# Building a QA System (IID SQuAD Track)

Stanford CS224N Default Project

**Aniruddh Ramrakhyani**
Department of Computer Science
Stanford University
anirram7@stanford.edu

## Abstract

The goal of the project is to build a deep-learning based question-answering (QA) system that performs well on the SQuAD dataset. For this, I implement three different models : (i) Enhance the provided baseline BiDAF [1] model with character embeddings, (ii) QANet [2] model and (iii) a 4-layer transformer encoder model. On the SQuAD dev set, adding character embeddings to basline BiDAF increases the EM and F1 scores to 59.6 and 63.12 respectively. The QANet model achieves EM score of 57.28 and F1 score of 60.59 while the transformer encoder model achieves EM score of 52.19 and F1 score of 52.19 on the dev set.

## 1 Key Information to include

- Mentor: Rachel Gardner
- External Collaborators (if you have any): N/A
- Sharing project: No

## 2 Introduction

The task of question-answering is considered to be an important benchmark in grading the performance of a NLP system. This is because the question answering task besides being practically useful in for example search engines, provides us a method to gauge as to how well a NLP system is able to understand and interpret the language. In the question answering task, the model is provided with two inputs : a question and a passage that may contain the answer. The model is required to predict the answer to the question given the passage or to indicate that there is no answer to the question in the passage. SQuAD is a popular question-answering dataset in which roughly half of the question have no answer. This makes the task harder for the model as it must now also correctly predict that there is no answer to the given question in the passage.

I implemented three different models for question-answering task:

- Baseline BiDAF model with character embedding : I added character embedding to the provided baseline BiDAF model. The char embeddings are enhanced using 1-D convolution followed by max pooling over the word width. This improved the F1 score from 58 to 63.14 and EM score from 55 to 59.6 and AvNA from 65 to 69.43 on the dev set.
- QANet model : The QANet model replaces the recurrence in the BiDAF model (due to LSTM) with self-attention. It learns the encoding of the question and passage using convolution and self-attention. This encoding is then fed to a context-query attention layer that is similar to the one used in BiDAF. The output of context-query attention layer is fed to a model encoder layer that contains convolutions and self-attention. The output of the model encoder is then used to predict the start and end positions of the answer in the passage. By replacing the recurrence with self-attention, the authors were able to achieve a speedup of 3x to 13x in training and 4x to 9x in inference. This model achieves F1, EM and AvNA scores of 60.59, 57.28, 67.52 respectively on the dev set.

- Transformer Encoder model: I implemented a 4-layer and 6-layer transformer encoder model with 6 and 10 attention heads respectively. The input to the model is fed in the format '<start><question><sep><passage>' where <start> is a special token that indicates the start of a question-answer sequence and <sep> is a special token that is used to separate the question from the passage. The output of the terminal encoder layer is fed to a feed-forward layer whose output is fed to two feedforward layers : one for the answer start position and the other for the answer end position. This models achieves a F1 score of 52.19, EM score of 52.19 and AvNA of 52.14 on the dev set.

  This project provides a comparison of the three different model architectures on the SQuAD task and thus is a useful replication of the models.

## 3 Related Work

Several models have been proposed over the past few years for the question-answering task targeting the SQuAD dataset. The BiDAF model proposed in 2017 uses bi-directional LSTM encoder on the character and word embeddings of both the passage and the question, which are then fed to Query to context and Context to query attention layer. The output of the attention layer is then fed to 2 bi-directional LSTM layers followed by a softmax to get the start and end probabilities of the answer. The authors achieve F1 score of 77 on the official SQuAD dataset. QANet, another recent work, improves upon the BiDAF architecture by replacing the LSTM encoder layers with depthwise separable convolution and self-attention. This change not only improves model runtime by more than 3x during training but also improves the F1 score to 82.7 on the official SQuAD dev set.

Transformer models, originally proposed in [3] have revolutionized the field of NLP. Since then a number of pre-training based models have been proposed (GPT [4], BERT [5], T5 [6], deep-self-attention [7]). These models are pre-trained on massive amounts of text data and then finetuned on the question-answering task using the SQuAD dataset. Variants of these pre-trained models have achieved state of the art performance on the SQuAD dataset that goes beyond human performance. While I did not use any kind of pre-training in my models, some of the hyper-parameter values for the transformer encoder model that I implemented were taken from GPT, BERT and the original transformer paper [3].

## 4 Approach

For this project, I implemented three different models. Details about the baseline model and architecture of the implemented models is described below.

### 4.1 Baseline

The baseline model was provided as a part of the IID track SQuAD task. This is a BiDAF model that uses only word embeddings. For details, please refer to the project handout or the BiDAF paper [1].

### 4.2 Char embeddings in BiDAF

I extended the provided baseline BiDAF model to process character embeddings along with word embeddings for both the passage and question. More specifically, I implemented the following algorithm to obtain character embedding for a word: pre-trained 64 dimensional character embeddings for all characters in the word are concatenated to create a tensor in $C \varepsilon R^{L \times 64}$ where L is the number of characters in the word. A 1-D convolution with 100 filters is applied on C to obtain $Q \varepsilon R^{L \times 100}$. Q is then max pooled over the word width to obtain character embedding $CE \varepsilon R^{L \times 100}$ for the word. The 300-dimensional GLoVE pre-trained word embeddings are concatenated with the character embedddings CE and then 1-D convolved to obtain final embedding for the word $E \varepsilon R^{100}$. E is then fed to a 2-layer highway encoder similar to that in the baseline model. The rest of the model is same as the baseline.

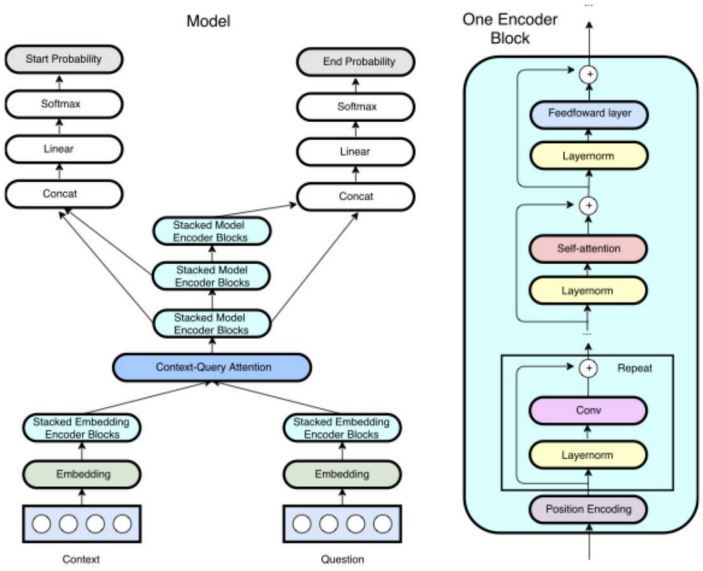


Figure 1: QANet model. Fig. taken from QANet paper [2].

### 4.3 QANet Model

QANet model (Fig 1) starts with an input embedding layer which encodes words and characters using the method described in 4.2. The same encoding layer is used for both the context and question to obtain embeddings $CE \varepsilon R^{CL \times 128}$ and $QE \varepsilon R^{QL \times 128}$ respectively where CL is the context length and QL is the question length. CE and QE are then fed to an embedding encoder block to obtain $C \varepsilon R^{CL \times 128}$ and $Q \varepsilon R^{QL \times 128}$. An encoder block (shown zoomed in on the right in Fig 1) consists of the following blocks: (i) a positional encoding layer that uses sine and cosine functions of different frequencies as used in the transformer paper [3], (ii) a stack of depthwise separable convolution layers [8], (iii) a self-attention layer (same as described in transformer paper [3]), and (iv) a feedforward layer. Each of convolution, self-attention and feedforward layer is preceded by a layernorm and followed by a residual connection. The depthwise separable convolution in encoder block uses 128 filters and kernel size of 7 while the self-attention layer uses 8 attention heads. For the embedding encoder layer, a stack of 4 convolutions is used in the encoder block. The dimension of output remain the same at each sub-layer in the encoder block ( $R^{L \times 128}$ where L is the sequence length).

C and Q are then fed to a Context-Query attention block which computes context-to-query and query-to-context attention. The context-query layer is the same as used in the baseline BiDAF model[1]. The output $A \varepsilon R^{CL \times 512}$ of the Context-query attention is then fed to a model encoder layer consisting of a 3-layer stack of 7 encoder blocks. The encoder blocks in the model encoder layer are the same as used in the embedding encoder layer except that the number of convolution layers is 2 and kernel size is 5. All 3 layers of the encoder stack in model encoder share weights. The outputs of first and second encoder stack is concatenated and the output of first and third encoder stack is concatenated to produce $S \varepsilon R^{CL \times 1024}$ and $E \varepsilon R^{CL \times 512}$ respectively. S and E are then fed to a linear layer followed by a softmax to get the answer start and answer end probabilites respectively. This final output layer is similar to the one used in baseline BiDAF model.

I implemented the entire QANet model including depthwise separable convolution, encoder block, self-attention, 1-D convolution for character encodings, feedforward layers and instantiating and joining correct number of layers in QANet model. The Context-Query attention layer, word embedding layer and Highway Encoder layers were taken from the baseline model. Sine-cosine positional

[1]This differs slightly from the QANet model in paper as that uses slightly different method to compute query to context attention

encoding implementation was used verbatim from the pytorch website. For training, adam optimizer with $\beta_1 = 0.8$ and $\beta_2 = 0.999$ was used. The learning rate was warmed up from 0 to 1e-3 exponentially in the first 1000 steps and remained constant after that. The model used dropout on word and character embedding with prob 0.1 and 0.05 respectively. A layer dropout is applied to each sub-layer l in the encoder block with prob $(0.1 \times l/L)$ where L is the last layer in stacked encoder blocks. In addition, the dropout rate between every two layers is 0.1. I implemented layer dropout and applied dropout at required places in the model. In addition, I instantiated adam optimizer from pytorch and implemented custom function for learning rate warmup.

### 4.4 Transformer Encoder Model

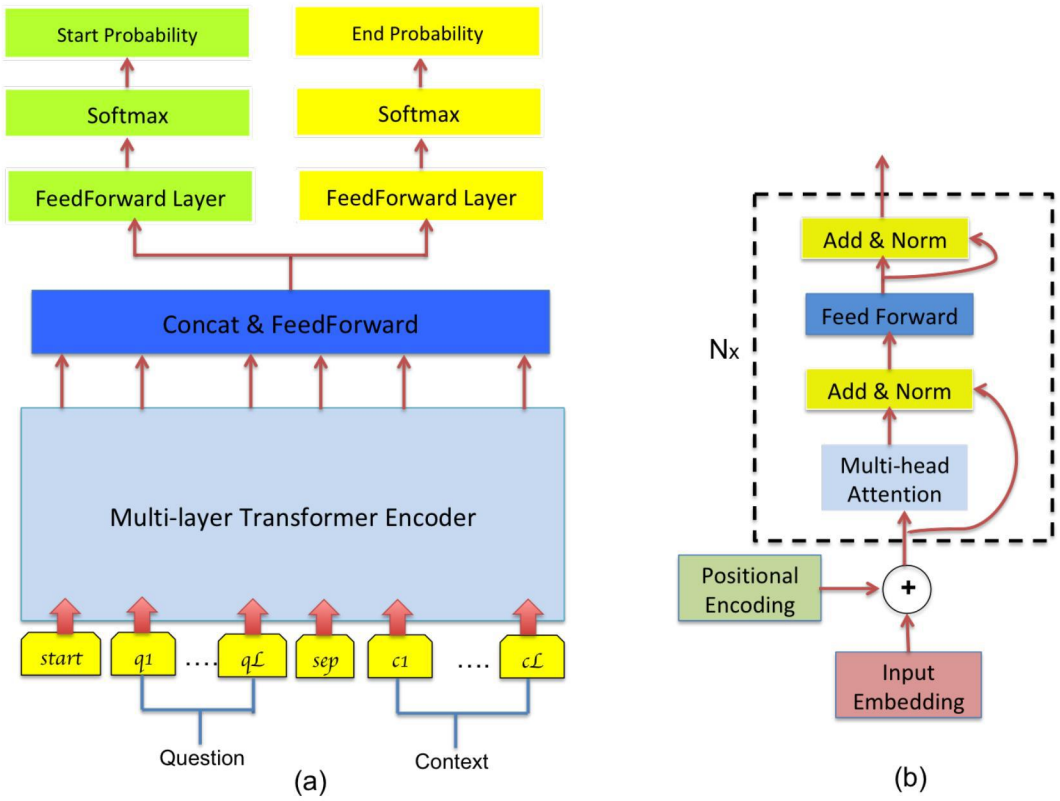


Figure 2: Transfomer Encoder model for SQuAD. (a) shows the overall model architecture where TE is a multi-layer transformer encoder block. (b) shows the architecture of a multi-layer transformer encoder where N is the number of layers.

I implemented a transformer encoder based model for SQuAD. Fig 2 (a) shows the overall architecture of the model. The input to the model is a sequence that is formed by concatenating question and context separated by a "sep" special token. Another special token "start" is prepended to this sequence to create the input for the model. The output of multi-layer transformer encoder at each position is concatenated to form a tensor $O \varepsilon R^{CL*512}$ (where CL is the context length) and fed to a single layer feedforward ($hidden\_size = maximum\_context\_length + 1$) with relu non-linearity. The output of this feedforward layer is fed to two different feedforward layers ($hidden\_size = maximum\_context\_length + 1$), one for answer start probability and the other for answer end probability followed by a softmax. The model predicts the probability of each word in the context being the start and end of the answer span. Finally, answer that maximizes $p_{start} * p_{end}$ such that start <= end is chosen. To allow the model to predict no-answer a special token Out-of-vocabulary is prepended to the context before the input to the model is created. This approach is similar to baseline BiDAF model.

The model uses 300 dimensional pre-trained Glove glove embeddings for words and 300-dimensional learnable positional encodings. For training, I tried both the adadelta optimizer provided with the

baseline code and adam optimizer. Following the advice in [9], the linear rate is warmed up from 1e-7 to 1e-3 exponentially in 4000 iterations and then annealed linearly to 3e-7. L2 weight decay is used with $\lambda$ of 0.01. The embedding, attention and residual dropout probability are all set to 0.01. I experimented with two configurations : 4-layer encoder with 6 attention heads and 6-layer encoder with 10 attention heads. Weights for feedforward layer in transformer block are initialized using xavier initialization [10] while the the linear layers at output use He initialization [10]. Positional encodings are initialized with normal distribution (with mean 0, var 1.0). The hidden size of feedforward layer in an encoder block is 2048. For the 6-layer model we use pre-trained word embeddings and allow the model to further train them.

I implemented the entire transformer encoder model[2] including self attention layer, learning rate warmup and decay, applying dropout, using correct weight initialization for layers, creating feedforward layers, trainable positional encodings, using Pytorch's adam optimizer and creating a collate function for the data loader to create input for the model in the form : "<start><question><sep><context>" and apply correct amount of padding.

# 5 Experiments

## 5.1 Data

SQuAD dataset provided as part of IID SQuAD Track is used for training and evaluation. The QANet model's input format is same as that of the provided baseline BiDAF model. In addition it also uses character embeddings for all words in both context and question. The input format for transformer encoder model is described in sec 5.3.2.

## 5.2 Experimental details

| Model Name | Batch Size | Num Epochs | Optimizer | Learning Rate |
|---|---|---|---|---|
| BiDAF with char embeddings | 64 | 30 | Adadelta | Held constant |
| QANet | 20 | 28 | Adam | Warmup to 1e-3 and then constant |
| Transformer Encoder (4 layers, 6 attention heads) | 80 | 30 | Adadelta | Held constant |
| Transformer Encoder (6 layers, 10 attention heads) | 64 | 30 | Adam | exponential warmup to 1e-3 and then linear decay to 1e-7 |

Table 1: Table showing training config for models.

F1 and EM scores along with AvNA (Answer vs No-answer) percentage are used to compare all the models. Table 5.2 lists the training config for the models. The config for the adam optimizer for QANet and 6 layer transformer encoder are in sec 4.3 and sec. 5.3.2 respectively. Model parameters found to perform best on the dev set were used for the test set.

## 5.3 Results

### 5.3.1 Model Comparison

Table 5.3.1 shows [3] the EM and F1 scores of the models on the dev and test set. Adding char embeddings to the BiDAF model increases the EM and F1 scores by 4.6 and 5.2 points respectively. AvNA also goes up 5%. Four and six layer transformer encoder models perform similarly achieving scores of 52.19 on the dev set. QANet model performs slightly worse than the BiDAF model with

[2]In the code, the classes are called TrnsformerDecoder. They are implementing transformer encoder model however.

[3]Due to limited tries allowed for the test set not all the models could be evaluated on the test set

| Model Name | Test set | | Dev Set | | |
|---|---|---|---|---|---|
| | F1 | EM | F1 | EM | AvNA |
| Baseline BiDAF | | | 58 | 55 | 65 |
| BiDAF with Char encoding | 62.3 | 59.6 | 63.14 | 59.60 | 69.43 |
| QANet | 59.62 | 56.23 | 60.59 | 57.28 | 67.52 |
| Transformer Encoder (4 layers, 6 attention heads) | | | 52.16 | 52.16 | 52.16 |
| Transformer Encoder (6 layers, 10 attention heads) | | | 52.19 | 52.19 | 52.14 |

Table 2: Table showing F1 and EM scores for the dev and test sets.

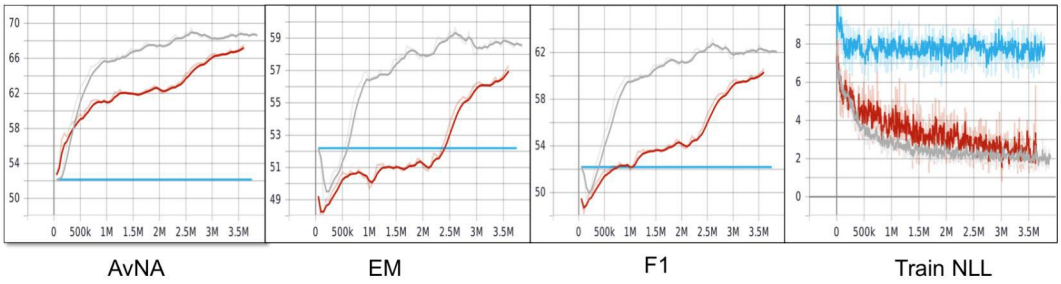


Figure 3: Dev set evaluation scores and training NLL for BiDAF with character embedding model (grey line), QANet model (red line) and 6 layer 10 attnetion heads transformer encoder model (blue line).

character embeddings achieving EM and F1 scores of 56.23 and 59.62 respectively on the test set. However, the performance of QANet model exceeds the performance of provided baseline model.

Increase in F1 and EM scores of BiDAF model after adding character embeddings were expected. However, it was surprising to see that QANet model didn't beat the BiDAF model with character embeddings. This may be because the QANet model was only trained for 28 epochs whereas in the QANet paper, it was trained for 40 epochs. In addition, the authors of QANet paper, used data generated from neural translation model in addition to the SQuAD dataset to train the model. Increasing the number of training epochs for my QANet implementation will likely increase the EM and F1 scores on the test set. This is reflected in Fig 3 where the F1, EM and AvNA on the dev set for the QANet model (red line) is increasing and approaching the respective scores for the BiDAF with character embedding model (grey line).

It was also surprising that the two transformer encoder models didn't perform as well on the SQuAD task. As I discuss in Sec 5.3.2, the 4-layer encoder model does have the capacity to learn all the training data. Maybe training it for even longer and with more data would have helped here since by pre-training BERT based models outperform other models like BiDAF and QANet.

### 5.3.2 Hyperparameter search for Transformer Encoder

Figure 4 shows dev set evaluation metrics and training NLL over the course of training for the 4-layer transformer encoder model with different dropout probability. Blue line is without any dropout applied. After 1M iterations, the training loss begins to drop and goes below 1 at 4M iterations. The model starts to overfit the training data which shows up as a drop in EM and F1 scores of the dev set. This serves as a quick sanity check for the implementation and for the fact that model has enough capacity to memorize the training dataset. Despite the F1 and EM scores falling, the AvNA however continues to rise indicating that the model is able to correctly learn when the answer may be present in the context. Towards the end F1 and EM scores start to rise as model training continues. Adding residual and embedding dropout of 0.1 (grey line) or 0.05, reduces the model capacity and the model

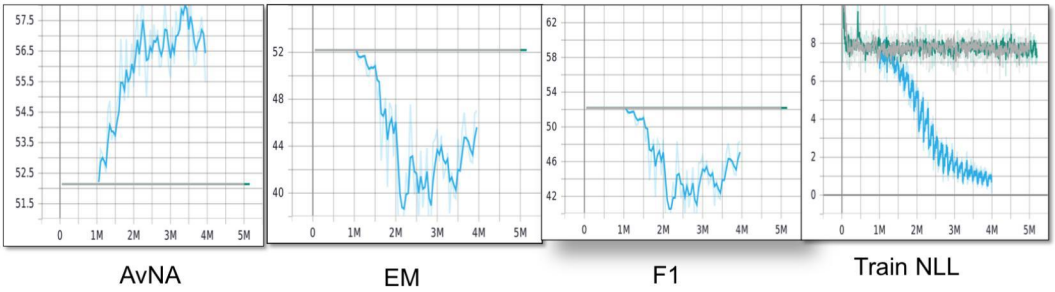


Figure 4: Dev set evaluation scores and training NLL for 4 layer 6 attention heads transformer encoder model. X-axis is no. of iterations while Y axis for each grpah is the respective metric. Blue line is with no dropout, grey line is with dropout of 0.1 and green line is with dropout of 0.05

is not able to learn much which is reflected by the saturated values of scores and training loss. Six layer transformer encoder model (blue line in Fig 3) also shows similar training and dev evaluation characteristics. However, using adam optimizer with xavier initialization in the 6-layer transformer encoder provides better training dynamics as unlike in the training loss in the 4 layer encoder model (which uses adadelta optimizer with normal weight initialization), the train loss doesn't spike up after 500K iterations, but instead decreases smoothly and then saturates.

## 6 Analysis

The 6-layer transformer encoder model and 4-layer transformer encoder model get stuck in a local minima where they mostly output N/A or No Answer to almost every question. The 4 layer transformer encoder model without any dropout, is able to learn and get better as training progresses. For example, after 3M iterations,the model output is:

**Question**: What is the Chinese name for the Yuan dynasty? **Context**: The Yuan dynasty (Chinese: ; pinyin: Yuán Cháo), officially the Great Yuan (Chinese: ; pinyin: Dà Yuán; Mongolian: Yehe Yuan Ulus[a]), was the empire or ruling dynasty of China established by Kublai Khan, leader of the Mongolian Borjigin clan. Although the Mongols had ruled territories including today's North China for decades, it was not until 1271 that Kublai Khan officially proclaimed the dynasty in the traditional Chinese style. His realm was, by this point, isolated from the other khanates and controlled most of present-day China and its surrounding areas, including modern Mongolia and Korea. It was the first foreign dynasty to rule all of China and lasted until 1368, after which its Genghisid rulers returned to their Mongolian homeland and continued to rule the Northern Yuan dynasty. Some of the Mongolian Emperors of the Yuan mastered the Chinese language, while others only used their native language (i.e. Mongolian) and the 'Phags-pa script.
**Answer**: Yuán Cháo
**Prediction**: ; pinyin

The model gets close and at 3.8M iteration it improves this answer to
**Prediction**: Yuan dynasty (Chinese: ; pinyin: Yuán Cháo)

The model is also able to recognize that the question is asking for a place.
**Question**: What country was under the control of Norman barons? **Context**: Subsequent to the Conquest, however, the Marches came completely under the dominance of William's most trusted Norman barons, including Bernard de Neufmarché, Roger of Montgomery in Shropshire and Hugh Lupus in Cheshire. These Normans began a long period of slow conquest during which almost all of Wales was at some point subject to Norman interference. Norman words, such as baron (barwn), first entered Welsh at that time.
**Answer**: Wales
**Prediction**: in Cheshire

The QANet model performs well on the dev set and is able to better determine whether answer exists in the passage. Sometimes its answers are more accurate than the provided answers in the dev set.

For example,

**Question**: When did Germany invade Poland and in doing so start World War II?
**Context**: After the German Invasion of Poland on 1 September 1939 began the Second World War, Warsaw was defended till September 27. Central Poland, including Warsaw, came under the rule of the General Government, a German Nazi colonial administration. All higher education institutions were immediately closed and Warsaw's entire Jewish population – several hundred thousand, some 30% of the city – herded into the Warsaw Ghetto. The city would become the centre of urban resistance to Nazi rule in occupied Europe. When the order came to annihilate the ghetto as part of Hitler's "Final Solution" on 19 April 1943, Jewish fighters launched the Warsaw Ghetto Uprising. Despite being heavily outgunned and outnumbered, the Ghetto held out for almost a month. When the fighting ended, almost all survivors were massacred, with only a few managing to escape or hide.
**Answer**: September 1939
**Prediction**: 1 September 1939

However, it is not able to get it right when the answer is more involved. For example, differentiating between adjectives "oldest" and "most famous":
**Question**: What is the oldest work of Norman art? **Context**: By far the most famous work of Norman art is the Bayeux Tapestry, which is not a tapestry but a work of embroidery. It was commissioned by Odo, the Bishop of Bayeux and first Earl of Kent, employing natives from Kent who were learned in the Nordic traditions imported in the previous half century by the Danish Vikings.
**Answer**: N/A
**Prediction**: Bayeux Tapestry

Or when the question is specifically asking about "generators":
**Question**: What percentage of electrical power in the United States is made by generators?
**Context**: The final major evolution of the steam engine design was the use of steam turbines starting in the late part of the 19th century. Steam turbines are generally more efficient than reciprocating piston type steam engines (for outputs above several hundred horsepower), have fewer moving parts, and provide rotary power directly instead of through a connecting rod system or similar means. Steam turbines virtually replaced reciprocating engines in electricity generating stations early in the 20th century, where their efficiency, higher speed appropriate to generator service, and smooth rotation were advantages. Today most electric power is provided by steam turbines. In the United States 90% of the electric power is produced in this way using a variety of heat sources. Steam turbines were extensively applied for propulsion of large ships throughout most of the 20th century.
**Answer**: N/A
**Prediction**: 90%

## 7 Conclusion

In this work, I created three models for the question-answering task on the SQuAD dataset and compared their performance. The models perform reasonably well and can be improved further by training them longer and with more data.

## References


[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[2] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[4] Radford Alec et al. Improving language understanding by generative pre-training.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[7] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention, 2018.

[8] Lukasz Kaiser, Aidan N. Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation, 2017.

[9] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.

[10] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, 13–18 Jul 2020.