Generating Synthetic Question-Answer Pairs for Transfer Learning in Biomedical Question Answering

by

Darshan Bhavin Thaker

A thesis submitted in partial satisfaction of the
requirements for the degree of
Bachelor of Science in Computer Science, Turing Scholars Honors

at

The University of Texas at Austin

Committee in charge:

Professor James Scott
Professor Gregory Durrett
Professor Emmett Witchel

Generating Synthetic Question-Answer Pairs for Transfer Learning in Biomedical Question Answering

Abstract

Biomedical semantic question answering deals with answering questions about biomedical-related concepts. We propose a system that tackles the problem of biomedical semantic question answering in the BioASQ challenge, where the task is to answer biomedical questions given a small set of relevant snippets in biomedical articles. Our approach relies upon transfer learning from a large general question-answering dataset, the Stanford Question Answering Dataset. A naive transfer learning technique of pre-training a deep neural network on the Stanford Question Answering Dataset and finetuning on the BioASQ dataset leads to a significant plummet in performance, from around 80% in the general question-answering domain to 30% in the biomedical domain. Our hypothesis for why this drop occurs is twofold. First, we believe the biomedical domain is inherently harder and requires more domain knowledge to correctly answer questions. Second, the BioASQ dataset is limited in size, and so finetuning does not do enough to allow the system to adapt to the new domain. In this thesis, we explore a solution to the latter problem. To combat small amounts of data in the biomedical domain, we derive an approach based on two-stage synthesis networks to perform generative data augmentation and generate synthetic question-answer pairs in the biomedical domain. We evaluate the quality of generated question and answer pairs both qualitatively through human analysis as well as quantitatively through their effect on transfer from the Stanford Question Answering Dataset to the BioASQ dataset. We explore several ways to improve the quality of synthetic question-answer pairs in the biomedical domain. While these improvements help increase validation accuracy on the BioASQ dataset, we find that ultimately, these approaches yield negative transfer.

Contents

Acknowledgments

I would like to thank my advisor, Professor James Scott, for being very helpful and supportive in the time I have gotten to do research with him and for giving me the freedom to explore many different topics in this research. I would also like to thank Professor Greg Durrett for valuable guidance and advice on this project and for insightful brainstorming help. Finally, I would like to thank Prabhat Nagarajan for the countless discussions about this project and Kapil Krishnakumar, as well as Ewin Tang for helpful discussions about matrix completion.

Chapter 1

Introduction

As information and knowledge has been made more available on the Internet over the last few decades, many techniques in the field of biomedicine have been developed to leverage these massive amounts of data. With large corpora and knowledge databases such as PubMed Central and MEDLINE, biomedical workers are tasked with the tough task of combing through and synthesizing hundreds of thousands of domain-specific articles when looking for answers to specific questions. One approach to facilitating this search is developing better search engines and query handling in large databases such as MEDLINE, but leaving it up to biomedical experts to extract relevant snippets and answer questions from a set of filtered biomedical articles. A more compelling approach is to develop a biomedical question-answering system that uses the power of machine learning and data-driven approaches to retrieve correct answers to biomedical questions. The BioASQ challenge was developed to allow researchers to tackle the problem of biomedical question answering [35]. In this thesis, we develop a technique to address the problem of biomedical question answering specifically for the BioASQ challenge.

The BioASQ dataset consists of around 650 question-answer pairs and a set of relevant snippets from biomedical articles. For example, the following triplet is an example of a question, answer, and snippet from the BioASQ dataset.

Example Snippet: GM1 and GM2 gangliosidosis are associated with deficiency of β-galactosidase and β-hexosaminidase respectively.

Question: Which enzyme deficiency can cause GM1 gangliosidoses?

Answer: β-galactosidase.

It is non-trivial to develop a biomedical question-answering system to answer questions of this form, since this system must identify relevant snippets to the question from the set of given snippets and parse these snippets to obtain the correct answer. In the above question, giving the correct answer to the given question requires reasoning about the question with respect to the snippet. A successful approach would be able to identify the words 'cause' and 'associated with' as related and then examine relationships between the objects in the snippet: GM1 and GM2 gangliosidosis and β-galactosidase and β-hexosaminidase. The word 'respectively' gives us the correct relationship between these objects. Additionally, to answer this question, a system might have to reason about the concept ontologies of the biomedical phrases β-galactosidase and β-hexosaminidase to determine whether they are enzymes.

With data of the form in the BioASQ dataset, traditional machine learning approaches can be employed to learn a mapping from questions to answers using the BioASQ dataset for training. Question-answering as a field has been an area of interest in Natural Language Processing for many years, and recent approaches in the field have seen tremendous successes with the use of deep learning techniques [31]. However, these deep learning approaches typically require large amounts of data, so training these models on the BioASQ dataset is infeasible. Thus, Wiese et al. propose leveraging other similar question-answering datasets that have large amounts of data using methods from transfer learning. Transfer learning aims to leverage knowledge gained from one source task in another related, but separate target task. In cases where labeled data for the target task is rare, transfer learning has

clear value. For the BioASQ dataset, Wiese et al. propose performing transfer learning from a popular general question-answering dataset, the Stanford Question Answering Dataset (SQuAD) [39, 28]. SQuAD contains over a hundred thousand labeled question-answer pairs, so training a deep learning model on this data is feasible. The following triplet is an example of a snippet, question, and answer from the SQuAD dataset.

> *Example Snippet*: Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24?10 to earn their third Super Bowl title.
>
> *Question*: Which NFL team represented the AFC at Super Bowl 50?
>
> *Answer*: Denver Broncos.

Even though the two datasets, SQuAD and BioASQ, are significantly different with the domain of snippets, the motivation of transfer learning is that, on SQuAD, a model can still learn syntactic structures of language and meta-reasoning about relationships between words that might help this model answer questions in the biomedical domain. Most simple approaches to deep transfer learning concern *pre-training* [24]. In this method, when training a neural network for a target task, weights of a neural network are initialized to the weights of a network trained on a related source task. This has shown remarkable results in the computer vision domain, where neural networks are initialized to a convolutional neural network trained on the large ImageNet dataset of object images ([18], [32]). After training on ImageNet, these neural networks are then fine-tuned on the target dataset of choice with the hope that the initialization of the network to a model trained on ImageNet will help improve classification accuracy in the target domain. Weise et al. find that a deep learning

approach of pre-training a model on the SQuAD dataset improves accuracy on the BioASQ dataset by a significant margin [39].

Despite the success of the transfer learning approach, transfer learning comes with its own set of challenges, especially when applied to the BioASQ dataset. Pre-training a deep neural network on SQuAD and finetuning on the BioASQ dataset leads to a significant plummet in performance, from around 80% in the general question-answering domain to 30% in the biomedical domain. We hypothesize that this gap can occur because of two main reasons. First, the BioASQ dataset contains inherently tougher questions to answer because of the biomedical domain knowledge required to answer questions. This is illustrated in the example question from the BioASQ dataset shown above, where a system might be required to know that β-galactosidase is an enzyme. Secondly, we believe that since the BioASQ dataset has approximately only 650 question and answer pairs to train on, a model pre-trained on SQuAD overfits to the types of questions and answers from the SQuAD dataset and does not have enough data in the biomedical domain to finetune on and adapt to the style of questions in the BioASQ dataset. In this thesis, we explore a solution to the latter problem through data augmentation in the biomedical domain. Data augmentation is an approach that adds new data to an existing dataset to increase the number of training examples in the dataset or increase the diversity of examples in the training set. Typically, data augmentation consists of adding generated or synthetic data to a dataset, and it is a well-known approach to combat overfitting [26]. In computer vision tasks, simple data augmentation schemes apply random rotations, flips, and transformations to images in the training dataset. This generates many different variations of the same image in the dataset, allowing a model to see a larger distribution of images during training. In the context of transfer learning, data augmentation can also aid transfer learning from a source task to a target task. Because of a lack of data in the target domain, a model can overfit to the source task and perform poorly on the target task. To address this, data augmentation can be employed in the target dataset so a richer data distribution is available to finetune on.

Our goal is to perform generative data augmentation for biomedical question answering,

specifically in the BioASQ dataset, by generating synthetic question-answer pairs. Our hypothesis is that if we can generate high-quality synthetic question-answer pairs and augment the BioASQ dataset with these pairs, a model pre-trained on SQuAD will perform better on the BioASQ dataset during finetuning. To that end, our contribution is a pipeline for question and answer generation in the biomedical domain, adapted from an approach proposed by Golub et al. (coined the two-stage synthesis network) [9]. This thesis is focused on improving the quality of synthetic question-answer pairs on the BioASQ dataset with the end goal of better transfer. We find that using two-stage synthesis networks on the biomedical domain yields noisy and poor generated questions and answers, which end up hurting the performance of a model during finetuning. We explore methods for improving the quality of these synthetic question and answers, but find that even with these improvements, performance on the BioASQ dataset falls during finetuning. We discuss reasons for why this drop might be occurring and evaluate possible solutions.

Chapter 2

Background

In this chapter, we describe the background for statistical natural language processing and deep learning necessary for our method.

2.1 Artificial Neural Networks

2.1.1 Feedforward Neural Networks

Neural networks are powerful function approximators. At the heart of an artificial neural network is a basic computation unit, a *neuron*. Given an input vector x, a neuron computes the following function:

$$f(x) = a(w^T x + b) \tag{2.1}$$

where $x, w \in \mathbb{R}^n$, $b \in \mathbb{R}$, and a is a function $a : \mathbb{R} \to \mathbb{R}$. A neuron can be thought of as first applying a linear transformation on x through some set of weights w and a bias b [4]. Then, a function a (canonically known as the *activation function*) is applied to the result of the linear transformation. Typically, the activation function is some nonlinear function, such as the sigmoid function or tanh function.

Feedforward neural networks consist of layers of neurons stacked on top of each, as shown in Figure 2.1. A feedforward neural network is shown as a directed acyclic graph, where neurons are vertices and directed edges represent inputs and outputs of neurons. An input vector of dimension d into a neuron is represented as d arrows pointing to that neuron. Thus, generalizing the computation of one neuron in Equation 2.1, the computation at each layer can be performed through a matrix multiplication, where the input is a matrix, a weight matrix stores weight vectors for every neuron in the layer, and the bias is a vector. In this way, when fed an input matrix, a neural network can successively apply matrix multiplications with the weights of each layer to obtain some output. Since each neuron computes some non-linear function (based on the activation function non-linearity), the neural network as a whole can approximate complex non-linear functions. Layers in the neural network where every neuron in one layer is connected to every neuron in the next layer is denoted as a *fully connected layer*.

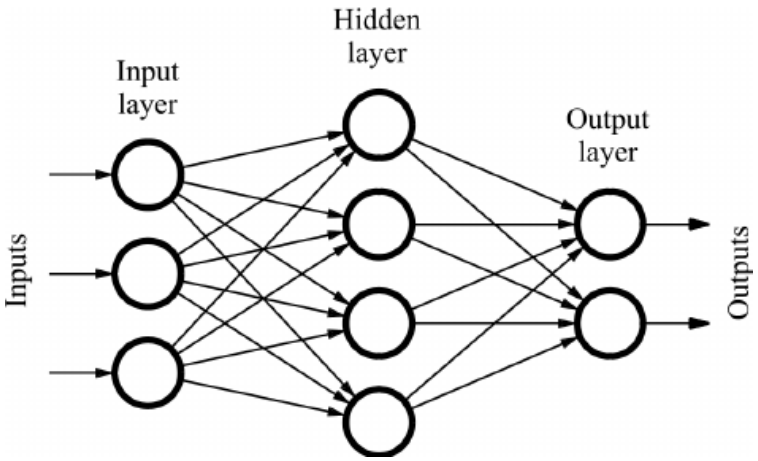


Figure 2.1: Feedforward neural network with 3 layers (Figure from [27])

Feedforward neural networks may be trained using *supervised learning* methods to learn weights and biases in the neural network. In the traditional supervised learning framework, a training set of n examples $[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$ is used for the *training* phase of learning. Each x_i is an input to the neural network and the learning task is to find weights for the neural network such that when x_i is given as input to the neural network, the network

will output y_i. More concretely, the goal of the neural network is to minimize a loss function that measures how well the network performs on the training set. The network is trained through *gradient descent* (typically *stochastic gradient descent*) and the weights are modified during each iteration of gradient descent until convergence.

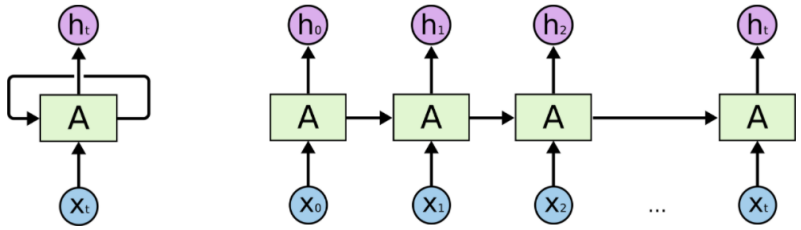
2.1.2 Recurrent Neural Networks

For tasks that involve sequential information such as natural language data, feedforward neural networks struggle to capture relationships between separate inputs since they assume that inputs are independent of each other. *Recurrent neural networks* (RNNs) utilize this sequential information by storing state, or keeping *memory*, of inputs passed into the network. This memory is also denoted as the *hidden state* of the RNN. Specifically, at every time step, a RNN takes in as input not only the current input at timestep but the output of the RNN at the previous time step $t - 1$.

Pictorially, this can be represented as a loop in the graph depicting the neural network, as shown in Figure 2.2a. However, this loop can be *unrolled* to view a RNN as a directed acyclic graph, as shown in Figure 2.2b. Unrolling over t timesteps shows the hidden state of the RNN evolving over time as new inputs are passed in. This can be thought of as memory since the RNN has the ability to find correlations between information in previous iterations and the new inputs. For input such as natural language, this can be helpful to encode syntactic structure of the sentence into the input instead of feeding in a single character or word at a time.

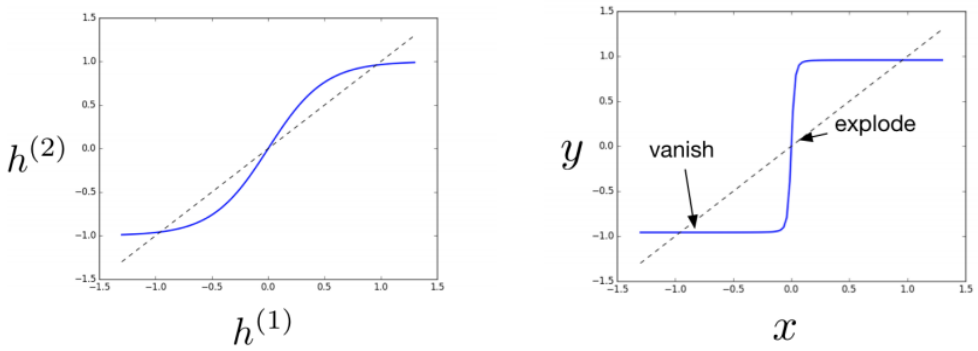
Limitations

Modeling natural language text data using a recurrent neural network has yielded impressive results for applications such as text generation [33], however there are two main problems with a standard recurrent neural network model.



(a) RNN Graph (b) Unrolled RNN

Figure 2.2: Recurrent Neural Network Graphs (Figure from [23])



(a) Tanh Activation Function (b) Function computed by RNN

Figure 2.3: Instability of RNNs (Figure from [12])

First, as a RNN is unrolled over time, inputs are fed through many hidden layers of multiplicative updates and activation functions. As the number of these hidden layers increases, inputs can either get prohibitively large or close to 0. These especially become an issue for computing gradients in deep networks. If the difference between weights across layers is too large, the computed gradient will be too large and an update to the weights in the neural network using a gradient descent step will drastically change weights. This can lead to unstable models as well as numerical instability issues in implementation. On the other hand, if gradients are too small, weights will not update and convergence of gradient descent-based algorithms will slow down rapidly. These two problems are denoted the *exploding gradient problem* and the *vanishing gradient problem* respectively. In an RNN with tanh activation

functions, this problem is illustrated in Figure 2.3. As the tanh function is iteratively applied, we see the function computed by the RNN in Figure 2.3b. It is clear from the graph that the gradient either explodes or vanishes in this figure.The exploding gradient problem is easily combatted by clipping gradients during every iteration to ensure they do not get very large. The vanishing gradient problem requires more complex approaches to tackle, as discussed in the following section.

Second, in practice, RNNs struggle to capture long-term dependencies in inputs [2]. Specifically in language data, this can cause issues in tasks such as text generation where previous sentence context for generating a new word is crucial. If this context is lost over time, the model will struggle to predict a relevant word.

2.1.3 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks, proposed by Hochreiter et al. [13], demonstrate solutions to both of the problems of RNNs mentioned in the previous section. Consider the RNN in Figure 2.2a. In an LSTM, the RNN A is replaced by a special LSTM cell, designed to handle long-term dependencies. An LSTM cell consists of four main components, divided into gates that handle inputs. These components produce outputs like a RNN cell, however they also modify a special state, denoted the *cell state*, a vector of information where each element can be thought of as a memory. The cell state is shared across multiple forward passes to the LSTM, and gates modify information in the cell state, enabling long-term memory retention given proper updates. The four main gates for the LSTM are:

1. *Input gate*: The input gate is the first step in determining how the cell state is updated given a new input. The input gate is a one-layer neural network with sigmoid activation neurons that takes in the current input and previous output of the LSTM from the preceding timestep. The output intuitively represents the values in the cell state should be updated, since the sigmoid function outputs values between 0 and 1 (which can be viewed as probabilities across the values in the cell state).

2. *Cell gate*: The cell gate is the second step in determining how the cell state is updated given a new input. The cell gate is a one-layer neural network with tanh activation neurons that takes in the current input and previous output of the LSTM from the preceding timestep. This intuitively represents new values for the cell state, since the tanh function outputs values between -1 and 1.

3. *Forget gate*: The forget gate handles which elements in the cell state should be removed or forgotten. The forget gate is a one-layer neural network with sigmoid activation neurons that takes in the current input and previous output of the LSTM from the preceding timestep. This intuitively gives probabilities of retaining cell state information for every element in the cell state.

4. *Output gate*: The output gate decides what elements in the cell state should be outputted as the output of the LSTM. The output gate is a one-layer neural network with sigmoid activation neurons that takes in the current input and previous output of the LSTM. This intuitively represents probabilities for which values of the cell state should be outputted.

The outputs from these gates are combined in the following ways to update the cell state as well as generate output for the LSTM cell:

1. *Updating cell state*: First, the values from the forget gate are used to update the cell state. A Hadamard product (element-wise product) of the forget gate output and the cell state allows a way to effectively remove memories for which the forget gate layer outputted a value close to 0. Afterwards, the updated cell state is added to the combined values from the input gate and cell gate. Combining the input gate and cell gate gives a way to determine which values in the cell state should be updated and what they should be updated to. Specifically, this combination is performed through a Hadamard product between the output of the input gate and cell gate. Since the output of the input gate can be thought of as a vector of probabilities and the output

of the cell gate can be thought of as new values for the cell state, multiplying these probabilities with the cell gate gives a new cell state.

2. *Generating output*: The value of the output gate can be interpreted as probabilities over the cell states for which elements should be updated. Performing a Hadamard product with the current cell state gives the output of the LSTM. To squash the values of the output between -1 and 1, the cell state is fed through an tanh function before performing a Hadamard product with the output of the output gate.

Mathematically, given current input x_t at timestep t, previous output h_{t-1}, and previous cell state C_{t-1}, the above gates can be described tersely as the set of following updates

$$
\begin{aligned}
f_t &= \sigma(W_f^T[h_{t-1}, x_t] + b_f) && \text{(Forget gate)} \\
i_t &= \sigma(W_i^T[h_{t-1}, x_t] + b_i) && \text{(Input gate)} \\
c_t &= \tanh(W_c^T[h_{t-1}, x_t] + b_c) && \text{(Cell gate)} \\
o_t &= \sigma(W_o^T[h_{t-1}, x_t] + b_o) && \text{(Output gate)} \\
C_t &= f_t \cdot C_{t-1} + i_t \cdot c_t && \text{(Updating cell state)} \\
h_t &= o_t \cdot \tanh(C_t) && \text{(Output value)}
\end{aligned}
$$

where σ is the sigmoid function, and W and b describe weight and biases for the one-layer neural networks representing the forget gate, input gate, cell gate, and output gate.

A traditional LSTM acts on a time sequence in one direction, storing results and context going forward in time. A simple modification of the LSTM yields the *bidirectional LSTM*, where a bidirectional LSTM cell contains two separate LSTM cells, acting on a time sequence in both directions [30]. Practically, this is implemented as running an LSTM in parallel on the input sequence as well as the reversed input sequence. In a natural language setting, this is evidently useful as context for a word can arise from words surrounding the word on

either side. LSTMs have shown impressive results in many areas, from text generation to speech synthesis to image captioning ([10], [11], [37]).

2.2 Sequence to Sequence Models

The *sequence to sequence* (seq2seq) model has been a popular approach for mapping sequences of data to different sequences of data. We motivate this model in the lens of *neural machine translation* [15]. Consider the task of translating a source sentence from English to French. Antiquated translation approaches explored rule-based systems as well as data-driven approaches for translating phrases in sentences independently; however, these approaches usually led to incoherent sentences with no real flow or meaning. Sequence to sequence models instead use an *encoder-decoder* model for neural machine translation ([34], [7]). The goal of these models is to first map an input sentence to a fixed-length vector representation of the sentence using an encoder. Then, this vector representation is mapped to a target sentence using a decoder. In the neural machine translation case, the encoder-decoder model can map a sentence from one language to a vector representation and into another language. Typically, encoders and decoders are RNNs or LSTMs that are able to capture dependencies in the sequence data over time and encode this sequential information in a vector representation of the data.

2.2.1 Attention Mechanisms

The encoder-decoder model yields good results in the neural machine translation, but a possible problem with this approach is that the encoder must condense all information from the sentence into a fixed-size vector. As the length of the source sentence grows, the encoder must lose some information and the performance of the translation model deteriorates [6]. Bahdanau et al. proposed a solution to this model by developing *attention mechanisms* for the decoder [1]. Instead of forcing the encoder to encode the entire source sentence in a

fixed-length vector, in this approach, an encoder only generates a vector output (also called annotation) for every word in the source sentence. When the decoder is generating a new word, it searches over all encoder annotations and finds annotations relevant to generating the current word. From these annotations, the decoder derives a *context vector* that describes relevant parts of the source sentence. This is equivalent to the decoder "paying attention" to certain portions of the source sentence over others when generating a new word. This intuitively makes sense, since certain portions of the source sentence provide much more context than other when translating word by word.

Luong Attention

We describe a specific global attention mechanism proposed by Luong et al. in which the decoder considers all hidden states of the encoder when deriving a context vector [20]. To formalize the Luong global attention mechanism, we employ the following notation. Consider the neural machine translation task from a source sentence $E = \{x_1, x_2, \ldots x_S\}$. Denote h_t as the hidden state for the decoder at any time t. Denote $\bar{h}_s$ as the hidden state for the encoder at time step s where $1 \leq s \leq S$. First, the attention mechanism must find the relevant encoder hidden states given the current decoder hidden state h_t. To do this, the attention mechanism computes an *alignment score* $a_t(s)$ between h_t and every source hidden state h_s, as shown in Equation 2.2.

$$a_t(s) = \frac{\exp\left(\text{score}(h_t, \bar{h}_s)\right)}{\sum_{1 \leq s' \leq S} \exp\left(\text{score}(h_t, \bar{h}_{s'})\right)} \tag{2.2}$$

The alignment score is essentially a softmax over individual scores between the target hidden state and a given source hidden state. This individual score is calculated from a *score-based* function, which can one of the following alternatives:

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & \text{dot product} \\ h_t^T W_a \bar{h}_s & \text{general} \\ W_a[h_t; \bar{h}_s] & \text{concatenation} \end{cases}$$

In the general and concatenation score functions, W_a parameterizes the weights of a one-layer feedforward neural network, which would be jointly learned with the weights of the network for the encoder and decoder.

After obtaining alignment scores between the target hidden state and every source hidden state, the decoder computes a context vector c_t as a weighted sum of all the source hidden states given the alignment scores as weights, as in Equation 2.3.

$$c_t = \sum_{s=1}^{S} a_t(s) \bar{h}_s \tag{2.3}$$

This attention mechanism is typically implemented as an extra layer at every timestep in the neural network for the decoder. For example, if the decoder was implemented as an LSTM, the output of the LSTM at each time step would be fed through an attention layer implementing the above logic, and the generated context layer would be the new output of the LSTM. This new output would then be fed into the LSTM again in the next time step.

Chapter 3

Transfer Learning for Biomedical Question-Answering

Our goal is to tackle transfer learning from a general question-answering dataset to a biomedical question-answering dataset. We describe our approach for this, which is a system built upon *two-stage synthesis networks* [9]. Before describing two-stage synthesis networks, we formally describe the setup for transfer learning. Consider two related, but distinct, tasks: source task S and a target task T, where data on target task T is limited and data for source task S is abundant. We adopt the popular method for transfer learning by pre-training a deep neural network on S and finetuning this model on T, as discussed in Chapter 1.

When data for T is limited, one approach to improving performance is *data augmentation* on the target task to get more data for the finetuning phase. In the context of question-answering, we wish to generate synthetic question-answers on the target task using a pipeline built upon two-stage synthesis networks [9]. We assume a basic reading comprehension model for question-answering. In the setup for regular reading comprehension, data is given in the form of tuples of context paragraphs and questions related to material in that paragraph, and a model is tasked with identifying the correct answer. Two-stage synthesis networks tackle the data augmentation problem by assuming an *extractive reading comprehension* model.

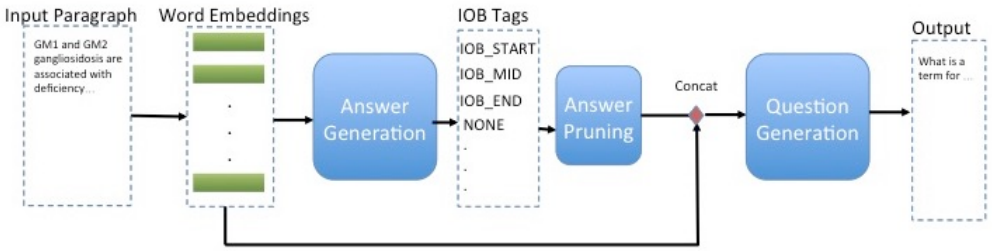


Figure 3.1: Full pipeline for generative data augmentation

Extractive reading comprehension relies upon the added assumption that answers are always exact phrases from the context paragraph, so the extractive reading comprehension task reduces down to searching for answer start and answer end indices in the given paragraph. Two-stage synthesis networks train a question/answer generation model from source task S, and use these generation models to perform *generative data augmentation* in target task T. After the target task data is augmented with the synthetic data, a model is pre-trained on S and finetuned on T with the goal of improved transfer.

We now describe our pipeline for transfer learning, specifically from the Stanford Question Answering Dataset (SQuAD) to the BioASQ dataset ([28], [35]). These datasets are described in more detail in Chapter 4. To generate synthetic question-answer pairs from a paragraph, we model the joint distribution of a question q and answer a given a paragraph p as $P(q, a|p) = P(a|p)P(q|a, p)$, so the two stages of the two-synthesis network can be split up into an answer generation phase and a question generation phase [9]. The full pipeline can be seen in Figure 3.1.

3.1 Input Representation

Formally, input to a reading comprehension model is a context paragraph $P = \{x_0, x_1, \ldots x_n\}$ and a question $Q = \{y_0, y_1, \ldots, y_m\}$, where each x_i and y_i is obtained by splitting the

paragraph and questions by spaces. These words are converted to embeddings using two pre-trained embeddings:

1. *GloVe vectors*: These are 300-dimensional vectors trained on the 840B Common Crawl corpus [25].

2. *Biomedical Word2Vec vectors*: These are 200-dimensional vectors trained using Word2Vec trained on 10 million PubMed abstracts ([14], [22]).

Each word's embedding is a 500-dimensional vector obtained by concatenating both embeddings above. If the word is not present in both of these embeddings, its embedding is initialized to an all zeros vector. Denote these word embeddings for the context paragraph and question as $P = \{p_0, p_1, \ldots, p_n\}$ and $Q = \{q_0, q_1, \ldots, q_m\}$ respectively. We will refer to this notation across the following sections.

We construct the vocabulary for our embedding space by joining a subset of the GloVe embedding vocabulary with a subset of the biomedical embedding vocabulary. Specifically, for training an answer and question generation model on SQuAD, Golub et al. use a subset of the GloVe embedding vocabulary, specifically one of size 110,179 containing words commonly found in the SQuAD dataset [9]. We augment this vocabulary with a subset of the biomedical embedding vocabulary obtained by only considering words found in the BioASQ dataset. [14]. Taking the union of these two subsets gives a vocabulary size of 117,644. Of the 18,103 words in the BioASQ dataset, 9842 of these words exist in both the GloVe and biomedical vocabulary, 680 of these words exist in neither vocabulary, 23 of these words exist only in the GloVe vocabulary, and 7558 of these words exist only in the GloVe vocabulary.

3.1.1 Low-Rank Matrix Completion

Because the GloVe and biomedical embedding vectors are in different spaces, simply concatenating them can lead to two main problems. To motivate the first problem, we provide a simple example. Consider the two words `infiltrate` and `infiltrates`. Both of

these words appear in the biomedical vocabulary, but only the word `infiltrates` appears in the GloVe vocabulary. In the biomedical embedding space ($\mathbb{R}^{200}$), we observe that, as expected, the two words `infiltrate` and `infiltrates` are highly similar in terms of the cosine similarity of their respective embeddings. However, in the concatenated space ($\mathbb{R}^{500}$), because the leading 300 elements of the embedding for the word `infiltrate` are 0, the cosine similarity between these two words is reduced significantly. This encourages a large split in the subspaces defined by the biomedical and GloVe vocabulary, since words that appear only in the biomedical vocabulary are likely to only be similar to other words that appear only in the biomedical vocabulary. Cosine similarity relationships between words that appear in both vocabularies and words that appear in precisely one of the vocabularies are destroyed with the simple embedding concatenation strategy.

Secondly, as will be discussed in further detail below, the distribution of the train and test sets for the answer generation and question generation phase lie in two different subspaces of the concatenated embedding space. Specifically, during training, words are likely to come from the GloVe embeddings vocabulary, and during testing, words are likely to arise from the biomedical embeddings vocabulary. If the subspaces defined by the biomedical and GloVe vocabulary are severely split, this feature mismatch can result in a poor test-time performance.

To tackle this problem, we wish to project the embedding space into a common subspace. The task of projecting into a lower-dimensional subspace could be performed by a traditional dimensionality reduction method such as Principal Component Analysis (PCA), however the zeros in the matrix make this problematic, since dot products of different rows will be biased by these values [4]. Instead, we treat this projection as a *low-rank matrix completion* problem. There are two assumptions in viewing this problem through the lens of low-rank matrix completion. First, we assume that the *ambient space* that the concatenated embedding lies in can be approximated via a lower-dimensional space. This is a reasonable assumption since we are concatenating two different spaces, so there is likely a subspace spanning both whose dimension is less than the sum of the dimensions of the two spaces. Secondly, we assume

that the 0 entries in the embedding matrix (which exist only for a word that is exclusively in one of the given vocabularies) can be treated as 'missing' values for which we wish to complete values by projecting into a similar space as other row entries. Low-rank matrix completion typically arises in the context of *recommendation systems*. For example, in the classic Netflix prize problem, the task of recommending new movies to users can be viewed as a matrix completion problem on the matrix containing user ratings for several movies across many different users [3]. We now describe a low-rank matrix completion algorithm that we will employ to project the embedding space into a common lower-dimensional subspace.

Let $X \in \mathbb{R}^{m \times n}$ be the concatenated embedding space, where m is the vocabulary size and n is 500. Denote $\Omega \subset \{1, \ldots, m\} \times \{1, \ldots, n\}$ as the sets of indices whose values are 0 because the word for that row is only present in one of the vocabularies. The low-rank matrix completion problem can phrased as the following optimization problem [5].

$$\begin{aligned} \min: \ & \text{rank}(Z) \\ \text{subject to}: \ & \sum_{(i,j) \in \Omega} (X_{ij} - Z_{ij})^2 \leq \delta \end{aligned} \tag{3.1}$$

Intuitively, Z is the low-rank approximation for X, so we wish to minimize the rank of Z while maintaining the property that Z is also close to X as possible. The constraint enforces that the Frobenius norm of $X - Z$ is less than some small value δ. The optimization problem in Equation 3.1 is NP-hard to solve, so instead, Candes et al. propose solving a variant of the problem:

$$\begin{aligned} \min: \ & \|Z\|_* \\ \text{subject to}: \ & \sum_{(i,j) \in \Omega} (X_{ij} - Z_{ij})^2 \leq \delta \end{aligned} \tag{3.2}$$

In this optimization problem, $\|Z\|_*$ is defined to be the *nuclear norm* of Z, or the sum of the singular values of Z. Taking the Lagrangian of the above optimization problem, we

Algorithm 1 Low-Rank Matrix Completion for Embeddings: `Soft-Impute`

Given: $X \in \mathbb{R}^{m \times n}$ embedding matrix, $\Omega \subset \{1, \ldots, m\} \times \{1, \ldots, n\}$, $\Omega_0 \subset \{1, \ldots, m\}$
$k \in \mathbb{Z}$ rank for low-rank approximation

$E \leftarrow X[\overline{\Omega_0}] \in \mathbb{R}^{|\overline{\Omega_0}| \times n}$ ▷ Set complement in universe $\{1, \ldots, m\}$
$E \simeq \text{SVD}_k(E) = \widetilde{U}_k \widetilde{D}_k \widetilde{V}_k^T$ ▷ Take first k singular vectors of SVD
$Z^{old} \leftarrow X \widetilde{V}_k \widetilde{V}_k^T$ ▷ Projection
while True **do**
  $X[\Omega] \leftarrow Z^{old}[\Omega]$ ▷ Fill in missing values
  $X \simeq \text{SVD}_k(X) = U_k D_k V_k^T$ ▷ Take first k singular vectors of SVD
  $proj \leftarrow U_k D_k$ ▷ Projected space in $\mathbb{R}^{m \times k}$
  $Z^{new} \leftarrow U_k D_k V_k^T$ ▷ Use low-rank SVD to approximate matrix
  if $\frac{\|Z^{new} - Z^{old}\|_F^2}{\|Z^{old}\|_F^2} < \epsilon$ **then**
    break
  end if
end while
return *proj*

can convert the constrained optimization of Equation 3.2 to Equation 3.3, where $\|X - Z\|_F^2$ denotes the Frobenius norm of $X - Z$, or the L2 norm of the vector of entries for every entry in the matrix.

$$\min : \ \frac{1}{2}\|X - Z\|_F^2 + \lambda \|Z\|_* \tag{3.3}$$

For our approach, we consider the algorithm for low-rank matrix completion proposed by Mazumder et al. [21]. This algorithm is denoted as the `Soft-Impute` algorithm for matrix completion. We adapt this algorithm for our use case to perform low-rank matrix completion - the algorithm is described in Algorithm 1. Denote Ω_0 as the indices of the rows which contain missing entries. This Expectation-Maximization-style algorithm works by iteratively computing a low-rank Singular Value Decomposition (SVD) of the embedding matrix, using this low-rank SVD to impute values into the embedding matrix, and repeating. The stopping criterion on this algorithm thresholds the ratio of the Frobenius norm between two successive approximations Z^{new} and Z^{old}. Recall that the Singular Value Decomposition

of a matrix X is a matrix factorization of X such that $X = UDV^T$, where U and V are unitary matrices, and D is a diagonal matrix. We can interpret elements on the diagonal of D as the *singular values* of X. For this algorithm, a low-rank SVD is computed by simply taking the first k singular vectors of the full SVD of the matrix. We make a key modification to the algorithm proposed by Mazumder et al., namely the initial Z^{old}. Mazumder et al. propose using a zero-initialization for Z^{old}. Instead, since we know information about the structure about the concatenated embedding space, we utilize the information based on words that intersect both spaces. Specifically, we compute a low-rank SVD of the submatrix of the embedding matrix that is fully specified (i.e. the rows for which we have entries in both the GloVe and biomedical embeddings). Then, we perform an orthogonal projection of every element in the given embedding matrix into this space. The intuition for this modification is that we are taking advantage of the information about which rows are fully specified. If we compute an SVD of the entire embedding matrix, the singular vectors will be very close to 0 for the rows containing missing entries. Taking an SVD of the fully specified submatrix and imputing back into the embedding matrix gives an initialization of missing values that is closer to the space defined by the fully specified submatrix, which is desirable.

3.2 Answer Generation

Given a paragraph, the answer generation phase attempts to identify key salient concepts in the paragraph as answers. To implement this, we wish to tag each word in the input paragraph as being part of an answer or not. Specifically, the answer generation model is an Inside-Outside-Beginning (IOB) tagger, where the model takes in as input a word from the input paragraph and outputs a tag from the set {IOB_START, IOB_MID, IOB_END, NONE} [29]. This answer generation model is trained on the source dataset S, so we assume that for supervised learning training, the source dataset (SQuAD) has ground truth labels for IOB tags, where answers in the paragraph are of the tagged form {IOB_START, IOB_MID, IOB_MID, . . . , IOB_MID, IOB_END}. Since we assume an extractive reading comprehension

setup, the answer span for every question lies somewhere in the context paragraph.

We employ the same answer generation model as in two-stage synthesis networks [9]. Specifically, given a context paragraph $P = \{p_0, p_1, \ldots, p_n\}$ of n words, we pass this paragraph through a bidirectional LSTM to obtain forward and backward hidden state outputs for each p_i. Concatenating both forward and backward hidden state outputs gives a representation h_i for each p_i that takes into account context of the words surrounding p_i in the paragraph. These hidden states are then passed through two fully connected layers, where the number of neurons in the second fully connected layer is equal to the number of output tags, 4. The output is a matrix of dimension $n \times 4$, denoting probabilities over the output tags, and the maximum likelihood tag is chosen as the output for every word in the paragraph. This network is trained using a cross entropy loss function between the output probabilities of the network and the true tag labels for every paragraph in the training set.

It is important to note that a traditional IOB tagger has a coherency restriction that IOB_MID tags must be preceded by a IOB_START tag, so previous word tag outputs must be taken into account when predicting the tag for a given word. This can be implemented through running an algorithm such as the Viterbi Algorithm on the sequence of probabilities outputted by the model across a paragraph to enforce this coherency restraint [8]. For speed and simplicity purposes, the model independently predicts a tag per word, and we post-process answer segments, as detailed in the following section.

3.2.1 Answer Extraction and Pruning

A key part of the answer generation module is predicting answer segments from the tags outputted for each word in a context paragraph. To output better answer segments, we propose two simple methods of *answer extraction* and *answer pruning*: *Binary Probability Pruning* and *Biomedical Tag Probability Pruning*, which we describe in more detail below.

Binary Probability Pruning

Since the model does not enforce a coherency restraint to ensure segments of the form IOB_START, IOB_MID, ..., IOB_END, a simple way to identify answer segments would be to consider contiguous words whose tag outputs are not NONE. This ignores any spatial information about the IOB_START, IOB_MID, and IOB_END tag. To prune away answers that are not good, we first assign a value to each word in an answer segment, where this value is equal to the negative log likelihood outputted by the model that this word is not NONE (i.e. the sum of the negative log probabilities of IOB_START, IOB_MID, and IOB_END). Then, the rank of an answer segment is the average negative log likelihood of the words in the answer segment. This gives us a total ordering on all the identified answer segments, and we can take the top k answers (where k is a hyperparameter) as answers outputted for a context paragraph.

Biomedical Tag Probability Pruning

Another technique to extract answers would be to use the tag information outputted by the model to enforce IOB tag coherence. Specifically, answers would be consecutive tags of the form IOB_START, IOB_MID, ..., IOB_END. This implicitly assumes that answers are of length greater than 1, which can be modified by allowing answer segments of only the IOB_START or IOB_END tag to be classified as potential answer segments. To tailor this towards the biomedical domain, we introduce an additional constraint for answer extraction. To formalize this constraint, let G be the set containing the GloVe vocabulary, let B be the set containing the biomedical vocabulary. Then, an answer segment A is considered a valid answer segment if and only if there exists at least one word $w \in A$ such that $w \in B \wedge w \notin G$. Intuitively, this restriction is to extract more answers that contain biomedical concepts. To prune away specific answers, we proceed by assigning a value to each word in an answer segment as in Binary Probability Pruning. This value is equal to the highest negative log likelihood outputted by the model for this word (i.e. if the word is of tag t, the value is

the negative log likelihood of t). As in Binary Probability Pruning, the rank of an answer segment is the average negative log likelihood of the words in the answer segment.

3.3 Question Generation

The question generation stage deals with producing a question given a paragraph and answer. This is modeled as a encoder-decoder model with attention, similar to a sequence to sequence model. Specifically, the encoder takes in as input a paragraph, and the decoder takes in the output of the encoder. The decoder must generate a stream of words that will be the generated question, and it does so by taking in one word embedding in the question q_i and predicting the next token q_{i+1}. For the first and last tokens in the question, a special start and end token are used. For supervised learning training, a full question $Q = \{q_0, q_1, q_2, \ldots, q_m\}$ (where $q_0 = START$) is given to the decoder and the model must learn to output the correct tokens, which would be $\{q_1, q_2, q_3, \ldots, END\}$. For testing, only a paragraph and the special start token is fed into the model, and the model must predict the next token that follows the start token. The specific model of how it learns this mapping is described below. The encoder and decoder model specifics are the same for both training and testing, with the only difference being that for training, a full training set question is passed in, whereas in testing, only the start token is given as input.

3.3.1 Encoder

The encoder acts on the paragraph embeddings. To model the answer location in the paragraph, the embedding for each word in the paragraph is concatenated with a $0-1$ feature that indicates whether that word is in the span of the answer for which we are generating a question. The encoder is modeled as a bidirectional LSTM that acts upon the modified paragraph embeddings $P = \{p_0, p_1, \ldots p_n\}$. The output of the encoder is the concatenated forward and backward hidden states of the bidirectional LSTM for each $p_i \in P$. Denote this

output as $E = \{e_0, e_1, \ldots e_n\}$. The encoder can be thought of as producing context-based representations of each word in the given paragraph.

3.3.2 Decoder

The decoder implements an LSTM that at each time step, takes in as input the output of the encoder E and the embedding for a previously generated question token q_i (recall that questions are padded with a special start token in the beginning of the question, so a previous question token always exists). A Luong attention layer with a dot product score function (refer to Section 2.2.1) is added to the LSTM at each time step, so the network outputs a hidden representation for a question token, then searches for relevant representations in the encoder output to produce a modified hidden representation [20]. Thus, the output of the decoder is a set of hidden representations for every word in the given question. Denote this output as $D = \{d_0, d_1, \ldots, d_m\}$. Each d_i is a hidden representation of dimensionality $\mathbb{R}^h$ where h denotes the size of the hidden representation outputted by the LSTM.

3.3.3 Latent Predictor Network

Given the output of the decoder, the network must map this output to a set of tokens that represent the generated question. From the pre-trained embeddings used (refer to Section 3.1), we can construct a vocabulary of tokens, which we denote as V. The output tokens for each word of the generated question come from this vocabulary. Thus, we wish to learn a function $g : \mathbb{R}^h \to V$, and the generated question becomes $Q_g = \{g(d_0), g(d_1), \ldots, g(d_m)\}$. To learn this function g, Golub et al. describe an architecture motivated by *latent predictor networks* [19], which we describe in detail in this section. The goal of the latent predictor network is to generate a probability distribution over the vocabulary V for every hidden representation d_i outputted by the decoder. The assumption of the latent predictor network is that the conditional probability of any word y in the vocabulary given d_i and the encoder output E is dependent upon latent variables called *predictors*. For the two-stage synthesis

network, there are two predictors used: the *copy predictor*, which we denote as p_c, and the *word generation predictor*, which we denote as p_w. Intuitively, for every generated word, the assumption made is that this word is either copied from a word in the context paragraph (described by E) or newly generated from the given vocabulary. Thus, we can marginalize the conditional distribution of the modeled quantity $P(y|E, d_i)$ to obtain the marginal likelihood shown in Equation 3.4.

$$\log\left[P(y|E, d_i)\right] = \log\left[P(y, p_c|E, d_i) + P(y, p_w|E, d_i)\right] \tag{3.4}$$

Furthermore, using Bayes' theorem, we can factor the joint probability distributions above as shown in Equation 3.5 and 3.6. This suggests an approach that models the probability of the predictors given d_i first, and then given this predictor, find the probability of generating the word y. Given this probabilistic model, we describe modeling these quantities separately in the subsequent sections.

$$P(y, p_c|E, d_i) = P(y|p_c, E, d_i)P(p_c|E, d_i) \tag{3.5}$$

$$P(y, p_w|d_i) = P(y|p_w, E, d_i)P(p_w|E, d_i) \tag{3.6}$$

Predictor Probabilities

To model the quantities $P(p_c|E, d_i)$ and $P(p_w|E, d_i)$, we use a two-layer feedforward neural network that takes in d_i as input and outputs the desired probabilities. This assumes that the encoder output E, which is a representation of the context paragraph, is independent of the probability of choosing the predictors. This is a fair assumption since the probabilities of finding the predictor that a word came from depends only upon that word, so we can condition the probability of each predictor only on d_i. The two-layer neural network has the property that the number of the neurons in the second layer is 2, so the output is a vector in $\mathbb{R}^2$. After performing a log-softmax on the output of this neural network, the output can be interpreted as a log probability distribution over the two predictors p_c and p_w.

Copy Predictor

Next, we describe the model for the quantity $P(y|p_c, E, d_i)$. Denote the output distribution over the vocabulary V as $\mathcal{D}_c$. This model is based upon *pointer networks*, which uses an attention mechanism over the encoder output E to find the desired probability [36]. Specifically, the attention mechanism used is a Luong attention mechanism with a dot product score function (refer to Section 2.2.1) [20]. The alignment score between d_i and every vector in E gives a probability of how likely d_i is contextually related to every $e_j \in E$. This gives us a distribution over the encoder output $E = \{e_1, e_2, \ldots, e_n\}$. We can easily convert this distribution to a distribution over the vocabulary V by considering the corresponding paragraph tokens $P = \{x_0, x_1, \ldots x_n\}$. Note that each x_i is not an embedding, but rather the raw tokens (indices into the vocabulary) for each word in the context paragraph. For every x_j, the probability of x_j in $\mathcal{D}_c$ is equal to the alignment score between d_i and e_j. Thus, the output distribution over the vocabulary $\mathcal{D}_c$ has non-zero probability only for those words in the context paragraph, which is why this distribution intuitively represents the probability of copying a word from the context paragraph.

Word Generation Predictor

Finally we describe the model for the quantity $P(y|p_w, E, d_i)$. We use a one-layer feed-forward neural network that takes in as input d_i and outputs the desired probability. The number of neurons in this layer is the number of words in the vocabulary, or $|V|$. Thus, after performing a log-softmax on the output of the neural network, the output can be interpreted as a log probability distribution over the vocabulary, which intuitively represents the probability of generating a new word from the vocabulary.

Training

During training, given a hidden representation from the decoder, d_i, the probability of generating any question token q_i from the vocabulary is represented as the marginalization

shown in Equation 3.4, so we sum over the two latent predictors: the copy predictor and the word generation predictor. Thus, the output of the latent predictor network is a mixed log probability distribution over the vocabulary representing the sum of distributions of the probability of each predictor multiplied by the distribution outputted by that specific predictor. The network is jointly trained with the encoder and decoder using a cross-entropy loss between the log probability distribution over the vocabulary and ground truth labels of the correct question tokens.

Testing

At evaluation time, the network is only given one token at a time, starting with the special start token. The network then outputs a probability distribution over the vocabulary, and the token with the maximum likelihood is the new generated token. The network then takes in this newly generated token and predicts the following token. This process continues until some maximum question length is hit (chosen to be a hyperparameter) or the special end token is generated by the model. A key difference between the latent predictor network during training and during testing is that during testing, the outputted probability distribution is not a mixed probability distribution over both predictors. Instead, the network chooses the predictor with highest probability and outputs the distribution over the vocabulary outputted by that specific predictor.

3.4 Transfer Learning

After generating synthetic question-answer pairs in the BioASQ dataset for our use case, we augment the BioASQ dataset with these synthetic question-answer pairs. Then, the transfer learning task proceeds as canonical pre-training. Specifically, we first pre-train a FastQA question-answering model on the SQuAD dataset [39, 38]. Then, we finetune this model on the augmented BioASQ dataset, with the hope that the synthetic question and answers aid the limited existing examples of question-answer pairs in the BioASQ dataset.

3.5 Implementation Details

Our answer generation and question generation models were written in Python using Tensorflow. We use an existing implementation provided by Weise et al. of the FastQA model for question-answering [39]. The above models describe the training models given one question, answer, and paragraph at a time, but in practice, we use a batch size of 24 to feed in 24 such triplets. The details of the model can easily be extended to handle another dimension, the batch size. The hidden sizes for all the LSTMs are set to be 100, and the network is trained using the Adam optimizer with learning rate 0.001 for the answer generation model and 0.0001 for the question generation model [17]. The question generation model has ≈ 14 million parameters in the model to train.

Chapter 4

Experiments and Results

4.1 Datasets

4.1.1 BioASQ

The BioASQ dataset is a biomedical question answering dataset consisting of questions and answers to biomedical-related questions, along with given snippets and articles related to the subject matter of the question [35]. The training data consists of 1799 questions that can be divided into 4 types: *yes/no*, *factoid*, *list*, and *summary*. To answer each question, the data also contains a list of documents and list of relevant snippets for every question. Combining the snippets for a given question, we can view the biomedical question answering task through the lens of reading comprehension. After concatenating all snippets to form a passage, we observe that the mean length of a passage in the BioASQ dataset is ≈ 483 words, the mean length of an answer is ≈ 4 words, and the mean length of a question is ≈ 10 words. Our focus in this work is on exact answers to *factoid* questions for two reasons. First, we find that in the BioASQ dataset, of 618 factoid questions, 443 or (72%) of them contain answers that are exact matches to phrases from the given snippets, and 97.6% of the given answers have at least one word that was taken directly from the snippet. Thus, we assume an extractive question answering model, or, in particular, we assume that the factoid

answers are contained somewhere in the input snippets. Second, the evaluation measures are simple: exact match accuracy and mean reciprocal rank (described in more detail in Section 4.2.4).

4.1.2 Stanford Question Answering Dataset

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset where input paragraphs are snippets from Wikipedia articles and are accompanied with several questions and ground truth answers [28]. The dataset contains over 100000 question-answer pairs on over 500 Wikipedia articles. SQuAD has the restriction that all answers to questions are contained somewhere in the input text, so the problem is that of extractive reading comprehension.

4.2 Results

Our core results are in the context of evaluating transfer learning from the SQuAD dataset to the BioASQ dataset. Table 4.7 shows the results of pretraining the FastQA question-answering model on SQuAD and finetuning on the BioASQ dataset with and without generating synthetic question-answer pairs. The following sections investigate this in more depth, isolating specific parts of the pipeline described in Chapter 3 and evaluating them both qualitatively and quantitatively.

4.2.1 Embeddings

We find that performing low-rank matrix completion on the embedding matrix yields significant improvements in preserving word similarities across vocabularies. A key hyperparameter for low-rank matrix completion is picking the dimensionality of the projected space, or the rank of the low-rank matrix that approximates the embedding space. We experiment with a rank of $100, 250$, and 400.

Embedding Type	Word	Similar words (ranked)
GloVe	hands	hand, fingers, finger, knees, hold
	hemoglobin	albumin, myoglobin, glucose, platelet, hba1c
	tumor	tumour, tumors, malignant, cancers, cancer
Biomedical	hands	feet, fingertips, forearms, toes, fingers
	hemoglobin	haemoglobin, hba1c, ferritin, rbc, deoxygenated
	tumor	tumour, tumors, tumours, metastasis, melanoma
Concatenated	hands	hand, fingers, finger, arms, face
	hemoglobin	albumin, hba1c, myoglobin, glucose, platelet
	tumor	tumour, tumors, malignant, cancers, cancer
Projected 400	hands	hand, fingers, finger, arms, face
	hemoglobin	albumin, hba1c, myoglobin, glucose, platelet
	tumor	tumour, tumors, malignant, cancers, cancer
Projected 250	hands	hand, fingers, finger, knees, hold
	hemoglobin	albumin, myoglobin, glucose, platelet, hba1c
	tumor	tumour, tumors, malignant, cancers, pancreatic
Projected 100	hands	hand, him, fingers, face, arms
	hemoglobin	albumin, lipid, glucose, ldl, lipids
	tumor	tumors, tumour, malignant, pancreatic, prostate

Table 4.1: Embedding Word Similarities for words in GloVe and Biomedical vocabularies

First, we find that without the proposed initialization of the `Soft-Impute` algorithm to a projection based on the fully specified entries of the embedding matrix, the completed values in the original embedding matrix are very close to 0, and the projected space is almost equivalent to using a traditional dimensionality reduction algorithm such as PCA [4]. With the proposed initialization that exploits the structure of the embedding matrix, we see significant improvements in the quality of the resulting projection. All the results demonstrated below assume this initialization to the `Soft-Impute` algorithm.

Table 4.1 shows top 5 ranked words in the embedding space for three words: `hands`, `hemoglobin`, `tumor`, all of which appear in both the GloVe and biomedical vocabulary. We see that in the concatenated embedding space, there is a strong bias towards words in the GloVe embedding space. For example, the top similar words for the word `hemoglobin` in the concatenated embeddings are almost identical to the top similar words in the GloVe

embedding space and the top similar words in the biomedical space are lost. Even in the projected subspaces of rank 400, 250, and 100, we see that the words have a bias towards words in the GloVe embedding space. This could be due to two reasons. First, because the biomedical words are likely in a different space than the GloVe embedding vocabulary words, when evaluating the raw cosine similarities between two words, these similarities in different spaces are not directly comparable. Secondly, the total vocabulary contains many more words from the GloVe embedding vocabulary than the biomedical vocabulary, so it is natural for a low-rank approximation of the full embedding matrix to optimize on capturing latent factors that capture similarities across the GloVe embeddings.

To more thoroughly evaluate the quality of the low-rank approximation, we examine the specific example of the words `infiltrate` and `infiltrates`, which was the example provided in Section 3.1 as the motivation for performing low-rank matrix completion. Recall that both of these words appear in the biomedical vocabulary, but only the word `infiltrates` appears in the GloVe vocabulary. In the concatenated embedding space, the cosine similarity between the two words `infiltrate` and `infiltrates` drops from 0.89 in the biomedical embedding space to 0.39, due to the leading zeros in the embedding for the word `infiltrate`. To evaluate the quality of low-rank matrix completion, we can observe the cosine similarity between these two words before and after performing low-rank matrix completion. For the specific pair of words `infiltrate` and `infiltrates`, Table 4.2 shows the raw cosine similarities of the two words as well as the rank of the word `infiltrate` from `infiltrates` in a nearest neighbors search across the full embedding space. This shows that for $k = 250$, the low-rank matrix completion algorithm projects the two words `infiltrate` and `infiltrates` closer together in the embedding space, suggesting that the low-rank matrix completion approach is a large step towards projecting both embeddings into a lower-dimensional subspace. The 400-dimension and 100-dimension projected spaces do not perform as well, which highlights the traditional bias-variance tradeoff. While a 100-dimensional projection yields the highest raw cosine similarity for these words across the different projections, the word `infiltrate` is ranked at position 13 in a nearest neighbor search. This indicates that while the words

Embedding Type	Cosine Similarity	`infiltrate` Rank
Concatenated	0.39	19
Projected 400	0.34	67
Projected 250	0.56	4
Projected 100	0.58	13

Table 4.2: Embedding Word Similarities for words in GloVe and Biomedical vocabularies

`infiltrate` and `infiltrate` were projected closer to each other, other extraneous words might also be mapped closer. Intuitively, since the GloVe and biomedical vocabularies are of rank 300 and 200 respectively, there is unlikely to be a low-rank subspace with rank less than 200 that spans both of these spaces. After examining the singular values of the GloVe and biomedical embeddings, we find that both matrices are relatively well-conditioned (the ratio of their highest singular value to their lowest singular value is fairly low), which supports our hypothesis that there is unlikely to be a low-rank subspace of rank less than 200 that spans both spaces. For all experiments that follow, we use a word embedding space projected down to 250 dimensions using the Soft-Impute method [21], since that gives the best results relative to the other low-rank embedding projections.

4.2.2 Generated Answers

Before discussing the experimental evaluation of the answer generation module on the BioASQ dataset, it is prudent to define a heuristic for the quality of a "good" answer when given any context paragraph in the BioASQ dataset. Examining the BioASQ dataset ground truth answers, we define four classes of answers that define good answers. To get an insight into the distribution of the data, we sample 50 answers at random from the BioASQ dataset and examine their distribution over these classes. These four classes of answers are:

1. *Nouns*: Biomedical terms that are objects of sentences in the context paragraph. These are typically answers where a noun phrase (some biomedical syndrome, disease, en-

zyme, etc.) is an object being acted upon in a sentence. Of the 50 sampled answers, 36 are of this form.

2. *Adjectives*: In particular, these adjectives typically describe an important biomedical term. For example, in the phrase `autosomal recessive mode of inheritance`, an answer term might be `autosomal recessive`. Of the 50 sampled answers, 3 are of this form.

3. *Causal relationship*: An answer could be a phrase indicating the function of some biomedical concept, or a relationship between two terms. Of the 50 sampled questions, 8 are of this form.

4. *Number*: In the BioASQ dataset, these answer phrases are typically answers to questions starting with the phrase "How many". Of the 50 sampled questions, 3 are of this form.

The above distribution of answer types indicates that the two most frequently found answe classes are noun phrases as well as answers that help identify causal relationships. Intuitively, answers that encode causal relationships require much higher-level reasoning to identify as possible answers, so our answer generation model outputting independent tags per word in the paragraph is unlikely to capture this class of answers. However, if we extrapolate from these classes of answers to the distribution of answers over the whole dataset, we conclude that approximately 75% of the dataset contains answers of the form of biomedical answers. We guide our qualitative analysis of generated answers using these principles.

We first qualitatively evaluate the answer generation module by examining output tags for sample paragraphs in the BioASQ dataset and discussing quality. The answer generation module was trained for 15 epochs on the SQuAD dataset. We experimented with increasing the number of epochs for training, but found that increasing the number of epochs too much causes overfitting on the SQuAD dataset. This causes the phenomenon that answers

Paragraph Snippets (answers are bolded)	
Without Answer Pruning	Binary Probability Pruning
QT interval is shortened when **QTc is less than 350 ms** (**1st degree of shortening**) ...	QT interval is shortened when **QTc is less than 350 ms** (**1st degree of shortening**) ...
The major tumour suppressor protein , **p53** , is **one** of **the most** well **- studied proteins** in **cell biology** ...	The major tumour suppressor protein , p53 , is **one** of **the most** well - studied proteins in cell biology ...
... Thus , **flumazenil provides a safe and effective** means of **attenuating** or **reversing the CNS - depressant effects** of benzodiazepines whenever indicated , e . g . following benzodiazepine - induced **general anaesthesia** , **conscious sedation , or** after **benzodiazepine overdose** , either alone or in combination with other **agents** ...	... Thus , flumazenil provides a safe and effective means of attenuating or reversing the CNS - depressant effects of benzodiazepines whenever indicated , e . g . following benzodiazepine - induced general anaesthesia , conscious sedation , or after benzodiazepine overdose , either alone or in combination with other agents ...

Table 4.3: Identified Answers with and without Binary Probability Pruning

predicted on paragraphs from the BioASQ dataset tend to be words that appear frequently in the SQuAD dataset. These words are usually not biomedical terms or nouns, but instead commonly used adjectives, so these answers do not model the intended distribution of the BioASQ dataset.

As discussed in Section 3.2.1, we employ a variety of answer pruning techniques to extract relevant answer snippets that model the distribution of answers in the BioASQ dataset. Given the naive strategy discussed by Golub et al., we find that the answer generation model over-predicts and labels many of the words in a given paragraph as potential answers, as shown in Table 4.3. The model makes several mistakes in including partial sentences as answers or partial phrases. For example, the phrases `QT interval is shortened` and `- studied proteins` split sentences unnaturally and do not serve as good answers.

We now compare the naive approach with our approach of introducing various techniques of answer pruning. The first answer pruning approach is Binary Probability Pruning (refer

to Section 3.2.1). Over an entire paragraph, we ranked each answer segment by its average likelihood over the segment of each word being tagged as not NONE, as outputted by the model. As shown in Figure 4.3, this strategy performs poorly at identifying answers and prunes answers that could fall into noun or causal relationship answer classes (for example, the phrases `p53` and `reversing the CNS-depressant effects`). To combat this issue, we use the answer pruning approach of Biomedical Tag Probability Pruning, which ranks segments by average likelihood of the specific tag assigned to each word, eliminates segments not of the form {IOB_START, IOB_MID, ..., IOB_END}, and eliminates segments where there is not at least one word in the answer segment exclusively present in the biomedical vocabulary and not present in the GloVe embedding vocabulary. In addition, empirically, we find that restricting the answer segment to be a segment of size larger than 1 gives more relevant answer terms. As shown in Figure 4.4, this approach yields more terms that are biomedical terms and fall in the *Nouns* answer class. Specifically, phrases such as `QT interval` and `benzodiazepine overdose` are good biomedical phrases that could be potential answers. This approach is also beneficial in splitting answer segments. For example, without answer pruning, the phrase `QT interval is shortened` was one answer. However, because we allow segments to end with the IOB_END tag, the Biomedical Tag Probability Pruning approach was able to prune the part of the answer that was a trailing verb, yielding a better high-quality answer. However, this approach also has its limitations. For example, a few generated answers might still span across sentence boundaries and include partial phrases that are incoherent such as a begin parentheses with no matching end parentheses.

To get a quantitative evaluation of the generated answers, we manually annotate generated answers as one of the answer classes or as a special *irrelevant* answer class. We sample 50 generated answers and find the following distribution:

1. *Nouns*: 30 of the 50 answers are in this answer class. An example generated answer (bolded) of this form in the dataset is: ***Benzodiazepine (BZD)*** *overdose (OD) continues to cause significant morbidity and mortality in the UK.*

Paragraph Snippets (answers are bolded)	
Without Answer Pruning	Biomedical Tag Probability Pruning
QT interval is shortened when **QTc is less than 350 ms** (**1st degree of shortening**) ...	**QT interval** is shortened when QTc is less than 350 ms (1st degree of shortening) ...
The major tumour suppressor protein , **p53** , is **one** of **the most** well **- studied proteins** in **cell biology** ...	**The major tumour suppressor protein** , p53 , is one of the most well - studied proteins in cell biology ...
... Thus , **flumazenil provides a safe and effective** means of **attenuating** or **reversing the CNS - depressant effects** of benzodiazepines whenever indicated , e . g . following benzodiazepine - induced **general anaesthesia** , **conscious sedation , or** after **benzodiazepine overdose** , either alone or in combination with other **agents** ...	... Thus , flumazenil provides a safe and effective means of attenuating or reversing the CNS - depressant effects of benzodiazepines whenever indicated , e . g . following benzodiazepine - induced general anaesthesia , conscious sedation , or after **benzodiazepine overdose** , either alone or in combination with other agents ...

Table 4.4: Identified Answers with and without Biomedical Tag Probability Pruning

2. *Adjectives*: 2 of the 50 answers are in this answer class. An example generated answer (bolded) of this form is: *Caspases are the ultimate executors of **the apoptotic programmed** cell death pathway.*

3. *Casual relationship*: 1 of the 50 answers is in this answer class. An example generated answer (bolded) of this form is: *The emerging clinical implication , supported by recent epidemiological studies, is that βAR - blockers and drugs interfering with RANKL signaling , such as Denosumab , could increase patient survival if used as adjuvant therapy to inhibit **both the early colonization of bone by metastatic breast cancer cells** and the initiation of the " vicious cycle " of bone destruction induced by these cells.* Note that this example answer is not perfect, since the word "both" is followed by only one reason, but we include it in this category nonetheless.

4. *Number*: 2 of the 50 answers is in this answer class. An example generated answer

(bolded) of this form is: *Flumazenil was administered to 80 patients in **4504 BZD***.

5. *Irrelevant*: 15 of the 50 answers were classified as phrases ill fit to be good answers. This class can be segmented into errors that exclude certain important words in an answer, errors where biomedical terms in a sentence that are not the object of the sentence are identified as answers, and answers include extraneous words such as trailing verbs in the identified answer.

This distribution over the answer categories is mostly similar to the distribution of the answers observed in the BioASQ dataset, with a key difference being the lack of casual relationship answers. As discussed earlier, this is expected due to the simple nature of this model.

4.2.3 Generated Questions

After generating answers in the BioASQ dataset from the answer generation module, the next step in the pipeline is to generate corresponding questions for each generated answer using the question generation module described in Section 3.3. Because a quantitative metric for evaluating a specific question is difficult to come up with, we qualitatively judge generated questions.

Our question generation model is trained for 20 epochs on the SQuAD dataset. Similar to the answer generation model, the hyperparameter of the number of epochs to train on SQuAD was crucial to ensuring high quality generated questions. This is likely because during training, the model overfits to the SQuAD data and questions. Additionally, since the word embeddings matrix is finetuned during training, the word embeddings would likely become more skewed towards the GloVe vocabulary, making evaluation on the BioASQ dataset tougher.

When evaluating the quality of questions, we observed that the quality of answers has a very strong correlation with the quality of questions. This is expected since the model

cannot generate a coherent sentence whose answer is a stream of unimportant words. Thus, we observe that given the answer classes mentioned in the previous section, the number of noisy or poor generated questions is lower bounded by the size of the *Irrelevant* answer class.

Another key observation for our question generation model was that the copy predictor in the latent predictor network dominated most of the words generated in a question. Keyword phrases such as the beginning question word such as `What` or `Which` were typically generated using the word generation predictor followed by a stream of copied words from the paragraph. This is encouraged during training as well, because the loss function for training only measures a cross entropy loss on the outputted question tokens against the ground truth, which is easily optimized by copying words from the input paragraph. This implies that the size of the input context paragraph would strongly influence the quality of the generated question, which is what we observed empirically. When the size of the input paragraph was too large, the model was more likely to output incoherent sentences. This is likely because the copy predictor is outputting a probability distribution over a larger amount of words (the size of the context paragraph), so it is less likely to have a relevant exact phrase copied word for word into the generated question. To solve this, for any answer segment in the context paragraph, we feed in only the surrounding two sentences to the question generation model rather than the entire context paragraph, which we find significantly improves question coherence.

Table 4.5 and Table 4.6 show examples of successes and failures for generated questions. In this table, we only show one or two sentences around the answer segment for readability. The successes of the question generation model are usually dominated by the copy predictor. The model clearly learns a syntactic grammar structure for the questions, as seen by the questions in Table 4.5. Additionally, the model learns to identify actors in sentences as words that should be copied to the generated question. For example, for the question `The Type III is associated with homozygosity for what?`, the model learns to only copy the noun phrases `Type III` and `homozygosity` to the generated question instead of the full phrase, while simultaneously maintaining coherency. However, these questions are still very

simple and typically involve copying large phrases from the input paragraph. To generate questions that approach the quality of questions in the BioASQ dataset, we believe the model must capture long-term dependencies between biomedical entities and incorporate domain knowledge into the model. We hypothesize that because the model is trained on SQuAD data which is of a lesser difficulty than the biomedical question answering domain, the model only generates questions that would be regarded as trivial compared to the difficulty of questions in the BioASQ dataset.

In Table 4.6, we explore the various limitations of the question generation model. A large theme of the generated questions is the difficulty of the model to capture long-term dependencies in the context paragraph. This can be seen very clearly in the generated question: `What is shortened when QTc is less than QTc?`. The last word should be 350ms instead, and this small error renders the question meaningless. Manually investigating questions reveals a common trend of phrases repeated indefinitely. For example, the phrase `the paternal of the of the deleted of the deleted of the paternally paternal` appears in one of the generated questions, which highlights the inability of the model to capture long-term dependences in the paragraph. To investigate this further, we examined the output of several generated questions on the SQuAD test set during training. We found that, in this dataset, the issue of long-term dependencies is much less prevalent. This suggests that the LSTM in the question generation module might be overfitting to the SQuAD dataset. This also highlights the importance of the approach discussed earlier of using an embedding low-rank approximation to project the word embedding distributions of the SQuAD and BioASQ dataset into a common space. In some of the generated questions, the question generated is not relevant purely because the answer phrase is irrelevant, which we regard as a failure of the answer generation module.

Paragraph Snippets (answers bolded)	Generated Question
... Mutations of the TYR gene lead to Oculocutaneous Albinism type 1 (OCA1) , the most common type of albinism in humans (OMIM accession number 203100) . In humans mutations in the TYR gene are associated with type 1 **oculocutaneous albinism** (OCA1) that leads to reduced or absent pigmentation of skin , hair and eye ...	What are mutations in the TYR gene associated with ?
... Necrobiosis lipoidica diabeticorum is a rare disease of unclear etiology , that occurs in about 1 % of diabetic patients . **Necrobiosis lipoidica diabeticorum** (NLD) is a rare , granulomatous inflammatory skin disease of unknown origin , sometimes associated with diabetes mellitus ...	What is NLD ?
... Commotio cordis is a rare type of blunt cardiac injury in which low impact chest trauma causes sudden cardiac arrest , usually occurs from being struck by a projectile during sports . Commotio cordis due to blunt trauma to **the precordium** is a rare cause of death in young athletes , occurring less frequently than all of the other athletics - related deaths ...	What is the rare cause of death in young sports ?
... **cubense (Foc1**) which caused widespread Panama disease during the first half of the 20th century but is susceptible to Tropical Race 4 of Foc (Foc TR4) which is threatening world banana production ...	What caused the Panama disease during the 20th century ?
... Type III hyperlipoproteinemia is characterized by delayed chylomicron and VLDL remnant catabolism and is associated with homozygosity for the apoE - 2 allele ...	The Type III is associated with homozygosity for what ?

Table 4.5: Generated Questions Successes

Paragraph Snippets (answers bolded)	Generated Question
... Numerous other drugs are in earlier stages of development for HNSCC treatment , including novel **anti - EGFR mAbs** (MEHD7945A , necitumumab , and RO5083945) , small - molecule TKIs (vandetanib , icotinib , and CUDC - 101) , EGFR antisense , and drugs (temsirolimus , everolimus , OSI - 906 , dasatinib , and PX - 866) intended to overcome resistance to anti - EGFR agents ...	What is a term used for MEHD7945A , MEHD7945A , and other are of the novel named resistance ?
... This contrasts with the Prader - Willi syndrome (PWS) in which a similar deletion of the paternally contributed chromosome 15 is observed . However , in contrast to the paternal inheritance of the deleted chromosome 15 observed in the majority of **PWS patients** ...	What is the paternal of the of the deleted of the deleted of the paternally paternal contrast to PWS ?
... CMT4D disease is a severe autosomal recessive demyelinating neuropathy with extensive axonal loss leading to early disability , caused by mutations in **the N - myc downstream** regulated gene 1 (NDRG1) ...	What was the severe early location of the motor of molecular disease ?
... 70 - gene signature . The 70 - gene signature (MammaPrint) is a prognostic test used to guide **adjuvant treatment** decisions in patients with node - negative breast cancer . 70 - gene signature ...	What decisions is a prognostic test used to use in patients ?
... **QT interval** is shortened when QTc is less than 350 ms (1st degree of shortening) ...	What is shortened when QTc is less than QTc ?

Table 4.6: Generated Questions Failures

Approach	Exact Match Score	Mean Reciprocal Rank
SQuAD	17.5%	0.297
BioASQ	30%	0.349
SQuAD + BioASQ	35%	0.427
SQuAD + A_{gen} + Q_{gen}	12.5%	0.184
SQuAD + A_{gen} + A_p + Q_{gen}	12.5%	0.192
SQuAD + A_{gen} + A_p + Q_b	8.75%	0.176
SQuAD + A_{gen} + A_p + $Q_{gen-trunc}$	13.75%	0.216
SQuAD + BioASQ + A_{gen} + A_p + $Q_{gen-trunc}$	**35%**	**0.43**

Table 4.7: Core Transfer Learning Results on the BioASQ Dataset

4.2.4 Transfer Learning

Table 4.7 shows our main results for evaluating transfer learning from the SQuAD dataset to the BioASQ dataset. For pre-training, we train a FastQA model (using an existing implementation) on the SQuAD dataset for 12000 iterations [38]. After training, the model achieves a 70% test F1 score on the SQuAD dataset. We keep this model fixed and finetune on various datasets and evaluate performance on a held-out validation set from the BioASQ dataset during finetuning. Due to the small nature of the BioASQ dataset, this validation set consists of only 80 questions. Our evaluation measures are two-fold, and they are the same evaluation measures used for evaluating submissions to the BioASQ challenge [35]. First, since we are only considering factoid questions, we report Exact Match Score. An exact match occurs when the predicted answer is exactly the labeled answer in the dataset. The Exact Match Score is the percentage of exact matches on the validation dataset. The second evaluation measure is Mean Reciprocal Rank. The FastQA model outputs a list of answers ranked by the probability of each answer being correct. The reciprocal rank is 1 divided by the rank of the correct answer in the list of answers produced by the FastQA model. The mean reciprocal rank in the validation set is the average reciprocal rank over all answers in the validation set.

We now describe the various models described in Table 4.7. All models for the answer

generation and question generation module use a projected embedding space of size 250. The SQuAD approach is the FastQA model pre-trained on SQuAD and results are directly evaluated on the BioASQ dataset without any finetuning. The SQuAD + BioASQ is our baseline of pre-training on SQuAD and finetuning on the BioASQ dataset. The A_{gen} is the answer generation module described in Chapter 3, with only basic Binary Probability Pruning. The model $A_{gen} + A_p$ indicates the answer generation module with Biomedical Tag Probability Pruning The Q_{gen} model is the question generation module described in Chapter 3 trained on the $SQuAD$ dataset and evaluated on full paragraph snippets from the BioASQ dataset. The $Q_{gen-trunc}$ model is the same question generation model, but during evaluation, we only pass in as input the two sentences surrounding the identified answer. The Q_b model is a naive baseline model for question generation that does not use the question generation module described, but instead uses a rule-based system to generate questions. This system simply replaces an identified answer segment in a sentence by the word `what`. This baseline serves as an extreme case for the copy predictor in the question generation module. The BioASQ + A_{gen} + A_p + $Q_{gen-trunc}$ model is a model that is finetuned on the synthetic data along with the golden BioASQ training data. Specifically, every 4 minibatches during training, we sample a batch of synthetic data. This serves as a form of regularization to avoid overfitting on the synthetic data during training. For all experiments, the generated synthetic data is around 6 times larger than the BioASQ dataset, with around 3000 synthetic question-answer pairs on the BioASQ dataset.

We see minor improvements in performance due to some of the methods described in Chapter 3, specifically answer pruning and paragraph truncation. This improves mean reciprocal rank from 0.184 to 0.216. Additionally, we find that the question generation module still beats the naive question generation baseline of copying entire sentences from the input paragraph. This suggests that the question generation model is learning some additional information about which keywords belong in the question, which is promising for improving transfer.

We have two overall baselines for testing transfer, which are the SQuAD and SQuAD +

BioASQ approach described above. With the exception of the last model, all models were trained only on synthetic data, so for the method to yield better transfer, the performance of our data-augmented models would be between the SQuAD and SQuAD + BioASQ approach. Since the BioASQ data is the ground truth training data, we would not expect our synthetic data to be of higher quality than that. However, we find that training only on synthetic data performs worse overall than the SQuAD model. This strongly implies that the synthetic training data is unlike the distribution of the BioASQ training set, so the model is overfitting to the synthetic data during training. This can also be related to catastrophic forgetting, a commonly observed phenomenon in transfer learning tasks [16]. Catastrophic forgetting occurs when a model is trained for a specific task, asked to learn a new task by finetuning, and the model ends up forgetting the first task by performing worse on the source task after finetuning. In our example, catastrophic forgetting might result in an improved quality of answering questions that the BioASQ synthetic data contains, but the model might forget other nuances of the SQuaD dataset that allowed it to perform well on the BioASQ dataset. We hypothesize that even without catastrophic forgetting, the quality of the synthetic data is poor, as described in the preceding sections. This likely stems from the poor quality of questions generated by the question generation module. Recall that the number of irrelevant questions is likely bounded the number of irrelevant answers, which from a random sampling of 50 generated questions, we observed to be around 30%. This would imply that at least 30% of our training data consists of noisy questions and answers. In practice, we observed that up to 50% of the training data consists of noisy questions and answers due to additional mistakes by the question generation model that were discussed in Section 4.2.3. A model that learns to overfit on these questions will clearly have a problem generalizing to the validation set. Our best-performing model is the model trained on interleaved synthetic and true data batches. This model performs almost identically to the SQuAD + BioASQ model, and we believe the difference in mean reciprocal rank (0.43 vs 0.427) is not significant enough to make a case for improved transfer. It is expected that these two models are almost identical, since the synthetic data is seen rarely during training (once every 4 batches).

Chapter 5

Conclusion and Future Work

In this thesis, we explored an approach for generative data augmentation in the BioASQ dataset by generating synthetic question-answer pairs. The goal of this was to improve transfer learning on a model pre-trained on SQuAD by increasing the amount of data the model could utilize during finetuning. Using an approach derived from two-stage synthesis networks, we found that generating high-quality synthetic question-answer pairs was a very difficult task on the biomedical domain. Compared to domains such as the NewsQA domain, where two-stage synthesis networks have shown success in generative data augmentation, the overall quality of synthetic question-answer pairs in the biomedical domain is low.

We explored several arguments as to why the biomedical task is harder as well as some potential solutions. Firstly, because the vocabularies between the SQuAD dataset and BioASQ dataset are so different, we employed two different word embeddings, the GloVe embedding and Biomedical Word2Vec embedding. Simply concatenating these embeddings would result in a severe feature mismatch between training and testing, so we used a low-rank matrix completion algorithm to project both embeddings into a common subspace. Empirically, inspecting nearest neighbors in the embedding space of words shows that the low-rank matrix completion approach is a significant step towards merging the two embedding spaces. Future work in this vein could include jointly training an embedding space over the GloVe

and Biomedical vocabularies using Word2Vec. Additionally, a more in-depth exploration can be done into the theory of picking the rank of matrix that approximates the ambient space of the concatenated embeddings. These approaches could yield significant improvements for the quality of generated answers and questions, since the input to the model would be in a common space during both training and testing.

To improve answer generation, we considered two main strategies for answer extraction and pruning, Binary Probability Pruning, and Biomedical Tag Probability Pruning. We found that Biomedical Tag Probability Pruning significantly improves the quality of generated answers. These answers are still typically biomedical concepts or entities. Considering the answer classes in the BioASQ dataset, a future area of work would be to focus on generating answers from each of these classes individually. A separate answer generation model could be trained for each of these answer classes: Nouns, Adjectives, Casual Relationships, and Numbers. Then, we could sample from these answer generation models independently based on the observed distribution of answer classes in the BioASQ dataset. A key insight into why the model's performance deteriorates during finetuning is likely that the data distribution of the synthetic question-answer pairs is different from the data distribution of the ground truth questions and answers in the BioASQ dataset. Resolving the difference in distribution for generated answers is the first step in tackling this problem. Another area of future work is qualitatively comparing the output of the answer generation module with a baseline such as a Named Entity Recognition system.

With better generated answers, we believe the question generation module will simultaneously improve in quality. Many of the generated questions are meaningless purely because the answer span identified by the answer generation module is incomplete or irrelevant. Even for feasible answer spans, we think there is large scope for improvement in the quality of generated questions. One key problem is the inability of the question generation model to tackle long-term dependencies - this could be due to lack of LSTM hyperparameter tuning such as the size of the hidden layers. Additionally, the generated question distribution likely does not match the distribution of the questions in the BioASQ dataset. The question

generation module relies heavily on the copy predictor in the latent predictor network, so relevant questions are mostly copied from the input paragraph. Future work would be to find ways to artificially introduce question diversity, perhaps by introducing a regularization constraint in the loss function during training to bound the number of words copied from the input paragraph. Finally, during training for both the answer and question generation module, picking the number of epochs to train on the SQuAD dataset is a key parameter in determining the quality of generated answers and questions at test time on the BioASQ dataset. This is because the number of training epochs controls the degree of overfitting of the answer and question generation modules to the SQuAD dataset. Future work would include coming up with a quantitative evaluation of generated answers and questions to measure the degree of overfitting and more precisely pick the number of training epochs.

The task of transfer learning for biomedical question answering is a difficult task, complicated by the difficulty of the biomedical domain as well as the lack of labeled data. Introducing biomedical domain knowledge into the answer and question generation modules would likely yield the most improvements for transfer, since the model would be able to more precisely identify relationships between biomedical entities. A naive method for doing this would be to generate questions using the current question generation module and mapping biomedical entities and words to synonyms using a concept ontology. We believe that despite the negative transfer demonstrated by using synthetic question-answer pairs in the biomedical domain, the data augmentation for transfer learning problem formulation is still a very strong and promising direction of research. We hope that the pipeline proposed in this thesis is a step towards smarter data augmentation for biomedical question-answering.

References

[1] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks 5*, 2 (1994), 157–166.

[3] Bennett, J., Lanning, S., et al. The netflix prize.

[4] Bishop, C. M. Pattern recognition and machine learning.

[5] Candès, E. J., and Tao, T. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory 56*, 5 (2010), 2053–2080.

[6] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[8] Forney, G. D. The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (1973), 268–278.

[9] Golub, D., Huang, P.-S., He, X., and Deng, L. Two-stage synthesis networks for transfer learning in machine comprehension. *arXiv preprint arXiv:1706.09789* (2017).

[10] Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[11] Graves, A., Jaitly, N., and Mohamed, A.-r. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on* (2013), IEEE, pp. 273–278.

[12] Grosse, R. Exploding and vanishing gradients, 2017.

[13] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[14] Ioannis Pavlopoulos, A. K., and Androutsopoulos, I. Continuous space word vectors obtained by applying word2vec to abstracts of biomedical articles. *http://bioasq.lip6.fr/tools/BioASQword2vec/* (2014).

[15] Kalchbrenner, N., and Blunsom, P. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (2013), pp. 1700–1709.

[16] Kemker, R., Abitino, A., McClure, M., and Kanan, C. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072* (2017).

[17] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[19] Ling, W., Grefenstette, E., Hermann, K. M., Kočiskỳ, T., Senior, A., Wang, F., and Blunsom, P. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744* (2016).

[20] Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[21] Mazumder, R., Hastie, T., and Tibshirani, R. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research 11*, Aug (2010), 2287–2322.

[22] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[23] Olah, C. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[24] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2010), 1345–1359.

[25] Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543.

[26] Perez, L., and Wang, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

[27] Quiza, R., and Davim, J. P. Computational methods and optimization. In *Machining of hard materials*. Springer, 2011, pp. 177–208.

[28] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[29] Ramshaw, L. A., and Marcus, M. P. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*. Springer, 1999, pp. 157–176.

[30] Schuster, M., and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing 45*, 11 (1997), 2673–2681.

[31] Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603* (2016).

[32] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[33] Sutskever, I., Martens, J., and Hinton, G. E. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), pp. 1017–1024.

[34] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112.

[35] Tsatsaronis, G., Balikas, G., Malakasiotis, P., Partalas, I., Zschunke, M., Alvers, M. R., Weissenborn, D., Krithara, A., Petridis, S., Polychronopoulos, D., et al. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC bioinformatics 16*, 1 (2015), 138.

[36] Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems* (2015), pp. 2692–2700.

[37] Wang, C., Yang, H., Bartz, C., and Meinel, C. Image captioning with deep bidirectional lstms. In *Proceedings of the 2016 ACM on Multimedia Conference* (2016), ACM, pp. 988–997.

[38] Weissenborn, D., Wiese, G., and Seiffe, L. Fastqa: A simple and efficient neural architecture for question answering. *arXiv preprint arXiv:1703.04816* (2017).

[39] Wiese, G., Weissenborn, D., and Neves, M. Neural domain adaptation for biomedical question answering. *arXiv preprint arXiv:1706.03610* (2017).