Building a Robust QA system

CS 224N Default Final Project

Andrew Hwang, Max Pike, Pujan Patel
Department of Computer Science
Stanford University
{ahwang22}, {mpike27}, {pujan} @stanford.edu

Abstract

Researchers today prioritize their time by building increasingly complex models that are harder to interpret and debug. The goal of this project is for us to discover how noninvasive techniques can be equally as effective. Namely, we explore how accuracy improves with hyperparameter tuning, various different methods of learning rate decay, and layer freezing. We also analyze the effects of data-side augmentations such as backtranslation, synonyms, masked learning, and upsampling. The last area of exploration is an altered loss function that biases against length. Our results indicate that layer-wise fine-tuning and data augmentation each lead to better accuracy and generalization, and the new loss function leads to predictions closer in length to the ground truth response length.

1 Key Information to include

- Mentor: No external mentors
- External Collaborators: N/A
- Sharing project: N/A

2 Introduction

The holy grail of nlp question answering for years has been creating a model that can generalize well to shifts in domain. In other words, no matter what kind of questions you ask, the model should be able to adjust well to new kinds of questions. The Robust QA track for the default project emulates this problem by giving two types of datasets - an in-domain and an out-of-domain (ood) dataset. The in-domain dataset is substantially larger than the train and validation subsets of the ood dataset, but the test set is solely ood.

The most difficult aspect to this type of problem is to be able to learn general question answering features as well as domain specific aspects without overfitting to the specific datasets. It is a fine line, because you want to learn question answering from the thousands of examples in the in-domain, but you don't want to memorize how in-domain questions are answered. Similarly for ood, you want to learn the patterns of the ood dataset, but you don't want to memorize the few ood examples in the train and validation sets.

Our approach was five-pronged: exhaustive hyperparameter fine-tuning, data augmentation, mask-language modeling, loss function adaptation, and upsampling. The main goal of each of these methods is to help the model learn the general question answering framework of the in-domain training set and recognize the patterns of the ood training set, all while not overfitting. Our experiments show that data augmentation, upsampling, and hyperparameter tuning can increase F1 and EM by a couple points. Additionally, masked learning and custom loss functions both show promise and is an area of future work for robust question answering models.

3 Related Work

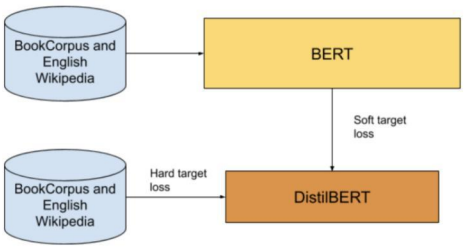

Figure 1: BERT Knowledge distillation: DistilBERT

3.1 DistilBERT

BERT is a popular, transormer-based model for question answering, and Sanh et al. created a more lightweight version of BERT called DistilBERT (1). Compared to BERT, DistilBERT is 40% smaller and 60% faster than BERT while remaining 97% as accurate. The way that they were able to accomplish this was by creating a triple loss during pretraining which accounts for the biases learned by the larger models. The model accuracy can also be attributed to the knowledge distillation approach which uses the larger model to guide the smaller model during the learning period. The DistilBert model we use has 102 different parameters with an embedding layer, 6 transformer blocks with multi-head self attention, followed by a linear output layer.

3.2 Hyperparameter Finetuning

Zhang et al., aim to understand some of the problems of few-sample fine-tuning of BERT contextual representation. Despite being considered one of the most state-of-the-art performing pre-trained NLP models, $BERT$ and $BERT_{Large}$ remain challenging for practitioners to fine-tune (1). Specifically, when training $BERT$ on smaller datasets where pre-training would be most advantageous, it's often seen that identical training processes can result in significantly different and sometimes degenerate models. This instability found in the training process has forced researchers to resort to repeatedly rerunning experiments for model selection, increasing model deployment time and costs while simultaneously making model performance comparisons difficult (2). As a result this paper aims to investigate why these instabilities in fine-tuning occur in BERT by analyzing commonplace BERT fine-tuning techniques and offer improvements on these techniques to improve remedy these instabilities.

This paper identifies three proposed methods to improve few-sample fine-tuning with BERT and finds that other recently proposed methods to improve fine-tuning do not actually have as strong relative impact when modifying the fine-tuning process to include the three proposed methods outlined. The researchers benchmark the performance of their improvements to fine-tuning methods by studying BERT's fine-tuning on four datasets that cover natural language (RTE), paraphrase detection (MRPC), sentiment classification (STS-B), and linguistic acceptability (CoLA), all datasets with fewer than 10k training samples where BERT fine-tuning on these datasets is known to be unstable (2). The three proposed methods Zheng show to improve performance are correcting the gradient bias commonly used Adam optimizer that was omitted in many BERT packages, reinitializing the top L layers rather than just one specialized output layer can increase model performance, and fine-tuning BERT for more training epochs can increase both model performance and stability.

3.3 Data Augmentation

Data augmentation has become a popular approach for many machine learning and NLP specific projects. Wei and Zou present four easy data augmentation techniques to use to boost performance for text classification. The techniques are synonym replacement, random insertion, random swap, and random deletion. For the study, they looked into the effects that these techniques have on convolutional and reccurent neural networks. For both models, they were able to increase accuracy

by 1 percentage point on average without changing any other parameters. Additionally, they show through latent space analysis that their augmentations conserve the accuracy of the labels.

Longpre et al. also explored the impacts of data augmentation but in the context of domain agnostic question answering (3). They show that data sampling techniques and augmentation via back translation can increase the performance of question answering models. They recognized that roughly half of the examples at test time would be negative examples (in other words, the answer would be "NO ANSWER"), so they sampled negative examples at a much higher rate at train time to reflect this distribution. For back translation, they translated the queries and contexts into a pivot language and then translated them back into the source language in order to create noisy training examples. Both techniques performed better than the baseline, and showed promise for the efficacy of data augmentation and sampling.

3.4 Masked Learning Model

Gururangan et al. demonstrate the powerful affects of task-adaptive pretraining, TAPT, when working with little amounts of unlabeled task specific data. Specifically, they recognized that TAPT provides large gains for ROBERTA [4]. They found that even with a far smaller pretraining corpus that is very task-specific, TAPT helped boost performance on task-specific predictions. Recognizing the parallels of having a small out of domain corpus as well, we set to use TAPT to see if the additional pretraining task can indeed help boost performance.

4 Approach

We go into an overview of our baseline model that we train and the following four main approaches that we take for improving DistilBert Model performance on question answering on various domain shifts.

4.1 Baseline Model:

Our baseline model was obtained by taking the vanilla DistilBertForQuestionAnswering and training on the in-domain datasets using early stopping with varying patience and learning rates. We select our baseline as the model which has the best F1 on the out of domain validation dataset. Our baseline chosen achieved a **F1: 70.09** and **EM: 53.87** on the in-domain validation, a **F1: 48.35** and **EM: 32.46** on the out of domain validation, and a **F1: 57.19** and a **EM: 37.89** on our upsampled out of domain validation. The weights for the best baseline model were then saved and used for all subsequent experiments.

1. Hyperparameter fine-tuning: We performed grid search on the learning rate and analyzed the impacts of layer-wise learning rate decay, weight decay, layer-wise learning rate initialization, layer freezing, and layer reinitialization. The goal of each of this techniques is to focus the learning on the outermost layers of the transformer architecture which are more prone to shifts in domain, as opposed to the inner layers which should remain relatively unchanged.
2. Data Augmentation: We used two methods: backtranslation and synonyms. The goal of this is to prevent general overfitting and reinforce learning relationships between words.
3. End-to-End Masked Language Modelling: Utilized as a pretraining task on the indomain and the ood datasets so as to further reinforce relationships between words and to reduce overfitting
4. Custom Loss Function: Our loss function includes an extra term that penalizes the length of the predictions. The benefit of this approach is that it guides the model towards making predictions according to the average length of predictions
5. Upsampling : We increased the size of the the ood train and validation sets so as to reflect the actual distributions in the test set.

4.2 Hyperparameter fine-tuning

We implement two proposed methods from Zheng et al (2) to improve few-sample fine-tuning with BERT, namely reinitializing the top L layers of our model and fine-tuning BERT for greater training

epochs as many researchers have been found to not training for sufficiently long even despite few samples. We confirm that reinitializing the top L layers rather than just one specialized output layer can increase model performance. In theory this makes sense as object recognition transfer learning shows that lower pre-trained layers learn more general features while high layers specialize to the pre-training task at hand. We conduct experiments of reinitializing the top $0, 1, 2$ layers and the output layer with the original BERT weight layer initializations. Also, for our experiments we conduct various tests of running for varying number of epochs with early stopping patiences of 2,5, and 10 determined by failure to improve in F1.
In addition to the above two ideas we implement other common fine tuning practices of layer wise decay as well as performing a grid search to identify optimal hyperparameters. For our layer wise decay, we set varying learning rates based for the top 2 layers of our transformer, the middle two layers of our transformer, and the bottom two layers of our transformer. Likewise, we also experiment with freezing the bottom L layers. These are all motivated by the understanding that the top L layers are more important to learn domain specific transfer on our out of domain dataset and our bottom layers should be limited by how much they change.

4.3 Data Augmentation

Since our train and validation sets for ood are so small, it is commonplace for researchers to artificially create more data to feed into the model. The focus of the data augmentation is to create noise in both the question and the context, but not the answers. For this project, we utilized two different approaches for created new, augmented data.

One method was via back translation. Back translation is the process by which you convert a series of text into a "pivot" language, and then translate it back into the original language. The goal of this method is to create noise in your dataset while also keeping the same structure of the sentences. We utilized Google's Google Translate API for translations, and originally, we chose spanish as our original pivot langauge. However, after qualitative analysis, we found that the back translation was much too similar to the original text. This can be attributed to how comprehensive Google Translate has become. To add another layer of noise, we decided to use two pivot languages: Russian and Japanese. This create an ample amount of noise while still preserving the structure of the questions and contexts.

The second method as proposed by Wei et al. (5), is synonym replacement. To do this, you choose some parameter α such that $0 \leq \alpha \leq 1$. The number of synonyms that you replace, n can be expressed as $n = \alpha * len(sent)$. The words that are replaced are randomly selected, and the synonyms themselves come from the python dictionary API. If no synonym is found, then we keep the original word in.

Below is an example augmentation for a context sentence.

- Original: An enraged posse of men descend on the isolated Seven Doors Hotel deep in the swamps.
- Back Translation: Restored Posse men are down on the 7 door hotel isolated on the depth of wetlands.
- Synonym Replacement: An angered posse of personnel flop on medication isolated cardinal exterior stay over deep interior the swamps.

4.4 Masked Language Modeling

Due to the scarcity of the out of domain data, we need to be able to learn general features key to question answering, so that it can be translated well during a domain shift. By adding an additional pretraining stage of Masked Language Modeling (MLM) on the out of domain datasets, the goal is to learn these general features. We performed experiments of incorporating this additional pretraining stage by leveraging the DistilBertForMaskedLM model from huggingface. This model was then trained on the contexts of the out of domain datasets. 15% of the words in the context paragraphs were randomly chosen to be replaced. Of the words to be replaced, 80% were replaced with a [MASK] token, 10% were replaced with another random word, and the remaining 10% were chosen to be the same word. The model was then trained to predict the true context paragraph using a traditional MLM loss function similar to Figure 2. The process of transferring an input sequence into

a masked sequence was followed using Gururangan et al. Due to time and resource constraints the MLM pretraining stage was only run for 50 epochs as opposed to 100 epochs in TAPT [4].

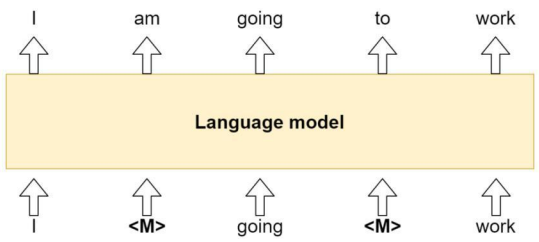

Figure 2: DistilBERTMaskedLM process

4.5 Loss Function Modification

We experiment with an alternative loss function than the standard cross entropy loss function. Currently the loss function adapted to our BERT model is

$$loss(start - logits, start - positions) + loss(end - logits, end - positions)$$

where the loss is defined as the standard cross entropy loss of $H_{y'}(y) := -\sum_i y'_i \log(y_i)$ and codified by

$$loss(x, class) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

In examining our model performance, however, we note that the distribution of predicted start and end positions produces answers that while accurate, are often times significantly longer than true answer. Nearly all of the true answers are shorter than 5 words while our model yields answers of up to 10 to 20 words. As such, we were motivated to use a brevity regularization term that reinforces our model to predict shorter answers.

Due to the nature of our model yielding start and end logits and the length of the answer being determined by the difference of the argmax of both logits, we are unable to create a direct penalty due to the lack of gradient produced by argmax. As such, we offer an alternative approach to reinforce this shorter length. We add an additional loss term with regularization hyperparameter λ of

$$loss(x, mean - positions)$$

where

$$x = \frac{1}{2}(start\_logits + end\_logits)$$

$$mean - positions = \frac{1}{2}(start\_positions + end\_positions)$$

The current loss functions drive our model to correctly identify given some context the distribution of mean and variance that our true start and end are positioned at. Mathematically, the correct value f of the support of the BERT model can be computed from our model that predicts p and has variance σ^2, namely as such:

$$f(x; p, \sigma) = -\frac{1}{2}\left(\log\left(2\pi\sigma^2\right) + \frac{(x - p(x))^2}{\sigma^2}\right)$$

In examining this, we see that while this can collectively drive our model to predict the correct distributions of the start and end positions as well as understanding their respective variances - this process is largely done independently aside from the shared context and only implicitly drives the model to improve the predicted variance. Our loss function adapts both the start and end logit positions conditional on their final prediction and enforces that their distribution should collectively center around the correct answer. While this step of centering is implicitly achieved by the original loss function, this added loss incorporates the conditional variance of typical response answer by learning the average distribution between start and end directly. Mathematically, instead of learning simply σ_{start} and σ_{end}, we also place greater weight on our model to learn $\sigma_{lengthofresponse}$. We see that in adding this loss function, we do in fact see brevity constraint enforced and shorter answers.

4.6 Upsampling training and validation datasets

This was a specific approach we took to solely improve testing performance. We note that the distribution of the testing dataset on the robustqa task is substantially different than the distribution of the traininig and validation datasets provided. The out of domain datasets include samples from DuoRC, RACE, and RelationExtraction datasets and the breakdown of representation from the training, validation, and testing datasets is 127/127/127, 126/128/128, and 1248/419/2693 respectively. We can see that the training distribution and validation distribution are largely identical with even splits however the testing distribution is closer to a 3-1-7 split. To remedy this, we upsample our training and validation datasets.

Note that a weighted loss function where missed predictions for classes with a higher frequency in the test dataset receive a higher penalty could have been utilized as well. However, we argue that upsampling can be more stable than weighted loss when training using mini batches. Instead upsampling and weighted loss are effectively the same when training using a single batch and updating your parameters once a epoch but without the unstable learning. We consider downsampling as well but considering the relative small size of our dataset, we simply upsample our datasets to reflect the same testing distribution.

5 Experiments

5.1 Data

The datasets that we will be using are the Stanford Question Answering Dataset (SQuAD), Natural Questions, and NewsQA as our in domain information. Both the SQuAD and NewsQA datasets were crowdsourced while Natural Questions was stripped from Search Logs. The passage sources for both SQuaD and Natural Questions were from Wikipedia and NewsQA was sourced from various News articles. The train dev split for the SQuAD, Natural Questions, and NewsQA datasets are 50,000/50,000/50,000 and 10,507/4,212/12,836 respectively.

We will also be utilizing three different out-of-domain datasets to see how well our model can react to shifts in domain for testing. The out of domain datasets are DuoRC, RACE, and RelationExtraction. The question sources were crowdsouroced, from teachers, and built synthetically respecitvely and were collected from movie reviews, examinatiionis, and wikipedia respectively. The train dev test splits for the DuoRC, RACE, and RelationExtraction datasets are 127/127/127, 126/128/128, and 1248/419/2693 respectively.

For our augmented datasets, we perform back translation and data augmentation for each sample in the ood train sets. If the augmented data causes the answer index to fall out of range, we remove that sample from the dataset. Once we have the augmented data, we upsample it according to the test distribution.

5.2 Evaluation method

The main evaluation metrics for this project are Exact Match (EM) and F1. However, it is important to note that the EM and F1 scores vary depending on which datasets and distributions you are evaluating on. The train and validation ood datasets have even distributions with the average F1 on the leaderboard around 50. The test set is from a distribution, and the average F1 on the leaderboard is around 60. Due to this disparity, we changed our evaluation metric to be representative of the test distribution. For the train and val ood datasets, we multiplied the duorc dataset by 3 and the relation extraction dataset by 7 which is a rough approximation of how they are distributed in the test dataset. When the EM and F1 on the validation set are mentioned, note that these numbers are referring to the upsampled ood validation set unless specified otherwise.

We also look into the efficacy of our loss function in how it impacts the length of its predictions. So, we calculate the average length of the predictions with and without the custom loss function. We want to see how our model performs on each of the datasets specifically, so we look at EM and F1 scores for each of the ood datasets. Lastly, we perform qualitative analysis on the predictions of our model to find patterns on the misses of our model.

	F1	EM
Baseline	57.19	37.89
Finetune output layer	58.35	38.39
Finetune output layer + 1 transformer layer (1L)	58.76	38.52
Finetune output layer + 2 transformer layer	57.07	37.04
Finetune 1L with frozen bottom layer (1LF)	58.95	38.81
Finetune 1LF with layerwise decay (1LFD)	58.95	38.81
Finetune 1LFD and MLM	57.15	37.39
Finetune 1LFD and back translation (1LFDB)	56.41	38.09
Finetune 1LFD and synonym translation (1LFDS)	60.27	38.44
Finetune 1LFD and both data augmentations (1LFDBS)	58.89	39.73
Finetune 1LFDS with added loss (1LFDSL)	59.35	38.24

Table 1: Results of various experiments with optimized learning rate parameters and early stopped on upsampled out of domain dataset

	DuoRC F1	DuoRC EM	Relation Ex F1	Relation Ex EM	Race F1	Race EM
Baseline	39.63	26.19	74.46	49.22	33.13	19.53
Finetune 1LFDS	35.93	25.81	75.03	51.56	32.75	19.53
Finetune 1LFDSL	40.28	30.16	69.13	43.75	35.53	23.44

Table 2: Breakdown of performance on specific datasets

5.3 Experimental details

For our experiments we created a script to conduct a grid search over every added feature to try and identify the strongest combination of features and report our best models as a function of their F1 performance on the upsampled dataset. For example, we conduct a grid search over the following features batch sizes of [8,16], learning rates of [1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4], patience [2,4], number of epochs, weight decay, number of output layers to fine tune on, and flags of whether we were or were not using MLM, back translation, synonym translation, or an added loss function. We note that we train to exhaustion, stopped only by a patience factor determined by F1 as suggested by the paper Zheng et al which advises training for as long as possible.

5.4 Results

In table 1, we report our best F1 and EM of our various models on our upsampled dataset. For our submission to the testing leaderboard, we selected to submit our baseline, Finetune 1LFDS, and Finetune 1LFDSL. Our best model on the testing leaderboard was our Finetune 1LFDS model with an E1 and FM of 38.165 and 57.608. This was slightly worse than anticipated from our prediction of performance compared to table 1. Overall, we see also that on the validation dataset, when using our new loss function, we have an average response length of 2.8 instead of 3.5. We analyze these results in the following section.

6 Analysis

One potential reason we perhaps seeing these underestimates of performance is that our upsampled dataset reinforces the collective overfitting on the validation Relation Extraction dataset which only had few samples to begin with and as such may suffer from relative performance on the testing dataset. We tried to diagnose why this occurred by analyzing performance on each of the individual datasets per table 2. We see that overall, we are seeing significant focus of performance on Relation Extraction dataset regardless of finetuning as indicted by the baseline. We see that this was likely due to these baselines being all together trained on the indomain datasets which were mostly consisting of samples drawn from Wikipedia, just as in the case of the relation extraction dataset. As such, our upsampling may have not been as effective as initially anticipated and instead lending our model to actually train more on the other two datasets may have been more effective to overall improving our model's performance. In analyzing our average response length on the validation set, we see that

incorporating our added loss successfully drove down the average response length from the baseline model. We see that this loss also regularizes and generalizes improvement on the other two datasets, possibly supposing that understanding the underlying distribution of the response length of the minor out of domain datasets is important and a much more defining parameter for performance on these datasets.

One pattern that we saw in the dataset was that most of the answers were 4 words or less. By looking at our model's output, we saw that a lot of times, we were locating the right area of the text but the answer was too long. This is why we implemented the custom loss function so that the predictions are closer in length to the actual answers. An example of our loss function working was an example where the answer was "Santa Rosa". Before the loss function, our model predicted "Santa Rosa, California" and afterwards it predicted "Santa Rosa." An area of future work is to further optimize and refine the lambda parameter which penalizes the length of the predictions.

Another pattern that we saw in our model's misses was it really struggled with questions with long contexts. This can be attributed to the fact that the model is still overfitting to the data and lacks comprehension of the large contexts. There are examples in the dataset where the model gets tripped by synonyms. For example, one of the questions is "Who does J reveal the mission to?" The answer starts with "J tells K." However, the model predicts "Agent W. He notifies them of Griffin." It most likely does this because 3 words earlier, the word "reveal" appears. The model is most likely searching equally for each of the words in the question. It should be able to recognize that "tells" and "reveals" are synonymous. This is a downfall with this model and is another area for future improvement.

Lastly, When visualizing text predictions when finetuning our model on the out of domain datasets after MLM pretraining, we observed that many of our answers were far too short and often just one word. We also observed that the addition of this new pretraining step of performing MLM slightly reduced our performance in both F1 and EM in various setups. When pretraining using a different objective than the traditional task, it is important to analyze which layers to freeze or reduce learning rate on so no extra noise is added to the weights learned. Although we tried various setups with freezing different layers as well as varying learning rates during our grid search, we noticed the incorporation of MLM pretraining consistently provided worse results than the omission of this step when holding all other variables constant. We believe this to have been due to noise possibly in our new weights due to being adapted to a new task. This problem could be further exacerbated for two reasons: 1) due to the sparse nature of our out of domain datasets when finetuning for QA and 2) due to additional noise that can result from our data augmentation techniques to the out of domain datasets.

7 Conclusion

Our main conclusions support that fine tuning and data augmentation methods were the most critical in improving performance on question answering systems under domain shifts. We see that data augmentation (back translation and synonym translation) however can sometimes be too noisy depending on how many sequences of languages we filter through, suggesting that future work look into understanding an opotimal number of languages. We have inconclusive results on the quality of MLM and upsampling our dataset as we see marginal improvement at best from these methods, potentially suggesting that they are not worthwhile pursuing for such few sample finetuning. Lastly, we see that for future work further investigation into our added loss function could be potentially useful in regularizing response length. Lastly, given the large difference in response F1 score based on class, we suggest that by adding a classifier that first predicts the type of dataset the question came from can be useful and then having three separate models for each class could be one strong area of future research.

8 Code

https://github.com/pujanpatel24/cs224-robust-project

References

[1] Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," NAACL-HLT, 2020.

[2] T. Zhang et al, "Revisting few-sample BERT fine-tuning" ICLR, 2021.

[3] S. Longpre, Y. Lu, et al, "An Exploration of Data Augmentation and Sampling Techniques for Domain-Agnostic Question Answering" Apple Inc., 2019.

[4] S. Gururangan, A. Marasović, et al, "Don't Stop Pretraining: Adapt Language Models to Domains and Task" ACL 2020., 2020

[5] J. Wei, and K. Zou, "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks," Dartmouth College and Georgetown University, 2019.

[6] K. Lee, J. Devlin, et al, "BERT: Pre-training of deep bidirectional transformers for language understanding," NAACL-HLT, 2019.