AI 829 NLP Project: Context-Based Open-Domain Question Answering system

Akshay Mahesh Gudiyawar (MT2020137)

Vikrant Punia (MT2020049)

Shubham Jha (MT2020514)

Problem Statement: A model which answers general questions based on a context in text based answering format. We intend to use Wikipedia texts, NCERT textbooks, and Stanford's SQUAD Dataset for training (link given below). The answer to every question would be a segment of text, phrase. For e.g., when trained on context about Amazon Rainforest, the model should be able to answer general questions like - In what country can most of the amazon rainforest be found? Where is the drainage basin of Amazon river? etc.

Dataset Link: https://rajpurkar.github.io/SQuAD-explorer/

The Stanford Question Answering Dataset (SQuAD) is a dataset for machine comprehension based on Wikipedia. The dataset contains 87k examples for training and 10k for development. Each example is composed of a paragraph extracted from a Wikipedia article and an associated human-generated question. The answer is always a span from this paragraph and a model is given credit if its predicted answer matches it. Two evaluation metrics are used: exact string match (EM) and F1 score, which measures the weighted average of precision and recall at the token level. For the task of evaluating open-domain question answering over Wikipedia, we use the SQuAD development set QA pairs only, and we ask systems to uncover the correct answer spans without having access to the associated paragraphs. That is, a model is required to answer a question given the whole of Wikipedia as a resource and not by just giving the relevant paragraph.

Solution Approach: We utilize the Document Reader architecture in DrQA et al. [1] to predict answer text span in the context for the given questions. The approach presented in [1] combines a search component based on bigram hashing and TF-IDF matching with a multi-layer recurrent neural network model trained to detect answers in Wikipedia paragraphs. Using Wikipedia articles as the knowledge source causes the task of question answering (QA) to combine the challenges of both large-scale open-domain QA and of machine comprehension of text. It treats Wikipedia as a collection of articles and does not rely on its internal graph structure. The system DrQA system for MRS (Machine reading at scale) which consists of two components: (1) the Document Retriever module for finding relevant articles and (2) a machine comprehension model, Document Reader, for extracting answers from a single document or a small collection of documents as show in Figure [1].

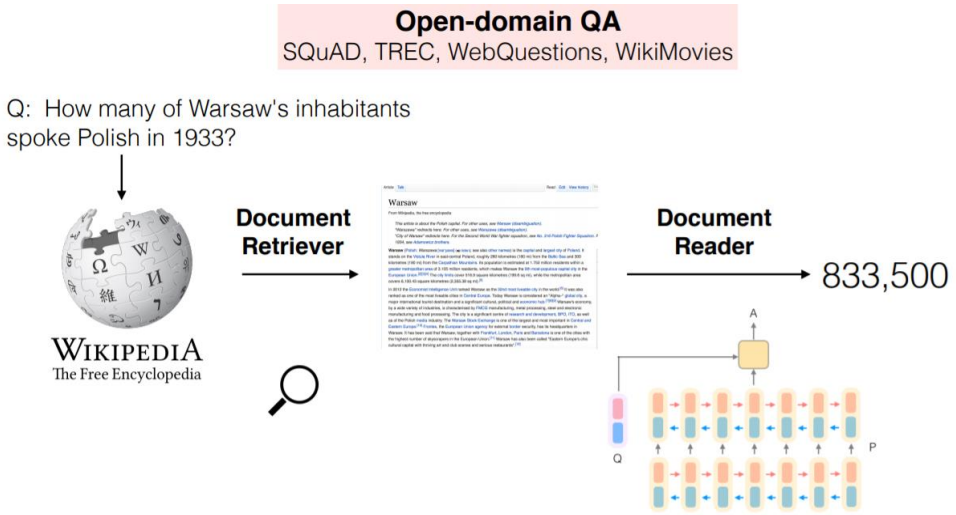


Figure [1] Overview of question answering system DrQA

In Document Retriever, a simple inverted index lookup followed by term vector model scoring is used for the retrieving task. Articles and questions are compared as TF-IDF weighted bag-of-word vectors. Its further improved by taking local word order into account with n-gram features. The Document Retriever is the first part of the full model, by setting it to return 5 Wikipedia articles given any question. Those articles are then processed by Document Reader.

In Document Reader part, given a question q consisting of i tokens {q1, . . . , qi} and a document or a small set of documents of n paragraphs where a single paragraph p consists of m tokens {p1, . . . , pm}, an RNN model is used which is applied to each paragraph in turn and then finally aggregate the predicted answers. To encode a question, we apply another recurrent neural network on top of the word embeddings of qi and combine the resulting hidden units into one single vector. By this we compute q = Summation (bj.qj) where bj encodes the importance of each question word. At the paragraph level, the goal is to predict the span of tokens that is most likely the correct answer. We take the paragraph vectors {p1, . . . , pm} and the question vector q as input, and train two classifiers independently for predicting the two ends of the span. We use the unnormalized exponential and take argmax over all considered paragraph spans for our final prediction.

The two metrics we will be using are exact match and F1. The exact match metric measures the percentage of predictions that match any one of the ground truth answers exactly. The F1 score is a looser metric, which measures the average overlap between the prediction and ground truth answer. The metrics are defined as follows:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \qquad Precision = \frac{TP}{TP + FP} = \frac{\text{\# of matching tokens}}{\text{\# of predicted tokens}}$$

$$Recall = \frac{TP}{TP + FN} = \frac{\text{\# of matching tokens}}{\text{\# of gold tokens}}$$

These are then averaged with the "has answer" prediction where we replace the individual question-answer pair score with a 1 if it correctly predicts that the question is not answerable, and keep the original score otherwise. We use the basic pretrained bert-base-uncased model from huggingface transformer to get the representation for tokens in the (context, question) pairs. Then use a linear classifier to classify answer start/end for tokens in context. The (context, question) pair is pre-processed via huggingface transformer's bert-base-uncased tokenizer to get token_ids, attention mask, sequence mask (whether is in context or question sequence), then fed into our model. Our model is inherited from huggingface transformer's BertPreTrainedModel class, which is a template class for developers to make new classes based on pre-trained models. Both linear classifier and bert model are fine-tuned.

For hyper-parameters, initial learning rate is 1e-3 and batch size used is 16.

Results:

On validation set, the following result metrics were obtained:

EM = 57.756 and F1 = 67.182 as shown in Figure [2]

```
[15] device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
     model_bert = Bert_QA.from_pretrained('/content/drive/MyDrive/IIITB/NLP Project/models/bert_qa_model_uncased')
     model_bert.to(device)
     model_bert.eval()
     evaluate(model_bert,val_dataset)

     100% 13422/13422 [14:30<00:00, 15.42it/s]

     EM: 0.57756
     F1: 0.67182
     (0.5775592311130979, 0.6718219871153087)
```

Figure [2] EM and F1 scores on Validation data

The EM scores on has-ans, and no-has groups are as shown in Figure [3]

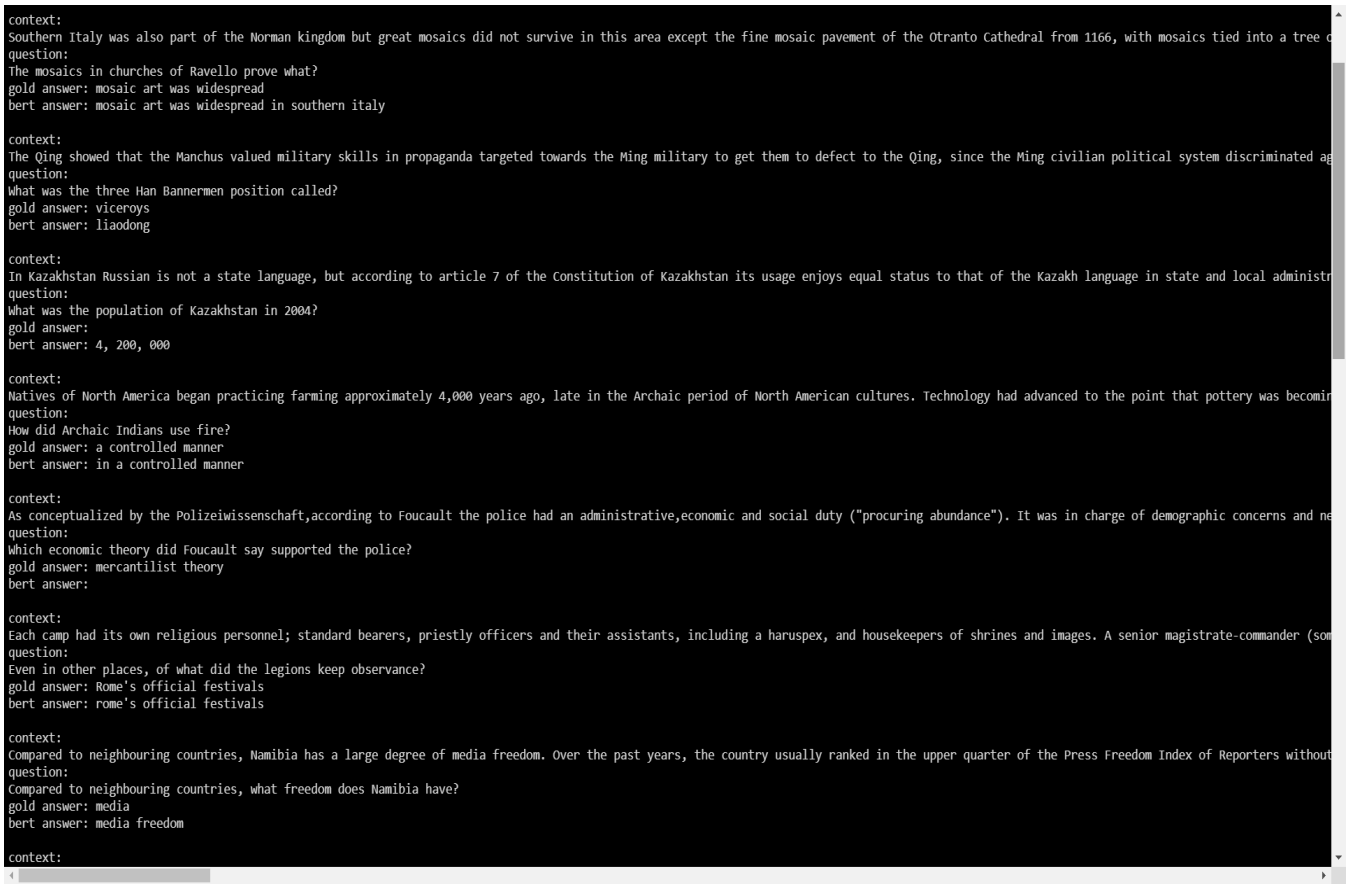
Has ans EM = 51.83, Has ans F1 = 65.53

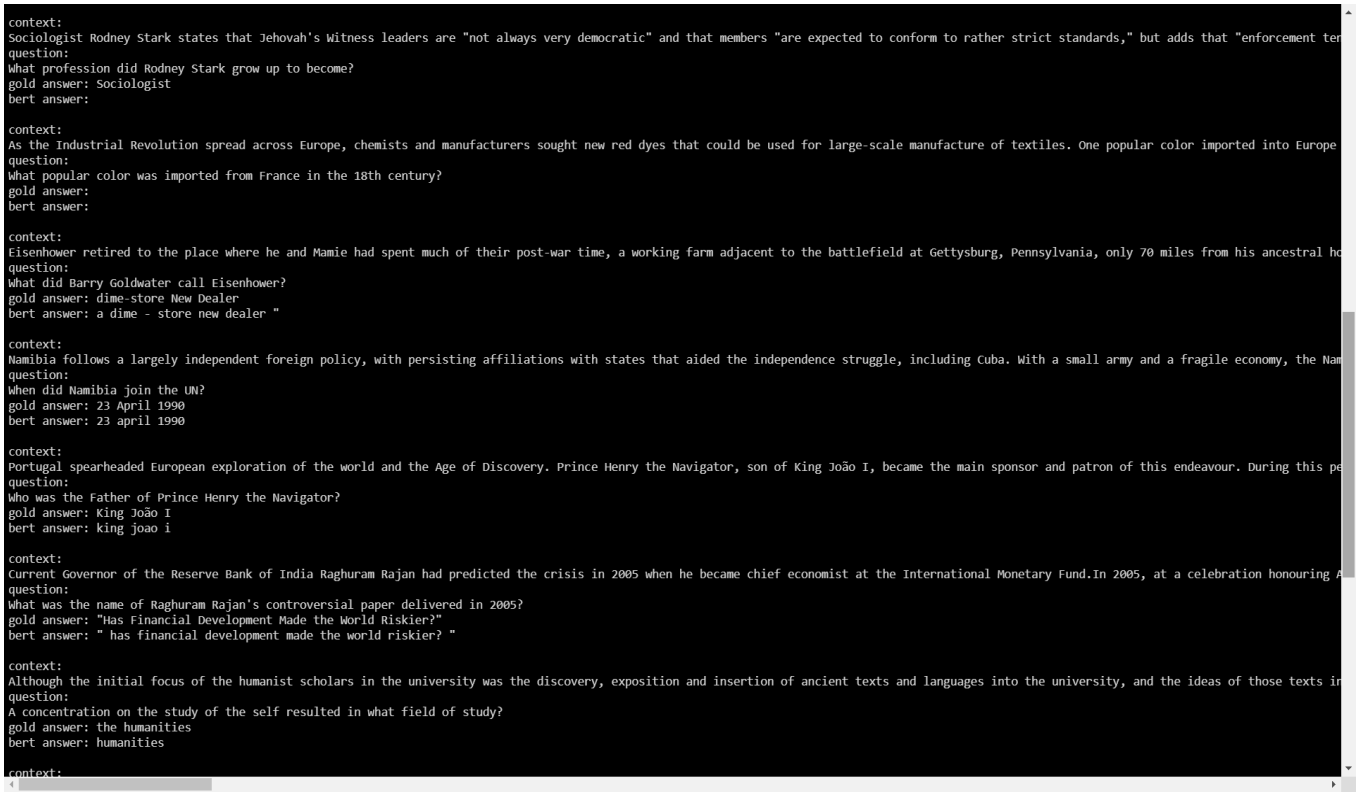
no-ans EM = 70.88, no-ans F1 = 70.88

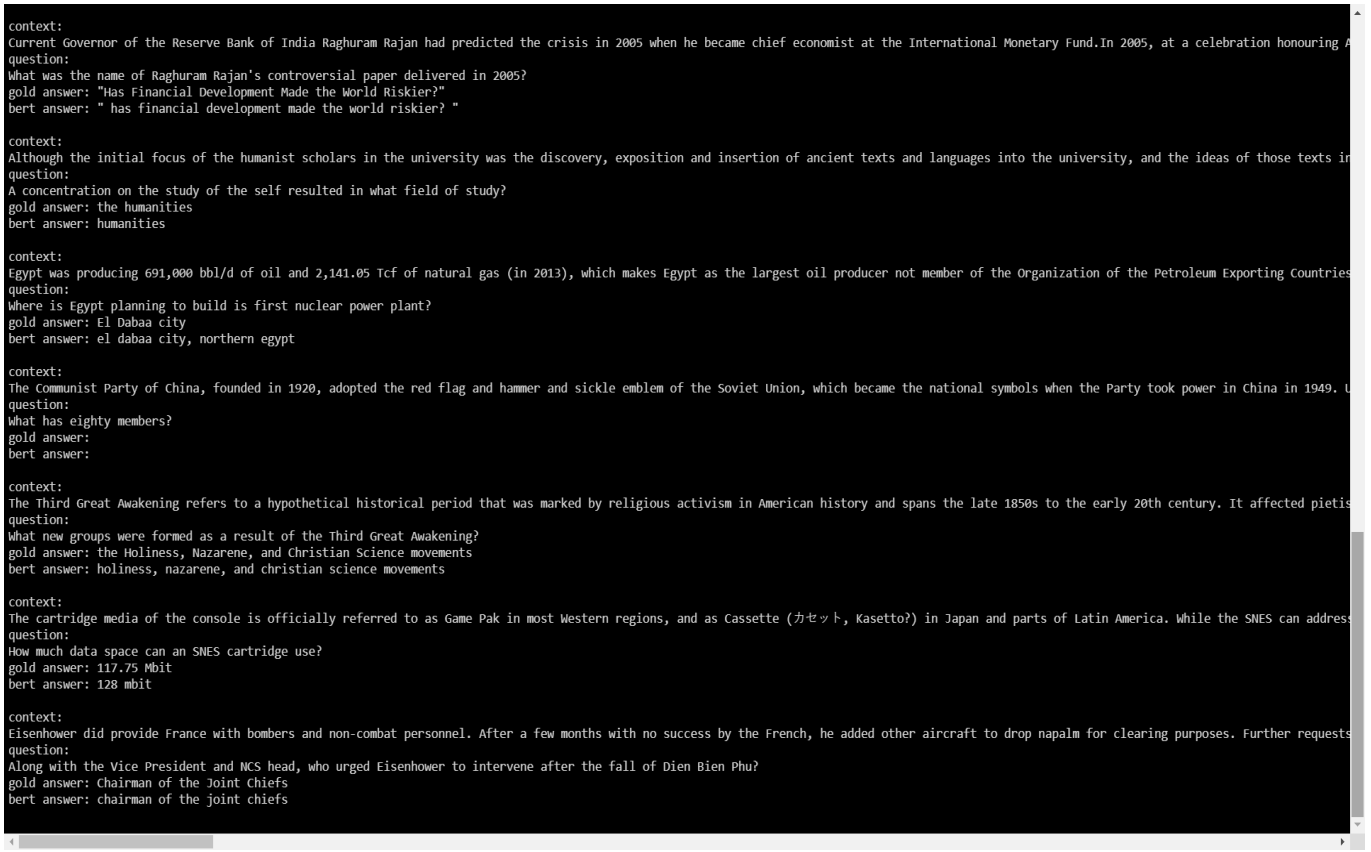
```
hasans_em = []
hasans_f1 = []
noans_em = []
noans_f1 = []
for i in tqdm(range(len(val_questions))):
    pred,_ = online_predict(val_questions[i],val_contexts[i])
    gold = val_answers[i]['text']
    if gold == '':
        noans_em.append(compute_exact(gold,pred))
        noans_f1.append(compute_f1(gold,pred))
    else:
        hasans_em.append(compute_exact(gold,pred))
        hasans_f1.append(compute_f1(gold,pred))

print('has-ans em: %.2f'%(100*np.mean(hasans_em)))
print('has-ans f1: %.2f'%(100*np.mean(hasans_f1)))
print('no-ans em: %.2f'%(100*np.mean(noans_em)))
print('no-ans f1: %.2f'%(100*np.mean(noans_f1)))

100% 13422/13422 [08:26<00:00, 26.50it/s]

has-ans em: 51.83
has-ans f1: 65.53
no-ans em: 70.88
no-ans f1: 70.88
```

Figure [3] EM and F1 scores on has-answer and no-answer groups

Here are some example cases where answers were predicted as phrases to questions asked on context. We can compare the bert predicted answer with ground truth answer (named as gold answer).







References:

[1] https://arxiv.org/abs/1704.00051 (Reading Wikipedia to Answer Open-Domain Questions, by: Danqi Chen, Adam Fisch, Jason Weston, Antoine Bordes)