

# Pretraining of Transformers on Question Answering without External Data

Stanford CS224N Default Project (IID Squad Track)

**Jeffrey Shen**

Department of Computer Science  
Stanford University  
jshen2@stanford.edu

## Abstract

Recent studies have found great success on a wide variety of natural language processing (NLP) tasks by leveraging Transformer architectures on large amounts of unlabeled data using a pretrain-finetune approach. In this paper, we investigate the performance of these techniques on the Stanford Question Answering Dataset (SQuAD) task in a computational and language resource-restricted setting. Without external data, we find that it is very difficult for Transformer architectures to meaningfully predict more than the majority class, even with RoBERTa or ELECTRA-style pretraining. By reducing the vocabulary size from 50k to 5k and then augmenting the set of non-answerable questions, we are able to improve the F1 score on the dev set from 52.17 to 56.86 and then to 60.46, respectively. However, the final model still performs worse than a baseline BiDAF model, suggesting that the ability of current state-of-the-art training objectives and model architectures to learn effectively from limited data is still severely lacking.

## 1 Key Information to include

- Mentor: Rachel Gardner
- External Collaborators (if you have any): N/A
- Sharing project: N/A

## 2 Introduction

Many recent approaches in NLP use large amounts of unlabeled text to build representations of language, which can be then applied to downstream tasks. Over time, many of these approaches leverage more powerful architectures to build more effective representations of language, from word vectors (e.g. GloVe; Pennington et al. [1]) to LSTMs [2] (e.g. ELMo; Peters et al. [3]) to the now ubiquitous Transformer architecture [4] (e.g. BERT; Devlin et al. [5]). Furthermore, in contrast with a handcrafted approach, these approaches can make relatively simple assumptions about the structure of language, e.g. that parts of text can be predicted given information about other parts of the text, and they can also use a single model architecture across many different NLP tasks. These approaches then obtain huge success on a wide variety of these tasks, such as SQuAD [6] or GLUE [7].

However, it is not clear whether such large amounts of external information should actually be required and how these approaches would perform without such external data. For example, in this paper, we look at the Stanford Question Answering Dataset version 2.0 (SQuAD 2.0) [6]. In SQuAD, since a context passage is given for each question, and the answer is contained in each passage, it is often the case that external knowledge about the subject is not necessary to produce the answer. However, it is not clear whether the 442 articles in the training set contains enough information about the English language to deduce structure and meaning.

In this project, we investigate whether recent Transformer-based pretraining approaches, such as RoBERTa and ELECTRA, can derive enough information to answer SQuAD 2.0 questions solely from the SQuAD 2.0 dataset itself. We find that despite making changes that should help ensure the SQuAD 2.0 task is not underspecified given the limited data, e.g. reducing the vocabulary size or augmenting the set of nonanswerable questions, a Transformer model is woefully unable to learn better than a baseline task-specific BiDAF model. The Transformer model mostly learns textual clues between the question and context, rather than using an understanding of the text. We suspect that better pretraining approaches are needed for these general architectures to more effectively learn about the structure of language.

### 3 Related Work

Pennington et al. [1] describe an approach (GloVe) that learn word vectors over a large corpus of text, such as Wikipedia, using co-occurrence statistics. Peters et al. [3] (ELMo) uses a bidirectional LSTM architecture to learn contextual word embeddings by performing language modeling to predict words given it’s surrounding context in a given text. These features can then be derived from any context and used in task-specific architectures to boost performance in several different NLP tasks. Later, Radford et al. [8] (GPT) found that by the recently introduced Transformer [4] decoder architecture could be used instead to perform unidirectional language modeling. Devlin et al. [5] (BERT) found that a single bidirectional Transformer encoder architecture could be pretrained with a masked language modeling (MLM) objective, i.e. asking a model to predict masked out tokens, and then finetuned, usually with a simple task-specific output layer, to achieve significant increases in performance on downstream NLP tasks.

A number of variations since BERT have been proposed, following a pretrain-finetune approach, such as XLNet proposed by Yang et al. [9], which uses a permutation-based autoregressive pretraining objective. However, it is not fully clear exactly how much of these gains can be attributed solely due to additional computational and language resources, as RoBERTa, a replication study of BERT by Liu et al. [10], showed that BERT, with simply more data and pretraining time, in addition to some minor hyperparameter tuning, could beat all of the approaches prior to the introduction of RoBERTa. Since then, several approaches that perform better than RoBERTa, even with similar or fewer amounts of training time. ALBERT, proposed by Lan et al. [11], uses a factorized embedding and shared parameters between layers with a more difficult sentence ordering prediction objective. ELECTRA, introduced by [12], only uses a different pretraining objective, by jointly training a small MLM generator to produce inputs for a discriminator model, which can perform comparably to RoBERTa using less than 1/4 the compute, as it can learn from each token rather than just the masked ones.

### 4 Approach

Aside from the initial starter code for this project<sup>1</sup>, we implement in PyTorch all the approaches listed essentially from scratch, using external code as a reference only. We make the code available on Github.<sup>2</sup>

#### 4.1 Model Architectures

**BiDAF.** As an initial baseline, we use a provided implementation of the BiDAF architecture [13]. The provided BiDAF implementation uses only pretrained GLoVe word embeddings instead of character-level embeddings.

**Transformer.** For the rest of our approach, we reimplement a Transformer encoder architecture [4]. For our PyTorch implementation, we didn’t use the library’s provided Transformer layers (but did use the MultiheadAttention module), and we try to follow the implementation details used by Liu et al. [10] for RoBERTa as closely as possible, using open-source code<sup>3</sup> as a reference. In particular, we use learned positional embeddings and GELU activation functions [14]. To obtain outputs for SQuAD, we use a single linear layer with one start and one end logit as the question answering

<sup>1</sup><https://github.com/minggg/squad>

<sup>2</sup><https://github.com/jeffdshen/squad>

<sup>3</sup><https://github.com/pytorch/fairseq>

head. In contrast with the standard RoBERTa implementation, we perform layer normalization before computing the residual connections, as many approaches have found it leads to more stable training, e.g. Xiong et al. [15].

## 4.2 Pretraining Objectives

**Masked Language Modeling (MLM).** We also reimplement a masked language modeling (MLM) objective [5]. We again try to follow the details from Liu et al. [10] as closely as possible. In particular, we split our randomly shuffled text samples into input blocks of 512 tokens, regardless of text boundaries, and add special classification and separator tokens. The context samples show up once per instance it is used in SQuAD. We then mask tokens individually and ask the model to predict all the masked tokens. We mask 15% of the tokens, with a 10% chance of leaving the token unchanged, a 10% chance of masking with a random token, and otherwise masking with a special mask token. For this objective, we use a language modeling head with an additional feedforward network followed by a linear projection using the same weights as the input embedding.

**ELECTRA-style MLM.** We reimplement an ELECTRA-style MLM objective, which is similar to the All-Tokens MLM objective that Clark et al. [12] introduce to compare their main approach against, as they found that pretraining with a loss on all tokens accounted for most of their improvement. That is, on each MLM step, we sample  $m = 3$  outputs from the MLM pretraining objective, selected randomly using the softmax output where the input is divided by a sample temperature  $\tau = 2.0$ , and then train the model to predict all the original tokens. We increase the mask probability to 25%. All the losses are averaged together, including the MLM loss, with a weight of  $\lambda = 16$  for each All-Tokens MLM loss, so that the total All-Tokens MLM loss has approximately the same weighting against the MLM loss as the ELECTRA discriminator weighting in Clark et al. [12].

## 4.3 Text Encoding

**BPE.** We reimplement a byte-level byte pair encoding (BPE) [16], similar to Liu et al. [10]. Our implementation of BPE in Python splits text into words based only on whitespace, appending the space ASCII character as an end of word token. Then the most frequent bytes are merged repeatedly until we obtain a vocabulary of 5k tokens.

## 4.4 Data Augmentation

**Question MLM.** Since some contexts do not have any unanswerable questions, we augment the original set of questions by first finetuning the MLM objective on only the SQuAD training set questions for a fixed number of epochs, before then sampling 2 MLM outputs per question with a sample temperature  $\tau = 1.0$ . Note that for augmentation, we do not concatenate questions together for input blocks, as in the original MLM objective. Since perturbing a question should result in a non-answerable question, we assign each augmented question a non-answerable label, unless it is identical to any original question with the same context, in which case we remove it.

# 5 Experiments

## 5.1 Setup

**Data.** We use the provided train, development, test split of SQuAD 2.0 [6], where the development and test set split together compose the official development set. Note that we do not use any external data, using only the provided train set for pretraining.

**Evaluation method.** We use the best F1 score on the provided SQuAD dev set to evaluate different approaches that we take. We denote the exact match score as EM.

**Experimental details.** For the transformer models, we generally default to the RoBERTa base settings (12 layers, 768 hidden size, 3072 FF size, 12 attention heads, 0.1 dropout). We use the Adam optimizer with  $\epsilon = 1e-6$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.98$ , unless we specify that we use Adam’s default settings, and an inverse square root learning rate schedule  $1r/\sqrt{n}$  with 10k linear warmup optimizer steps. This may be a different formulation than other papers, so as an example, for 10k warmup steps, the learning rate increases during the warmup period until it hits the peak learning rate of  $1r/100$ ,

and will later halve at 40k steps. For finetuning, we reduce the warmup steps to 1300. For each run, we select the best model on the dev set during training, and may also stop the run early manually.

## 5.2 GloVe Embeddings

We first evaluate the effect of GloVe word embeddings on both the baseline BiDAF model and a transformer model. We use the same word tokenization for all models, but either use a pretrained GloVe word embedding, or a learned word embedding. For the Transformer model, we adapt the 300 dimension GloVe word embedding by adding a learned linear layer.

We tried a number of different hyperparameter settings, of which we select a few to report in Table 1. In this section, we use a linear learning rate decay with a warmup ratio of 0.06 for each of the Transformer models (except the one with the pretrained GloVe word embedding, where we use an inverse learning rate decay with a warmup ratio of 0.12). Furthermore, all learning rates for the Transformer model in this section are reported as the peak learning rate, rather than the initial learning rate.

We manually stopped most of the Transformer models early, and none of them were able to reach an F1 score better than 52.17, which for this dev set is approximately the same as predicting every question as non-answerable. Of particular note, the Transformer model with 12 layers hit the lowest F1, which was 33.97 when it was stopped. All the BiDAF models, however, even without pretrained GloVe word embeddings, were able to perform better than just guessing the majority class. In particular, the baseline BiDAF model obtains an F1 score of 60.79.

| <b>BiDAF</b>           | F1    | 1M F1 | Epoch/Max | LR    | Batch Size | EMA decay |
|------------------------|-------|-------|-----------|-------|------------|-----------|
| GloVe*                 | 60.79 | 57.17 | 30/30     | 0.5   | 64         | 0.999     |
| GloVe (default Adam)   | 60.38 | 57.14 | 30/30     | 0.001 | 256        | 0.0       |
| Learned (default Adam) | 57.98 | 53.84 | 30/30     | 0.001 | 256        | 0.0       |
| <b>Transformer</b>     | F1    | 1M F1 | Epoch/Max | LR    | Batch Size | Layers    |
| GloVe                  | 52.17 | 45.88 | 9/30      | 5e-4  | 192        | 4         |
| Learned (default Adam) | 52.17 | 42.32 | 29/30     | 4e-4  | 192        | 4         |
| Learned                | 52.17 | 38.25 | 18/30     | 2e-4  | 192        | 4         |
| Learned (base)         | 52.17 | 39.26 | 12/30     | 5e-5  | 48         | 12        |

Table 1: SQuAD 2.0 F1 scores for models based on GloVe word embeddings. To better compare some of these models, we also report the F1 score at 1M examples (approximately 7.7 epochs), as some of them perform worse over the course of training.

## 5.3 Pre-norm RoBERTa

For RoBERTa pretraining, we first tried placing layer normalization after the residual connection (post-norm), but we found that this caused training to be unstable for RoBERTa with the base number of layers. In this section, the BPE vocabulary size used was 50k rather than 5k. We report the best MLM loss (negative-log-likelihood) on the development set for each run in Table 2.

For each of the RoBERTa models with the post-norm setting, we used a linear learning rate decay with a warmup ratio of 0.06, and report the peak learning rate. We also report the maximum number of 512 token blocks seen.

| <b>RoBERTa (post-norm)</b> | NLL   | Max Blocks  | LR   | Batch Size | Layers |
|----------------------------|-------|-------------|------|------------|--------|
| Small (H=512)              | 5.332 | 750k        | 2e-4 | 32         | 6      |
| Base                       | 7.425 | 250k        | 1e-4 | 48         | 12     |
| Base                       | 7.32  | 750k        | 2e-4 | 32         | 12     |
| <b>RoBERTa (pre-norm)</b>  | NLL   | Blocks/Max  | LR   | Batch Size | Layers |
| Base                       | 4.369 | 1.49M/4.37M | 0.04 | 12         | 12     |

Table 2: The MLM losses for the RoBERTa models with layer normalization after or before the residual connection.

## 5.4 ELECTRA-style MLM

Similar to RoBERTa pretraining, the vocabulary size in this section is also 50k. For ELECTRA-style pretraining, we selected the best model based on the All-Tokens MLM loss, and give the results in Table 3. Compared to RoBERTa, we find that ELECTRA-style pretraining can require much fewer steps and less training time (likely even more so if we only sampled only  $m = 1$  sample per MLM step rather than  $m = 3$ ).

However, ELECTRA-style pretraining did not reduce the final MLM loss on the development set. Since Clark et al. [12] use a discriminator and a generator with separate weights rather than two generators, it is difficult to know why our ELECTRA-style approach results in worse overfitting during pretraining than RoBERTa. However, we note that the large size of the dataset in the original approach means it's very difficult to overfit the training dataset and thus see this phenomenon. Furthermore, we also note that the model begins to overfit on the All-Tokens MLM loss sooner than the MLM loss and suspect it could negatively affect the model's ability to perform MLM generally.

| <b>RoBERTa (pre-norm)</b> | NLL         | Training Time | Samples/Max | LR   | Batch Size |
|---------------------------|-------------|---------------|-------------|------|------------|
| Base                      | 4.369       | 8.5h          | 1.49M/4.37M | 0.04 | 12         |
| <b>ELECTRA-style MLM</b>  | NLL         | Training Time | Samples/Max | LR   | Batch Size |
| Base ( $\lambda=1$ )      | 4.645/1.889 | 5.5h          | 287k/287k   | 0.08 | 12         |
| Base ( $\lambda=16$ )     | 4.642/1.926 | 5.5h          | 287k/350k   | 0.08 | 12         |

Table 3: The pretraining results for the ELECTRA-style models. For comparison against RoBERTa, we report both the best MLM loss as well as the All-Tokens MLM loss, and the sample number at which the best MLM loss was achieved, even though the best model is selected at a different sample. All models were run on a single V100 GPU.

## 5.5 Vocabulary Size

Despite adding pretraining, we found that the performance of the Transformer models was still worse than guessing the majority class. By reducing the BPE vocabulary size from 50k to 5k, we were able to improve the development set performance to 56.86. We present the results after finetuning for RoBERTa and ELECTRA in Table 4 against ELECTRA with 5k vocab.

In this case, the standard vocabulary size used for RoBERTa and ELECTRA was likely too large. Because BPE merges the most frequent token pairs at each step, after enough steps, many subword tokens will no longer appear in the training set. However, the development set may then contain novel words composed of subwords that never were updated during training, resulting in overfitting. The MLM dev loss for ELECTRA with 5k vocab was 2.281 compared to 4.642 with 50k vocab, despite the training losses reaching comparable values at certain times during training, showing that there is less of a discrepancy between the train and development sets with a smaller vocabulary.

| <b>Model</b>              | F1    | F1 (Epoch 1) | Max Epoch | LR    | Batch Size |
|---------------------------|-------|--------------|-----------|-------|------------|
| RoBERTa (50k vocab)       | 51.37 | 44.13        | 14        | 0.004 | 12         |
| ELECTRA-style (50k vocab) | 51.82 | 40.32        | 2         | 0.004 | 12         |
| ELECTRA-style (5k vocab)  | 56.86 | 53.74        | 11        | 0.004 | 16         |
| No pretraining (5k vocab) | 52.17 | 49.35        | 3         | 0.004 | 16         |

Table 4: SQuAD 2.0 results for Transformer models with different vocab sizes. We also report the F1 score after just one epoch, or 130k examples, since most of the F1 scores converge almost immediately. The exception is the Transformer model without pretraining with a BPE vocabulary, which reaches a low of 37.58 after 400k examples.

## 5.6 Iterative Denoising

In order to see if we could improve the pretraining objective further, rather than taking each of the All-Tokens MLM inputs from just the MLM output, we also experimented with iteratively feeding the output samples back into the model. We conjectured that this might force the model to focus more on

difficult inputs, thus converging faster or to a better minimum. Before feeding the inputs back into the model, we also correct a small portion of the incorrect tokens to try to make sure that the task is still possible without memorization. We also asked the model to predict using the special classification token which iteration the input was from in order to try to help the model learn the coherency of the text and to predict answerability. However, as shown in Table 5, we found that pretraining with this iterative denoising objective took much longer, reached worse values, and resulted in a worse F1 score.

We had tried multiple settings other than reported for the sample temperature, loss weighting, and correction probability, but found it was difficult to control the relative difficulty of the MLM tasks at each stage. We suspect there may be better iterative denoising pretraining objectives, e.g. by fixing the relative difficulty of the tasks, or changing the nature of each successive task, that lead models to learn MLM better, but were not able to explore additional objectives due to time constraints.

| Model               | F1    | MLM NLL | Pretraining Time | Outputs sampled |
|---------------------|-------|---------|------------------|-----------------|
| ELECTRA-style       | 56.86 | 2.281   | 8h               | m=3             |
| Iterative denoising | 53.17 | 2.466   | 10h              | m=3             |

Table 5: SQuAD 2.0 and MLM pretraining results for inputs taken independently or iteratively. For iterative denoising we sampled with  $\tau = 0.5$ ,  $\lambda = 3.25^i$  for iteration  $i = 1, 2, 3$ , and correct the incorrect tokens with probability  $p = 0.1$ .

## 5.7 Data augmentation

Due to the class imbalance, we found our models were guessing answerable questions much more frequently. To assess this, we report the sensitivity (number of correctly identified answerable questions), which we denote as A, and the specificity (number of correctly identified non-answerable questions), which we denote as NA, for the models trained on non-augmented and augmented data in Table 6. We find that the model trained on augmented data have a closer sensitivity and specificity, higher accuracy on answerability (AvNA), and an F1 of 60.46. Interestingly, the F1 and EM on only correctly identified answerable questions (F1\_A, EM\_A) was also higher, meaning that when the model was answering a question, it was more likely to answer with the correct span of text.

| Model         | F1    | EM    | AvNA  | A (A EMA)     | NA (NA EMA)   | F1_A  | EM_A  |
|---------------|-------|-------|-------|---------------|---------------|-------|-------|
| Non-augmented | 56.86 | 53.64 | 64.41 | 71.07 (80.22) | 58.30 (48.56) | 77.80 | 68.83 |
| Augmented     | 60.46 | 57.86 | 65.32 | 57.01 (60.55) | 72.95 (69.03) | 82.19 | 72.69 |

Table 6: SQuAD 2.0 results for ELECTRA-style models finetuned on either non-augmented or augmented data. Since the A and NA scores can vary more than 10 percentage points over the course of training, we also give an exponentially weighted moving average (decay=0.9) at the end of training which is more representative of the average over the course of training.

## 5.8 Final Results

Our final Transformer model with ELECTRA-style pretraining, a vocabulary size of 5k, and finetuning on augmented data, was still not able to beat the baseline on the development set (see Table 7). When evaluated on the test set, the F1 also dropped from 60.46 to 56.96, suggesting either overfitting on the development set, or that the test set has some domain shift that the Transformer model couldn't generalize to.

| Model               | Dev F1 | Dev EM | Test F1 | Test EM |
|---------------------|--------|--------|---------|---------|
| BiDAF (baseline)    | 60.79  | 57.50  | 60.32   | 56.81   |
| Transformer (final) | 60.46  | 57.86  | 56.96   | 54.56   |

Table 7: SQuAD 2.0 dev and results for the final Transformer model against the baseline BiDAF model with pretrained GloVe embeddings.

Even though the baseline BiDAF model does use external data in the form of pretrained word embeddings, we note that even the model with learned embeddings was able to score an F1 of 57.98 on the development set, which is higher than our model’s Test F1 if the performance were to also hold on the test set. In our approach, we attempted to get a more powerful model, i.e. the Transformer, to learn better from limited data not by constraining the architecture, but by improving the data and training objectives. However, in this case, constraining the architecture, e.g. with a task-specific BiDAF architecture, is able to confer enough inductive biases to the model that it is able to learn more from limited data.

## 6 Analysis

We notice that while our model, when it answers a question that is answerable, is usually able to identify the correct span of text, it often is not able to determine when a question is answerable or not answerable. To look at the issue, we present two representative examples where the model classifies the answerability of the question incorrectly in Table 8 and Table 9. Overall we find that the model is most likely relying too much on textual similarity, rather than on an actual understanding of the text.

**Question:** What is the approximate condenser temperature in stainless steel?

**Context:** The efficiency of a Rankine cycle is usually limited by the working fluid. Without the pressure reaching supercritical levels for the working fluid, the temperature range the cycle can operate over is quite small; in steam turbines, turbine entry temperatures are typically **565 °C** (the creep limit of stainless steel) and condenser temperatures are around 30 °C. This gives a theoretical Carnot efficiency of about 63% compared with an actual efficiency of 42% for a modern coal-fired power station. This low turbine entry temperature (compared with a gas turbine) is why the Rankine cycle is often used as a bottoming cycle in combined-cycle gas turbine power stations.[citation needed]

**Answer:** N/A

**Prediction:** 565 °C

Table 8: An example where the model incorrectly answers an unanswerable question. The incorrect span in the text is bolded.

In Table 8, we see that the model outputs an answer span that is in a similar location to the phrases "stainless steel" and "condenser temperature" in the text. However, it does not understand that these phrases are referring to different quantities, and do not make sense when aggregated together, and mistakenly predicts "565 °C" which is the "turbine entry temperature," or the "creep limit of stainless steel." The model also fails on various other types of question constructions that result in a textually similar question, such as negating a question.

In Table 9, we can see that despite the context sharing some textual similarity to the question, e.g. "60 Hertz power," "speed," and "turbine," the model classifies the question as unanswerable. It is possible that the model either (a) doesn’t recognize "USA" in the context as "the United States", which does not show up in the context, (b) is unable to associate "speed" with "turbine," which is in a different sentence, or (c) is unable to recognize the synonym "typical" of "usual", which also does not appear in the context, except in the form "usually."

Thus, we suspect that model is not fully learning the meaning and structure of language, instead only relying on simple textual clues to produce a prediction.

## 7 Conclusion

While current Transformer architectures and pretraining objectives are powerful enough to learn more effectively from vast amounts of unlabeled data than task-specific architectures, we find that these performance gains do not extend to low-resource settings.

We did find that a few minor task-specific changes drastically improved performance, e.g. lowering the BPE vocabulary size and augmenting data to correct class imbalance, but we assert that, while inelegant, these changes primarily ensure that the tasks we give to the model are not underspecified. Without them, it is arguable whether the tasks are actually solvable without heavy prior assumptions

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Question:</b> In the United States, what is a usual turbine speed with 60 Hertz of power?</p> <p><b>Context:</b> A steam turbine consists of one or more rotors (rotating discs) mounted on a drive shaft, alternating with a series of stators (static discs) fixed to the turbine casing. The rotors have a propeller-like arrangement of blades at the outer edge. Steam acts upon these blades, producing rotary motion. The stator consists of a similar, but fixed, series of blades that serve to redirect the steam flow onto the next rotor stage. A steam turbine often exhausts into a surface condenser that provides a vacuum. The stages of a steam turbine are typically arranged to extract the maximum potential work from a specific velocity and pressure of steam, giving rise to a series of variably sized high- and low-pressure stages. Turbines are only efficient if they rotate at relatively high speed, therefore they are usually connected to reduction gearing to drive lower speed applications, such as a ship’s propeller. In the vast majority of large electric generating stations, turbines are directly connected to generators with no reduction gearing. Typical speeds are <b>3600 revolutions per minute</b> (RPM) in the USA with 60 Hertz power, 3000 RPM in Europe and other countries with 50 Hertz electric power systems. In nuclear power applications the turbines typically run at half these speeds, 1800 RPM and 1500 RPM. A turbine rotor is also only capable of providing power when rotating in one direction. Therefore, a reversing stage or gearbox is usually required where power is required in the opposite direction.[citation needed]</p> <p><b>Answer:</b> 3600 revolutions per minute</p> <p><b>Prediction:</b> N/A</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Table 9: An example where the model classifies an answerable question as unanswerable, despite a high degree of text similarity. The correct span in the text is bolded.

about the data. However, we suspect there may be better ways in the future to reduce the train-test discrepancy, e.g. by using a character-level embedding, or better pretraining objectives that ensure that the model first uses the coherency of the text to predict answerability, rather than spurious correlations in the majority class.

We still conjecture that the task as given is well-specified enough to be solvable with a general Transformer-based model architecture. Over the course of training, the Transformer-based model seems to mostly learn low-level statistical information, such as textual similarity, rather than high-level information about the structure and meaning of language.

In the future, we would like to see other Transformer variations explored, such as character-level embeddings, relative position embeddings [17], or an encoder-decoder architecture. Furthermore, we also would like to see if other pre-training objectives improve the performance, such as span corruption or our proposed iterative denoising. Though we were unable to compare the downstream task performance of ELECTRA-style pretraining with RoBERTA, as we saw with ELECTRA, a different pretraining objective allows the model to train faster with less data. We hope that a general model architecture and objective, that are able to perform well in a low-resource setting, should also lead to approaches that learn more quickly, effectively, and generally by learning patterns, rather than correlations, that capture the meaning of language.

## References

- [1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- [2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.
- [3] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL <https://www.aclweb.org/anthology/N18-1202>.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon,

- U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://www.aclweb.org/anthology/N19-1423>.
  - [6] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.
  - [7] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL <https://www.aclweb.org/anthology/W18-5446>.
  - [8] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.
  - [9] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf>.
  - [10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL <http://arxiv.org/abs/1907.11692>.
  - [11] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=H1eA7AEtvS>.
  - [12] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020. URL <https://openreview.net/pdf?id=r1xMH1BtvB>.
  - [13] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL <https://openreview.net/forum?id=HJ0UKP9ge>.
  - [14] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. URL <http://arxiv.org/abs/1606.08415>.
  - [15] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. 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 10524–10533. PMLR, 13–18 Jul 2020. URL <http://proceedings.mlr.press/v119/xiong20b.html>.

- [16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL <https://www.aclweb.org/anthology/P16-1162>.
- [17] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL <https://www.aclweb.org/anthology/N18-2074>.