

Building a QA system (Robust QA track)

Stanford CS224N Default Project

Robert Heckerman Department of Computer Science Stanford University robheck@stanford.edu	Ozioma-Jesus Anyanwu Department of Computer Science Stanford University ozijesus@stanford.edu	William Park Department of Computer Science Stanford University wpark73@stanford.edu
--	---	--

Abstract

While there have been great strides made in solving fundamental NLP tasks, it is clear that the models which tackle these problems fail to generalize to data coming from outside the training distribution. This is problematic since real-world applications require models to adapt to inputs coming from previously unseen distributions. In this paper, we discuss our attempt to create a robust system for extractive question answering (QA). We use a BERT variant as our baseline, and attempt four methods to improve upon it. Our first method is a model that uses the Mixture-Of-Experts (MoE) technique described in [1] and [2]. The second is an original inference-time procedure which predicts the answer span that maximizes the expected F1 score. The third approach is to produce more out-of-domain training examples via data-augmentation. Our final and best-performing method is an Adversarial Training model described in [3]. The MoE model and expected-F1-maximization strategy fail to outperform the baseline's F1 score of 47.098, achieving F1 scores of 44.870 and 44.706 on the validation set respectively. Training the baseline with augmented data produces an F1 score of 48.04. Domain Adversarial Training gives the best results when coupled with data augmentation, yielding an F1 score of 51.17 on the validation set. However, we see that on the test set, none of our models were able to beat the baseline's F1 score of 60.240.

1 Key Information to include

We have no external collaborators nor a mentor. And we are not sharing this project with another class.

2 Introduction

Powerful question answering systems have far reaching applications stretching across many aspects of life and industry. While many of the best systems have equivalent or better performance than humans on their training distributions, recent work ([4], [5]) has demonstrated that these systems fail to generalize well on data that is dissimilar to the training set. Section 3 goes in to more detail about possible reasons for this failure to generalize. This failure presents a large problem, as most if not all real world applications of such systems will be presented with inputs different from those encountered during training.

In this paper, we explore four main approaches to creating a robust system for the task of extractive question answering described in Section 1.2 of [the default project handout](#). We train our systems on inputs coming from three different "in-domain" training sets, and evaluate their performance on previously unseen "out-domain" datasets. See Section 5.1 for more information on the datasets.

We discuss two model-based approaches to undertake the problem. First, we present a Mixture-of-Experts model, in which we create three QA models that each individually specialize in answering questions from one of the three in-domain datasets. We train a gating function which decides how much weight to give each of our three model's output in the final answer-span prediction.

Our second model-related approach is a domain-agnostic QA Answering system trained with the adversarial learning framework discussed in [3]. The main idea is to directly combat overfitting on the training set by producing a system that is penalized for encoding domain-specific features of the input within its hidden states.

The first of our non-model based approaches is a novel inference algorithm which attempts to better maximize our utility function – the average F1 score on the evaluation set. It does this by choosing the start and end positions that maximize the expected F1 score of our selected answer, rather than the start and end positions that have the highest probability. Our hope with this approach is that the uncertainty the model faces when making predictions on out-of-domain data can be handled better by taking an expectation during inference.

In the other non-model based approach, we augment the limited examples of the out-of-domain data using the backtranslation technique detailed in [6]. This approach gives the model "more" examples of the kind of data it will actually see at inference time.

In the end, we found that we could achieve the best performance on the dev set by combining the Adversarial Training model with data augmentation.

3 Related Work

We conducted a detailed literature review in order to understand what techniques have been successful in building a robust question answering system. Once we knew which approaches were successful and, to some extent, why they were successful, we synthesized these approaches to build an effective, robust model ourselves.

In [5], Yogatama et. al neatly summarize the main problem that many extractive question answering systems face when it comes to generalization: these systems overfit to the quirks of a particular dataset and learn domain-specific features that do not generalize to all question answering datasets. With this knowledge, we focused on finding methods which helped with the model’s overfitting on the in-domain data.

[3] directly addresses the problem of overfitting by implementing a model which is penalized for learning domain-specific features. In particular, the authors implement a discriminator which takes in the hidden representation of the CLS token at the top layer of the BERT model and attempts to guess what domain the data come from. If it can accurately guess which domain the data come from, then the model is punished. The idea here is that the model will still attempt to do question answering, but it won’t learn domain-specific features, reducing overfitting.

Other papers attempt to improve robustness by augmenting in-domain training data. [7] attempts various data augmentation techniques on in-domain data but finds them all unsuccessful. This makes sense: additional in-domain data won’t help the model generalize, since it does nothing to stop the model from overfitting on the quirks of the in-domain datasets.

Our setup is different from that of [7], however: we have a small amount of training data from the out-of-domain sets. We think that a logical next step to try would be to augment the out-of-domain data to encourage the model to learn more general features that help performance on the out-of-domain data. [6] provides us with the specifics of data augmentation for question answering via backtranslation.

One last approach we studied for improving robustness was Mixture-of-Experts. First proposed in [1], Mixture-of-Expert models generally try to solve problems more effectively by mixing together answers from "experts" which each learn to handle some subset of the training data. [The default project handout](#) provides an implementation of MoE specifically for question answering.

4 Approach

We use the provided baseline, whose details are in the [the robust QA default project handout](#).

Beyond the baseline, we implement four alternative approaches. Two of our approaches are model-related, while the others deal with data and inference:

1. A Mixture-Of-Experts model.
2. An original inference procedure that chooses the start and end positions of the answer which maximize the expected F1 score according to our model.
3. A Domain Adversarial Training (DAT) model.
4. Data augmentation of the out-of-domain training data through back-translation.

We also combine (3) and (4) to investigate the performance of our adversarial training model when given more out-of-domain training data.

4.1 Mixture-of-Experts

The Mixture-Of-Experts [1] model we implement contains three separate distilBERT models (our experts) whose predictions are mixed together using a simple MLP gating function. In particular, we finetune three DistilBertForQA models on the three in-domain datasets described in Section 5.1. We then combine these three models together with a gating function parameterized by a two layer feed forward network into a large network.

Each expert takes in the subword embeddings for the context, and produces two logit vectors representing probability distributions for the start and end index of the answer. The gating function also takes in the subword embeddings of the context, and produces a 3-class softmax output where the i^{th} entry is the weight that the i^{th} expert's output is assigned to in the final model output. The final model output is the weighted sum of the experts' start and end logits.

More concretely, let $f_1(x), f_2(x), f_3(x)$ (where $f_i(x) \in \mathbb{R}^{d \times 2}$) represent the output logits of the three distilBERT models given the context x . And let $g(x) \in \mathbb{R}^3$ represent the 3-class output of our gating function. We then compute the final output logits $y \in \mathbb{R}^{d \times 2}$ as $y = \sum_{i=1}^3 g_i(x) f_i(x)$. Notice that $f_i(x) \in \mathbb{R}^{d \times 2}$ (where d is the size of the context) holds because the models output both logits for the start and end indices of the answer.

It's also important to note that within our network, the parameters of all but the top layer of distilBERT models were frozen, and thus only those and the parameters of the gating function g were being trained. This was not done by choice, but by circumstance, as our VM instance did not have enough memory to train parameters for 3 large DistilBertForQA networks. Our gating function is a feed-forward network with 2 hidden layers each with 8 hidden units with a ReLU activation for the first layer, and a softmax for the second to produce the 3-class outputs. We apply dropout to the last hidden layer with a probability of 0.1.

4.2 F1-score Maximization

The baseline model's approach to inference is fast and effective: it picks the start and end logits that have the highest sum, subject to some additional constraints described in Section 4.2 of the [default final project handout](#). There are some problems with this approach, though, that we think can be solved by paying closer attention to the utility function we're trying to maximize. In our case, that's the F1 score of our model.

To see where the baseline inference performs poorly, consider an example where our model is relatively unsure of where to start and end its answer span. So consider a probability distribution over a 10-word context that is $[0.19, 0.2, 0.2, 0.2, 0.21, 0, 0, 0, 0, 0]$ for the start index and $[0, 0, 0, 0, 0, 0.21, 0.2, 0.2, 0.2, 0.19]$ for the end index. Here, the baseline inference method would pick the fifth and sixth word as the answer when the true answer is, at least according to the model, more likely something longer (the probability mass for the start has a mean and median to the left of position 5 and the probability mass for the end has a mean and median to the right of position 6). Since we're unsure of where to start and end, the highest probability start and end points are fairly arbitrary, meaning that we could select start and end points that either exclude too much of the correct answer (like in the example above), or include too much surrounding junk. Doing either of those things will hurt our F1 score. A better way to choose start and endpoints would be to find the points that maximize the *expected* F1 score, and use those as our prediction. By taking an expectation, we make our choices more robust to uncertainty in the model. With the example above, for instance, choosing the start and end points that maximize expected F1 score will give us more of the surrounding words, making our selected answer closer to what the model believes is the "true" answer.

To be more concrete, our objective is to predict best start and end indices s^*, e^* according to the following equation:

$$s^*, e^* = \arg \max_{s, e} \mathbb{E}[\text{F1-score} | s, e]$$

We define $\mathbb{E}[\text{F1} | s, e]$ as $\mathbb{E}[\text{F1} | s, e] = \sum_{s', e'} \text{F1}(s', e' | s, e) \cdot P(S = s', E = e')$ where S and E are the true start and end indices. To simplify this equation and make it more tractable to code, we make the naive assumption that the start and end points are independent of one another, and thus have $P(S = s', E = e') = P(S = s')P(E = e')$. Also notice that we obtain $P(S = s')$ and $P(E = e')$ according to the probability distribution output by our model. So, in the numerical example presented earlier, $P(S = 0)$ is 0.19. Algorithm 1 gives pseudocode for this procedure.

4.3 Domain Adversarial Training

We build the domain-agnostic question answering system described in [3] using adversarial training. In contrast to the baseline BERT model whose only objective is to predict the correct answer span, the classifier in domain adversarial training has two objectives. (1) As before, it must learn to predict the correct answer span given the input x (where x is a question-context pair). But additionally, it must (2) do so by projecting the input x into a hidden representation h that does not encode domain-specific features of the input. To this end, the adversarial learning framework includes a discriminator (the adversary to the QA classifier) whose only objective is to predict which domain x comes from given the hidden representation h of this input produced by the QA classifier. The hope is that if the classifier is able to fool the discriminator, then it indicates that it does a good job of throwing out domain-specific information of x when producing the hidden state h , and only learns domain-invariant features of the input to make its answer span predictions.

To be more specific, let f be our QA model (we use DistilBERT for Question Answering described in the default project handout with all the default hyperparameters). Let g be the discriminator and let $x = ([CLS]q[SEP]c[SEP])$ be the input tokens to the QA Model (where q and c are the question and context tokens). When we input x through f , we get p_{start} and p_{end} as usual, which are probability distributions for the start and end positions of the answer. $f(x)$ also yields $h = (h_{[CLS]}h_qh_{[SEP]}h_ch_{[SEP]})$ which are the last layer's hidden representations of each of the input tokens. Our discriminator g takes in $h_{[CLS]}$ and produces a probability distribution over domains $d = g(h_{[CLS]}) \in \mathbb{R}^k$, where k is the number of domains of the training data.

Let $l \in \{1, \dots, k\}$ be the actual domain of the input x , then the loss of the discriminator g for a single example is defined as:

$$\mathcal{L}_g = -\log d_l$$

This is equivalent to the cross entropy between the probability distributions over domains d output by g and the actual probability distribution, which is a one-hot vector $y \in \mathbb{R}^k$ where $y_l = 1$.

The loss function of our QA Classifier f has two parts which correspond to the classifier's two objectives described in the beginning of this section. For the objective of producing the correct answer spans, the loss function $\mathcal{L}_{QA}$ is the same as what is described in the default project handout. Namely, if i and j are the indices of the correct start and end tokens respectively, then

$$\mathcal{L}_{QA} = -\log p_{\text{start}}(i) - \log p_{\text{end}}(j)$$

For the objective of making the discriminator uncertain of its predictions, the loss function $\mathcal{L}_{adv}$ is the Kullback-Leibler divergence (D_{KL}) between the probability distribution d output by the discriminator, and the uniform probability distribution over possible domains $u \in \mathbb{R}^k$ where $u_t = \frac{1}{k}$ for $t \in \{1, \dots, k\}$:

$$\mathcal{L}_{adv} = D_{KL}(u||d)$$

Intuitively, the KL divergence between two probability distributions is a measure of how different the two distributions are. A lower KL divergence means the distributions are more similar. So in minimizing $\mathcal{L}_{adv}$, the QA classifier tries to output hidden states that force the discriminator's prediction to be no better than a uniformly random choice between all the domains. The final loss for our QA classifier f is:

$$\mathcal{L}_f = \mathcal{L}_{QA} + \lambda \mathcal{L}_{adv}$$

where λ is a hyperparameter. Through experimentation, we found that a value of $\lambda = 0.005$ works best when our model is fed data from 3 domains (SQuAD, NewsQA, and Natural Questions).

To train the model, we alternate between optimizing our classifier f and adversary g using the loss functions $\mathcal{L}_f$ and $\mathcal{L}_g$ as described in [3]. Our implementation closely mirrors the implementation linked in the paper, but we wrote our own procedure to generate the domain labels for the discriminator.

4.4 Data Augmentation via Backtranslation

We augment each of the three out-of-domain training sets using back-translation, with ten different languages as our pivot languages. We also create 10 repeats for each pivot language translation. So for each of the three out of domain training sets, we end up with 100 new training sets generated from that domain. We use Spanish, German, French, Danish, Chinese, Japanese, Indonesian, Arabic, Danish, and Romanian as our pivot languages.

To produce the translations, we follow the procedure described in [6]. Specifically, for each question, context, answer triple (q, c, a) in the out-of-domain training sets, we back-translate the context c to create c' , and leave

the question q intact. To generate c' , we split up the context c into its sentences, back-translate each sentence independently using the same pivot language for all of them, and recombine them. We must then generate a new label a' since the translation may have changed the position of the answer span. To do this, we use a 2-step heuristic. Let s be the sentence in c containing the answer a , and let s' be the translated version of that sentence. Then the heuristic goes as follows:

Step 1: Compute character-level 2-gram similarity between each word in s' and the start/end words of a (a_{start} and a_{end}). Keep the words with the highest similarity to a_{start} as candidates for the start position, and the words with the highest similarity to a_{end} as candidates for the end position.

Step 2: For each valid candidate answer span produced in Step 1, compute its character-level 2-gram similarity to the answer a and pick the best one as the new label a' .

Our original implementation of this back-translation procedure is located in the GitHub repository. We use [PyPi's BackTranslation library](#), (which extends Google's Translation API) to perform the translations.

The benefit of augmenting the out-of-domain training data is clear. Since our model is being evaluated on out-of-domain data, it certainly helps to see as much data from these distributions as possible during training.

5 Experiments

5.1 Data

The dataset consists of 6 separately collected QA datasets. We are provided with 3 "in-domain" reading comprehension sets (Natural Questions, NewsQA, and SQuAD) for training and 3 "out-domain" sets (RelationExtraction, DuoRC, and RACE) for evaluation. Within the training data, we have a total of 150,000 training examples (50,000 examples each) and an additional 57,555 examples in the dev set of our training data. Within the test data, we have a total of 4360 test examples.

We are also provided with 127 training examples from the out-domain datasets, which we augment using the backtranslation strategy described 4.4 to produce 12,700 additional examples from each out-domain dataset.

5.2 Evaluation method

There are two evaluation metrics that are used for this project: F1 and Exact Match (EM).

Although both forms of evaluation are used in this project, we focus more on the F1 measure as evaluation via Exact Match seems to be a relatively strict measure that does not accurately assess the abilities of the model. We believe that F1 evaluation more accurately summarizes the power of a model.

5.3 Experimental details

We trained the baseline with an AdamW optimizer using default hyperparameters. This training took roughly 19 hours, as we were evaluating after every 2000 iterations of training.

We trained each individual distilBERT in our MoE model using the default AdamW hyperparameters. Cumulatively, this took about 10 hours. We then trained the gating function and the top layer of our distilBERT model, again with AdamW and default hyperparameters, and that took about 20 hours.

Using our Expected-F1-Maximizing algorithm for inference took about 2 hours on the out-domain evaluation dataset.

The only hyperparameter we tune throughout our experiments is the value of λ in our domain adversarial models. We begin our search over λ by using best value in [3], which was found to be $\lambda = 0.01$. We then test larger and smaller values, 5 and 0.005, to give us some sense of which direction we should search in. We find that, in general, smaller λ 's are giving us better results on the dev set, so on each new iteration we try a smaller value. Each iteration took roughly 20 hours to train.

We run our data-augmentation code locally to generate our new datasets. This process takes about 2 hours running 2 machines in parallel. We then train our baseline model using our original and these new datasets, taking about 22 hours.

Lastly, we observe data-augmentation and domain adversarial training perform best, and combine them. We change the adversarial model to predict 6 classes, and try with $\lambda = 0.01$ and $\lambda = 0.001$. This training takes 24 hours per run.

5.4 Results

Our results on the test and validation sets are as follows:

	DuoRC	RACE	RelationExtraction	Overall
Mixture-of-Experts	-	-	-	44.870
$\mathbb{E}[\text{F1-score}]$ Max	-	-	-	44.706
Baseline	40.31	36.76	64.12	47.10
Baseline Augmented	40.53	31.53	73.74	48.64
DAT $\lambda = 5$	40.09	35.00	68.15	47.79
DAT $\lambda = 0.01$	41.02	36.61	68.28	48.68
DAT $\lambda = 0.005$	43.22	38.90	67.58	49.94
DAT Augmented $\lambda = 0.01$	39.14	32.99	76.39	50.97
DAT Augmented $\lambda = 0.001$	44.29	32.76	76.36	51.17

Table 1: F1 Scores on Out-Domain Validation Sets

	F1	EM
Baseline	60.240	42.362
Baseline Augmented	59.029	42.156
DAT $\lambda = 0.005$	59.717	40.894
DAT Augmented $\lambda = 0.001$	57.814	41.009

Table 2: F1 and EM scores on Out-Domain Test Set

We are very surprised with our test-set results. None of the three models manage to beat out the baseline, in either evaluation metric. One thing we did not have time to do was train our models on the out-domain validation datasets. Given more time, we would have augmented these examples and trained our most promising models with these examples included for better results.

6 Analysis

Our analysis will focus on validation set results since we have the gold answers for this set. We observe that all models perform best on RelationExtraction, and worst on RACE. This makes a lot of sense, as RACE is the most difficult dataset for question answering of the three presented. Specifically, RACE is a dataset constructed from written examinations of middle school and high school students. Many of the questions do not have a word-for-word answer within the text, and require reading between the lines. As a result, there is a good amount of subjectivity in the answers. For example, one question we randomly sampled from the validation set outputs was the following: "What is the best title of this passage?", where the passage was referring to the context. The fact that there is no specific answer span within the context that would explicitly state the title of the passage makes answering this meta-question a much more difficult task. Relation Extraction, on the other hand, contains simple questions that are directly answerable from the one-sentence context. There is little to no subjectivity in the answers, making it a much easier task.

On the topic of relation extraction, we observe an interesting trend in the examples where the baseline fails. When there are several candidate answers to the given category of the question, the baseline's prediction is completely wrong. In other cases, the baseline's answer span at least includes some of the words in the gold answer. For example, if the question asks for place, and there is only one named entity in the context that is a place, the baseline produced that entity in its output. But if there are several named entities that can be categorized as places, the baseline typically gets this wrong. Out of 20 randomly sampled RelationExtraction questions in the validation set, the baseline gets 5 completely wrong, and all of these manifest this very issue. Two examples are shown in Figure 1.

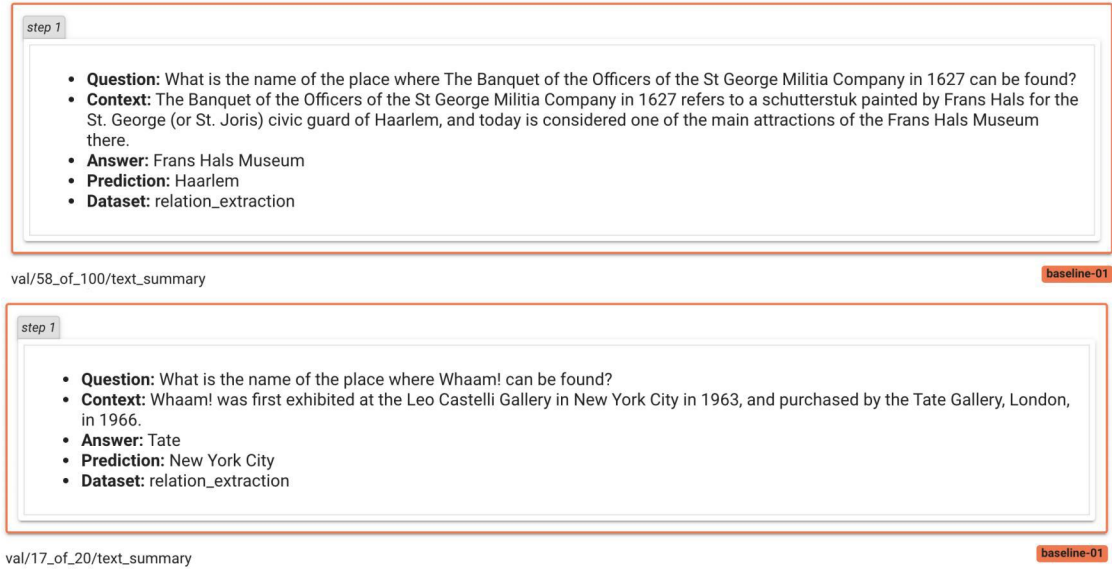


Figure 1: Baseline Failures on RelationExtraction (Visualized with TensorBoard)

One explanation is that our pre-trained BERT model has adequate world knowledge to recognize when a word is, to use the previous example, a place, but is still unable to parse dependencies within sentences to understand which of the candidates has the specific relationship to the question. Notice that the incorrect answers produced by the baseline are both still "valid" answers in the sense that they are both places—indicating that the model does indeed have the world knowledge to identify candidate answers, so clearly this is not the issue.

Augmenting the RelationExtraction dataset solves the main problem. The augmented baseline never completely misses any of the 20 randomly sampled RelationExtraction questions discussed earlier. It certainly seems as though providing more data for the simple task of relation extraction drastically improves the transformer’s ability to understand the relationships between the words in the context and the question. Figure 2 shows the augmented baseline’s performance on the two examples where the baseline failed.

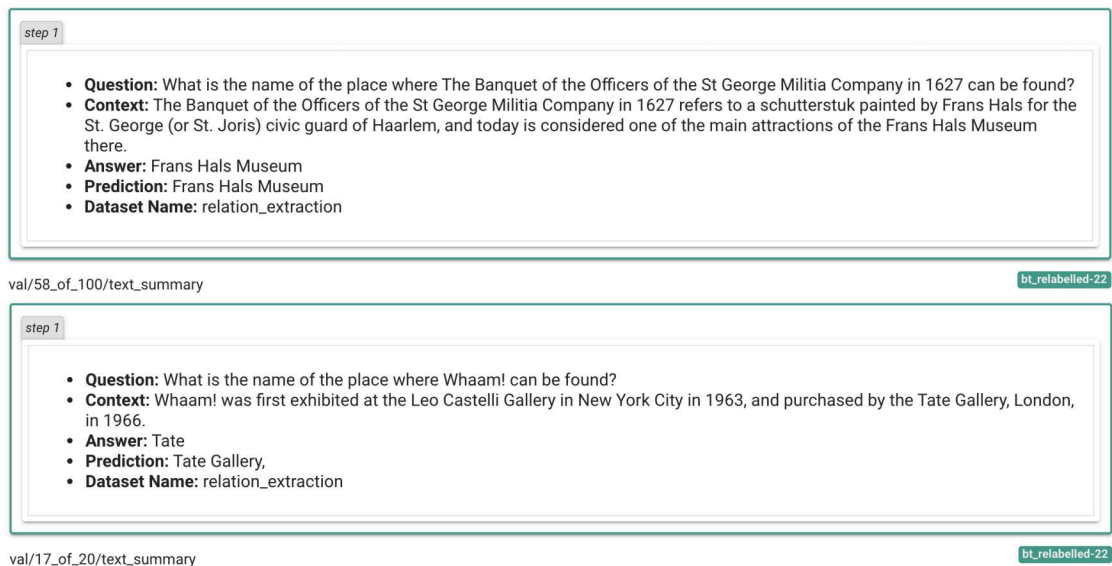


Figure 2: Augmented Baseline Successes on RelationExtraction (Visualized with TensorBoard)

The results of our other experiments with data augmentation are surprising, however. Our expectation is for data augmentation to help with performance on all of the out-of-domain dev sets. Being exposed to examples from each of the out-of-domain datasets should help the model make predictions on those datasets. That is,

if our model now gets to see some examples from, say, the RACE dataset where it didn't get to before, it should know better how to operate on it. Yet, this isn't what we see. What we see, instead, is that the model's performance on DuoRC remains relatively the same, its performance on RACE falls quite significantly, and as discussed earlier its performance on RelationExtraction receives a large boost.

One explanation is that the examples from the out-of-domain train and dev sets are drawn from different parts of the distribution for the DuoRC and RACE datasets, while they are drawn from similar parts in the RelationExtraction dataset. If this is the case, then while the model would actually perform better on the entire DuoRC and RACE datasets, its performance appears worse because the examples its being evaluated on are substantially different than the ones it trained on. We think this might be the case because our dev sets are so small, and likely not representative of the entire dataset.

Another possible explanation is that seeing examples from the DuoRC and RelationExtraction datasets actually hurts the model's performance on RACE. As discussed, the reasoning required for the RACE dataset is more complicated than that required for the RelationExtraction dataset. Thus, we conjecture that seeing the examples from the RelationExtraction dataset encourages the model to use simple reasoning and do worse on the RACE dataset. With more time, we would like to run an experiment where we augment all the out-of-domain datasets except for RelationExtraction, and see if we still get the drop in performance in RACE.

The results of our experimentation with different values of λ are somewhat surprising as well. We expect that greater values of λ should regularize our model more – that is drive it to learn more domain-invariant features –, leading to a more evenly spread out performance across the held out dev sets. However, we don't see this. Instead, we see that the most even performance on the dev sets actually occurs with the smallest value of $\lambda = 0.005$, while the least even performance occurs with the greatest value of $\lambda = 5$. We suspect that there is a small range of values of λ where the model can effectively balance the dual learning objectives, leading to an accurate and well-regularized model. If λ is too large, the penalty incurred by encoding domain-specific features becomes too large, preventing the model from actually learning to perform the main objective of question answering.

The disappointing performance of our novel inference algorithm has a couple of likely explanations. The first possible explanation is that the independence assumption we make is more inaccurate than we intuit, leading to incorrectly calculated expectations. Another possible explanation, and what we think is actually happening, is that the outputs of our distilBERT model aren't calibrated probabilities. It is possible that distilBERT actually outputs some monotonic transformation of the probabilities, in which case our expected F1 equation is very inaccurate since it assumes that the i^{th} value in the output of the BERT model is the *probability* that the answer starts/ends at that position. This is quite likely, actually, since there is no reason for the distilBERT to actually output probabilities as long as the position with the highest probability has the highest output score.

Lastly, we choose to do no analysis on our MoE model. We don't feel that our implementation of the Mixture-of-Experts model is complete, and thus its results are not fully representative of MoE's power.

7 Conclusion

We develop various models to see how well each of them generalizes to out-of-domain data on the extractive question answering task. On the evaluation set, we see that the adversarial training model with data augmentation performs the best overall. However, on the test set the baseline has the best performance.

One of the main lessons we take away is that it's difficult to iterate on modern machine learning models, since their training takes many hours and it's almost impossible to know how your choices will influence your results.

We've also learned that dev set performance does not accurately reflect test set performance, as our best performing model on the dev set performed the worst on the test set.

Though the ultimate purpose of our research is to produce robust QA systems that can be applied in the real world, our findings only apply specifically to the scope of this extractive question answering task. We do not make any claims about the performance of our models outside of this setting and on datasets not discussed in this paper.

References

- [1] S. Nowlan R. Jacobs, Michael I. Jordan and Geoffrey E. Hinton. Adaptive mixtures of local experts. In *Neural Computation*, 1991.
- [2] CS 224N Staff. Cs 224n default final project: Building a qa system (robust qa track). 2021.
- [3] Donggyu Kim Seanie Lee and Jangwon Park. Domain-agnostic question-answering with adversarial training. In *MRQA@EMNLP*, 2019.
- [4] Alon Talmor and Jonathan Berant. Multiqa: An empirical investigation of generalization and transfer in reading comprehension. 2019.
- [5] Dani Yogatama et al. Learning and evaluating general linguistic intelligence. 2019.
- [6] Adams Wei Yu et al. Qanet: Combining local convolution with global self-attention for reading comprehension. 2018.
- [7] Shayne Longpre, Yi Lu, Zhucheng Tu, and Chris DuBois. An exploration of data augmentation and sampling techniques for domain-agnostic question answering. In *An Exploration of Data Augmentation and Sampling Techniques for Domain-Agnostic Question Answering*, 2019.

A Appendix (optional)

Algorithm 1: Expected F1-score Maximization

Input: start probability distribution $s\text{-probs}$, end probability distribution $e\text{-probs}$;
Set $s^* = e^* = \text{None}$. Set $\text{best-expected-f1} = -1$. Set $d = \text{len}(s\text{-probs})$;
for $s \in \{0, \dots, d\}$ **do**
 for $e \in \{s, \dots, d\}$ **do**
 Set $\text{expected-f1} = 0$;
 for $e' \in \{s, \dots, d\}$ **do**
 for $s' \in \{0, \dots, \min(e', e)\}$ **do**
 $\text{expected-f1} \leftarrow \text{expected-f1} + \mathbf{F1}\text{-score}(s', e', s, e) \cdot s\text{-probs}[s'] \cdot e\text{-probs}[e']$;
 end
 end
 if $\text{expected-f1} \geq \text{best-expected-f1}$ **then**
 $\text{best-expected-f1} \leftarrow \text{expected-f1}$;
 $s^* \leftarrow s$;
 $e^* \leftarrow e$;
 end
 end
end
return s^*, e^*
