Who Let the Dogs Out!? A Neural Approach to Question Generation

Jaspreet Kaur Bhamra
jbhamra@ucsd.edu
Prem Pathuri
ppathuri@ucsd.edu
Sai Komatineni
skomatin@ucsd.edu
Feiyu Yang
f8yang@ucsd.edu

March 17, 2022

Abstract

We address the task of Neural Question Generation (NQG) using the SQuAD dataset. We experiment with numerous encoder-decoder architectures such as seq-to-seq models as well as transformer based methods. We find that seq-to-seq models are able to capture common structure that questions may pose but have no ability to utilize relevant information from the passage. In comparison, transformer based models are able to produce structured and varied questions. We find that adding attention to seq-to-seq models worsens performance. Our results suggest that highly parameterized models are required for NQG tasks and that small-scale architectures are ineffective. Link to code: https://github.com/skomatin/cse251b_final

1 Introduction

Natural Language Processing (NLP) has rose to prominence in recent years. With the advent of large NLP systems trained on massive corpora, Natural Language Understanding (NLU) has risen in import as well. NLU refers to an NLP system's ability to understand the text it processes. The evaluation of systems' levels of understanding has become known as Machine Comprehension (MC).

Evaluating MC for NLP systems is executed in the same manner in which reading comprehension is tested in humans. Given a passage, a system with high levels of NLU should be able to produce relevant information when prompted. Such tasks can appear in multiple forms. Close-style tasks require systems to predict missing words in a given passage, an example of which is the CNN/Daily Mail dataset wherein a passage with numerous entities are given as input, and the system's job is to predict the missing

entity [7]. In comparison, Open-domain Question Answering (QA) involves producing an answer from a given set of passages or documents, where the answers lie within the documents. Our work addresses QA on the Stanford Question and Answer Dataset (SQuAD) [6].

An extension to the machine comprehension task is to be able to accurately predict whether a given question can be answered. This is important as systems should have the ability to abstain from answering if insufficient information is present. A further extension of this task is the ability to generate questions given a passage and answer, referred to as Neural Question Generation (NGQ). Two very useful applications of this task are i) fast generation of questions for games like Jeopardy ii) for educators to be able to quickly generate questions to quiz students from a pre-defined content. We claim that this task is more challenging than open domain QA as NGQ requires systems to produce a sequence of words which is not strictly contained within a passage.

A system that is able to solve this task must have a sound understanding of the passage as well as the relationship between an answer and the passage. With this information, systems must be able to creatively produce a variety of questions for the same answer-passage pair.

We focus on the SQuAD 1.1 dataset which contains passages extracted from Wikipedia alongside multiple questions per passage. Each question in this dataset has multiple answers, wherein each answer is a span of the passage. We invert the task posed by SQuAD by asking systems to produce plausible questions which may have prompted the answer given a passage.

2 Related Work

While there are many approaches to QA, most involve computing some contextual representation of both the question and the passage. A major advancement in open-domain QA came about as a result of BiDAF, a network which utilized Bidirectional Attention Flow [8]. BiDAF makes use of two contextual representations of the passage, produced by a ContextToQuery mechanism as well as a QueryToContext mechanism. The ContextToQuery (C2Q) mechanism uses the query to attend to query-relevant words within the context (ie. passage). In comparison, the QueryToContext (Q2C) mechanism, attends to the words within the context which are most similar to words within the query. In other words, C2Q allows a system to understand which parts of a question are relevant for each word and Q2C allows it to understand which words in each passage are relevant to the question. Lastly, this bidirectional query-aware contextual representation of the passage is fed into a decoder network along side the initial contextual embeddings. The authors of [8] do this in order to prevent information loss which can occur as a result of the attention mechanism wherein the passage is summarized heavily. Thus previous information is allowed to flow through the network and be combined with a bidirectional contextual passage representation, giving BiDAF its name.

While BiDAF had a major impact in QA systems, it did not address the task of unanswerable questions as BiDAF was released prior to the creation of SQuAD 2.0. In the literature, there are multiple paradigms of addressing the issue of unanswerable questions. These can be broken down into single-read or multi-read approaches.

Many single-read approaches solve this problem by adding an extra module whose task is to identify whether a question can be answered given a passage. The decision of this module is then used to output the prediction (ie. null string if there is no answer, or the span if an answer exists). These "verifier" modules are either used in the encoder or the decoder aspects of any system architecture.

Multi-read systems perform multiple passes through the passage in order to determine if a question can be answered. A simple solution is perform m reads of a passage, each time gaining new information until a decision is made. However, this approach is sensitive to the setting of m. ReasoNet [9] was introduced to solve this issue. ReasoNet turns the multi-read approach into reinforcement learning problem, wherein a separate network is trained to identify whether to read the passage another time, or if sufficient information has been captured to make a decision about whether the question can be answered. However this approach is expensive, as each question-passage pair may involve a variable number of reads, thus it is difficult to learn in batches.

Retrospective Reading [12] solves this issue of identifying unanswerable questions by using two parallel modules. It uses one module to perform an initial read and make some judgement regarding whether the question can be answered. A second module is run in parallel to produce an answer along with its confidence in the answer. Its confidence in the answer is combined with the first module's judgement to produce a measure of how well the question can be answered. The predicted span is produced as output if this quantity is greater than some threshold δ and a null-string is produced otherwise.

There are many works exploring the question of NGQ; both [11] and [2] develop a neural network to solve NGQ and demonstrate their performance on the SQuAD dataset. Both authors develop an attention-based encoder-decoder architecture. The encoder is a bi-directional LSTM that consists of a forward LSTM and a backward LSTM. The decoder is a uni-directional LSTM network. A vocabulary is first built with the passage and questions and they are then on-hot encoded to represent inputs as vectors. These inputs are then encoded using self-attention based transformer blocks. Wang et al. maintain a hidden representation of the input passage and proposes a decoder that takes this as input and uses an auto-regressive method to generate questions. The model is trained to minimize the negative log likelihood of the data under the model distribution. Kim et al. address the problem of the answer frequently appearing in the generated question by proposing a answer-independent decoder.

3 Dataset and Preprocessing

Since each passage can have multiple questions and each question can have multiple answers, we treated each unique combination of passage, question, and answer as a separate data point. We plan to address this redundancy (by repeating passages) in the future. Basic filtering was first performed on the dataset, eg: adding consistent spacing for each punctuation mark, adding ¡start¿ and ¡end¿ tags to specify the structure of the text. We also fixed the length of each passage, question, and answer to fixed lengths. We tokenized words using the pytorch tokenizer which creates a separate identity (number for each word).

The SQUAD dataset provides a training and development dataset of sizes 90,000 and 35,000 samples approximately, respectively. We split the development dataset into validation and testing sets such that the test had 10k samples and validation set had 25k samples. We used passages, questions, and answers from the train and validation set to build the vocabulary. We used a threshold to ensure that only words with a frequency above the threshold are considered for the vocabulary, so as to avoid words with poor representation.

3.1 Concise Representations

The SQuAD dataset is an open-domain QA dataset wherein the answer is a span of the passage. As such, we produce another version of the dataset where the passage is truncated on both ends to include a certain window W of words around the answer. That is, if a passage contains the answer in the middle of the passage, W words before the answer and W words after the answer will be included, and all other words from the passage will be ignored. This produces a shortened passage which is much more relevant to the answer. We experiment with using this method rather than using the full passage in addition to the answer. It can be seen that by using the concise representation, the answer may not be necessary as it is much more evident within the passage, thus when using the concise passages, we do not use the answer as an input. We find that performance is generally improved when using shortened passages, with focus on information that is pertinent to the answer.

4 Network Architectures

4.1 Baseline Architecture

We apply basic encoder-decoder architecture to our baseline models. Specifically, we use a bidiredctional LSTM as an encoder, whose output is passed to an LSTM decoder for question generation. The baseline has two different models as shown in figure 1 and 2.

Both models take passage and answer as input and concatenate them as a pair to pass through embedding layer. In the first mode, the output of embedding layer gets

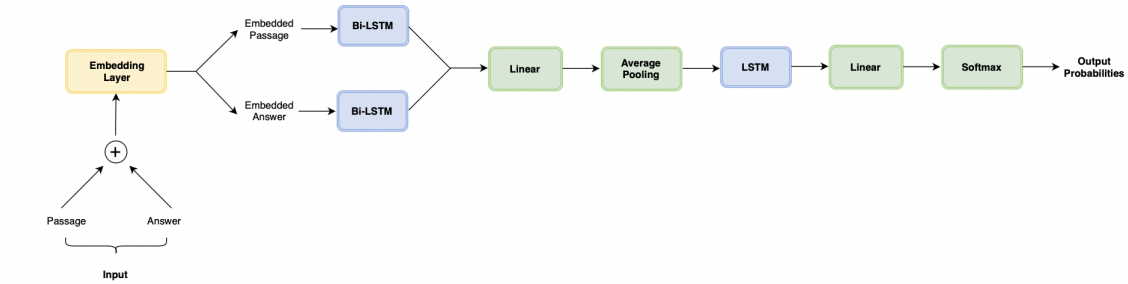

Figure 1: Baseline 1

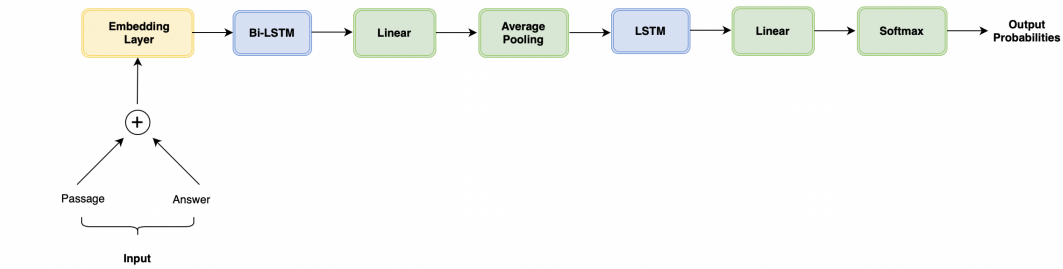

Figure 2: Baseline 2

separated and sent to two Bi-LSTM to be encoded. Then the outputs of two Bi-LSTMs are concatenated again. Now it's dimension is $N \times (P + A)$, where P, A and A are the passage, answer length and H is the hidden size. It then pass through a feed forward network to change the dimension to $N \times (P + A)$, where E is the embedding size. Then it pass through a average pooling layer to change the dimension to $N \times 1$. The output then goes to LSTM then full connection layer and softmax to predict the possible output. The output is also embedded to be sent to LSTM.

The second model architecture is very similar, except it doesn't separate the result after embedding layer. Instead, the concatenated passage and answer are encoded together. In addition to these two architectures, we also experimented with replacing the LSTM with GRU unit for the first Baseline architecture.

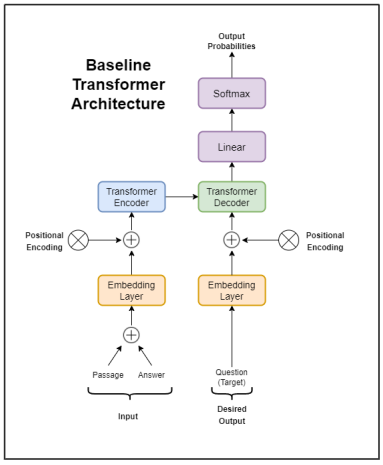

Figure 3: Transformer Architecture

4.2 Transformer-based Architectures

This section details transformer based architectures that were implemented in an attempt to fill in gaps that we thought our previous models had. We trained the models with the following hyperparameters (unless specified otherwise in the relevant subsections).
We used a learning rate of 0.0005 and used the Adam optimizer with a weight decay of 0.00001 and epsilon value of 1e-9 for numerical stability. We used a batch size of 8 due to computational constraints. In addition, the encoder and decoder blocks consisted of 6 encoder and decoder layers respectively with 5 attention heads in the multihead attention layers. The transformer model employed a feedforward block with 512 nodes and hidden layer with 512 nodes as well. Dropout of 0.2 was used for the sub-layers in the encoder and decoder blocks. The models were trained for 10 epochs, and we made use of early stopping.

Since our inputs and outputs have been padded in order to get consistent string length for the entire dataset, we used Cross Entropy as the loss such that the padding tokens are ignored during loss calculation. This was done in order to prevent rewarding the model for predicting padding tokens as the output.

4.2.1 Vanilla Transformer Encoder-Decoder (VTED)

We use a basic transformer encoder-decoder structure for our baseline transformer architecture. This closely follows the architecture described in [10]. A [passage, answer] pair forms the input of the model, while the associated question is the target output. The input pair is concatenated and passed through an embedding layer. The output of this layer is then sent through a positional encoding unit. This positional encoding is needed to force the model to make use of the order of the text sequence. These posi-

tional embeddings are generated following the paper by Vaswani et al. [10]

The embedded input and the positional encoding vectors are then concatenated and passed to a transformer encoder module. This block consists of multiple transformer encoder units (currently 6 units). Each transformer unit consists of a multihead attention layer and a feedforward layer with residual connections and normalization layers.

The output of the encoder, along with the desired target, are then passed to the transformer decoder block. Since the entire input sequence is being passed to the decoder, in order to prevent it from looking at future words during prediction, a word mask is employed. Masking is also used to mask padding tokens in both the input and output sequences in order to zero attention outputs.

Finally the output of the decoder block is sent through a linear layer (feed forward) and a softmax unit. The output is then un-embeddded to recover the word tokens.

4.2.2 VTED with pretrained GloVe Embeddings and Input Preprocessing

Using the same architecture as above, we replace the custom embedding layers in the model with pretrained layers with GloVe embeddings [5]. We choose the 50 dimensional embeddings trained on a corpus of 6B tokens. Along with this, the input to the encoder model is modified such that only part of the passage that contains the answer is passed to the model instead of a passage-answer pair. This is possible since the SQUAD dataset consists of (passage, question, answer) triplets such that the answer is always a substring of the passage. This is done to force the model to pay attention to the relevant words in the input to see if that improves the quality of predicted questions.

4.2.3 Dual Encoder - Single Decoder Transformer

Since the vanilla transformer encoder-decoder architecture seems to be unable to form mappings/relationships between the input passage-answer pair and the target question, we try to force it by passing the answer and the passage through separate transformer encoders. These inputs are then combined and passed to the decoder block which remains the same as the model in section 4.2.1.

4.2.4 Transformer Encoder-Decoder with Scheduled Sampling

In this architecture, we modify the best architecture so far by adding another decoding block as depicted in Fig 5 based on the work by Mihaylova et al. [4]. The 2 decoder blocks share parameters. The output of the first decoder block is combined with the actual target tokens (depicted as ”gold output” in the diagram and then passed to the second decoder at random points during training. This is to help the model recover from mistakes early on in the predicted string. This combination of prediction with the actual gold output is done after some warmup steps have passed via the normal teacher-forcing method used in the previous architectures.

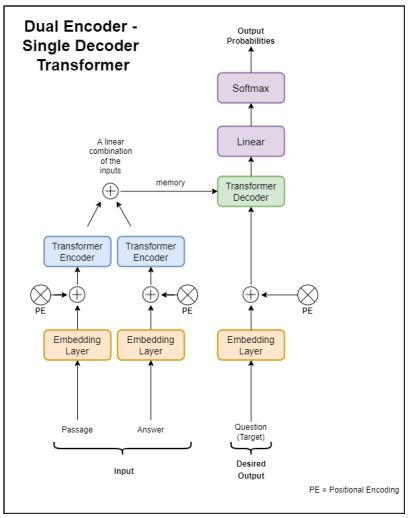

Figure 4: Dual Encoder - Single Decoder Transformer

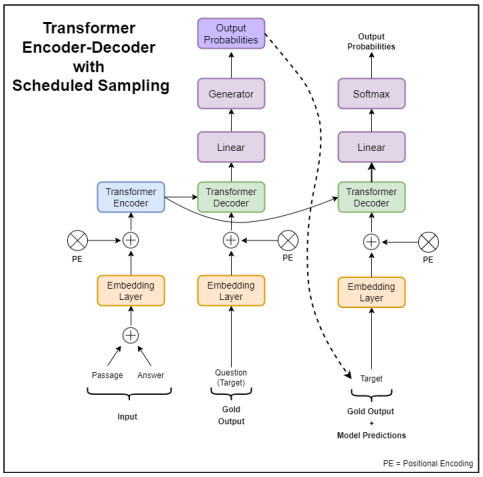

Figure 5: Transformer Encoder-Decoder with Scheduled Sampling

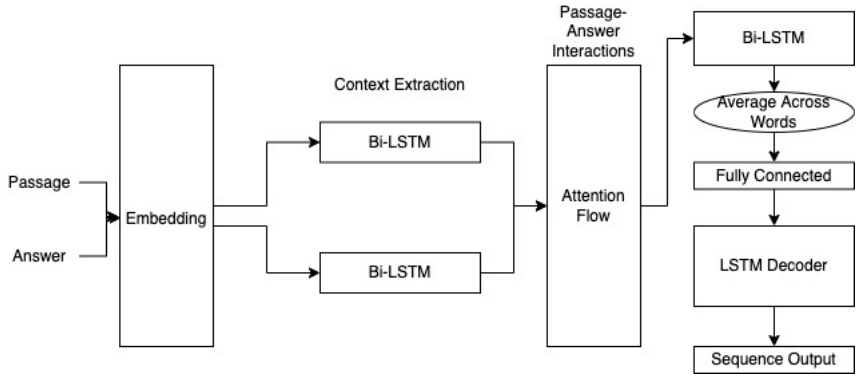

Figure 6: BiDAF LSTM Encoder-Decoder

4.3 Bi-Directional Attention Flow

We follow the same structure as our baseline architecture but use a bi-directional attention flow layer as described in [8]. The purpose of this layer is to utilize the answer to attend to parts of the passage, as well as use the passage to attend to parts of the answer. The details are described in [8]; in short, a similarity is computed between the passage and the answer and this similarity is used to extract representations of both the passage and the answer with the most important relevant information. The output of this attention flow layer is combined with previous embeddings to send to the next layer. This reduces the effects of early summarization. A depiction of this architecture can be seen in Figure 6.

As in all other models, the inputs are first embedded such that word representations can be extracted. These embedded inputs are sent through LSTM networks to extract context from the passage and answer separately. These contextualized embeddings are then sent through the Attention Flow mechanism which produces a representation of the passage, containing the passage-answer interactions. This representation is then fed through another bi-directional LSTM to produce features of the words within the passage which are relevant to the task. At this point, the output of this Bi-directional LSTM is of shape $N \times P \times 2H$, where P and H are the passage length and hidden size respectively. In order to produce an encoding which can be used to warm start the LSTM decoder, we must take the average across all words to produce an input representation of shape $N \times 1 \times 2H$ which is then sent through a fully connected layer to project the representation onto the embedding space. This encoding is then used to warm-start the LSTM decoder whose function is to produce a sequential output which we train to be a question that could have prompted the input answer, given the input passage.

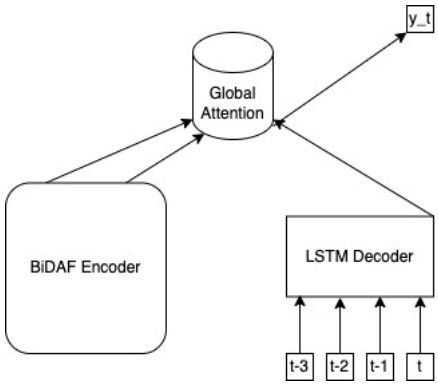

Figure 7: BiDAF Encoder-Decoder with Attention

4.3.1 BiDAF Experiments

The baseline BiDAF implementation involves an encoder with a BiDAF layer whose output is averaged across timesteps and sent through a linear layer to be fed in as the first input. This method is a simple encoder decoder and does not make any use of attention between the encoder and the decoder. Thus, we also train an Attentional BiDAF Encoder-Decoder which makes use of a Global Attention mechanism as described in [3]. This enables the decoder to attend to relevant information from the context extracted from the passage and the answer. In this scheme, a start token is fed to the decoder as the first input to prompt sequence generation. This can be seen in Figure 7.

In addition, we experiment with the use of pre-trained word embeddings. Specifically, we use the Stanford GloVe embeddings [5] trained on 6B tokens, with a vocabulary size of 400K. In order to reduce memory consumption on our GPUs, we choose the 50-dimensional embeddings.

For all of our experiments with BiDAF, we use a batch size of 60, a learning rate of 5e-4, a hidden size of 512, and 2 layers for both the encoder and the decoder.

4.4 Suggestor Network

Lastly, we implement a suggestor network, inspired by [1]. This suggestor network follows a similar encoder structure as our previous models. However during decoding, we create two separate modules. The first module acts as a classifier, whose task is to predict the first word. In order to make this feasible, we limit our dataset to those questions whose first word are among the top 10 most frequent starting words. We refer to these as *interrogatives*. The predicted interrogative is then passed to the decoder alongside the input token during question generation. Thus, the question generator should be guided to generate an appropriate question given the relationship between the answer and the passage. In order to make use of the full context during question generation, we

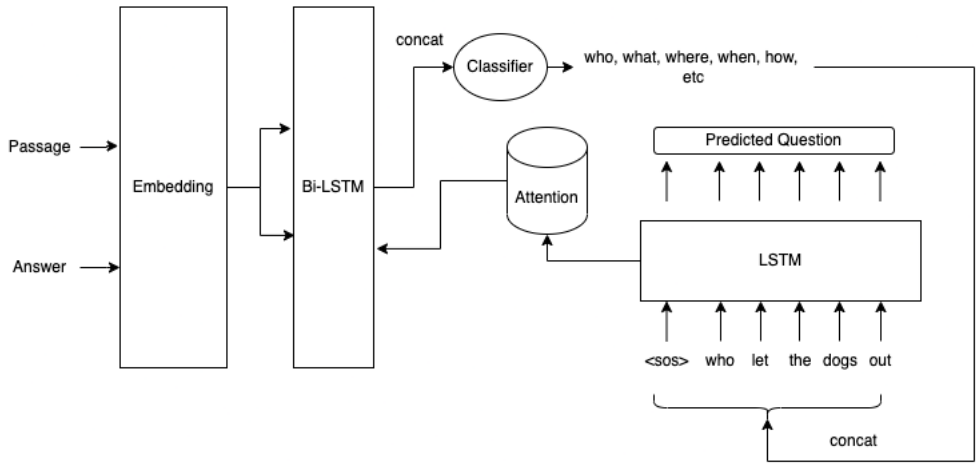

Figure 8: Suggestor Network

employ a predictive local attention mechanism as described in [3]. This architecture can be seen in Figure 8.

We train this suggestor network using a batch size of 128, a learning rate of 1e-4, a hidden size of 512, an encoder with 4 layers, a decoder with 2 layers, a window size W (for local attention) of 10, as well as a *p-size* of 512. The *p-size* refers to the dimensions of the matrix operation performed to compute the mean location of the Gaussian function used during predictive local attention. Details can be inferred from [3].

5 Results

5.1 Hyperparameters

Although slightly varied for each experiment, the hyperparameters we used are as follows: vocabulary_threshod: 2, batch size = 32, number of workers for data processing = 4, number of epochs = 5-10, learning rate = 5e-4, hidden layer size = 512, number of hidden layers = 2, embedding size for text = 300, temperature = 0.1, maximum question length = 60, maximum answer length = 46, maximum passage length = 851

5.2 Baseline Architectures

The vanilla encoder decoder architecture provides reasonable results with the loss values for the train and val sets ranging from 0.02 to 0.05. Figure 9 displays the loss

curves for this model. We only performed the tests over 3 epochs due to limited compute resources. However, in previous runs we observed minimal difference after epoch 3. We obtained a test loss of 0.021, BLEU1 score of 0.087, and BLEU4 score of 0.0. Examples of generated questions along with acutal question for this model:

```
   Prediction1:  what what the the the
Actual1:  at what time is the harvard yale rivalry set aside


   Prediction1:  what is the name of the name of the first
of the first of the first of the united of the first of the
united of the british empire
Actual1:  what type of education was assessed during this time


   Prediction1:  what is the name of the name of the first
of the first of the united of the united of the world of the
united
Actual1:  the adaptive immune system must distinguish between
what types of molecules
```

Figure 10 displays the loss curves for a very similar model with GRU units instead of LSTM units. We obtained a test loss of 0.018, BLEU 1 score of 0.077, BLEU4 score of 0.0058. Examples of generated questions along with actual questions for this model:

```
   Prediction1:  what is the name of the name of the first
of the first of the first of the united
Actual1:  what type of education was assessed during this time


   Prediction1:  what is the name of the name of the first
of the first of the first of the united of the first of the
united of the british empire
Actual1:  the adaptive immune system must distinguish between
what types of molecules


   Prediction1:  what is the name of the name of the first
of the first of the united of the united of the world of the
united
Actual1:  at what time is the harvard yale rivalry set aside
```

Figure 11 shows similar plots for the baseline model 2 (with LSTM units). We obtained a test loss of 0.02, BLEU 1 score of 0.06, BLEU4 score of 0.0. Examples of generated questions along with actual questions for this model are exactly the same as those for Baseline model 1 wth LSTM units.

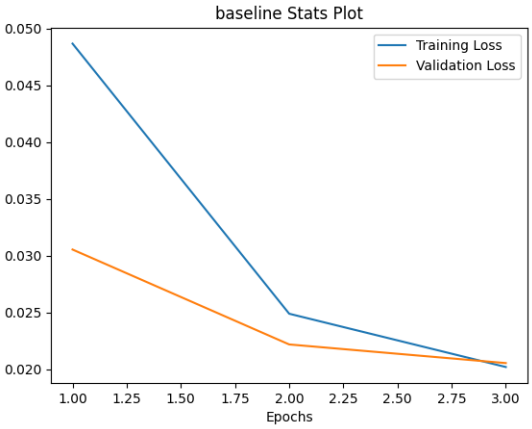

Figure 9: Loss Curves for Encoder-Decoder Baseline 1 with LSTM units

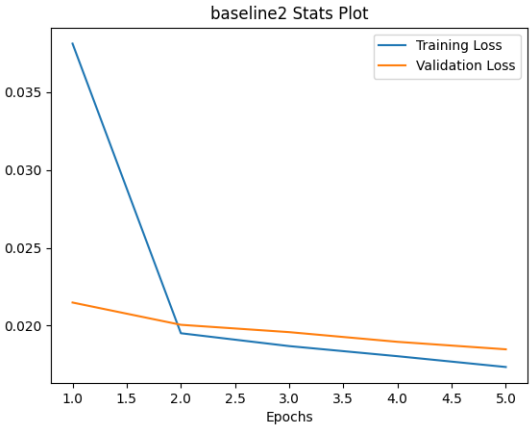

Figure 10: Loss Curves for Encoder-Decoder Baseline 1 with GRU units

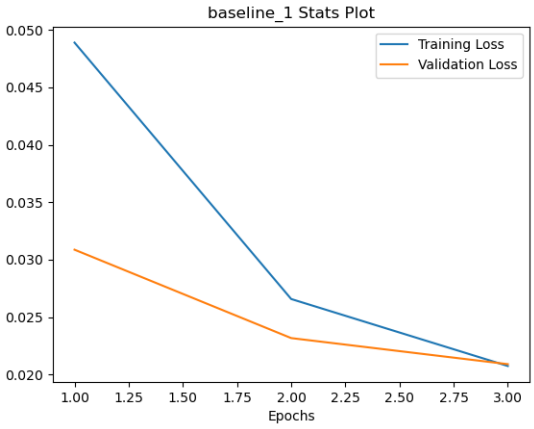

Figure 11: Loss Curves for Encoder-Decoder Baseline 2 with LSTM units

5.3 Transformer-based Architectures

The vanilla transformer encoder-decoder based architecture does not perform exceptionally well on the test set. We got a BLEU1 score of 0.16 and BLEU4 score of 0.0058 and a test loss of 0.076. A few sample predictions along with the true target:

```
Prediction1:  what was the name of the war that was the first
to be the city
Actual1:why did warsaw gain the title of the phoenix city

Prediction2:  what is the name of the process that is used
to make a capacitor
Actual2:  what are the two major subtypes of t cells

Prediction3:  what was the name of the region that was the
first part of the republic of the south of the south
Actual3:  when did the ottoman empire fall
```

As we see, the outputs are fairly structured with respect to grammar but there is very little variation in the kinds of questions being asked. A large population of the predicted questions consist of phrases like ”what is/was the name of”, ”united states”, ”church”. We believe that this could be an issue due to the model being unable to properly capture the relation between the passage-answer pair and the question. For this reason, we use multiple encoders in the second model proposed in section 4.2.3.

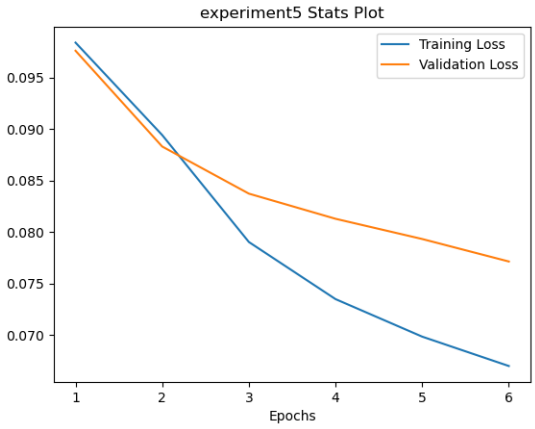

Figure 12: Loss Curves for Vanilla Transformer Encoder-Decoder

5.3.1 VTED with pretrained GloVe Embeddings and Input Preprocessing

When the embedding layer was replaced with pretrained layer using GloVe embeddings, we observed a drop in BLEU scores. The scores dropped to 0.079 and 0.003 respectively. But there was an observed increase in the variety of questions generated by the model. This model was trained on a subset of the dataset in order to reduce time taken to run experiments.

```
Prediction1:  what is the most common type of uranium
Actual1:roughly how much oxygen makes up the earth crust

Prediction2:  what is the term for the term used to describe
the antenna
Actual2:  a non deterministic turing machine has the ability
to capture what facet of useful analysis

Prediction3:  what was the name of the first european government
in the us
Actual3:  how much gold did victoria produce in the years of
1851 1860
```

5.3.2 Dual Encoder - Single Decoder Transformer

This model does not perform well. Here, we concatenate the encoded answer and passage together before passing them to the decoder. It is possible that a richer representation of the combined input is needed in order for the model to leverage the information being generated by the encoder. We ran our experiments with a small subset of 4000 training samples in order to test the architecture and got BLEU scores of 0.0003 and

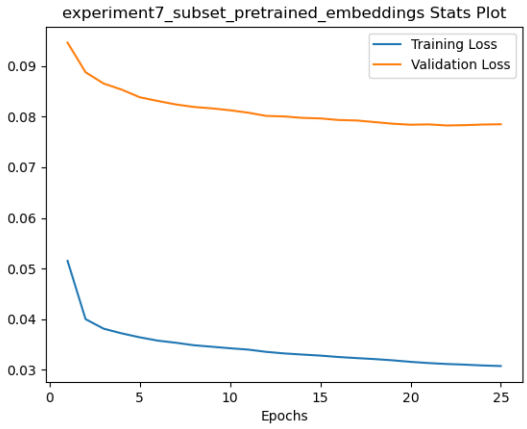

Figure 13: Loss Curves for Vanilla Transformer Encoder-Decoder with pretrained GloVe embeddings.

1.322-05 respectively. Thus, we conclude that simply an addition of an encoder to encode the passage and answer separately does not help to generate a richer encoding.

5.4 BiDAF

5.4.1 Baseline BiDAF

The baseline method of using BiDAF produced better results than many experiments involving BiDAF. Achieving a BLEU1 of 0.195 and a BLEU4 of 0.022, it outperformed many of our models in regards to BLEU.

However, this does not imply that the model was any better than the rest. In taking a look at a few predicted captions:

```
Prediction1:  what is the name of the that the system?
Actual1:what type of vote is required for the security council
to commit to military action in korea.

Prediction2:  what is the name of the that is the largest language
of the?
Actual2:  what are the main materials of cell walls of plants
and fungi?
```

As one can see this baseline model has learned to always start predictions with "what is the name of the". Increased training would not have changed this behavior as indicated by the loss curves in Figure 14. Model training was close to convergence and would not have changed drastically with more epochs. This model was trained with a

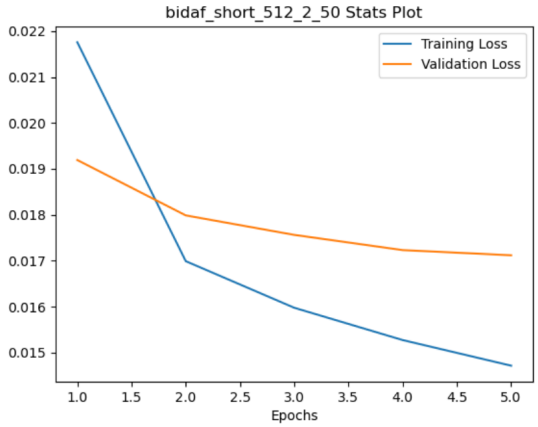

Figure 14: Baseline BiDAF Model

hidden size of 512 using 2-layer LSTMS, with 50 dimensional embeddings. Increasing the hidden size was not feasible as BiDAF requires large tensor concatenations that scales with the hidden size which would result in high GPU memory consumption.

5.4.2 Attentional BiDAF

We found that adding a Global Attention module decreased the performance of the BiDAF model. This scheme produced a BLEU1 and BLEU4 of 0.088 and 0.0 respectively. This is clearly evidenced by the questions it generated which had no structure in comparison to the baseline BiDAF model. An example of a generated question is shown below:

```
Prediction:  a many distinguishes what is as do island caused
story was the in
Actual:  what are women rewarded with for kissing the giant
phalluses?.
```

As one can see, the output is highly random and follows no structure. In comparison to the baseline, this model was unable to learn any patterns from the data.

5.4.3 BiDAF with GloVe

We find that using pretrained GloVe word embeddings did not have any positive impact on performance. While the BLEU1 and BLEU4 scores did reduce to 0.178 and 0.0168 respectively, this does not seem to be indicative of any change in model behavior. An example output is shown below:

```
Prediction:  what is the name of the that been been used to
```

```
be used?
Actual:  besides carnival , what other major cruise line parks
its cruise ships in southampton water?
```

As such, we find that the word embeddings themselves did not impact the baseline BiDAF model.

5.4.4 Attentional BiDAF with GloVe

The BiDAF model with Attention performed slightly better than its counterpart. Achieving BLEU1 and BLEU4 scores of 0.033 and 0.0 respectively, it is able to create the starting structure of questions, but trails off into random generation quickly. An example generated question is shown below:

```
Prediction:  what would ethnic serving are exhibition what
percentage shareholder snow orbit character notable caribbean
municipalities are bird diesel paid yers massaliote king an
accreditation traffic what emmett framework 21st city are carolingian
sphere czech legend braga art singapore mexico did can battle
aristotle goes vendee prime minister minister minister shell
1664 when
Actual:  what is one type of capacitor that has a specified
rating value for maximum ripple current ?
Predicted:  when did purchased cry 1907 whic exclusives
Actual:  what business did baldwin and mccurdy start after
the aea folded?
```

While the predictions are not similar to the true questions, the starting structure of a question is present in the predictions of this model. In comparison, an attentional BiDAF trained without GloVE embeddings produced random sequences of words throughout the question.

5.5 Suggestor Network

Our suggestor network achieved a BLEU1 of 0.103 and a BLEU4 of 0.0. While it had lower performance than our other networks, we noticed that it did have more variation in the starting words used in the generated questions. This can be seen below:

```
  Sample Question 1:  when did kowsar in 57th what was ghetto
new forney barcella in scotiabank who was phalle 63 in united
did reliefs what was voicing in witness who the 'standard advertisements
the perennial in
Sample Question 2:who was kleines istic how
```

6 Discussion and Evaluation

Baseline model 1 resulted in very obvious poor results where the predicted question was mostly "what what the the the". This implies that the model was unable to learn well using the teacher forcing technique as it is prioritizing outputting words with higher frequency rather than paying to the words that were predicted before it. It was interesting to see the GRU model perform better than the LSTM model. We speculate that this is due to the large number of repetitions of the passage in the dataset. Since the GRU does not have a memory unit, we expect it to perform worse on long passages and in this case since we are repeating the passages there is less of a need for an ability to parse long text structures. We also observed only a slight difference between the baseline model 1 and 2 meaning that splitting the embedded input before passing into a BiLSTM did not necessarily help. The fundamental problem here could be the concatenation of passage and answer because the answer is always a substring of the passage. An improvement to this model would be to completely separate the passage and answer at input level till the decoder.

We also noted that the validation loss curve is generally lower than the train loss curve for the baeline models and this could be because of the model memorizing the words in the passage rather than learning the sentence and paragraph structures.

In general we found that our BiDAF architecture worked best, when it did not make use of any attention mechanisms. It was suprising to see that attention hurt performance of our networks. This is further validated by the transformer networks' performance also being worse than our BiDAF network. We believe this is due to the patterns that our models have learned. We noticed that the most common questions were of the form `what is the name of the...` and our BiDAF model picked up on this and started every question with this phrase. As such, it scored higher than our other models. None of the other models were able to pick up significant patterns that would help in question generation. As such, by using mechanisms such as attention, or a suggestor network, we force the networks to introduce variation in their question generation, by referencing the context. With our simple models, this hurts performance in terms of BLEU scores. However, the captions that these models generate are not qualitatively any worse in that, even our best network architecture was not posing coherent and relevant questions.

As such, our main finding is that simple network architectures trained on a small-scale dataset may not be able to capture the nuanced interactions within passages and between passages and answers. In the future, we propose starting with pre-trained language models such as BERT or T5 that have already been proven to achieve state-of-the-art performance on many language tasks. Much of the success of vision systems stems from the use of transfer learning, which is missing in our work. Thus our main finding is that these simple networks, with no prior knowledge will be unable to generate relevant sequences of text in a NQG task.

7 Individual Contributions

7.1 Prem

I implemented the BiDAF and Suggestor network models and was responsible for testing their performance.

7.2 Sai

Setup the dataset, implemented its preprocessing. Also worked on implementation of baseline models and testing them. I also experimented with trainers with auto models.

7.3 Feiyu

Worked on the implementation and debug of two baseline models.

7.4 Jaspreet

Worked on transformer based models

References

[1] Junmo Kang, Haritz Puerto San Roman, and Sung-Hyon Myaeng. Let me know what to ask: Interrogative-word-aware question generation. *CoRR*, abs/1910.13794, 2019.

[2] Yanghoon Kim, Hwanhee Lee, Joongbo Shin, and Kyomin Jung. Improving neural question generation using answer separation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6602–6609, 2019.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[4] Tsvetomila Mihaylova and André F. T. Martins. Scheduled sampling for transformers. *CoRR*, abs/1906.07651, 2019.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[6] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[7] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017.

[8] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[9] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *CoRR*, abs/1609.05284, 2016.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[11] Bingning Wang, Xiaochuan Wang, Ting Tao, Qi Zhang, and Jingfang Xu. Neural question generation with answer pivot. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9138–9145, 2020.

[12] Zhuosheng Zhang, Junjie Yang, and Hai Zhao. Retrospective reader for machine reading comprehension. *CoRR*, abs/2001.09694, 2020.