Building a QA system (Robust QA track)

Stanford CS224N Default Project

Ammaar Adam
Department of Computer Science
Stanford University
ammaar@stanford.edu

Coleman Smith
Department of Computer Science
Stanford University
csmith89@stanford.edu

Takao Yatagai
Department of Computer Science
Stanford University
yatagait@stanford.edu

Abstract

The goal of this project is to investigate the effects that meta-learning has on improving model accuracy in the context of a question answering system. We've based our work on findings from [1] [2]. The DistilBERT baseline model seems to not be able to adapt to out-of-domain texts, and we use meta-learning to get the model to better adapt to new domains. We were able to contribute a codebase and results that allowed us to test our meta-learning methods using DistilBERT and two classification head on its ability to adapt to out-of-domain questions. We were able to see that the meta-learning methods had better generalization errors, and with more training epochs, shows potential to out-perform the baseline on the out-of-domain validation sets.

1 Key Information to include

- Mentor: Elaine Yi Ling Sui
- External Collaborators (if you have any): N/A
- Sharing project: No

2 Introduction

The Question Answering (QA) problem in the natural language processing community has gained much attention in recent years due to how useful an effective QA system would be in our daily lives, such as for smart assistants and for search engines. One of the difficulties of the QA system is that the question and answers can span so many different domains, ranging from financial questions to how to cook. This variability in the domains that the QA system must perform well on makes the problem quite difficult to tackle. The RobustQA track emulates this challenge by having to perform well on out-of-domain datasets, given many in-domain samples, and a very small set of out-of-domain samples.

Data scarcity is a main factor that limits many machine learning approaches, and is especially limiting in the QA systems. The range of domains that the QA system must perform on is limitless, and we want to be able to perform well on most domains, even if we have very little data to learn about a new domain. To tackle this issue of domain changes, we decided to explore meta-learning. Meta-learning effectively trains the model how to learn, rather than traditional supervised models that trains the model what to learn. In this project, we explore how we can use the few-shot framework to make our QA system robust to the out of domain samples.

3 Related Work

We've based our work on findings from two different papers. [1] [2], with more specifics on [2] in the following section.

3.1 MAML

The main meta-learning framework that we based our ideas off of is the Model-Agnostic-Meta-Learning (MAML) model [2]. The MAML model poses the problem at hand as a few-shot classification problem, consisting of a support set and query set for each new distribution of samples. The first was based on ideas from [2] which we discuss later in our approach section. We base our meta learning approach on ideas from [2]. The main approach from [2] is represented in Figure 1.

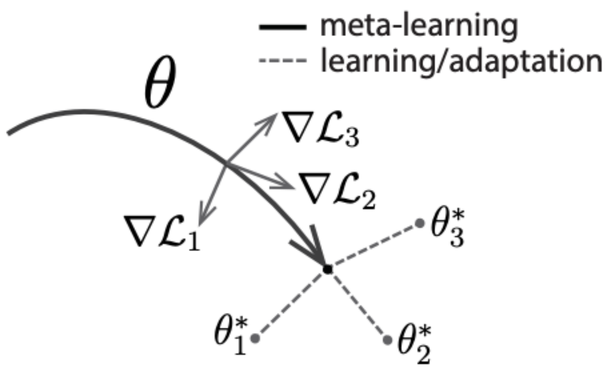

Figure 1: Diagram of the model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks. Taken from [2].

The main idea behind the MAML algorithm is that we are trying to train a model that can quickly adapt to new tasks, given a small subset of examples from the new task. Training using MAML mainly consists of two steps: First, we use the support set as examples from the new task, to adapt our model to this new domain. The adaptation is shown as θ_i in the diagram. Once our model has adapted to the new task, we then see how well the adapted model performs on the query set, another set of samples from the new task. We perform forward propagation on the query set and calculate the loss we get from the query set on this new task. The key point behind meta-learning is that we aggregate the losses we get from adapting and evaluating on different domains, and use the aggregated losses for back-propagation. This allows the model to learn parameters that can effectively learn to perform well on different domains.

4 Approach

The main challenge that the baseline model posed was that it was able to perform decently well on the in-domain datasets, but its performance significantly dropped on the out-of-domain datasets. This signifies that the baseline model is not robust to distribution changes, especially given far fewer samples of a new domain.

In order to combat the inability for the baseline model to adapt to new domains, we have decided to investigate framing this problem as a few-shot learning problem. We see that the out-of-domain training set is much smaller than the training set of the in-domain datasets, and we see that the inability of the baseline model to learn from this new domain is due to the smaller sample size. Therefore, we define the few-shot problem as adapting to a new domain in $K = 127$ shots. As we have seen from the MAML paper [2], MAML is able to adapt to a variety of different tasks using the few-shot learning approach.

Given that the nature of the question-answering problem is ultimately a classification problem—specifically, classifying which index is the start and end index of the answer from the given corpus—

we frame the problem as a 127 - shot, 384 - way problem, since the maximum sequence length is 384 tokens long.

The main changes that we implemented to the baseline was adapting our problem to the meta-learning framework, following the framework described in the above section.

For the backbone of our MAML model, we are still using DistilBERT [3] which is a smaller, distilled version of the original BERT built on a pre-trained transformer.

In modifying/adapting the baseline to a meta-learning framework, we had to make two main alterations to the baseline code which we coded ourselves, which were to re-write a few-shot classification data loader such that it can feed data into our MAML model appropriately, and to implement the meta-learning framework using the DistilBERT backbone and multiple classification heads. Specifics about our implementation of these two components can be found below.

4.1 Few-Shot Dataloader

We implemented a new data loader which separated our data into support and query sets, iterable over the domain. Since we're doing batch learning, we also randomize the order in which domains examples are provided with a batch, i.e. in which order the model is trained on the domains, per-batch. Some design decisions we have made is when training, we split the in-domain train-set into support and query sets. In validation on in-domain val-set, we use the in-domain train set as the support set, and the in-domain val-set as the query set. In validating the out-domain val-set, we use the out-domain train set for the support set, and the out-domain dev/test set for the query set.

Specifically, during training, each batch consists of a batch for each domain in the in-domain datasets. Then, in each domain batch, there is a support and query set, where the support set size is 127, and that for the query set a hyper-parameter. Therefore, each batch has shape $(3, \text{batch\_size} + \text{query\_size}, \text{max\_sequence\_length})$. Note that the order in which we see the domain batches are randomized in training.

During evaluation, we adapt to the support set we get from the train set of the out-of-domain datasets, and feed-forward the dev/test samples as the query set to evaluate our model.

4.2 Meta-Learning Question Answering Model

For our meta-learning QA model, we had to implement a meta-learning wrapper around `DistilBertForQuestionAnswer` such that we could make incremental learning changes (via the support set examples), while still being able to reset to the pre-domain-specialized model before training in the next domain. This introduced a significant amount of complexity relating to PyTorch's use of data-graphs for gradient descent, and our need to change weights in the intermediate time-frame between forward and back propagation.

What made this wrapper particularly complex was that our loss from each domain needed to be aggregated at the end of the forward step of each batch, before being applied via back-propagation all at once at the end of each batch of learning. This was specifically to prevent the model from making incremental changes as domains were encountered, which might have skewed the resulting model.

The algorithm of our model is as follows:

```
1. $B_i \leftarrow$ from dataloader - loads i-th batch
2. Losses$\leftarrow$ - Initialize loss list (used to aggregate losses over domains) $\theta_i$ - get initial model parameters
3. for $d$ in num_domains:
    (a) $X_{s_d}, y_{s_d} \leftarrow B_i[d]$[support] - Support set features and labels for domain d
    (b) $X_{q_d}, y_{q_d} \leftarrow B_i[d]$[query] - Query set features and labels for domain d
    (c) $\theta_d^* \leftarrow$ inner_update$(X_{s_d}, y_{s_d}, \theta_i)$ - Learn the new domain from the support set examples
    (d) Loss$_d \leftarrow \theta_d^*(X_{q_d}, y_{q_d})$ - Forward propagate and get loss on query set
    (e) Append Loss$_d$ to Losses
    (f) $\theta_i \leftarrow \theta_i$ - Reset support set learning
```

4. Loss← mean(Losses)
5. Back-Propagate aggregated losses to both classifier and DiltilBERT components.

To create our meta-learning wrapper, we took a `DistilBertForQuestionAnswer` from a pre-trained model and saved the `DistilBertModel` contained within as an component of our `MAML` model. We then built our own classifier, and had `MAML` feed `DistilBertModel`'s results into our classifier to get our results. We chose this approach firstly because it allowed us to make use of the already-implemented `DistilBertModel`, while also allowing us to take a meta-learning approach via a functionally implemented classification head. The need to learn the support set before query propagation, before unlearning it and learning the new support set. It also mirrored the implementation of `DistilBertForQuestionAnswering`, which is composed of a `DistilBertModel` and a single linear layer classifier.

For training this model, we first trained our classifier on the support set to specialize our model for the new domain before forward propagating the query set through the entire model. By disabling PyTorch gradients during the support set training, we were able to avoid data-graph errors with the HuggingFace `DistilBertModel` implementation, while still making meta-learning updates to the classifier head of our model. Because the classification head was implemented functionally as well, we were able to load and reset the model weights as-needed during the intermediate steps of our `forward` method.

This all means that our `DistilBertModel` is not primed to a new domain via the support set, but our classification head is primed to a new domain via the output of our `DistilBertModel` on that domain. Thus, we applied two slightly different forms of training with meta-learning to the two parts of our model: regular feedforward and backpropagation training on `DistilBertModel` with a more generalized loss due to the specialization of the classification head via temporary support set training and; learning with a meta-learning approach on our classification head, with temporary and permanent learning on the support and query sets, respectively. Overall, this gave us a novel approach to applying meta-learning to composite functional and non-functional models.

Furthermore, by separating the classification head from our `DistilBertModel`, we were able to use multiple types of classifiers. Specifically, we tested both a linear classifier—as a vanilla comparison between baseline and our approach with meta-learning—as well as a multi-layer perceptron, hoping to attain greater accuracy by adding depth to our classifier.

5 Experiments

5.1 Data

We are provided with three in-domain reading comprehension datasets. The three datasets all have 50000 training examples and between 5000-10000 dev examples. We use the training examples from these three in-domain datasets to train our model with meta-learning, splitting the 50000 examples into support and query sets. Specifically, during meta-training, we set the support size to be 127, as this will be the size of the out-of-domain training set when we try to predict on the out-of-domain validation and dev and test sets. We then use the in-domain dev sets for hyper-parameter tuning.

We are also provided with three small out-of-domain datasets from different domains. These datasets have 127 training examples, 128 dev examples, and between 400 and 2700 test examples. We use our 127 training examples as a pre-evaluation support set to train our model on the new domain using meta-learning. This support-trained model is then evaluated on our dev and test sets.

5.2 Evaluation method

Our performance is measured via two metrics: Exact Match (EM) score and F1 score. EM is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly. F1 is the harmonic mean of precision and recall. Our EM and F1 scores are averaged across the entire evaluation dataset to get the final reported scores.

5.3 Experimental details

There were two main steps in getting our final test scores. The first part was training our model on the entire in-domain dataset, using meta-learning. Then, having trained on our in-domain dataset, we evaluate on the dev and test set of the out-domain datasets. In particular, we use the entire train set of the out-of-domain datasets as the support sets which is used for the model to adapt to the out-of-domain datasets, then evaluated on the dev/test splits. We use the evaluations of the dev set of the out-of-domain datasets for hyper-parameter tuning. After performing some grid-search on hyper-parameters, we were able to get the following hyper-parameters:

1. learning rate: 0.01
2. number of inner updates during training: 10
3. number of inner updates during evaluation: 50

The hyper-parameters that were most influential were the number of inner updates. The number of inner updates is the number of times we back-propagate the loss we get from the support set when we are trying to adapt the model to the new domain. If we have too many inner updates, our classifier would over-fit to the support set, and not perform as well on the query set. However, if we under-fit the support set, we would not perform well on the query set because our classifier wouldn't have learned enough about the new task. An interesting point to note is how the number of inner updates differs between when we train versus when we evaluate. We reason that during evaluation, since we must adapt to the totally new out of domain datasets, we need many more updates than we did in training. In training, we were training our model on the same in-domain query data, so we do not need too many inner updates because each batch (task) is not so different from one another.

5.4 Results

Method	Dev		Test	
	F1	EM	F1	EM
In Domain Datasets				
Baseline	70.18	54.30	N/A	N/A
LinearLearner	65.01	48.07	N/A	N/A
MLPLearner	68.57	52.15	N/A	N/A
Out of Domain Datasets				
Baseline	47.57	31.41	N/A*	N/A*
LinearLearner	42.55	24.87	55.063	35.872
MLPLearner	44.77	29.32	58.331	39.358

*We did not evaluate the baseline on the test set as we did not want to use up one of our three submissions on the baseline.

Our MLP classification head performed better than our linear classification head in both in- and out-domain settings, likely due to the MLP classifier being able to capture more generality and complexity than the its linear counterpart.

Sadly, both underperformed compared to the baseline results. However, we believe that this is likely due to funding and time constraints—with the latter exacerbated by the former—which prevented our models from reaching their full potential.

6 Analysis

During development and training, we paid close attention to both the in-domain validation scores and out-of-domain validation scores over the training epochs. One of our major challenges in implementing these methods was training our model with a few-shot classification data-loader while still evaluating using the original `QADataset` given to us in the baseline code. Thus, we used the baseline results in these metrics as a sanity check in an effort to ensure that all parts of our model were working.

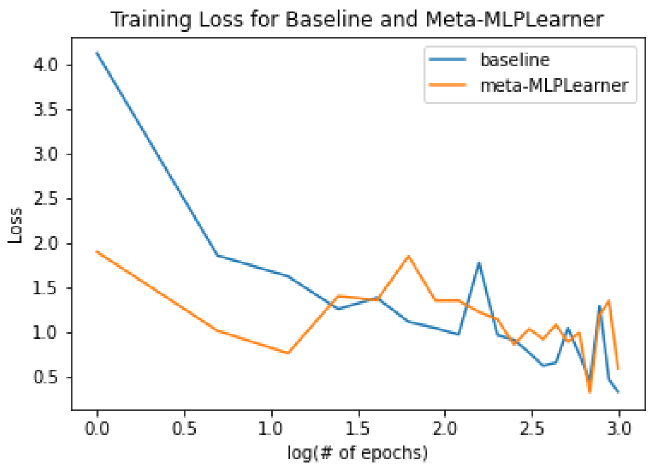

Figure 2: Loss curve for Baseline and Meta-Learning Distilbert Model + MLP

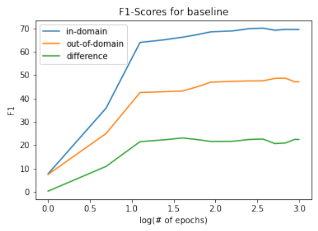

Figure 3: F1 Scores for Meta-Learning Distilbert Model + LinearLearner

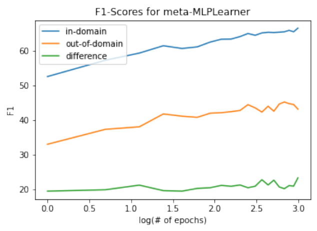

Figure 4: F1 Scores for Meta-Learning Distilbert Model + MLP

Taking a look at both in-domain and out-of-domain validation F1 scores for the meta-learning distilBERT model with classification heads (Linear and Multi-Layer Perceptron), we see that for both models, although the in-domain F1 score did not reach the same level as our provided baseline model, the trajectory of improvement remained positive. Further, although the loss' rate of decrease was slowing, it had clearly not yet reached a steady plateau. We can observe that the training loss for baseline decreases faster because the model easily overfits the training data, whereas meta-learning takes longer to reach training convergence, because it learns a more general model, which results in a higher out-of-domain performance in the same number of epochs. This indicates that we need more epochs to train our meta-learning methods than our baseline to reach similar levels of training loss.

Additionally, the part that we are most excited about is that our out-of-domain F1 scores are also still increasing, and the discrepancy between the in and out of domain performance is similar, or less than that of the baseline model. This indicates that the meta-learning algorithm is able to achieve better generalization to the out-of-domain datasets. We see that although the F1 scores are increasing over epochs, the difference does not increase nearly as much, which means that the generalization errors do not increase with more epochs, especially for the meta-learning models.

We also note that our MLP classifier performed better on both in- and out-domain settings. This is likely due to the greater depth of the classifier network being used, but shows that there is potential for improvement upon the current DistilBertForQuestionAnswering via the use of a deeper/more complex classifier.

Although we see a decreased discrepancy between our in-domain and out-of-domain validation scores on the meta-learning methods than the baseline, the discrepancy still exists, which still shows the limitation of meta-learning in reducing generalization errors. We may be able to see a greater decrease in generalization errors if the problem were slightly different. Specifically, we think that comparing the baseline and meta-learning methods on datasets consisting of a far greater number of domains, and with fewer number of samples per domain would show greater generalization improvement from the meta-learning method. We believe that a meta-learning approach will work well as it has inherent advantages in capturing more general information from training data, is less prone to over-fitting, and does not suffer from divergence due to evenly spread directionality within the search space between the model parameters and loss over these many domains.

7 Conclusion

In our investigation, we have shown that a meta-learning approach to the generalized question-answer task clearly shows potential for improving upon more well-established techniques in the field. As we see from the three figures of our training loss, in-domain validation scores, and out-domain validation scores, our model clearly is able to adapt well to the in-domain datasets, without plateauing on the out-domain validation set scores. This shows that the idea of using the small train-set of the out-of-domain dataset serves as a good way of adapting to new out-of-domain tasks, and that meta-learning during training is a viable way for the model to adapt to different domains better. The main bottleneck of the project has been the compute resources and time. We spent the majority of the time coding up the very intricate and complex meta-learning algorithms, especially given the complexity of DistilBERT, and as noted in our analysis, our model may have performed better if given more time to train. Our main contributions is the adoption of the meta-learning framework for the robust QA problem. We were able to overcome the large roadblock of implementing meta-learning that stemmed from the nature of PyTorch/TensorFlow and their use of data-graphs in gradient descent calculations.

Given our results and the improved generalization of our meta-learning model, we would like to further investigating attaining higher scores for both in-domain validation scores and for out-domain validation scores. We could implement deeper networks for our classification head to achieve better performance on our in-domain datasets. We could then evaluate how much the model overfits to the in-domain datasets by validating on the out-of-domain datasets, and explore how we can use techniques such as data augmentations and regularization techinques (such as dropout) in decreasing the generalization errors such that the out-of-domain validation scores could improve with the improvement on the in-domain datasets.

Additionally, since we are only testing for 3 out-of-domain datasets, there are not that many new tasks that the model must adapt to. Some future directions to take this would be to evaluate on a similar QA task with many more different classifications of domains, with much fewer train samples per domain. Then, we can compare how the meta-learning framework would compare against the baseline in this even more challenging problem of adapting to many more tasks with fewer samples.

References

[1] Arzoo Katiyar Kilian Q. Weinberger Tianyi Zhang, Felix Wu and Yoav Artzi. Revisiting few-sample bert fine-tuning. In *The International Conference on Learning Representations (ICLR)*, 2021.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *arXiv preprint arXiv:1703.03400*, 2017.

[3] Julien Chaumond Victor Sanh, Lysandre Debut and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *arXiv preprint arXiv:1910.01108*, 2019.