position tag. A positional encoding is added onto the vector once the embeddings have been concatenated. The positional encoding used in the encoder is the frequency-based positional encoding designed by Vaswani et al. (2017), which we detailed in section 2.5.

We utilize the spaCy Python package (Honnibal and Montani, 2017)[5] to tokenize the text in our dataset and perform named entity recognition on each sample's context paragraphs and reference question. As discussed in section 4.4.2, each token identified as a named entity is replaced with a placeholder token. For each sample in the dataset, we create a dictionary that maps each placeholder token back to its original token. We utilize separate embedding layers for placeholder and non-placeholder (standard) tokens. This design choice allows us to individually control whether the embeddings corresponding to each token type are frozen or learned during the training process. It also allows us to experiment with using pre-trained word embeddings for the standard tokens while the embeddings for the placeholder tokens are learned from the training data.

We further utilize the spaCy Python package (Honnibal and Montani, 2017) to generate part-of-speech tags for each token in the input context. These tags are embedded with vectors that are learned from the training data.

Answer position tags are generated for each token in the input context using the BIO tagging scheme (Zhou et al., 2017). The BIO tagging scheme labels each token with one of three tags: [B], [I], and [O]. When a token within the input context is tagged with the [B] (beginning) tag, this token represents the beginning of the answer sequence. The [i] (inside) tag indicates that a token is part of the answer sequence but is not the first token of

[5]https://spacy.io/

the answer sequence. The [O] (outside) tag means that the token is not a part of the answer sequence. The answer sequence appears in multiple locations throughout the input context in some samples. In such cases, each answer sequence is tagged using the BIO scheme. The answer position tags are embedded with vectors learned from the training data.

4.4.4 Decoder and Pointer-Generator

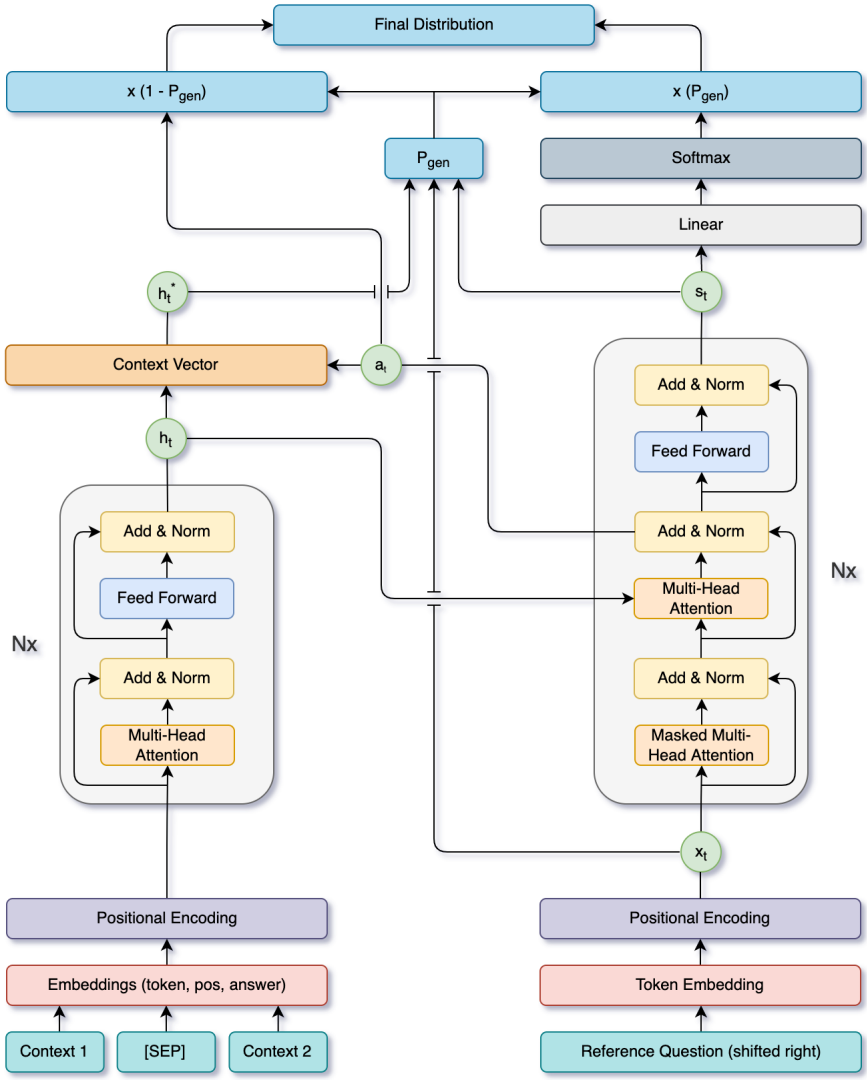


Figure 4.2: Overview of the proposed model.

The decoder is based on the standard transformer decoder detailed in section 2.5. Similar to the encoder, separate embedding layers are used for placeholder and standard tokens.

The positional encoding utilized by the decoder is the frequency-based positional encoding introduced by Vaswani et al. (2017), which we detailed in section 2.5. Unlike the encoder, no additional input features are utilized, and the positional encoding is added to the decoder's input.

We modify the decoder with a pointer-generator, allowing the model to choose between selecting a token from its fixed vocabulary or copying an out-of-vocabulary token from the input context. As discussed in section 4.4.2, neural question generation models are usually limited to a vocabulary consisting of a fixed number of tokens, which results in many of the tokens within the input context being represented by the [unk] token. For this reason, copy mechanisms are ubiquitous in sequence-to-sequence question generation models. Furthermore, Scialom et al. (2019) demonstrated that copy mechanisms provide a slight benefit even when using placeholder tokens to represent named entities since no named-entity recognition model has perfect accuracy. Suppose the named-entity recognition model fails to recognize a named-entity token within the input context. In that case, that token will be represented by the [unk] token if it is not a member of the model's vocabulary.

The pointer-generator we utilize is based on the implementation introduced by See et al. (2017), which was designed to be used with an RNN-based sequence-to-sequence model for the text summarization task. Previous works from different areas of natural language processing have adapted this pointer-generator design to the transformer model (Prabhu and Kann, 2020; Jiang et al., 2021). Within these adaptations, the input to the pointer-generator is mapped from the original components in the RNN-based implementation to analogous elements in the transformer model.

During the generation process, a soft switch is utilized for each position in the generated question to determine whether to select a token from the model's fixed vocabulary or copy an out-of-vocabulary token from the input context.

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}) \tag{4.3}$$

$$P(w) = p_{gen} p_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \tag{4.4}$$

The soft switch $p_{gen} \in [0,1]$ for position t in the generated question is shown in equation 4.3, where w_{h^*}, w_s, and w_x are learnable vectors, and b_{ptr} is a learnable scalar. s_t represents the decoder hidden state, x_t represents the input to the decoder, and h_t^* represents a context vector calculated by summing the encoder hidden states, which are weighted by an attention distribution a_t. We derive a_t by averaging the heads of the encoder-decoder attention layer from the last decoder block. For each sample, an extended vocabulary is formed by taking the union of the tokens in the model's fixed vocabulary and the tokens that appear within the input context. The probability of a token w from the extended vocabulary appearing at position t in the generated question is calculated by summing its generation probability and copy probability, as shown in Equation 4.4, where i represents each position in the input context where the token w occurs. Notice that if the token w is not a member of the model's fixed vocabulary, then $p_{vocab}(w)$ is zero. If the token w does not appear in the input context, then $\sum_{i:w_i=w} a_i^t$ is zero.

4.5 Summary

We began this chapter by describing the motivation behind multi-hop question generation. Next, we formally defined the task and the model's training objective. Lastly, we introduced our proposed model based on the transformer architecture and detailed its various components. In the next chapter, we provide details on the dataset, our model's implementation, and the results of our model's performance.

Chapter 5

Experiment Details and Discussion

5.1 Introduction

We begin this chapter by detailing the contents of the HotpotQA dataset, which we use to train and evaluate our proposed model. We then describe how the dataset is cleaned and divided into various splits. Next, we discuss the implementation details of our model. Lastly, we discuss the performance of our model and the results of the automated evaluation metrics.

5.2 Dataset

For training and evaluating our model, we utilize HotpotQA (Yang et al., 2018), which was created for the parallel task of multi-hop question answering. HotpotQA consists of 113k samples derived from Wikipedia articles. Each sample consists of the following:

- Ten context paragraphs, including titles.
- A reference question.
- An answer.
- A list of supporting sentences.
- The question's difficulty.
- The question's type (bridge or comparison).

- A unique ID number.

While ten context paragraphs are provided for each sample, only two of those paragraphs contain supporting sentences which create the path of reasoning that connects the question to the answer. The remaining eight paragraphs serve as distractors for the question answering task, which we discard for the question generation task.

We also discard the supporting sentence information. Gupta et al. (2020b) utilized this information to predict which sentences from the context paragraphs should serve as input to their question generation model. We feel that excluding supporting sentence information and learning to generate questions from the complete context paragraphs is a more challenging and valuable task, a sentiment shared by other researchers (Su et al., 2020).

Context 1: The Androscoggin Bank Colisée (formerly Central Maine Civic Center and Lewiston Colisee) is a 4,000 capacity (3,677 seated) multi-purpose arena, in Lewiston, Maine, that opened in 1958. In 1965 it was the location of the World Heavyweight Title fight during which one of the most famous sports photographs of the century was taken of Muhammed Ali standing over Sonny Liston.

Context 2: The Lewiston Maineiacs were a junior ice hockey team of the Quebec Major Junior Hockey League based in Lewiston, Maine. The team played its home games at the Androscoggin Bank Colisée. They were the second QMJHL team in the United States, and the only one to play a full season. They won the President's Cup in 2007.

Answer: 3,677 seated.

Reference question: The arena where the Lewiston Maineiacs played their home games can seat how many people?

Figure 5.1: Bridge-type question from the HotpotQA dataset (Yang et al., 2018).

Samples in the HotpotQA dataset are divided into two different question types: *bridge* and *comparison*. Samples containing bridge questions identify an entity that connects the two context paragraphs and utilize that connection to form a question. Figure 5.1 demonstrates a bridge-type sample, where Androscoggin Bank Colisée is the bridge entity that connects the two context paragraphs. To answer the reference question, two separate rea-

Context 1: Henri Leconte (born 4 July 1963) is a former French professional tennis player. He reached the men's singles final at the FrenchOpen in 1988, won the French Open men's doubles title in 1984, and helped France win the Davis Cup in 1991. Leconte's career-high singlesranking was world No. 5.

Context 2: Jonathan Stark (born April 3, 1971) is a former professional tennis player from the United States. During his career he won two Grand Slam doubles titles (the 1994 French Open Men's Doubles and the 1995 Wimbledon Championships Mixed Doubles). Stark reached the World No. 1 doubles ranking in 1994.

Answer: Jonathan Stark.

Reference question: Which tennis player won more Grand Slam titles, Henri Leconte or Jonathan Stark?

Figure 5.2: Comparison-type question from the HotpotQA dataset (Yang et al., 2018).

soning hops must be performed, which requires answering two smaller questions in turn. The first reasoning hop is "where did the Lewiston Maineiacs play their home games?" The answer, the Androscoggin Bank Colisée, can be found within Input Context 2. The second reasoning hop is "How many people can the Androscoggin Bank Colisée seat?" The answer, 3,677, can be found within Input Context 1 and serves as the final answer to the reference question.

Questions belonging to comparison samples are formed by selecting a different entity from each context paragraph and asking about a property they share. Figure 5.2 demonstrates a comparison-type sample, where the question is formulated by comparing two professional tennis players, Henri Leconte and Jonathan Stark. Again, two separate reasoning hops must be performed. The first reasoning hop is "How many Grand Slam titles did Henri Leconte win?" By reading Input Context 1, we can infer that he has not won any. The second reasoning hop is "How many Grand Slam titles did Jonathan Stark win?" The answer, two, can be found within Input Context 2 and serves as the final answer to the reference question.

5.2.1 Data Cleaning and Splitting

We removed samples from the dataset containing a simple yes or no answer since these samples were not suitable for our task. Performing question generation in the answer-aware setting requires answer spans to be explicitly labelled within the input context. A simple answer of yes or no usually will not be literally found within the input context, which makes labelling the answer span impossible for these training samples. Removing these samples resulted in 91,911 usable samples. 78,909 of those samples contain bridge-type questions, while 13,002 are comparison-type. Our dataset was divided into three splits: *training*, *validation* and *test*. The training split was used to learn the model's trainable parameters, while the validation split was used for hyperparameter tuning. The test split was only used to evaluate the model once its hyperparameters were selected. The training split received 80% of the dataset's samples, while the validation and test split each received 10%. We shuffled the samples in the dataset before assigning them to a split and stratified the splits on the sample type and question difficulty. The unique ID number of each sample was used to ensure that there was no data leakage between the splits.

Since the HotpotQA dataset was derived from Wikipedia articles, the data was reasonably clean, and minimal preprocessing was necessary. Although it is possible to debate the correctness of the contents of Wikipedia articles, the articles themselves tend to be well written. They do not contain slang and are not littered with unnecessary punctuation or emojis. Other datasets used for question generation research, such as the Amazon question/answer dataset, have been known to contain such material since its samples are crowdsourced from product pages on its e-commerce platform, where there are no standards for the style of writing that is acceptable.

We utilized regular expressions to remove punctuation from the splits, except for apostrophes, hyphens and question marks. The text belonging to each sample was not converted to lowercase since utilizing cased text resulted in better performance.

5.3 Implementation Details

Our proposed model was implemented in the Python programming language using PyTorch (Paszke et al., 2019)[6]. This open-source machine learning framework implements tensor operations and deep neural networks with extensive support for hardware acceleration. We experimented with many different hyperparameter configurations to achieve the best possible results during the training process. The chosen hyperparameters and other implementation considerations are discussed below.

Deep learning models that handle natural language processing tasks utilize vocabularies to map tokens to numeric values. Our proposed model uses a single vocabulary shared by the encoder and the decoder. This vocabulary consists of placeholder tokens, which replace named entities, and non-placeholder (standard) tokens. In natural language processing models, it is common practice to restrict the token vocabulary to the n most common tokens in the training data. It is also common to exclude tokens from the vocabulary if they appear with a frequency less than a specified threshold. Restricting the token vocabulary reduces the number of trainable parameters in the model and reduces noise in the input data. We built our token vocabulary by first deciding which standard tokens to include. We determined which standard tokens in the training data were present in the glove.840B.300d.txt[7] file and counted the frequency of those tokens within the training data. We then selected the top 40,000 tokens that appeared more than twice within the training data. Next, we selected the placeholder tokens that occurred more than five times within the training data. We found it beneficial to specify separate threshold frequencies for each token type since the embeddings for the placeholder tokens were learned, and the embeddings for the standard tokens were frozen. It is also the case that to learn a helpful embedding for a token, it needs to be seen an adequate number of times within the training data.

We created a separate vocabulary which mapped each part-of-speech tag to a unique numeric value. Since each part-of-speech tag appeared numerous times within the training

[6] https://pytorch.org/
[7] https://nlp.stanford.edu/projects/glove/

data, we did not find it necessary to restrict which tags were included in the vocabulary.

We initially hoped to utilize pre-trained word embeddings to represent non-placeholder tokens within our model. We experimented with initializing the embeddings of the standard tokens with the embeddings provided in the glove.840B.300d.txt file. We found that using these pre-trained embeddings was less effective than using randomly initialized ones, even when fine-tuning the pre-trained embeddings. We also found that the large number of trainable parameters that resulted from training the randomly initialized embeddings caused the model to overfit, even when a more significant dropout probability was applied. Interestingly, the model performed best when the placeholder token embeddings were learned from a random initialization while the standard token embeddings were initialized randomly and frozen. We utilized 300-dimensional vectors for the token embeddings.

The embeddings representing the part-of-speech tags consisted of 16 dimensions, while the answer position embeddings consisted of 4 dimensions. We experimented with both learned and static positional encodings in our model, but static positional encodings performed slightly better.

Taking the size of the word embeddings, part of speech embeddings and answer position embeddings into account, we arrive at a hidden dimension of 320. The size of the position-wise feed-forward layers in both the encoder and decoder is set to 640. The encoder and decoder each consist of 3 blocks. We utilized 8 heads in each multi-head attention layer. Finally, the dropout was set to .10 for all layers in the model. The model consists of $\sim$19 million trainable parameters, which were initialized using the Xavier initialization scheme (Glorot and Bengio, 2010).

We utilized the Adam optimizer (Kingma and Ba, 2014) in conjunction with the learning rate schedule shown in equation 5.1, which was originally introduced by Vaswani et al. (2017). In this equation, d_{model} represents the hidden dimension of the model, $step\_num$ represents the current training step, and $warmup\_steps$ is a hyperparameter that allows the learning rate to increase linearly for the first $warmup\_steps$ training steps. Similar to

Vaswani et al. (2017), we set the number of warmup steps to 4000. We trained our model for a total of 50 epochs.

$$lrate = d_{model}^{-0.5} \cdot min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-0.5}) \tag{5.1}$$

We tested the results of our model using greedy decoding and beam decoding with several beam widths. For some hyperparameter configurations, beam decoding produced superior results. For the hyperparameter configuration we have described so far, we found that greedy decoding produced the best results in terms of the automated evaluation metrics.

5.4 Evaluation Results and Discussion

To evaluate the performance of our proposed model, we utilize automated evaluation metrics to score the predictions produced for the samples in the test split. The evaluation metrics used consist of BLEU (Papineni et al., 2002), ROUGE-L (Lin, 2004), and METEOR (Lavie and Agarwal, 2007). These metrics were chosen for their popularity within the question generation research community. Evaluating the proposed model with these metrics will allow its performance to be directly compared to previous work on multi-hop question generation.

Table 5.1: Performance comparison between our model and MulGQ (Su et al., 2020).

	BLEU 1	BLEU 2	BLEU 3	BLEU 4	ROUGE-L	METEOR
MulQG	40.15	26.71	19.73	15.20	35.30	20.51
Our Model	**42.13**	**30.44**	**23.84**	**19.42**	**39.26**	**22.78**

Table 5.1 contrasts our model's performance on the test split with the results of the MulQG model proposed by Su et al. (2020). We compare our proposed model to this work because it matches our task more closely than either of the other previous works on multi-hop question generation. Namely, both our model and the model proposed by Su et al.

(2020) utilize the same dataset and perform multi-hop question generation in an answer-aware setting free of supporting sentence information. The scores for our model were generated using the evaluation script created by Sharma et al. (2017)[8]. This is the same evaluation script utilized by Su et al. (2020) to evaluate the performance of their MulQG model. Our model shows a notable improvement in each of the automated evaluation metrics.

One of the main motivations of the work presented in this thesis was to reduce the time required to train a multi-hop question generation system by utilizing a Transformer-based model instead of relying on recurrent neural networks. Since the multi-hop question generation task utilizes input contexts that are longer than those seen in traditional question generation research, the amount of training time required can increase dramatically when using RNNs. Our proposed model achieved the results listed above with a modest 19 million trainable parameters. The MulQG model, on the other hand, consists of $\sim$57 million. This low number of trainable parameters combined with The Transformer's high degree of parallelization results in our model taking much less time to train. Our model takes approximately 5 hours to train on a Titan Xp GPU, while MulQG requires about 41 hours on a comparable GPU.

Our proposed model is not only quick, it also produces very high quality questions. The generated questions demonstrate a high degree of fluency and many of them contain more detail than the reference questions provided by the dataset. An example of a system-generated bridge-type question is shown below.

Reference Question: The horse - collar tackle is most closely associated with a professional football player who was drafted by what team in 2002 ?

Generated Question: The horse - collar tackle is most closely associated with a former American college and professional football player who was drafted by what team in 2002 ?

[8]https://github.com/Maluuba/nlg-eval

The generated question is very similar to the reference question, but it elaborates upon it by specifying that the professional football player was a former American college football player as well. An example of a system-generated comparison-type question is shown below.

Reference Question: Which was released first Point of Order or The Celluloid Closet ?

Generated Question: Which documentary was released first Point of Order or The Celluloid Closet ?

Again, the system-generated question exhibits greater detail than the reference question by specifying that the films being compared are documentaries.

5.5 Summary

We began this chapter by detailing the contents of the HotpotQA dataset. We then explained how we cleaned the dataset and divided it into training, validation and test splits. Next, we discussed the implementation details of our model. Lastly, we discussed the results of the automated evaluation metrics, the amount of time required to train the model, and the quality of the generated questions.

Chapter 6

Conclusion

6.1 Introduction

This chapter begins by summarizing the work presented in this thesis. We then discuss potential future work that can be completed for the multi-hop question generation task.

6.2 Summary

In this thesis, we proposed a transformer-based model for the multi-hop question generation task. We incorporated components into our model including a pointer-generator and placeholder tokens to limit the negative effects of out-of-vocabulary tokens. We trained and tested our model using the same dataset as the previous works, and evaluated our model's performance using the same evaluation script. The results of the automated evaluation metrics demonstrated a clear improvement over the previously published research. Our proposed model also required significantly less time to train than the previous work.

6.3 Future Work

Since multi-hop question generation is a relatively new area of research, there are many possible improvements for future work. While carrying out the work presented in this thesis, we identified several improvements that may be worth exploring.

While testing the performance of our proposed model, we found the dataset to be lacking in a couple of ways. Each sample in the HotpotQA dataset consists of only a single ground-truth question. This can be problematic since, in multiple cases, we found that the

question generated by our model was equivalent to the ground-truth question. Still, the evaluation metrics did not assign a perfect score since they did not match completely. This issue particularly affected comparison-style questions, an example of which is shown below.

Generated question: Which film was released first The Castaway Cowboy or College Road Trip?
Ground-truth question: Which film was released first College Road Trip or The Castaway Cowboy?

As discussed in section 5.4, some of the questions generated by our proposed model included more detail than the ground-truth questions, yet they were also penalized.

Generated question: Frederick Cleveland Hibbard graduated from what public land - grant research university located in Columbia Missouri US?
Ground-truth question: Frederick Hibbard graduated from what public land - grant university?

The HotpotQA dataset could be improved by adding additional ground-truth questions for each sample. For the comparison-style samples, an additional reference question could be added where the two named entities being compared are swapped. Additional reference questions containing more detail could be added for the bridge samples.

Adding a new reference question to each of the 79k bridge samples would require a significant amount of human labour since this task would have to be completed manually. As for the comparison-style samples, we initially thought it may be possible to automate the process of swapping the named entities being compared. Unfortunately, this is not possible, and this task would require a considerable amount of human labour as well. Since the named entities being compared are not explicitly labelled within the training data, an automated implementation would require these entities to first be manually annotated. Uti-

lizing a named-entity recognition model to complete this annotation would be inadequate since not all entities would be detected accurately and the swap would not be performed correctly. Even if each named entity could be detected automatically with perfect accuracy, there is often more than two named entities per reference question, so it is not trivial to choose which entities must be swapped.

Another way to improve the model's performance is to experiment with different pre-trained word embeddings, including context-dependent embeddings, which may be better suited than the GloVe embeddings we attempted to leverage in our implementation.

Finally, we think it would be promising to determine how to incorporate a knowledge graph into the encoder of a transformer-based model. Su et al. (2020) utilized a graph convolutional network in conjunction with an LSTM-based model, and it appears to be highly effective at handling the extended input context associated with multi-hop question generation.

Bibliography

Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, Cham, 2018. ISBN 978-3-319-94462-3. doi: 10.1007/978-3-319-94463-0.

Husam Ali, Yllias Chali, and Sadid A. Hasan. Automatic question generation from sentences. In *Actes de la 17e conférence sur le Traitement Automatique des Langues Naturelles. Articles courts*, pages 213–218, Montréal, Canada, July 2010. ATALA. URL https://aclanthology.org/2010.jeptalnrecital-court.36.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL https://arxiv.org/abs/1409.0473.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, mar 2003. ISSN 1532-4435.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016. URL https://arxiv.org/abs/1607.04606.

Jason Brownlee. How to choose loss functions when training deep learning neural networks. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/, Jan 2019.

Zi Chai and Xiaojun Wan. Learning to ask more: Semi-autoregressive sequential question generation under dual-graph interaction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 225–237, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.21. URL https://aclanthology.org/2020.acl-main.21.

Yllias Chali and Sadid A. Hasan. Towards topic-to-question generation. *Computational Linguistics*, 41(1):1–20, March 2015. doi: 10.1162/COLI_a_00206. URL https://aclanthology.org/J15-1001.

Wei Chen and Gregory Aist. Generating questions automatically from informational text. 2009.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using

rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

A. Clark, C. Fox, and S. Lappin. *The Handbook of Computational Linguistics and Natural Language Processing*. Blackwell Handbooks in Linguistics. Wiley, 2012. ISBN 9781118347188. URL https://books.google.ca/books?id=6BJOwNHD1osC.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Xinya Du and Claire Cardie. Identifying where to focus in reading comprehension for neural question generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2067–2073, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1219. URL https://aclanthology.org/D17-1219.

Xinya Du and Claire Cardie. Harvesting paragraph-level question-answer pairs from Wikipedia. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1907–1917, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1177. URL https://aclanthology.org/P18-1177.

Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1342–1352, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1123. URL https://aclanthology.org/P17-1123.

Nan Duan, Duyu Tang, Peng Chen, and Ming Zhou. Question generation for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 866–874, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1090. URL https://aclanthology.org/D17-1090.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. Question generation from SQL queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1188. URL https://aclanthology.org/D18-1188.

Deepak Gupta, Hardik Chauhan, Ravi Tej Akella, Asif Ekbal, and Pushpak Bhattacharyya. Reinforced multi-task approach for multi-hop question generation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2760–2775, Barcelona, Spain (Online), December 2020a. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.249. URL https://aclanthology.org/2020.coling-main.249.

Deepak Gupta, Hardik Chauhan, Akella Ravi Tej, Asif Ekbal, and Pushpak Bhattacharyya. Reinforced multi-task approach for multi-hop question generation, 2020b. URL https://arxiv.org/abs/2004.02143.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Michael Heilman and Noah A. Smith. 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*, pages 609–617, Los Angeles, California, June 2010. Association for Computational Linguistics. URL https://aclanthology.org/N10-1086.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://arxiv.org/abs/1207.0580.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

Weiwei Jiang, Junjie Li, Minchuan Chen, Jun Ma, Shaojun Wang, and Jing Xiao. Improving neural text normalization with partial parameter generator and pointer-generator network. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7583–7587, 2021. doi: 10.1109/ICASSP39728.2021.9415113.

Yanghoon Kim, Hwanhee Lee, Joongbo Shin, and Kyomin Jung. Improving neural question generation using answer separation, 2018. URL https://arxiv.org/abs/1809.02393.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Wei-Jen Ko, Te-yuan Chen, Yiyan Huang, Greg Durrett, and Junyi Jessy Li. Inquisitive question generation for high level text comprehension. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6544–6555, Online, November 2020. Association for Computational Linguistics.

doi: 10.18653/v1/2020.emnlp-main.530. URL https://aclanthology.org/2020.emnlp-main.530.

Vishwajeet Kumar, Manish Joshi, Ganesh Ramakrishnan, and Yuan-Fang Li. Vocabulary matters: A simple yet effective approach to paragraph-level question generation. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 781–785, Suzhou, China, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-main.78.

Igor Labutov, Sumit Basu, and Lucy Vanderwende. Deep questions without deep understanding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 889–898, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1086. URL https://aclanthology.org/P15-1086.

Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, USA, 2007. Association for Computational Linguistics.

Jingjing Li, Yifan Gao, Lidong Bing, Irwin King, and Michael R. Lyu. Improving question generation with to the point context. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3216–3226, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1317. URL https://aclanthology.org/D19-1317.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

David Lindberg, Fred Popowich, John Nesbit, and Phil Winne. Generating natural language questions to support learning on-line. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 105–114, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/W13-2114.

Ming Liu, Rafael A. Calvo, and Vasile Rus. Automatic question generation for literature review writing support. In *Proceedings of the 10th International Conference on Intelligent Tutoring Systems - Volume Part I*, ITS'10, pages 45–54, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3642133878. doi: 10.1007/978-3-642-13388-6_9. URL https://doi.org/10.1007/978-3-642-13388-6_9.

Ming Liu, Rafael Alejandro Calvo, and Vasile Rus. G-asks: An intelligent automatic question generation system for academic writing support. *Dialogue Discourse*, 3:101–124, 2012.

Karen Mazidi and Rodney D. Nielsen. Linguistic considerations in automatic question generation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 321–326, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-2053. URL https://aclanthology.org/P14-2053.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

Mao Nakanishi, Tetsunori Kobayashi, and Yoshihiko Hayashi. Towards answer-unaware conversational question generation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 63–71, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5809. URL https://aclanthology.org/D19-5809.

Preksha Nema, Akash Kumar Mohankumar, Mitesh M. Khapra, Balaji Vasan Srinivasan, and Balaraman Ravindran. Let's ask again: Refine network for automatic question generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3314–3323, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1326. URL https://aclanthology.org/D19-1326.

Santanu Pal. Qgstec system description–juqgg: A rule based approach. 06 2010.

Liangming Pan, Wenqiang Lei, Tat-Seng Chua, and Min-Yen Kan. Recent advances in neural question generation. *CoRR*, abs/1905.08949, 2019. URL http://arxiv.org/abs/1905.08949.

Liangming Pan, Wenhu Chen, Wenhan Xiong, Min-Yen Kan, and William Yang Wang. Unsupervised multi-hop question answering by question generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5866–5880, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.469. URL https://aclanthology.org/2021.naacl-main.469.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://doi.org/10.3115/1073083.1073135.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning

library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

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

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. URL https://arxiv.org/abs/1802.05365.

Nikhil Prabhu and Katharina Kann. Making a point: Pointer-generator transformers for disjoint vocabularies. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 85–92, Suzhou, China, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-srw.13.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

Vasile Rus, Brendan Wyse, Paul Piwek, Mihai Lintean, Svetlana Stoyanchev, and Christian Moldovan. The first question generation shared task evaluation challenge. In *Proceedings of the 6th International Natural Language Generation Conference*. Association for Computational Linguistics, July 2010. URL https://aclanthology.org/W10-4234.

Thomas Scialom and Jacopo Staiano. Ask to learn: A study on curiosity-driven question generation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2224–2235, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.202. URL https://aclanthology.org/2020.coling-main.202.

Thomas Scialom, Benjamin Piwowarski, and Jacopo Staiano. Self-attention architectures for answer-agnostic neural question generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6027–6032, Florence, Italy,

July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1604. URL https://aclanthology.org/P19-1604.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks, 2017. URL https://arxiv.org/abs/1704.04368.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016. URL https://arxiv.org/abs/1611.01603.

Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *CoRR*, abs/1706.09799, 2017. URL http://arxiv.org/abs/1706.09799.

Noah A. Smith and Michael Heilman. Automatic factual question generation from text. 2011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Dan Su, Yan Xu, Wenliang Dai, Ziwei Ji, Tiezheng Yu, and Pascale Fung. Multi-hop question generation with graph convolutional network. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.416. URL https://doi.org/10.18653%2Fv1%2F2020.findings-emnlp.416.

Sandeep Subramanian, Tong Wang, Xingdi Yuan, Saizheng Zhang, Adam Trischler, and Yoshua Bengio. Neural models for key phrase extraction and question generation. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 78–88, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2609. URL https://aclanthology.org/W18-2609.

Xingwu Sun, Jing Liu, Yajuan Lyu, Wei He, Yanjun Ma, and Shi Wang. Answer-focused and position-aware neural question generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3930–3939, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1427. URL https://aclanthology.org/D18-1427.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

Stalin Varanasi, Saadullah Amin, and Guenter Neumann. CopyBERT: A unified approach to question generation with self-attention. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 25–31, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlp4convai-1.3. URL https://aclanthology.org/2020.nlp4convai-1.3.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Siyuan Wang, Zhongyu Wei, Zhihao Fan, Zengfeng Huang, Weijian Sun, Qi Zhang, and Xuanjing Huang. PathQG: Neural question generation from facts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9066–9075, Online, November 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.729. URL https://aclanthology.org/2020.emnlp-main.729.

Zhen Wang, Siwei Rao, Jie Zhang, Zhen Qin, Guangjian Tian, and Jun Wang. Diversify question generation with continuous content selectors and question type modeling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2134–2143, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.194. URL https://aclanthology.org/2020.findings-emnlp.194.

Xiuyu Wu, Nan Jiang, and Yunfang Wu. A question type driven and copy loss enhanced frameworkfor answer-agnostic neural question generation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 69–78, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.ngt-1.8. URL https://aclanthology.org/2020.ngt-1.8.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Xuchen Yao, Gosse Bouma, Yi Zhang, Paul Piwek, and Kristy Boyer. Semantics-based question generation and implementation. *Dialogue & Discourse*, 3, 03 2012. doi: 10.5087/dad.2012.202.

Qian Yu, Lidong Bing, Qiong Zhang, Wai Lam, and Luo Si. Review-based question generation with adaptive instance transfer and augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 280–290, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.26. URL https://aclanthology.org/2020.acl-main.26.

Hamed Zamani, Susan Dumais, Nick Craswell, Paul Bennett, and Gord Lueck. Generating clarifying questions for information retrieval. In *Proceedings of The Web Conference 2020*, WWW '20, pages 418–428, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380126. URL https://doi.org/10.1145/3366423.3380126.

Shiyue Zhang and Mohit Bansal. Addressing semantic drift in question generation for semi-supervised question answering. In *Proceedings of the 2019 Conference on*

Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2495–2509, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1253. URL https://aclanthology.org/D19-1253.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3901–3910, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1424. URL https://aclanthology.org/D18-1424.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question generation from text: A preliminary study, 2017. URL https://arxiv.org/abs/1704.01792.

Wenjie Zhou, Minghua Zhang, and Yunfang Wu. Question-type driven question generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6032–6037, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1622. URL https://aclanthology.org/D19-1622.

Appendix A

System-Generated Questions

Question Type: Bridge.
Reference Question: The first televised speech in the House of Commons of the United Kingdom was made by a politician that was assassinated by what group ?
Generated Question: The first televised speech in the House of Commons of the United Kingdom was made by a British Conservative politician who was assassinated by what organization ?

Question Type: Bridge.
Reference Question: The horse - collar tackle is most closely associated with a professional football player who was drafted by what team in 2002 ?
Generated Question: The horse - collar tackle is most closely associated with a former American college and professional football player who was drafted by what team in 2002 ?

Question Type: Bridge.
Reference Question: When was the American director producer and screenwriter born who co - founded Amblin Entertainment ?
Generated Question: When was the American director producer and screenwriter born who co - founded Amblin Entertainment with Kathleen Kandice Kennedy ?

Question Type: Bridge.
Reference Question: The Pangaea Expedition is a worldwide voyage led by a South African - born Swiss adventurer who studied what at Stellenbosch University ?
Generated Question: The Pangaea Expedition is a worldwide voyage led by a South African - born Swiss professional explorer and adventurer who studied what at Stellenbosch University in Western Cape ?

Question Type: Bridge.
Reference Question: In 2011 what was the population of the town in which Phil Nilsen was born ?
Generated Question: What was the population in 2011 of the town where Phil Nilsen was born ?

Question Type: Comparison.
Reference Question: Which was founded first Virginia Commonwealth University or

Batangas State University ?
Generated Question: Which university was founded first Virginia Commonwealth University or Batangas State University ?

Question Type: Comparison.
Reference Question: Which was released first Point of Order or The Celluloid Closet ?
Generated Question: Which documentary was released first Point of Order or The Celluloid Closet ?

Question Type: Comparison.
Reference Question: Which is taller Trump Tower or Americas Tower ?
Generated Question: Which skyscraper is taller Trump Tower or Americas Tower ?

Appendix B

SpaCy Part-of-Speech-Tags

Fine-grained Tag	Description
AFX	affix
JJ	adjective
JJR	adjective, comparative
JJS	adjective, superlative
PDT	predeterminer
PRP$	pronoun, possessive
WDT	wh-determiner
WP$	wh-pronoun, possessive
IN	conjunction, subordinating or preposition
EX	existential there
RB	adverb
RBR	adverb, comparative
RBS	adverb, superlative
WRB	wh-adverb
CC	conjunction, coordinating
DT	determiner
UH	interjection
NN	noun, singular or mass
NNS	noun, plural
WP	wh-pronoun, personal
CD	cardinal number
POS	possessive ending
RP	adverb, particle
TO	infinitival to
PRP	pronoun, personal
NNP	noun, proper singular
NNPS	noun, proper plural

Fine-grained Tag	Description
-LRB-	left round bracket
-RRB-	right round bracket
,	punctuation mark, comma
:	punctuation mark, colon or ellipsis
.	punctuation mark, sentence closer
”	closing quotation mark
“”	closing quotation mark
“	opening quotation mark
HYPH	punctuation mark, hyphen
LS	list item marker
NFP	superfluous punctuation
#	symbol, number sign
$	symbol, currency
SYM	symbol
BES	auxiliary “be”
HVS	forms of “have”
MD	verb, modal auxiliary
VB	verb, base form
VBD	verb, past tense
VBG	verb, gerund or present participle
VBN	verb, past participle
VBP	verb, non-3rd person singular present
VBZ	verb, 3rd person singular present
ADD	email
FW	foreign word
GW	additional word in multi-word expression
XX	unknown
_SP	space

Appendix C

SpaCy Named Entity Tags

Tag	Description
PERSON	People, including fictional
NORP	Nationalities or religious or political groups
FACILITY	Buildings, airports, highways, bridges, etc.
ORGANIZATION	Companies, agencies, institutions, etc.
GPE	Countries, cities, states
LOCATION	Non-GPE locations, mountain ranges, bodies of water
PRODUCT	Vehicles, weapons, foods, etc. (Not services)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK OF ART	Titles of books, songs, etc.
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods
TIME	Times smaller than a day
PERCENT	Percentage (including "%")
MONEY	Monetary values, including unit
QUANTITY	Measurements, as of weight or distance
ORDINAL	"first", "second"
CARDINAL	Numerals that do not fall under another type