Meta Learning with Data Augmentation for Robust Out-of-domain Question Answering

Stanford CS224N Default Project 2

Ken Wang
Department of Aero & Astro
Stanford University
weizhuo2@stanford.edu

Lisen Deng
Department of Aero & Astro
Stanford University
lisenddd@stanford.edu

Abstract

Natural language understanding problems has gain much popularity over the years and current models often has poor generalizability on the out-of-domain tasks. This robust question answering (QA) project aims to remedy this situation by using Reptile, a variant of meta learning algorithms. In this project, the primary goal is to implement Reptile algorithm for question answering tasks to achieve a better performance than the baseline model on the out-of-domain datasets. After the Reptile implementation is validated, the secondary goal of this project is to explore how various hyper parameters affect the final performance. After we believe that the Reptile is optimally tuned, we worked on the data provided for this project. First, we merged in-domain validation dataset to the training data, then we added data augmentation to further tap into the potential of the out-of-domain training data. Training results of Reptile outperforms vanilla BERT model and Reptile with data augmentation increases the score even further. The best F1 score is 59.985 and best EM score is 42.225. If we compare the performance on out-of-domain validation dataset, scores are more than 12% and 22% higher than the baseline score respectively.

1 Key Information to include

External collaborators: None | Mentor: None | Sharing project: None

2 Introduction

Natural language understanding(NLU) problems has became increasingly popular over the years for its applicability to various tasks that would either provide convenience to daily life or increase working efficiency. However, the performance of current models is often poor when the task is out of the training domain, limiting the applicability of those models. This robust QA project aims to remedy this situation by using Reptile, a variant of meta learning algorithms.

Concretely, the goal is to implement Reptile algorithm to work with BERT. Traditionally, a pretrained model works well for question answering on in-domain datasets, we want to improve this model so it works well on out-of-domain datasets too. The Reptile implementation is expected to achieve a higher performance metrics than the baseline model on the out-of-domain datasets.

This goal is quite promising given the results in the suggested paper [1]. Reptile is proved to achieve better performance on majority of the datasets compared to a vanilla BERT model. However the paper did not directly evaluate Reptile's performance on question answering. It would be interesting to validate Reptile's performance on question answering tasks.

Apart from improving the model itself, we also tried to improve the performance from the data side. First, we notice that the in-domain-validation dataset was not used anywhere in the training process.

Therefore we merged it with the training set, adding roughly 15% of extra data. Then we tried data augmentation as illustrated in the paper [2]. Basically, for each context in out-of-domain training set, we generate n new contexts using four techniques in the paper (random insertion, random swap, random deletion, synonym replacement). We hope that this augmentation would help the agent use the small out-of-domain dataset more efficiently and achieve a higher score.

3 Related Work

Natural language understanding problems has been a long standing challenge. Although there are some methods proposed to give okay results, it has been suspected that the model's understanding has never gone deeper than the superficial relations. This makes the model hard to apply to real world problems which involves a wide range of task distributions. In response to this problem, researchers are investigating every possibilities to increase the generalizability of the NLP systems. Previous works focused on learning a general language representations that would help downstream tasks. The other approach involves learning contextualized word embeddings, for example BERT model. In light of recent success of MAML on reinforcement learning tasks, it has been taken as a promising candidate. The concept of few shot learning naturally fits into this NLU problem as human languages are inherently unstructured and often chaotic. The paper [1] is particularly interesting as it tries to answer the question above by directly investigating the performance of MAML on the NLU tasks.

Data augmentation was widely used in the field related to learning with vision. As images are easy to augment, one could perform noise, down sampling, scale, rotation and distortion to an image to allow the model learn real life imperfections. We believe the same could be applied to language data. The core idea of a piece of text should not change despite the surrounding words are replaced with synonym or with a few words dropped. For the data augmentation, we referred to a method called Easy Data Augmentation (EDA)[2] that claims to be able to improve the performance of the model through 4 simple augmentations. In the paper, the method was tested on various datasets and each has gained 2% in performance metrics. Moreover, they also showed less performance drop when the dataset is small(<20% of original size). We believe this is perfect for our purpose, as we want to utilize the small out-of-domain dataset more efficiently.

4 Approach

4.1 Meta Learning

Inspired by the recent success, we decided to chase down the meta learning approach in this project. Meta learning, as the name suggests, learns how to learn fast. It has an edge on the few-shot problems and domain shifting problems since it aims to learn a set of parameters that would perform reasonably well across the task distributions after fine-tuning with little data.

Reptile[3] is a first order meta learning algorithm that aims to find a initial set of parameters that can be fine-tuned to a new task using a small amount of data. Let ϕ denotes parameter initialization, and W_τ denotes the set of optimal parameters for task τ which is from a distribution of tasks T. Reptiles tries to find ϕ such that the euclidean distance between ϕ and W_τ is small for all tasks within task distribution T. Specially, the algorithm tries to minimize $\mathbb{E}_{\tau \sim T}\left[\frac{1}{2} D\left(\phi, \mathcal{W}_\tau\right)^2\right]$, and Reptile corresponds to performing SGD on this objective. Training involves a inner training loop which is gradient descent for a task. The resulting parameters from the inner training loop are used for a outer loop update. A detailed training procedure for Reptile is shown in Algorithm 1

The official source code for Reptile is written in Tensorflow and does not interact with a BERT model. The suggested paper [1] mentioned using Reptile on question answering as a auxiliary task without directly investigating the performance. There was no code published for the suggested paper. Considering all the factors, we rewrote the algorithm in Pytorch and made it compatible with the project starter code.

Since meta learning algorithms like Reptile are known for hard to train as they tend to be unstable, it can be challenging to correctly implement Reptile and train it successfully. To alleviate this problem, the starting parameters is based on the suggested values written in the Reptile paper[3]. The hyperparameters presented in the suggested paper is less valuable to us as the authors used the

Algorithm 1: Reptile

ϕ initial model parameters;α outer learning rate; β inner learning rate; k inner training steps;
best score $\leftarrow$ 0;
for *n epochs* **do**
  sample a task, t, from a distribution of in-domain training tasks;
  train ϕ for k steps with task t using learning rate β, resulting parameters $\hat{\phi}$;
  $\phi \leftarrow \phi + \alpha(\hat{\phi} - \phi)$;
  if *reaches validation frequency* **then**
    finetune ϕ on out-of-domain training datasets, resulting parameters $\hat{\phi}$;
    evaluate on out-of-domain evaluation datasets using $\hat{\phi}$, resulting score s;
    if *s > best score* **then**
      save model parameters ϕ
    end
  end
end

batched version of Reptile. We used the serial version of Reptile that we only sample one task for each inner training loop. The batched version of Reptile samples a number of tasks and average their gradient as the result of the inner training loop. The number of tasks is set to 8 in the paper. Even with data augmentation, which will be described in the next section, we can only get 6 tasks. Therefore, we decided to use the serial version of Reptile.

4.2 Data augmentation

As discussed in the introduction section, we would like to use Easy Data Augmentation (EDA) to generate the augmented data. EDA involves four main techniques: Synonym Replacement (SR), Random Insertion (RI), Random Swap (RS), and Random Deletion (RD). The description for each technique is listed below. The $n\_per\_technique$ below means number of time each technique got applied to the context. This value is usually calculated by $n\_per\_technique = int(n/4) + 1$, where n is number of new contexts to generate for each context.

- Synonym Replacement (SR): Randomly choose $n\_per\_technique$ words from the context that are not in stop words list. Then replace each of them with a random synonyms.
- Random Insertion (RI): Find a word in the context, generate a random synonym, and insert it within 12 words radius of source word. Repeat this for $n\_per\_technique$ times.
- Random Swap (RS): Randomly choose two words in the context within 12 words radius and swap their positions. Repeat this for $n\_per\_technique$ times.
- Random Deletion (RD): Randomly remove each word in the context with probability p. Repeat this for $n\_per\_technique$ times.

Notice that the description is different from the original paper, as the original paper is operating on dataset consists of sentences. Here we have a paragraph to work with. To retain the core context in the paragraph, we decided to limit the operations of random insertion and random swap within the 12 words radius of the source word. Since it doesn't make much sense to swap a word at the beginning of a paragraph with the word at the end of paragraph.

Also, we do not want to modify the answer region, since any of these operations could potentially alter the meaning of the answer. Therefore we first replace the answer region with a unique token that will not be touched by any of the operation. Then after the four operations, answer block was replaced at the token location. This also allows us to record the new answer starting point while retaining the original answer.

The actual implementation of EDA is based on the published code of original paper. But we modified it according to the specifications we discussed above to make sure it fits our propose. That said, roughly half of the implementation is written by us and the other half is heavily modified from the published code.

4.3 Baseline

The baseline model would be the model trained using the starter code. It's a DistilBERT model that performed best on the in-domain validation dataset. The DistilBERT model is a smaller model compared to BERT. BERT has two times more parameters than DistilBERT and takes two times more inference time, but DistilBERT only performs slightly worse, about 5%, than BERT. The starter code for this project combines three in-domain training dataset and performs multitask learning on the combined dataset for 3 epochs. The learning rate is set to 3e-5 and batch size is 16. Other details of the baseline model can be found in the project handout.

5 Experiments

5.1 Data

Data is downloaded from the link provided in the starter code. The starter code combines three in-domain training data into one data loader. For meta learning purpose, we load each in-domain training data into a separate data loader, so we have three tasks to sample from. Evaluation for Reptile is performed using out-of-domain training and validation data. Therefore, we decided to merge in-domain training data with in-domain evaluation data. Data augmentation is performed only on out-of-domain training data and only used in training.

5.2 Evaluation method

The start code evaluate three out-of-domain datasets together without any finetuning. For our evaluation, we evaluate the performance of one dataset after finetuning the trained model using data from that particular dataset. Then, we revert the model to the state before finetuning, and perform finetuning and evaluation for another out-of-domain dataset. This evaluation procedure is kept the same during training time and testing time. In the end, we report the average score of the resulting three sets of scores.

5.3 Experimental details

The model is trained for 3 epochs for all experiments to ensure fair comparison with the baseline model. We tuned inner learning rate, outer learning rate, inner training steps, and the amount of data augmentation. Specific values and corresponding results are presented in the next section. In general, one experiment takes about a bit more than 3 hours on a NCsV3 virtual machine without data augmentation. If data augmentation is used, it takes less than 4 hours depending on the amount of augmented data.

5.4 Results

We submitted three results to the test leaderboard of RobustQA track. The experiments are summarized in Table 1, with the best score highlighted for each case. The first column of the table gives the default hyperparameter values for our experiments. The second column provide the hyperparameter value that is different from the default value. F1 scores and EM scores for validation data and test data are all shown. All three results outperform the baseline model. A cross comparison of trends on each hyper-parameter is presented in Figures 1, 2,and 3.

Default Value	Value Changed	Val F1	Val. EM	Test F1	Test EM
lr = 3e-5; inner lr = 1e-5	augmentation = 0	51.089	35.787	**59.985**	41.697
inner steps = 20; batch size = 16	augmentation = 16	51.611	37.173	59.465	41.720
augmentation = 32	N/A	**52.841**	**38.743**	59.616	**42.225**

Table 1: leaderboard scores

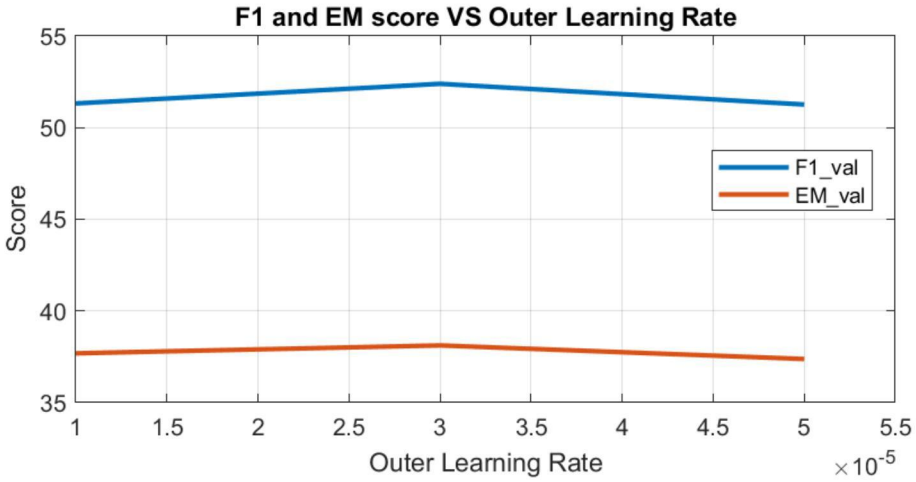

Figure 1: outer learning rate

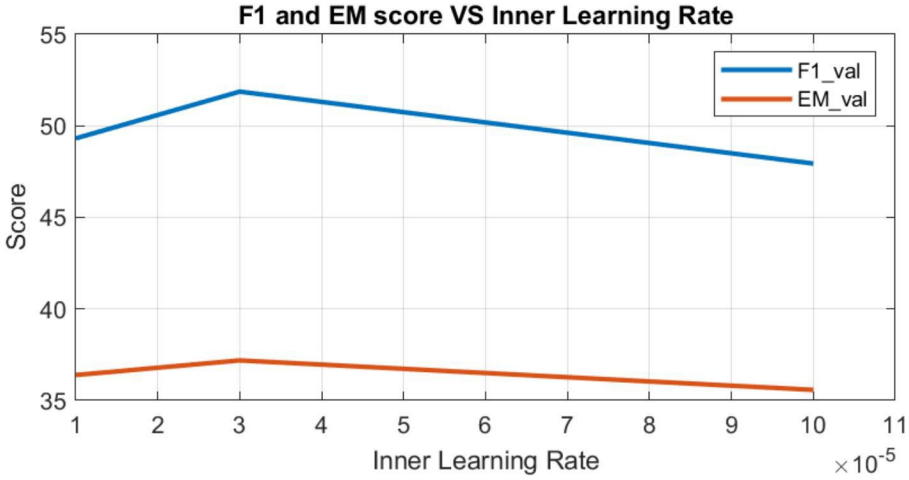

Figure 2: inner learning rate

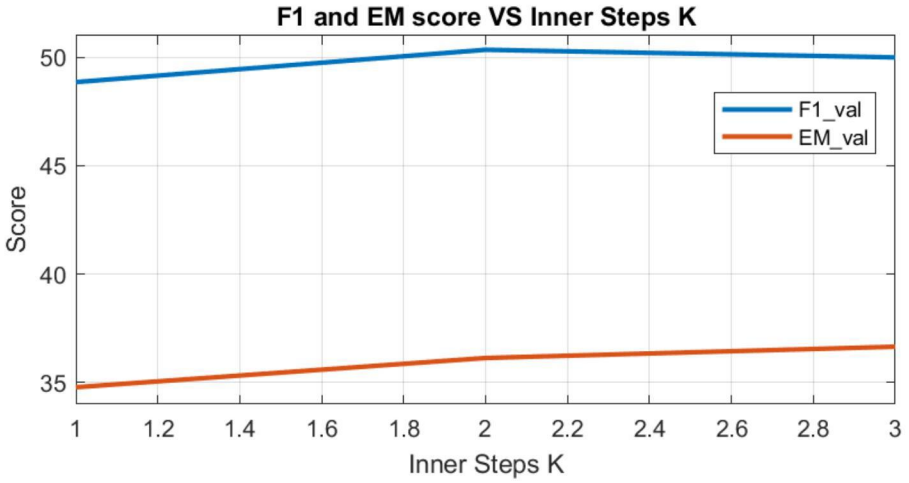

Figure 3: inner training steps

The best result is achieve through the combination of Reptile and data augmentation, though the improvement is marginal and less than what we expected. As we can see from the table, the performance increase by using data augmentation on EM score is more significant than the increase on F1 score. It might imply that our data augmentation strategy make our model model precise, but it helps little on making the model more robust. All four data augmentation strategies do not change the meaning of context too much, but they introduce some changes about the answer's position in the text. Therefore, trained on these data, the model can be more precise about the answer position if it actually knows a generally correct range for the answer span.

Another interesting phenomena is that we rank in the top 10 for the validation leaderboard, but for the test data, we rank among the 20th. One possible explanation is that our model is over-fitting to the small amount of out-of-domain training data. Another possible reason is that the amount of available validation data for each dataset is different from the number of test data. Each out-of-domain dataset has 128 or 127 data points, so they are evenly distributed. For test datasets, RelationExtraction dataset has 6 times more data than RACE dataset. Therefore, if one model does extremely good on RelationExtraction but performs bad on RACE, it actually receives a higher score than a model that perform mediocrely on both datasets.

Also, RelationExtraction is a easier dataset compared to other two datasets. This might be the reason that F1 and EM scores both increase by a noticeable amount from validation leaderboard to test leaderboard. From the presented figure, we can see that our model is hitting the sweet-spot of the hyper-parameters, since the scores are shown to decrease when we moving in either directions on all three plots.

6 Analysis

The model performs best on RelationExtraction dataset where it can get a F1 score of around 80. This dataset contains short contexts, mostly one long sentence, and straightforward questions as other two dataset involves lengthy paragraphs. This result makes sense. As the context becomes longer, it gets harder for the model to understand the context, resulting a performance loss. Also, the model does better on objective questions, such as facts about a person, than on subjective questions, such as opinions of the author. Performance is better for shorter answer span than long answer span. It implies that the model is not really understanding the text, and most likely only inferring from probabilities. This also explains why the baseline model's performance drop significantly when we shifted to out-of-domain datasets. Our intuition is that the model learns a prior distribution from the training data which can be used to calculate starting and ending point for the answer span. When this learned prior distribution does not match the actual underlying distribution, model would give erroneous results.

Pure Reptile achieves a high score within the first epoch of training, then the scores improves marginally. Reptile with data augmentation takes longer to converge on a good score, but the best score is higher. It implies that pure meta learning strategy can work well with very little data available. To get a better initialization of parameters, more training data and diverse tasks are necessary even though it takes longer to train the model. Depending on the data augmentation strategy, we might run the risk of over-fitting.

7 Conclusions & Future Work

In conclusion, we demonstrated that meta learning is an effective strategy that can alleviate the performance decrease due to shift of domain. Meta learning combined with simple data augmentation strategy such as EDA can further increase robustness of the model. Both methods outperforms the baseline model and we obtained satisfactory results on the out-of-domain validation datasets. However, the performance is not exactly ideal on out-of-domain test datesets. The reason behind this performance gap could be over-fitting, as our data augmentation can't introduce data points with new semantic meaning. It can only provide data with different syntactic structures. Another possible reason is the distribution mismatch between validation data and test data for the three out-of-domain datasets.

For future works, we can use model ensemble to further improve the performance. Model ensemble keeps a separate model for each dataset and combine their loss in a rather sophisticated way for

training. This strategy naturally works with meta learning as we conduct evaluation separately for each dataset. Another possible extension is to investigate other meta learning strategies such as model-agnostic meta learning or MAML [4] and its variant first-order MAML [5]. In the Reptile paper, MAML and first-order MAML outperform Reptile on some tasks by a small margin. We can also dig deeper into different data augmentation strategies such as masking some parts of the original context and use BERT model to directly fill those masked portions.

References

[1] Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. Investigating meta-learning algorithms for low-resource natural language understanding tasks, 2019.

[2] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[3] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[5] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *CoRR*, abs/1710.11622, 2017.