MixQA: Embedding and Answer Mixing for Question Answering

Anonymous

Abstract

When we attempt to search for answers to our questions in search engines, we often attempt to rephrase our questions if the answers are not what we expect. Therefore, we attempt to replicate this thought process with our method of mixing embeddings of questions and answer ensembling, which we call MixQA. We experiment on the Squad1.1 and Squad2.0 datasets using BERT. We investigate a variety of different questions to mix with the given test-time question, including similar training-time questions, randomly chosen training-time questions, and paraphrases of the given question. We find that all our methods increase performance over the baselines with no mixing, and do indeed take advantage of the information from the mixed questions. Our best performing method on both datasets involves using ROUGE-1 retrieved similar training-time questions for mixing, improving performance on Squad1.1 and Squad2.0.

1 Introduction

There has been extensive work and research on the task of question answering (QA). Humans ask and answer questions commonly in everyday life, as it is a process by which we interact with others and acquire information. In fact, QA is an essential part of our humans learn. The automation of QA has led to improved efficiencies in human learning, and has been incorporated in various applications and mediums such as sesarch engines, dialogue systems, and so forth.

As research in NLP and QA specifically increases, we find that much of the research has focused on improving models themselves or their training, and less attention has been paid to investigating efficient and effective test or inference-time methods. In this paper, we present MixQA: embedding and answer mixing for question answering, in hopes of capturing more possible answers to the question and improving the confidence and accuracy of selected answers by the model. We hope that both methods can introduce both information and controlled noise that the model can effectively utilize to increase its test-time performance.

We investigate two major approaches:

1. Mixing embeddings of the given test-time question with those of its paraphrases and/or training-time questions, and feeding this mixed embedding into the model in hopes of extracting a more confident and accurate answer.
2. Ensembling the answers of the given test-time question plus those of its paraphrases and/or training-time questions. In particular, we sum the predicted beginning and end of span logits for each, in hopes that the final selected answer using this method is a more confident and accurate.

The rest of our paper is structured as follows. Section 2 discusses related work that has been done regarding QA, paraphrasing, answer ensembling, and using training-time information during inference. Then, in Section 3, we discuss the Squad1.1 and Squad2.0 QA datasets that we perform our experiments on, followed by discussing our BERT QA model in Section 4. We then discuss our MixQA pipeline and metholodogy in-depth in Section 5. Finally, we present and analyze the results of our experiments in Section 6, and conclude with key takeaways and future work in Section 7.

2 Related Work

2.1 Question Answering

QA has many moving parts. First, the model needs to understand what the question is asking or looking for. Then, it must comb the either the passage or a database of information and pull relevant. This

answer could be extracted from the passage, or generated based on the passage. All of this is related to language understanding, which has seen a few language models use question answering as a downstream task. These include BERT (Devlin et al., 2018), XLNet (Yang et al., 2019b), RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2019). Some datasets that have come out of this task are WikiQA (Yang et al., 2015) for open-domain QA, HOTPOTQA (Yang et al., 2018) for multi-hop QA, and CoQA (Reddy et al., 2019) for conversational QA. QA is not limited to just text. Models can answer questions from images and video, tasks known as Visual-QA and Video-QA, respectively. What form or medium depends on the task at hand and the sorts of information the model needs to understand.

2.2 Paraphrasing

Questions may have many different forms. Changing one keyword or rephrasing the question can cause a search engine to find different results or many more results. Therefore, paraphrasing plays an important role in helping improve the robustness of QA systems. This appears mostly in the form of data augmentation, where many forms of a question will be generated and then passed through the model to try to find an answer. One form of data augmentation is back-translation (Xie et al., 2019), which involves translating a question into a language different and then back to the input language (e.g. English). That allows for some variation in how the question is phrased, and may even provide slightly different keyword choices. Another method is through adversarial training (Yang et al., 2019a). These methods also allow for more diversity within the questions.

2.3 Clustering

There are usually many documents that many open-domain or retrieval-based QA systems have to examine to find the relevant information to answer a question. However, if documents are clustered, QA systems can reduce their workload by first finding clusters of documents that are most similar to the queried question. There are a few clustering methods, but the document space is very large, especially if the task is open-domain. One method specifically made for querying large databases quickly is FAISS (Johnson et al., 2019). It can find the K nearest neighbors of a given point, or cluster all the given points in a very short period of time.

2.4 Utilizing Training Data and Embedding Mixing

In QA, it may be helpful for the model to look at how some of the training data answered similar questions in the past. Therefore, it may be useful to keep track of the training questions in a "datastore" so that the model can reference it (Khandelwal et al., 2019). These methods include embedding mixing, and do not require any retraining of the model; they are more efficient and *"plug-and-play"* solutions to improving the model's performance. They have been used for other tasks such as style transfer, language generation, and data augmentation. To the best of our knowledge, they have not been investigated specifically for QA, and we are the first to investigate their use with MixQA.

2.5 Answer Ensembling/Ranking

There is previous work that investigates the ensembling or ranking of multiple answers. The most straightforward is to pick the answer that best answers the question, or also known as answer ranking. There have been a few methods to do so including graphical models (Ko et al., 2007) and RankQA (Kratzwald et al., 2019). These methods look at certain metrics that can depend on the specific task. Note that our proposed answer ensembling approach first sums the predicted logits of the beginning and end of span tokens, rather than generating individual answers and then either ranking/picking the best one or ensembling them afterwards.

2.6 Applications

We see question answering in many forms, especially in systems such as Alexa, Siri, and Google Home. In dialogue systems, users can ask questions and issue commands, so the system has to figure out what action it needs to take first. Those systems are more open-domain, while some are closed-domain like "Let's Go" (Raux et al., 2005), which has users calling in for bus information. Sometimes, there is more emphasis on answer correctness and other times there is more emphasis on answer variety, for purposes such as keeping the user's attention. The applications of question answering are endless.

3 Datasets

We use two of the most widely-used and popular datasets in question answering: the Stanford Ques-

tion Answering Dataset (SQuAD), both versions 1.1 (Rajpurkar et al., 2016) and 2.0 (Rajpurkar et al., 2018). SQuAD 1.1 provides more than 100,000 questions, each with their own context passage. Each answer can be found in the passage, and is a span of words in the text. SQuAD 2.0 builds upon Squad1.1 and adds 50,000 questions that cannot be answered from looking at the context passage. This forces models to understand when a passage does not have an answer span that the question is looking for, which improves their robustness and performance. SQuAD 1.1 is evaluated using exact match (EM) and F1, where the former measures the percentage of answer spans that exactly match the ground-truth, and the latter measures the F1-score (combination of precision and recall) of the words/tokens in the chosen answer span compared to the ground-truth. Since SQuAD 2.0 has unanswerable questions, it has separate EM and F1 metrics for both the answerable and unanswerable questions. It also has an overall EM and F1 which is which is aggregated over both types of questions and is typically used to assess the final overall performance of models.

4 BERT Model

We use BERT for our experiments. Specifically, we finetune BERT-base-uncased using a batch size of 16 on both Squad1.1 and Squad2.0 to serve as models for our experiments on each dataset, respectively. We use a max length (of question plus context) of 384, document stride of 128, learning rate of 3e-5, weight decay of 0.05, and train for up to 5 epochs each, saving checkpoints per epoch. We choose the epoch 2 version of both models, as they resulted in the highest validation EM and F1 performance. Simply running these models for inference on their respective validation sets (with no mixing or answer ensembling) serves as the baselines for our experiments.

5 MixQA Pipeline and Methodology

Our MixQA pipeline consists of three major components as seen in Figure 1: the embedding module, the transformer for answer extraction, and answer ensembling. We explain each in detail in the following subsections.

5.1 Embedding Module

The first module in our pipeline is the embedding module. The purpose of this module is to take in the given test-time question plus its paraphrases and/or candidate training-time questions, and output a final mixed embedding to be fed into the following transformer/BERT model.

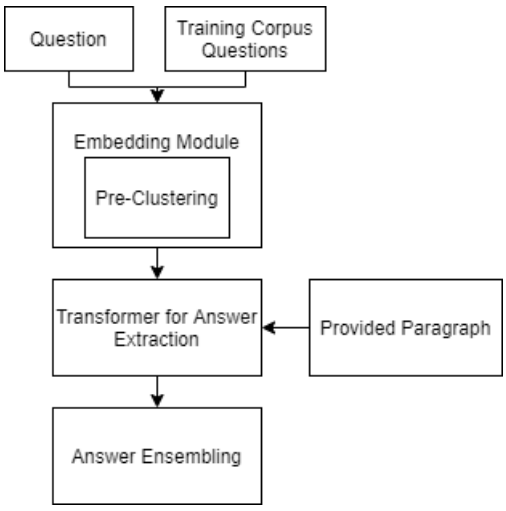

Figure 1: Illustration of our MixQA pipeline.

5.1.1 Extracting Similar Training-Time Questions

Before we mix the embeddings, we need to figure out which of the training corpus questions we want to use to mix. Therefore, we look at ROUGE-1, 2, L (Lin, 2004) and BERTScore (Zhang et al., 2019) to calculate the similarity of the test-time question with the training time ones (q). We also perform experiments where we look at the similarity between the concatenation of the test-time question and context (qc) with the training time concatenations. The different ROUGE metrics look more at token overlap, while BERTScore looks more at semantic similarity by measuring individual token-level similarity of BERT embeddings.

One issue is that these metrics take a long time to run, especially over the over one-hundred thousand training questions. Therefore, we look toward clustering to allow us to calculate these metrics over a subset of the training time questions. We use FAISS for clustering, and keep the clusters to around 100 questions each, resulting in 1300 clusters total. For each test-time question, we can first find the closest cluster centroids and then find the exact closest individual questions from these chosen clusters, which represent only a small fraction of the entire training-time questions.

Specifically, for ROUGE-q, we do a full comparison of each test-time question and every training-time question as question-only ROUGE similarities

Method	Top X sentences	Retrieved Sentence
ROUGE-1	1	In what country is Spoleto located?
ROUGE-1	10-20	What country is Salzburg in?
ROUGE-1	90-100	In what country is the Miami Amtrak Station?
ROUGE-2	1	In what country is Manila?
ROUGE-2	10-20	In what country is the beer Tella produced?
ROUGE-2	90-100	What country is the first football league forgotten in?
ROUGE-L	1	In what country is Spoleto located?
ROUGE-L	10-20	In what country is the Edwards Campus located?
ROUGE-L	90-100	In what part of the United States is Texas located?

Table 1: Sentences retrieved as similar using ROUGE looking only at the questions (q). The original test-time question is 'In what country is Normandy located?"

Method	Top X sent.	Retrieved Sentence
ROUGE-1	1	Which two vernaculars hold relation to the official language?
ROUGE-1	10-20	Who took control of the region in 710?
ROUGE-1	90-100	When were the Netherlands' colonies annexed by Napoleon?
ROUGE-2	1	What is associated with the fall of the Napoleonic Empire?
ROUGE-2	10-20	When was the second wave of neoclassical architecture?
ROUGE-2	90-100	What text embodies Avicenna's legacy in philosophy?
ROUGE-L	1	In what year did the subway begin operation?
ROUGE-L	10-20	What is to the north of Cameroon?
ROUGE-L	90-100	What is Portugal's Social Progress ranking?

Table 2: Sentences retrieved as similar using ROUGE using the concatenation of question plus context (qc). The original test-time question is 'In what country is Normandy located?"

are noticeably faster to compute. For ROUGE-qc, BERTScore-q, and BERTScore-qc, we find the top five closest clusters to each test time question (or qc) and compare against the individual questions (or qc) in those five clusters. Then, using the appropriate metric, we find and save the top 500 training-time questions that are most similar to the test-time question (ordered by decreasing similarity). Some examples of sentences that were picked using this method can be seen in Tables 1 and 2 for ROUGE and 3 for BERTScore.

Note that we also tried randomly sampling training-time questions. We randomly sample over three different seeds, and report the average metrics for each of the runs.

5.1.2 Question Paraphrasing

Other than training-time questions, we also looked at mixing the embeddings of paraphrases of the given test-time questions. There are two methods we used: unsupervised data augmentation (UDA) which we discussed before as back-translation and using the Text-to-Text Transfer Transformer (T5) (Raffel et al., 2019) [1]. For the latter, we choose a pretrained question paraphrasing model on the Quora Question Pairs (QPP) dataset. This T5 returns a list of up to 10 paraphrases for the given input question, which we choose from.

UDA allows us to control the paraphrase by providing a parameter called temperature. The higher the temperature, the more diverse the backtranslated paraphrases and greater change compared to the input, and the lower the temperature, the more that the original syntax and semantics are kept. The optimal temperature values are usually around 0.7 or 0.8, and we have chosen to explore temperatures of 0.6, 0.7, 0.8, and 0.9. It is important to note that for multi-mixing, we mix the four temperature paraphrases' embeddings together.

Some examples of paraphrasing using T5 and UDA can be seen in Table 4. One can notice that certain sentences produced by T5 are too specific with additional details that were not present in the original question, are different questions entirely, or are too vague. UDA also runs into the same problems, but it seems like its paraphrases are more stable with regards to keeping the semantics and meaning of the original input question.

5.1.3 Embedding Mixing

After we have the candidate questions to be mixed with the given test-time ones, we must mix them together. We mix the embeddings of the given test-time question with candidate questions in a ratio. We investigate two variations of this: only mixing the questions (keeping the current test-time context) and also mixing the contexts (mixing with

[1] https://github.com/ramsrigouthamg/Paraphrase-any-question-with-T5-Text-To-Text-Transfer-Transformer-

Type	Top X sent.	Retrieved Sentence
q	1	In what part of the United States is Houston located?
q	10-20	In what century did the Tibetan Empire fall?
q	90-100	What is the name of Charleston's brother city?
qc	1	What location did many of the Greek people decide to live in other than France after the 17th century?
qc	10-20	During what period did Mauretania exist?
qc	90-100	What resulted from the Persian Revolt?

Table 3: Sentences retrieved as similar using BERTScore with either question (q) or question and context (qc) as the similarity comparison. The original test-time question is 'In what country is Normandy located?"

Method	Generated Sentence
T5	Who was in battle with the Normans in Italy?
T5	Who did the Nomans fight?
T5	Who didn't fall to the Normans in Innsbruck, Italy?
T5	How many Norman forces did the Normans in Italy fight?
T5	Do Normans actually fight in Italy? If not why?
UDA temp 0.6	Who was fighting in Italy?
UDA temp 0.7	Who were the Normans fighting in Italy?
UDA temp 0.8	Who are the Normans fighting in Italy?
UDA temp 0.9	Who fought Normandy in Italy?

Table 4: Example paraphrases of input questions using both T5 and UDA. The input question was 'Who did the Normans fight in Italy?"

the training-time contexts). We can mix a variable number of questions or contexts. When working with paraphrases, we keep the test-time context (as there is no associated training-time context). The most simple form of the mixing equation is:

$$m_c * t_e + (1 - m_c) * c_e \tag{1}$$

where m_c is the mixing coefficient (e.g. 0.9, 0.95, or 0.99) and represents the weight of the final mixture to place on the given test-time question, t_e is the test-time embedding, and c_e are the candidate embeddings of the candidate questions to be mixed with the given test-time one.

For top-1 mixing, we simply mix a single paraphrase or the most similar training-time question with the given test-time question. For multimixing, we mix multiple candidates. We use two approaches for multimixing: equal weight to each candidate or normalizing the weight of each candidate based on its similarity to the given test-time question. Note that for paraphrases, only the former applies. Assuming n candidates to be mixed, we have the following equations:

$$m_c * t_e + (1 - m_c) \sum_{i=1}^{n} \frac{1}{n} c_e^{\{i\}} \tag{2}$$

$$m_c * t_e + (1 - m_c) \sum_{i=1}^{n} a_i * c_e^{\{i\}} \tag{3}$$

where $a_i = \frac{sim_i}{\sum_{i=1}^{n} sim_i}$ and $\sum_{i=1}^{n} a_i = 1$

In the above equations, $c_e^{\{i\}}$ refers to the embedding of the i^{th} candidate question, a_i to the weight given to the i^{th} candidate, and sim_i to the similarity score of the i^{th} candidate with the given test-time question. Equation 2 is the equation for equal weight to each candidate, and Equation 3 is the equation for normalized weight by similarity.

5.2 Transformer for Answer Extraction

The embedding module introduced in the previous section plays the role of finding proper candidates for the mixing process. Our pipeline supports mixing between one test-time example with multiple candidates. This architecture is illustrated in Figure 2. Tokenized test examples and multiple candidates for each are passed through the same BERT layer to get their embeddings, and then all candidate embeddings associated with each test-time question are combined using either equation 2 or 3. This combined candidate embedding is then mixed with the test question's embedding and this final mixed embedding is fed into the linear layer to compute answer span probability distributions.

For the implementation, we utilized the HuggingFace (PyTorch version) framework as our pipeline backbone. There are two main reasons why we chose HuggingFace as our main development framework. First, it is easy to load various SOTA pretrained language models and QA models. Second, the flexible dataset class in HuggingFace can load a wide range of QA datasets including

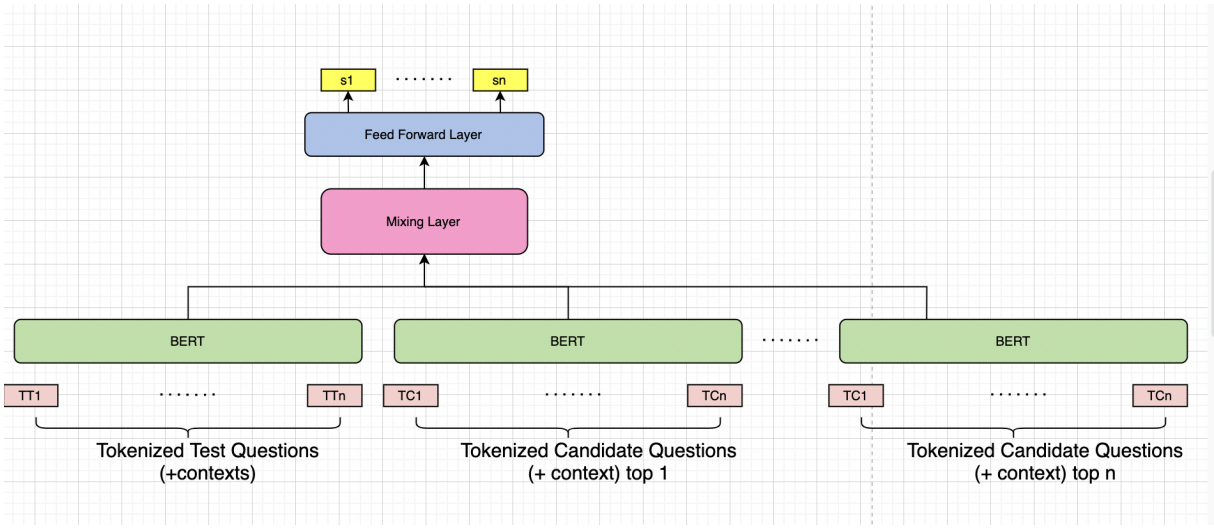

Figure 2: Diagram of our embedding mixing and answer extraction model.

SQuAD and MRQA and provides easy access to evaluation scripts.

However, the use of HuggingFace also posed several challenges for us. First, trade-offs between convenience and customization. HuggingFace is designed for fast experimentation, so most classes, including Dataset, Model, and Post-processing, are highly encapsulated to just a few lines of code. The modification of one part of the source code results in several changes to other parts. We encountered multiple challenges including the misuse of the ArrowTable data structure and the alignment of features for longer context.

An example is the alignment challenge for mixing. Usually for NLP tasks, we truncate the input to fixed lengths to enable efficient batch processing; in the domain of QA, the truncation is far more complicated since it is possible that the the answer span is in the second-half of the truncation and the label will incorrectly become "no answer". To solve this, we needed to map each test question to multiple computed features and then pad or truncate candidate features to make sure they have the same size as the test features.

Inference speed is an important criteria for any ML models and can drastically affect experiment times. For multimixing, we found that mixing with top 10 candidates can take more than 45 minutes to finish per experiment. We hence tried multiple optimization strategies to speed up the inference process by incorporating faster matrix computation libraries and vectorization techniques, and managed to reduce top 10 mixing inference time down to approximately 15 minutes.

Currently, our pipeline supports experimentation using a rather large configuration space with the following configurable parameters: mixing coefficient, mixing context type (train, test), retrieval method (question only (q), question plus context (qc)), number of candidates, whether we normalize the similarity scores (norm) or keep them equal (equal), and types of candidates to mix (e.g. T5 and UDA paraphrases, randomly selected or top similar training-time questions using ROUGE1,2,L and BERTScore). Most of the technical details of these configurations were delineated in Section 5.

5.3 Answer Ensembling

We also planned to incorporate answer ensembling as a separate method from embedding mixing as a form of *late fusion*. In particular, for each question plus either its paraphrases or retrieved similar training-time questions, we feed them through the model and get the prediction logits of each token like normal (basically, the beginning and end of span probabilities). Then, we essentially *ensemble* the answers from the various input questions by adding together the logits per aligned token and take argmax. The argmax beginning and end of span tokens after the summation would correspond to the final chosen answer. Just like with mixing, we can control the number of paraphrases/similar questions, the type of similar question/paraphrase, the weight (original vs. rest), and so forth.

However, we have faced many difficulties with this, and currently have not been able to successfully implement it. The major issue is that the logits are not aligned as they include tokens for the ques-

tion, and the lengths of the original question and its paraphrases or similar questions are all different. This is because logits are also produced for the question tokens (not just context tokens) and used to help decide if there is an answer or not. Hence, we cannot simply remove the question logits. Note that we had tried simply adding the logits and taking argmax and results were poor, likely due to everything being misaligned.

Looking deeper into Squad2.0 and the way it works, we found that it has several criteria for "NoAns" (predicting that a question has no answer). Below are simplified explanations for a few:

- Score of "impossible answer" greater than all valid answers
- Beginning token predicted after ending token
- Argmax predicted answer spans contain question tokens (main difficulty for us)
- Answers above a maximum length

Resolving this logit misalignment problem would be very challenging; lots of engineering work is required to align the logits, and multiple changes to the code (e.g. logit post-processing) is required. One idea we have is to pad logits, e.g. with 0s, up until the maximum length of the given test-time question and its paraphrases or similar training-time questions. However, this will lead to many issues with logit post-processing that will also need to be fixed.

6 Results and Analysis

We ran a search over many hyperparameters to find the best performing set per method of selecting candidate questions based off improvements to both the exact match and F1 scores over the baseline. The parameters are: the number of questions mixed (1, 4 for UDA and 1, 3, 5, 10 for all other methods), mixing coefficient (0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 0.95, 0.99), context (train or test), normalization (equal or normalized), and for similar-training time questions, the process by which to calculate the similarity (question-only (q) or question plus context (qc)). We first take a look at the results on the SQuAD 1.1 dataset in Table 5. We report the best set of hyperparameters for each method.[2]

We can see that all of our methods have a small improvement on the baseline both in terms of EM and F1. However, using the top-5 retrieved ROUGE1-q questions, mixing both questions and contexts, and using a 0.95 mixing coefficient resulted in the highest performance. We note that for ROUGE methods, the model performs better when choosing more questions to mix. The mixing coefficient chosen for most is also 0.95, which suggests that our model does appreciate the controlled noise and information from the mixed questions (by not choosing 0.99), but cannot take too much of it (0.9). The ROUGE scores also all use the train-time context instead of only the test-time context, which shows that adding information from the training-time contexts is also important. They also only use the question as the similarity comparison instead of the question and context. Random mixing appears to barely work here, and paraphrase mixing (T5 and UDA) performs worse than ROUGE. Also note that the EM improvements are statistically insignificant, which we will try to improve upon.

We now look at the performance of our model on SQuAD 2.0 in Table 6. The same trend for the mixing coefficient seems to apply, as most of the methods have 0.95 as the highest performing one. Interestingly enough, the paraphrase methods have a mixing coefficient of 0.99 (and also perform worse than random), which suggests that paraphrasing may not provide the controlled noise that we are necessarily looking for. Our methods provide increased performance on the unanswerable questions, but cannot outperform the baseline for the answerable questions. However, overall EM and F1 have increased for all methods, and increases the most with the ROUGE1-qc method. BERTScore has lower performance than ROUGE, but does perform better than random, which suggests that it is still a viable method. Random mixing does improve performance for most seeds but is unstable, so the averages are lower than the ROUGE methods. Multimixing improves on all corresponding top-1 mixing experiments, and we see that for both SQuAD 1.1 and 2.0, ROUGE1 provides the best improvement.

Most of our F1 scores are statistically insignificant. However, this is likely because we are improving F1 scores on NoAns examples and decreasing it for HasAns examples, thus making the overall improvements seem more like random noise (as some examples increase and others decrease). Hence, paired t-test may not be the best statistical significance test here, or we should look at statistical

[2]Note that for Squad1.1 we did not try BERTScore or ROUGE-qc yet for retrieving similar training-time questions.

Method\Params + Metrics	no. mixed	mix coeff.	temperature	context	normalization	EM	F1
Baseline	-	-	-	-	-	80.36	87.96
T5	5	0.95	-	-	-	80.44*	88.02*
UDA	1	0.99	0.7	-	-	80.43*	88.02
random	10	0.95	-	test	-	80.39*	87.99*
ROUGE1-q	5	0.95	-	train	norm	**80.45***	**88.08**
ROUGE2-q	10	0.95	-	train	norm	80.44*	88.05
ROUGEL-q	10	0.95	-	train	norm	80.44*	88.05

Table 5: Performance of the top/best hyperparameter configurations per method on SQuAD 1.1's dev set. Not all runs have every hyperparameter. Random refers to randomly selected questions for mixing, averaged over three seeds. Note that for all metrics, higher corresponds to better performance. Values higher than the baseline marked with * are statistically insignificant (using paired two-tailed t-tests).

Method\P+M	no. mixed	mix coeff.	context	norm.	EM	F1	AnsEM	AnsF1	NAnsEM	NAnsF1
Baseline	-	-	-	-	72.55	76.01	**71.17**	**78.11**	73.91	73.91
T5	5	0.99	-	-	72.70	76.12*	70.73	77.58	74.67	74.67
UDA	4	0.99	-	-	72.71	76.16	70.72	77.62	74.70	74.70
random	5	0.95	test	-	72.99	76.21*	67.75	74.20	78.21	78.21
ROUGE1-qc	10	0.95	test	equal	**73.14**	**76.30***	67.80	74.12	**78.47**	**78.47**
ROUGE2-qc	5	0.95	test	equal	73.05	76.29*	67.71	74.20	78.37	78.37
ROUGEL-q	3	0.95	test	equal	73.10	76.28*	67.81	74.19	78.37	78.37
BERTScore-q	3	0.95	train	norm	72.99	76.29*	68.52	75.14	77.44	77.44

Table 6: Performance of the top/best hyperparameter configurations per method on SQuAD 2.0's dev set. Not all runs have every hyperparameter. Random refers to randomly selected questions for mixing, averaged over three seeds. Note that for all metrics, higher corresponds to better performance. Values higher than the baseline marked with * are statistically insignificant (using paired two-tailed t-tests). Note that we only tested statistical significance on overall EM and F1 values.

significance of the NoAns and HasAns results separately, which we plan to do.

Moreover, there are some differences here compared to Squad1.1. Firstly, for most methods on Squad2.0, only mixing the questions (and keeping the test-time context) performs better than mixing both question plus context. Further, equal mixing weights per candidate perform better for most methods on Squad2.0 compared to normalized mixing weights by similarity.

We noticed other trends/observations from our Squad2.0 experiments that are not present in our tables. Firstly, mixing rates that are 0.8 and below reduce overall performance and below 0.5, performance drastically reduces including for unanswerable questions. Therefore, we do not report any of these results. We found that having a lower mixing coefficient (AKA weight on the original test-test question) results in lower answerable metrics but higher unanswerable metrics. When it is higher than 0.9, the increase of the metrics for the unanswerable questions more than offsets the decreases in the answerable question metrics, resulting in improved overall performance over the baseline. The unanswerable questions are more than just simple classifications as noted in Section 5.3, so we found this result acceptable. Further, as seen in the SQuAD1.1 results in Table 5, our methods can maintain or increase performance on all answerable questions, proving our method does not simply rely on increasing probability of NoAns.

7 Conclusion and Future Work

In conclusion, we have explored various ways of mixing embeddings from both training-time questions and paraphrases of the given test-time questions. Paraphrasing seems to help the model less than choosing training-time questions based on n-gram and semantic similarity metrics, and also slightly less than choosing the questions randomly from the training set. We saw slight performance increases for Squad1.1 which we will aim to further improve. Further, we saw higher performance increases for Squad2.0 mainly for the unanswerable questions, and we hope to continue exploring potential ways to also increase the performance for answerable Squad2.0 questions to further increase overall performance.

We plan to get answer ensembling working and run extensive experiments with it, and further tune our existing embedding mixing hyperparameters. In particular, for the best performing methods, we will try beyond top-10 mixing (e.g. up to top-100). We also plan to try ROUGE-qc and BERTScore

for Squad1.1 after getting clusters of the Squad1.1 training questions. Two other methods we would like to look at for finding similar training-time questions are using METEOR (Banerjee and Lavie, 2005) and Jaccard similarity. Further, we plan to perform experiments on more models and datasets, as we have only explored the SQuAD datasets using BERT so far. However, our initial findings seem promising, and we hope to improve upon them even further in the future.

For the pipeline side, our current experimentation is largely based on the Google Colab platform, which is limited both in terms of GPU resources and the extent of automation. As a next step, we could setup AWS compute clusters to facilitate further experiment automation.

References

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Jeongwoo Ko, Eric Nyberg, and Luo Si. 2007. A probabilistic graphical model for joint answer ranking in question answering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 343–350.

Bernhard Kratzwald, Anna Eigenmann, and Stefan Feuerriegel. 2019. Rankqa: Neural question answering with answer re-ranking. *arXiv preprint arXiv:1906.03008*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Antoine Raux, Brian Langner, Dan Bohus, Alan W Black, and Maxine Eskenazi. 2005. Let's go public! taking a spoken dialog system to the real world. In *Ninth European conference on speech communication and technology*.

Siva Reddy, Danqi Chen, and Christopher D Manning. 2019. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. 2019. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*.

Qian Yang, Zhouyuan Huo, Dinghan Shen, Yong Cheng, Wenlin Wang, Guoyin Wang, and Lawrence Carin. 2019a. An end-to-end generative architecture for paraphrase generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3132–3142.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 2013–2018.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019b. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.