Robust Question Answering System

Stanford CS224N {Default} Project

Helena Huang
Department of Electrical Engineering
Stanford University
`yhuang77@stanford.edu`

Abstract

Pretrained models like BERT achieves good performance when we fine-tune it to resourceful QA tasks like SQuAD. However, when we apply the model to out-of-domain QA tasks with different question and passage sources, the performance degraded badly. We discovered that the domain change in passage source is the main contributor to worse performance. We investigated ways to improve robustness of pretrained QA systems by experimenting on different optimizers, freezing and re-initializing model layers during training. We found that AdamW is the best optimizer for training on out-of-domain QA datasets, and freezing just the embedding block of DistilBERT improves model performance the most.

1 Key Information to include

- External collaborators (if you have any): NA
- External mentor (if you have any): NA
- Sharing project: NA

2 Introduction

Large pretrained language models like BERT [1] are proven to have state of the art performance when fine-tuned to question answering tasks with large training data sets (e.g. SQuAD). However, the performance of the model tends to fall off when we try to apply it to question answering tasks in other domains with less available training data. In fact, when we fine-tune DistilBERT on a mixture of high resource and low resource QA tasks, the model performs significantly worse on the QA tasks in the low resource domains.

In addition, previous research shows that fine-tuning pretrained models on low resource tasks involves notable instability [2]. Using different random seeds during fine-tuning could cause the model to have very different performance.

To better understand the impact of domain difference within the same NLP task and causes of instability in low resource learning, we carried out a set of experiments, hoping to find ways to improve our model's performance on QA task in low resource domains.

We analyzed the impact of question and passage source on generalizing in-domain training to out-of-domain application. We also experimented on different optimizers, freezing and re-initializing DistilBERT layers.

[1]

3 Related Work

Suchin Gururangan et al. [3] observe improvement in performance when they apply domain-adaptive pretraining and task-adaptive pretraining to RoBERT on various NLP tasks. Domain-adaptive pretraining refers to training the model on texts from the same domain as the target task with the original masked language modeling (MLM) loss. Task-adaptive pretraining refers to training the model on unlabeled texts from the same task domain with MLM loss. NLP datasets are usually created by labeling a subset of data available for the task, so there exist unlabeled texts from the same task domain. The paper also discovers that pretraining the model on texts from an irrelavent domain harms performance for most of the NLP tasks they investigate. Each of the NLP tasks Suchin Gururangan et al. evaluate come from a single domain, but our question answering tasks involves data from multiple domains. We have multiple sources for our passage and question texts. Therefore, we investigated the effect of different data domain on the same NLP task.

Tianyi Zhang et al. [2] investigate the contribution of optimizer, re-initialization of pre-trained layers and number of training epochs on the stability and performance of fine-tuning on small datasets. Specificallly, they discover that the bias correction term in Adam optimizer is important for the stability of the model. BERTAdam omits the bias correction step and results in larger variance in performance during different random trials. The difference seems to be insignificant for fine-tuning tasks with large training sets, but only manifests itself when the training dataset is small. This observation inspires us to investigate other optimizers, such as Adadelta and Adagrad. We are curious if the nuances in implementation of these optimizers affect performance in low resource fine-tuning.

According to the empirical results of Tianyi Zhang et al. [2], re-initializing pre-trained layers improves performance for most of the low-resource fine-tuning tasks. The effect appears to be more significant when they down-sample the training dataset to make it even smaller. This empirical finding could be explained by the paper of Jason Yosinski et al. on feature transferability[4]. Jason Yosinski et al. points out that deep neural networks trained on images usually learn general features like color blobs in their early layers and task specific features in layers closer to the output. Therefore, it is reasonable that reinitializing the top layers helps the model learn faster and better for low resource tasks, because it eliminates features biased toward the MLM task in the top layers. For tasks with ample training data, the benefit of reinitialization becomes less obvious since the model has enough data to overwrite the task specific features in the top layers anyway.

In our case, we do have enough data to train our model to the QA task, but limited training data for QA task in the domain we are interested in. Therefore, we are investigating the feature transferability across domains within the same task instead of across different tasks. We are interested in whether we can make the analogous argument that earlier layers of our model learn features general to all language domains, while the top layers learn features more specific to the domain of the training data.

4 Approach

Baseline. We will use the baseline provided by the default final project, which finetunes DistilBERT using AdamW on all training data for 3 epochs.

Optimizers. We will be focusing on three different optimizers: AdamW, AdaGrad and AdaDelta. The pseudo code for each algorithm is displayed in the appendix.

AdamW improves upon Adam optimizer by decoupling the regularization step from the gradient update step [5]. Adagrad is an optimization algorithm that picks up on rare features. It assigns lower learning rate to frequently occuring features and assigns higher learning rate to less frequent features [6]. Adadelta builds on Adagrad and includes learning rate adaptation across time in addition to the learning rate adaptation across features [7].

We will use the Pytorch implementation for all three optimizers.

Freezing and Re-initialization DistilBERT for question and answering has an embeddings block with 4 layers, 6 transformer blocks, and an output block with a dropout layer.

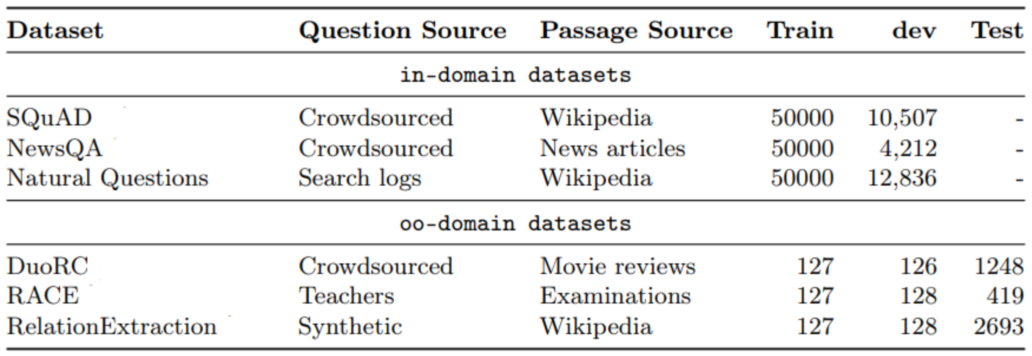
For the freezing experiments, we will disable gradient calculation for the embedding block and various numbers of transformer blocks. Disabling gradient calculation will stop the model from updating weights in the frozen layers.

For the re-initializing experiments, we will re-initialize the output layer's weight and bias with xavier uniform distribution and uniform distribution respectively.

5 Experiments

5.1 Data

We will be using the datasets given by the default project handout: SQuAD, NewsQA, Natural Questions, DuoRC, RACE, and RelationExtraction. SQuAD, NewsQA, and Natural Questions are considered the in-domain datasets, whereas DuoRC, RACE, and RelationExtraction are consider the out-of-domain datasets.



Dataset	Question Source	Passage Source	Train	dev	Test
		`in-domain datasets`			
SQuAD	Crowdsourced	Wikipedia	50000	10,507	-
NewsQA	Crowdsourced	News articles	50000	4,212	-
Natural Questions	Search logs	Wikipedia	50000	12,836	-
		`oo-domain datasets`			
DuoRC	Crowdsourced	Movie reviews	127	126	1248
RACE	Teachers	Examinations	127	128	419
RelationExtraction	Synthetic	Wikipedia	127	128	2693

Figure 1: **Statistics for datasets used for building the QA system for this project.** Table borrowed from the default project handout.

5.2 Evaluation method

We will use the EM score and F1 score to evaluate our model as specified in the default project handout.

5.3 Experimental details

Question and passage domain. We first trained DistilBERT to all in-domain datasets using the same setup as our baseline (opimizing with AdamW for 3 epochs). Then we trained the model to each of the out-of-domain datasets separately for 10 epochs using AdamW.

Optimizer. We trained DistilBERT to the in-domain datasets as described above. Then we tuned the learning rate for each optimizer in roughly multipliers of 3 (e.g. $3e^{-5}, 1e^{-4}, 3e^{-4}$...). We found the best learning rate for AdamW is $3e^{-5}$, the best learning rate for Adagrad is $3e^{-4}$, and the best learning rate for Adadelta is $3e^{-3}$. Using the optimized learning rate, we trained the model on all out-of-domain datasets with each optimizer on 3 different random seeds. We trained the model for 10 epochs because we found that the performance on development set generally peaks and NLL loss plateaus by 10 epochs.

Freezing out-of-domain. Using the best optimizer we found earlier, we experimented on freezing the embeddings block, and 0, 2, 4, 6 transformer blocks of DistilBERT while training to the out-of-domain datasets. We trained the model for 10 epochs for each experiment.

Freezing in-domain. We froze the embedding block and 2 transformer blocks of the DistilBERT model while training on the in-domain datasets using AdamW with learning rate $3e^{-5}$ for 3 epochs. With this model trained on in-domain QA task, we further trained it on the out-of-domain datasets for 10 epochs.

Re-initialization. We re-initialized the output layer's weight and bias with xavier uniform distribution and uniform distribution while training on the out-of-domain datasets for 10 epochs.

5.4 Results

Question and passage domain. The table below shows the F1 and exact match evaluation of the model when trained to the three out-of-domain datasets separately. The baseline is where we trained our model to all out-of-domain datasets at the same time.

dataset	F1	EM
RACE	39.60	26.56
DuoRC	41.62	29.37
RelationExtraction	73.70	54.69

Optimizer. The table below shows the evaluation on the development set. AdamW seems to be the optimizer that achieves the best result.

Optimizer	F1	EM
AdamW	50.09	35.08
Adagrad	49.17	33.77
Adadelta	49.25	35.34

Freezing and re-initialization. The table below shows evaluation on the dev set. We found that just freezing the embedding block of DistilBERT achieves the best performance, so we evaluate it on the test set. The ressult is also shown in the table below.

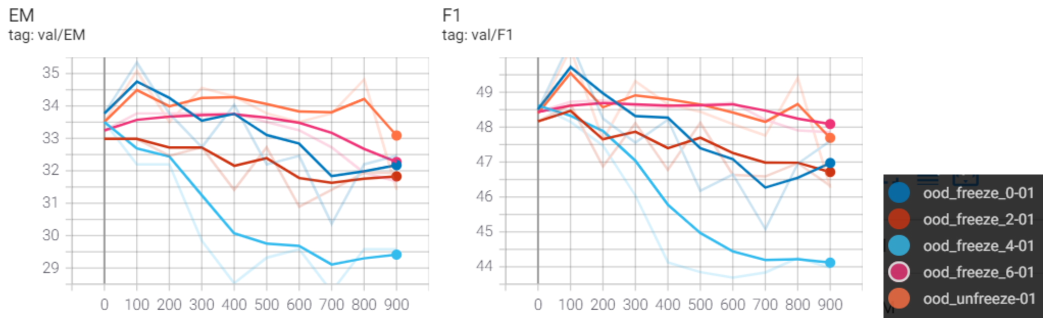


Figure 2: **Freezing Different Numbers of Transformer Layers for Out-of-Domain Training.** The plots show the exact match and F1 score during training. The shadowy curves show the actual values while the solid curve show the smoothed values.

(dev set)	F1	EM
ood-freeze-0	50.45	35.34
ood-freeze-2	48.65	32.98
ood-freeze-4	48.62	33.51
ood-freeze-6	48.76	33.77
ind-freeze-2	48.47	32.72
re-init	49.15	34.03

6 Analysis

Question and passage domain. DuoRC has the same question source with two of the large in-domain datasets. RelationExtraction has the same passage source with two of the in-domain datasets. RACE has different question and passage source from all in-domain datasets. The model performs the best on RelationExtraction QA task after training on the in-domain datasets, and worst on RACE. It seems that both question and passage source affect the performance of the model. The model seems to benefit most from training on data with same passage source as the target QA task domain. This could be because the passage contributes the most texts to the QA task. Nevertheless, training the model on data with similar quesion source also helps improve performance.

Optimizer.

Freezing out-of-domain. Freezing early layers of DistilBERT reduces the number of parameters to train, so we expect it to be more stable and less likely to overfit the training data. In Figure 2, we can see that the randomness in the evaluation scores decreases as we increase the number of transformer blocks we freeze. The curve where we freeze all 6 transformer layers appears to be the most smooth, which is what we expected. However, it doesn't out perform the baseline. The randomness seems to help it achieve a high peak during the dev set evaluation.

Freezing in-domain. We expect freezing early layers during in-domain training to be helpful because early layers might contain features general to all domains. It seems that only freezing the embedding layer helps the model perform better than the baseline.

Re-initialization. Re-initialization might help get rid of features biased toward the QA training domain, but it might also lose features helpful for QA tasks in general. From our empirical result, it seems that it doesn't achieve better performance than the baseline. Therefore, losing features for QA tasks outweights the benefit of removing domain bias.

7 Conclusion

We found out that the passage source domain is an important factor on how well in-domain learning can transfer to out-of-domain application for QA tasks. If we could find more training data with the same passage source as our targeted QA domain, then it would help improve the performance of our model on the targeted QA task.

We also discovered that out of all the freezing and re-initialization experiments, freezing just the embedding block achieves the best performance.

We spent a long time experimenting on freezing transformer layers when training on in-domain datasets, but later discovered a bug in the code so we had to redo most of our experiments. Yet we didn't have enough time to carry out all experiments we intended to do. If we have more time or computing resources in the future, we would experiment on freezing different numbers of transformer layers during in-domain QA task training. We might also try conducting the out-of-domain freezing and re-initialization experiments after freezing different numbers of layers during in-domain training.

We would also run all experiments on multiple random seeds for stability analysis in the future. There results we reported could be largely affected by the random seed we used, and we might achieve different results for different random seeds.

References

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

[2] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*, 2020.

[3] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

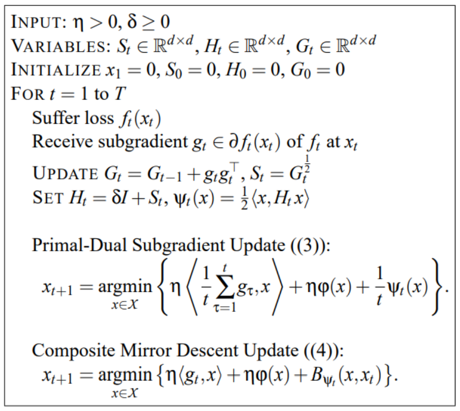
[4] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

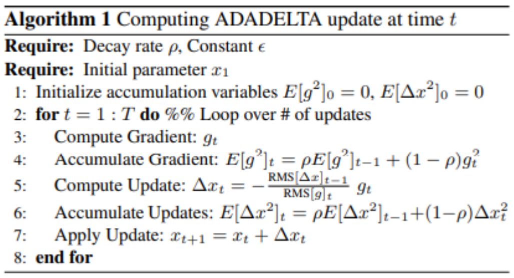
[7] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A Appendix (optional)

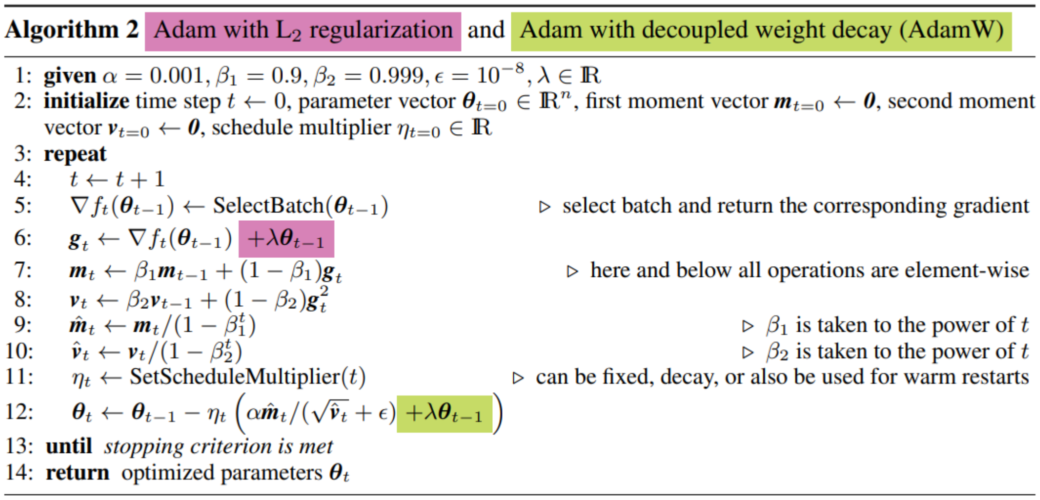


```
INPUT: $\eta > 0, \delta \geq 0$
VARIABLES: $S_t \in \mathbb{R}^{d\times d}, H_t \in \mathbb{R}^{d\times d}, G_t \in \mathbb{R}^{d\times d}$
INITIALIZE $x_1 = 0, S_0 = 0, H_0 = 0, G_0 = 0$
FOR $t = 1$ to $T$
  Suffer loss $f_t(x_t)$
  Receive subgradient $g_t \in \partial f_t(x_t)$ of $f_t$ at $x_t$
  UPDATE $G_t = G_{t-1} + g_t g_t^\top, S_t = G_t^{\frac{1}{2}}$
  SET $H_t = \delta I + S_t, \psi_t(x) = \frac{1}{2}\langle x, H_t x\rangle$

  Primal-Dual Subgradient Update ((3)):
    $x_{t+1} = \underset{x\in X}{\operatorname{argmin}} \left\{ \eta \left\langle \frac{1}{t}\sum_{\tau=1}^{t} g_\tau, x \right\rangle + \eta\varphi(x) + \frac{1}{t}\psi_t(x) \right\}.$

  Composite Mirror Descent Update ((4)):
    $x_{t+1} = \underset{x\in X}{\operatorname{argmin}} \{\eta\langle g_t, x\rangle + \eta\varphi(x) + B_{\psi_t}(x, x_t)\}.$
```

Figure 2: ADAGRAD with full matrices



```
Algorithm 1 Computing ADADELTA update at time $t$
Require: Decay rate $\rho$, Constant $\epsilon$
Require: Initial parameter $x_1$
1: Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta x^2]_0 = 0$
2: for $t = 1 : T$ do %% Loop over # of updates
3:     Compute Gradient: $g_t$
4:     Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$
5:     Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$
6:     Accumulate Updates: $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$
7:     Apply Update: $x_{t+1} = x_t + \Delta x_t$
8: end for
```



```
Algorithm 2 Adam with L$_2$ regularization and Adam with decoupled weight decay (AdamW)
1: given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2: initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment
   vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: repeat
4:     $t \leftarrow t + 1$
5:     $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$    ▷ select batch and return the corresponding gradient
6:     $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
7:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$    ▷ here and below all operations are element-wise
8:     $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
9:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1-\beta_1^t)$    ▷ $\beta_1$ is taken to the power of $t$
10:    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1-\beta_2^t)$    ▷ $\beta_2$ is taken to the power of $t$
11:    $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$    ▷ can be fixed, decay, or also be used for warm restarts
12:    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha\hat{\boldsymbol{m}}_t/(\sqrt{\hat{\boldsymbol{v}}_t}+\epsilon) +\lambda\boldsymbol{\theta}_{t-1}\right)$
13: until stopping criterion is met
14: return optimized parameters $\boldsymbol{\theta}_t$
```

Figure 3: **Pseudo code for AdamW, Adagrad and Adadelta.** Figures reproduced from [5], [6], [7]